

Bayesian Prognostic Framework for High-Availability Clusters

Premathas Somasekaram

Doctor of Philosophy

University of York
Computer Science

December 2021

Abstract

Critical services from domains as diverse as finance, manufacturing and healthcare are often delivered by complex enterprise applications (EAs). *High-availability clusters* (HACs) are software-managed IT infrastructures that enable these EAs to operate with minimum downtime. To that end, HACs monitor the health of EA *layers* (e.g., application servers and databases) and *resources* (i.e., components), and attempt to reinitialise or restart failed resources swiftly. When this is unsuccessful, HACs try to *failover* (i.e., relocate) the resource group to which the failed resource belongs to another server. If the resource group failover is also unsuccessful, or when a system-wide critical failure occurs, HACs initiate a complete system failover.

Despite the availability of multiple commercial and open-source HAC solutions, these HACs (i) disregard important sources of historical and runtime information, and (ii) have limited reasoning capabilities. Therefore, they may conservatively perform unnecessary resource group or system failovers or delay justified failovers for longer than necessary.

This thesis introduces the first HAC taxonomy, uses it to carry out an extensive survey of current HAC solutions, and develops a novel Bayesian prognostic (BP) framework that addresses the significant HAC limitations that are mentioned above and are identified by the survey. The BP framework comprises four *modules*. The first module is a technique for modelling high availability using a combination of established and new HAC characteristics. The second is a suite of methods for obtaining and maintaining the information required by the other modules. The third is a HAC-independent Bayesian decision network (BDN) that predicts whether resource failures can be managed locally (i.e., without failovers). The fourth is a method for constructing a HAC-specific Bayesian network for the fast prediction of resource group and system failures. Used together, these modules reduce the downtime of HAC-protected EAs significantly. The experiments presented in this thesis show that the BP framework can deliver downtimes between 5.5 and 7.9 times smaller than those obtained with an established open-source HAC.

Contents

List of Figures	xiii
List of Tables	xix
Nomenclature	xxiii
I Introduction and Background	1
1 Introduction	3
1.1 Context and Motivation	3
1.2 Research Hypothesis	5
1.3 Research Questions	6
1.4 Research Contributions	8
1.4.1 Taxonomy of High Availability Clusters	8
1.4.2 Survey of High Availability Clusters	8
1.4.3 Holistic Modelling Technique for High Availability	8
1.4.4 Bayesian Prognostic Framework Preparation	8
1.4.5 Bayesian Decision Network for Predicting Locally Manageable Resource Failures	9
1.4.6 Bayesian Network for Failure Propagation and Prediction	9
1.4.7 Bayesian Prognostic Framework for High-Availability Clusters	10
1.5 Thesis Organisation	10
2 Uses and Architecture of High-Availability Clusters	13
2.1 Key Concepts and Terminology	13
2.2 Layers of Enterprise Application	15
2.2.1 HAC Architecture	16
2.3 Disaster Recovery and High-Availability Clusters	20

Contents

3	Taxonomy and Survey of High-availability Clusters	21
3.1	Taxonomy	21
3.1.1	A: Deployment Pattern	22
3.1.2	B: Application Areas	25
3.1.3	C: Type of Cluster	26
3.1.4	D: Topology	27
3.1.5	E: Cluster Management	30
3.1.6	F: Failure Detection and Recovery	34
3.1.7	G: Consistency and Integrity	37
3.1.8	H: Data Synchronisation.	40
3.2	Survey of High-availability Cluster Solutions	43
3.2.1	Survey Methodology	43
3.2.2	Approach for Selecting the Surveyed HACs	44
3.2.3	HAC Analysis Methodology	47
3.2.4	Survey Results	48
3.2.5	Analysis of the Survey Results	52
3.2.6	Limitations, Challenges and Opportunities	57
3.2.7	Limitations	57
3.2.8	Challenges	59
3.2.9	Opportunities	60
3.3	Summary	62
4	Bayesian Networks	63
4.1	Bayes Theorem	63
4.2	Bayesian Statistics	63
4.3	Bayesian Networks	64
4.3.1	Representation of Probabilities	66
4.3.2	Types of Bayesian Networks	66
4.3.3	Learning in Bayesian Networks	68
4.3.3.1	Structure Learning	68
4.3.3.2	Parameter Learning	68
4.3.4	Learning Modes	70
4.3.5	Inference	70
4.3.6	Motivation for Using Bayesian Networks	70
4.4	Bayesian Decision Networks	72
4.4.1	Representation of Conditional Probabilities	73
4.4.2	Inference	73
4.5	Constructing and Using a Bayesian Network/Decision Network	73
4.6	Summary	74

II	A Bayesian Prognostic Framework for High-Availability Clusters	77
5	Holistic Modelling Technique for High Availability	79
5.1	Introduction	80
5.2	Related work	81
5.3	Holistic Modelling Technique for High Availability	82
5.3.1	Terminology	82
5.3.2	HHAM Model Definition	83
5.3.3	Mapping Table (M-table)	85
5.3.4	Translation Rules (T-rules)	86
5.4	Building the HHAM model and M-table of an IT application	89
5.5	Tool Support for the Holistic Modelling Technique for High Availability	94
5.6	Summary	95
6	Predicting Locally Manageable Resource Failures	97
6.1	Overview	98
6.2	Related Work	102
6.3	HAC Characteristics for Predicting Locally Manageable Resource Failures	103
6.4	General Variable and State Definitions	107
6.5	Relative Weight Assignment and Dimensionality Reduction	108
6.6	BDN-HAC-1	110
6.6.1	Variable and State Definition	110
6.6.2	Transformation into the Bayesian Decision Network	112
6.6.3	Conditional Probability Tables	114
6.6.4	Model Inference Example	116
6.7	BDN-HAC-2	118
6.7.1	Variable and State Definition	118
6.7.2	Transformation into the Bayesian Decision Network	119
6.7.3	Conditional Probability Tables	121
6.7.4	Model Inference Example	121
6.8	Causality and Decision Network	124
6.8.1	Reasoning with Incomplete Data	125
6.8.2	Influence of the Decision Node ‘Current State’	126
6.9	Summary	127
7	Bayesian Network for Failure Propagation and Prediction	129
7.1	Related Work	132
7.2	Method for Constructing the BN Model	133
7.2.1	Transformation Methodology	133
7.2.2	Transformation from HMTHA to BN-HAC	133

Contents

7.2.3	Variables and State Definition	135
7.2.4	Incorporating the Weak Node Concept	135
7.2.5	T-rules	135
7.2.6	Assigning Prior Probabilities	135
7.3	Failure Propagation and Prediction	136
7.3.1	Working with Latent Nodes	137
7.3.2	Working with Incomplete Data	137
7.3.3	Parameter Learning	138
7.3.4	Failure Propagation	140
7.3.5	Inference Using Production Data	141
7.3.6	Causal Reasoning with the BN-HAC Failure Propagation and Prediction Model	141
7.4	Summary	145
8	Bayesian Prognostic Framework Preparation	147
8.1	Related Work	148
8.2	Database for Storing Log and Configuration Data	148
8.3	Configuration Refinement	150
8.4	Log Interface	152
8.5	Transformation and Conversion	154
8.6	Database Table for Storing Model Data	156
8.7	Filter	157
8.8	Summary	157
9	Bayesian Prognostic Framework	159
9.1	Related Work	159
9.2	Bayesian Prognostic Framework	160
9.3	Implementation Steps	162
9.4	Summary	163
III	Implementation and Evaluation	165
10	Evaluation	167
10.1	Testbed	167
10.1.1	Virtual Machines	168
10.1.2	Network Configuration	168
10.1.3	Storage Configuration	169
10.1.4	HAC Solution	170
10.1.5	Enterprise Application	170
10.1.6	Enterprise Application Deployment in the Testbed	171

10.1.7	HAC Architecture	172
10.1.8	HAC Configuration	173
10.1.9	Quorum Configuration	174
10.1.10	Conditions of the HAC	175
10.2	Evaluation Methodology	176
10.2.1	Test Cases	177
10.2.1.1	Fault Injection Methodology	177
10.2.1.2	Test Cases	178
10.2.2	Data Sets	180
10.2.3	Data Set Analysis	183
10.2.4	Evaluation Metrics	185
10.2.4.1	Evaluation Metrics for Runtime Overhead and Execution Time	188
10.2.5	Expected Outcome	191
10.3	Evaluation of Bayesian Prognostic Framework Modules	192
10.3.1	Evaluation of the Holistic Modelling Technique for High Availability	192
10.3.1.1	Evaluation of the Model Construction	192
10.3.1.2	Runtime Overhead	195
10.3.2	Evaluation of the Locally Manageable Resource Failure Prediction	196
10.3.2.1	Evaluation of the Models	196
10.3.2.2	Prediction Quality	198
10.3.2.3	Runtime Overhead	205
10.3.3	Evaluation of the Bayesian Network model for Failure Propagation and Prediction	207
10.3.3.1	Evaluation of the Model Construction	207
10.3.3.2	Prediction Quality	212
10.3.3.3	Runtime Overhead	217
10.3.4	Evaluation of Bayesian Prognostic Framework Preparation	219
10.3.4.1	Runtime Overhead	219
10.4	Evaluation of Bayesian Prognostic Framework	219
10.5	Discussion	222
10.6	Threats to Validity	223
10.7	Summary	224
11	Conclusion and Future Work	227
11.1	Taxonomy of the High Availability Clusters	228
11.1.1	Research Contributions and Discussion	228
11.1.2	Further Research Directions	228
11.2	Survey of High Availability Clusters	228
11.2.1	Research Contributions and Discussion	228

Contents

11.2.2	Further Research Directions	229
11.3	Holistic Modelling Technique for High Availability	229
11.3.1	Research Contributions and Discussion	229
11.3.2	Further Research Directions	230
11.4	Bayesian Prognostic Framework Preparation	230
11.4.1	Research Contributions and Discussion	230
11.4.2	Further Research Directions	230
11.5	Bayesian Decision Network for Predicting Locally Manageable Resource Failures	231
11.5.1	Research Contributions and Discussion	231
11.5.2	Further Research Directions	231
11.6	Bayesian Network for Failure Propagation and Prediction	232
11.6.1	Research Contributions and Discussion	232
11.6.2	Further Research Directions	232
11.7	Bayesian Prognostic Framework for High-Availability Clusters	233
11.7.1	Research Contributions and Discussion	233
11.7.2	Further Research Directions	234
Bibliography		235
Appendix A High Availability Resource Characteristics		255
Appendix B Use of the Taxonomy		259
Appendix C Evaluation of Runtime Overhead of the Holistic Modelling Technique for High Availability		261
C.1	Experimental Setup	262
C.2	Results and Discussion	262
C.3	Threats to Validity	264
C.4	Summary	265
Appendix D Implementation and Evaluation of Bayesian Prognostic Framework Preparation		267
D.1	Implementation	267
D.1.1	Prepare the Environment	267
D.1.2	Implement the Database for Storing Log and Configuration Data	267
D.1.3	Apply Configuration Refinement	268
D.1.4	Implement the Log Interface	268
D.1.5	Enable Transformation and Conversion	269
D.1.6	Implement the Database Table for Storing Model Data	269
D.1.7	Apply Filter	269

D.2 Evaluation of Bayesian Prognostic Framework Preparation 270

 D.2.1 Experimental Setup 270

 D.2.2 Results and Discussion 270

D.3 Threats to Validity 272

D.4 Summary 273

Appendix E Implementation of the Bayesian Decision Network for Predicting Locally

Manageable Resource Failures **275**

E.1 Implement the Model 275

E.2 Change the Target Environment Details 275

E.3 Test the Model 276

E.4 Inference Using Production data 276

List of Figures

1.1	High-level view of the Bayesian prognostic framework.	7
2.1	Structure of the running example application and its components.	15
2.2	Architecture of a high availability cluster (HAC) with $n \geq 2$ nodes.	18
3.1	Top-level classes of the HAC taxonomy.	22
3.2	Deployment patterns.	22
3.3	Kubernetes architecture and communication flow between key components.	23
3.4	Application areas of HACs.	25
3.5	Type of cluster.	25
3.6	HAC topology.	27
3.7	Cluster management.	30
3.8	Cluster communication.	31
3.9	Failure detection and recovery.	34
3.10	Consistency and integrity.	37
3.11	Data synchronisation.	40
3.12	File systems related to shared storage. Key: ext4 – Fourth extended file system, ZFS – Z File System, OCFS2 – Oracle Cluster File System, vxCFS – Veritas Cluster File System, NFS – Network File System, IBM Spectrum Scale – A distributed file system, formerly called the General Parallel File System (GPFS)	41
3.13	Platform and operating system support of the surveyed high availability clusters (HACs) grouped by operating system.	53
4.1	The causal connections: (i) Serial, (ii) Diverging, and (iii) Converging connections.	65
4.2	Types of Bayesian networks.	67
4.3	Parameter learning approaches.	69
4.4	Learning scenarios and the corresponding learning methods.	69
4.5	Bayesian network model for the HAC from the running example. Latent nodes are highlighted in blue.	71
4.6	Components associated with Bayesian network model construction and inference.	73

List of Figures

5.1	Classification of major availability modelling formalisms.	81
5.2	Metamodel of the holistic availability modelling technique presented as a UML class diagram.	85
5.3	Simplified process for creating a holistic high availability model for high availability.	89
5.4	High availability model for an example IT application using a graphical notation to represent the different types of vertices and arcs from Definition 1. The different graphical elements are shown in the legend.	90
5.5	High availability cluster (HAC) of the running example.	93
5.6	The HMTHA tool showing the high availability model (HHAM) for the running example using the graphical notation for the different types of vertices and arcs from Definition 1.	94
5.7	HMTHA tool showing the second view to analyse the model.	94
6.1	Overview of locally manageable resource failure prediction. Dashed boxes and arrows indicate that the data are already available in the runtime environment and are retrieved from sources in the design phase.	99
6.2	Timeline of resource failure events showing the HAC activities in the lower level, and the related activities of the Bayesian decision network model for predicting locally manageable failures in the upper level.	100
6.3	Reducing dimensionality and adding relative weights to the variables in (i) the error-related properties set E , (ii) the dependency-related properties set D , and (iii) combining the outcomes of (i) and (ii) with criticality-related property set C and current status-related property set S	109
6.4	Bayesian decision network model BDN-HAC-1. Random nodes are depicted with a white background, blue shading indicates latent nodes, green shading indicates the decision node, and the utility node is shaded orange.	112
6.5	The Bayesian decision network model, BDN-HAC-2, showing the nodes and the related edges. White represents random nodes, green indicates decision nodes, and utility nodes are represented by red.	120
6.6	An illustrative example of the BDN-HAC model inference for the running example.	124
7.1	Timeline of resource failure events where HAC activities are shown in the lower level, and the activities of the Bayesian network model for failure propagation and prediction are displayed in the upper level.	129
7.2	Mapping the HHAM to the BN-HAC model: i) HHAM model; ii) mapped BN-HAC model. In the BN-HAC model, the nodes shaded in blue are latent nodes; all the other nodes are random nodes.	134
7.3	Failure propagation in the BN-HAC model when node A_3 fails. Red indicates a failed node, white represents running random nodes and blue indicates latent nodes.	138

7.4	Node failure probability distribution after parameter learning using the three data sets.	139
7.5	Joint probability distribution of the model for the running example.	140
7.6	An illustrative example of the BN-HAC model inference for the running example.	142
7.7	Identified causal connections in the BN-HAC model for the running example.	142
7.8	Failure propagation and the impact on the nodes in the same layer when node A_3 fails.	144
7.9	Diverging connection depicting nodes representing two resource groups. White represent random nodes and blue indicates latent nodes.	144
8.1	Components of the preparation environment are depicted, which show the log processing steps.	147
8.2	Database structures for storing configuration and runtime data.	149
8.3	High availability cluster log extract and the key elements.	152
8.4	An illustration of the log management approach by the log interface.	153
8.5	Database table for storing data for the BDN-HAC model.	156
9.1	Modules and components of the Bayesian prognostic framework, annotated to indicate whether they are used during D(esign), I(mplementation) and/or at R(untime), e.g., 'DI-' indicates that a component is used during design and implementation and is not used at runtime.	161
9.2	Bayesian prognostic framework development process.	162
10.1	Main layers of the enterprise resource planning (ERP) solution in the testbed.	170
10.2	Architecture of the high availability cluster (HAC) in the testbed application.	172
10.3	Extract from the quorum configuration file.	173
10.4	Distribution of the failed-resource types in data sets.	183
10.5	Distribution of mitigation actions based on the threefold strategy to handle failures by the HAC.	184
10.6	Contingency table for the basic metrics.	185
10.7	Holistic high-availability model (HHAM) for the testbed application.	194
10.8	Utility analysis results between BDN-HAC-1 and BDN-HAC-2 models for (a) Data Set 1 and (b) Data Set 2. The horizontal cutoff lines at zero utility for BDN-HAC-1 (cf. eq. (6.10)) and at utility 200 for BDN-HAC-2 (cf. eq. (6.17)) separate the positive outcomes (filled markers) and negative outcomes (empty markers).	197
10.9	Strength of influence for the parent-child node pairs from the two BDN-HAC models.	197
10.10	Receiver operating characteristic (ROC) curve showing prediction quality for Data Set 1.	199
10.11	Receiver operating characteristic (ROC) curve showing prediction quality for Data Set 2.	199
10.12	Receiver operating characteristic (ROC) curve showing prediction quality for Data Set 5.	200

List of Figures

10.13 Receiver operating characteristic (ROC) curve showing prediction quality for Data Set 7.	200
10.14 Changes in the prediction outcome based on nodes receiving data for (a) Data Set 1, (b) Data Set 2, (c) Data Set 5 and (d) Data Set 7. The labels on the horizontal axis show the last node that was supplied with data, e.g., the values for ‘ B_2 ’ were obtained when the nodes A_2, A_3, A_4, A_5 and B_2 were supplied with data, and all of the other BDN nodes were not.	202
10.15 Comparison of prediction quality for experiments in which only the critical or only the noncritical nodes were supplied with data for: (a) Data Set 1, (b) Data Set 2, (c) Data Set 5, and (d) Data Set 7.	203
10.16 Comparison of prediction quality between existing characteristics (ECs) and new characteristics (NCs) for (a) Data Set 1, (b) Data Set 2, (c) Data Set 5 and (d) Data Set 7.	204
10.17 Mean execution time for the three steps associated with the invocation of the BDN-HAC model.	205
10.18 Box plots of CPU utilisation of the BDN-HAC steps for (a) Data Set 1, (b) Data Set 2, (c) Data Set 5 and (d) Data Set 7.	206
10.19 Box plot of memory utilisation of the BDN-HAC steps for (a) Data Set 1, (b) Data Set 2, (c) Data Set 5 and (d) Data Set 7.	206
10.20 Bayesian network model for failure propagation and prediction (BN-HAC), representing the testbed high-availability cluster (HAC). The model details are presented in Table 10.22.	209
10.21 Strength of influence analysis for the BN-HAC model.	212
10.22 Receiver operating characteristic (ROC) curve showing prediction quality for Data Set 6 using Data Sets 3, 9 and 10 for parameter learning.	213
10.23 Receiver operating characteristic (ROC) curve showing prediction quality for Data Set 8 using Data Set 4 for parameter learning.	214
10.24 Failure probability distributions for each node in the BN-HAC model (in percentage) after parameter learning with different data set sizes, shown for Data Sets 3, 4, 9 and 10.	216
10.25 Mean execution times for the three steps associated with the invocation of the BN-HAC model.	217
10.26 CPU utilisation of the BN-HAC model invocation steps for inference (a) Data Set 6 and (b) Data Set 8.	218
10.27 Memory utilisation of the BN-HAC model invocation steps for inference (a) Data Set 6 and (b) Data Set 8.	218
10.28 TTR analysis with the recovery time when only HAC results are observed in blue and when the BP framework is employed is shown in red. Panels (a), (b) and (c) are from Data Set 11, and (d), (e) and (f) are from Data Set 12.	220

C.1	Utilisation of memory per vertex, cumulative memory, and absolute memory for (a) model 1, (b) model 2, and (c) model 3.	263
C.2	CPU (a) and memory (b) utilisation when adding vertices.	263
C.3	Mean CPU (a) and memory (b) utilisation per vertex type.	264
C.4	Mean CPU (a) and memory (b) utilisation per layer.	264
D.1	Configuration refinement for one record using the details from the holistic high-availability model (HHAM), mapping table (M-table), and high availability cluster (HAC) configuration details.	268
D.2	Extract from the log file from the testbed high availability cluster (HAC).	268
D.3	Extract from table <i>model_data</i> showing the result of the transformation, conversion, and filter application.	269
D.4	Box plot of the CPU utilisation of the BFPF components presented for (a) Data Set 1, (b) Data Set 2, (c) Data Set 5 and (d) Data Set 7.	271
D.5	Box plot of the memory utilisation of the BFPF components presented for (a) Data Set 2, (b) Data Set 3, (c) Data Set 5 and (d) Data Set 7.	272

List of Tables

2.1	Enterprise application (EA) layers with possible high availability (HA) solutions . . .	17
2.2	HAC terminology	19
3.1	Connection between deployment patterns, application areas and the rest of the taxonomy	24
3.2	Roles and responsibilities for service models in a public cloud, and for on-premises deployment	24
3.3	Type of clusters and potential configurations	26
3.4	Active-passive topology variants	29
3.5	Quorum implementation with Windows server failover cluster (WSFC)	39
3.6	Selection questions	44
3.7	Evaluation of selected HAC solutions	46
3.8	Eliminated HAC solutions in the six-step approach for selecting HACs for survey . .	47
3.9	Outcome of the survey	48
3.10	The surveyed HACs, versions and vendors	52
3.11	Comparison of Kubernetes and high availability cluster features	55
4.1	An example CPT	67
5.1	Modelling rules	86
5.2	Six-step approach to mapping an M-table to a BN model	87
5.3	Translation rules (T-rules) for each vertex type	87
5.4	M-table of the running example	92
5.5	HAC configuration for the running example	93
6.1	BDN-HAC model construction steps and step components	98
6.2	Resource properties of high-availability clusters grouped by sets	104
6.3	Basic variables representing the properties of high-availability clusters, related symbols, values, and variable groups (including group 4 of <i>derived variables</i>)	108
6.4	Description of all the nodes in the BDN-HAC-1 model	111
6.5	Probability distributions for all nodes in the model BDN-HAC-1 and their states . . .	115
6.6	Description of the BDN-HAC-2 model	119

List of Tables

6.7	Probability distributions associated with the nodes in the BDN-HAC-2 model	122
6.8	The impact of incomplete data on the prediction outcomes for model BDN-HAC-1 .	125
7.1	BN-HAC model construction steps and step components	131
7.2	Completed M-table	134
7.3	Prior probability distributions of the BN-HAC model	136
7.4	Data set with incomplete data	137
7.5	Data set with substituted values	137
7.6	Three data sets used during parameter learning experimentation	138
7.7	Prediction that node C_1 will fail (shown as a percentage) when each of nodes A_3 , A_4 and A_5 fails (individually) after parameter learning using the three data sets from Table 7.6	139
7.8	BD-HAC inference for the running example in five scenarios involving the individual failures of resources A_2 , A_3 , A_4 and A_5 (scenarios No 1–4), and the combined failure of A_3 and A_5 (scenario No 5)	141
8.1	Description of the table <code>resource_type</code>	149
8.2	Description of the table <code>cluster</code>	150
8.3	Description of the table <code>node</code>	150
8.4	Description of the table <code>group</code>	150
8.5	Description of the table <code>configuration</code>	151
8.6	Description of the table <code> hac_main</code>	151
8.7	A record entered in the table <code>configuration</code> as part of the configuration refinement . .	152
8.8	Basic variables that represent the properties of high-availability clusters, related symbols, values, types, value description and sources for obtaining the values	154
9.1	Reusability of framework components	163
10.1	Virtual machines used to enable high availability in the testbed	168
10.2	List of virtual IP addresses and associated resource groups	168
10.3	Storage configuration of the testbed	169
10.4	High availability cluster (HAC) configuration listing all resources, HAC names, resource types, resource groups, and short service descriptions	174
10.5	Applied policies for the testbed high availability cluster (HAC)	175
10.6	Resource-level configuration parameters for the testbed high availability cluster (HAC)	175
10.7	Overview of test cases	179
10.8	Full details of test case T1	180
10.9	Overview of the data sets	182
10.10	Mean execution time in seconds for three mitigation steps of the HAC in the testbed .	184
10.11	Metrics derived from the basic metrics, respective symbols, formulas and descriptions	186

10.12 Area under the curve intervals and their interpretations	187
10.13 Mapping values used in plotting the receiver operating characteristic (ROC) curves	187
10.14 Execution time metrics and the related notation associated with the high-availability cluster (HAC), Bayesian decision network (BDN-HAC) model and Bayesian network (BN-HAC) model	188
10.15 Runtime metrics and the related notation for the individual steps of the Bayesian decision network (BDN-HAC) model and Bayesian network (BN-HAC) model	188
10.16 Metrics and the related notation associated with the components of the Bayesian prognostic framework preparation (BFPF) module	189
10.17 Holistic high-availability model grouped by vertex type	193
10.18 First part of the mapping table (M-table) for the testbed application	195
10.19 Details of the two BDN models	196
10.20 Summary of the receiver operating characteristic (ROC) analysis grouped by data set	201
10.21 Completed M-table for the testbed high availability cluster (HAC) showing the structure of the mapped BN-HAC model	208
10.22 Details of the Bayesian network for the failure propagation and prediction (BN-HAC) model for the high availability cluster (HAC) in the testbed	209
10.23 Local probabilities and conditional probabilities of the Bayesian network for the failure propagation and prediction (BN-HAC) model	210
10.24 Summary of the prediction results grouped by the inference data set	215
10.25 Performance of the parameter learning for Data Sets 3, 4, 9 and 10	215
10.26 Mean failover time in seconds for different types of resource groups	220
10.27 Availability analysis results between the standalone HAC and the HAC supported by the BP framework	221
A.1 Excluded resource properties of high-availability (HAC) clusters	255
B.1 Application of the taxonomy to the testbed application	259
C.1 Evaluated HHAM models—vertex types	261
C.2 Evaluated HHAM models—vertices in each layer	262
D.1 Mean execution time in seconds for different components of the BFPF	270

Nomenclature

Acronyms / Abbreviations

ALU Additive Linear Utility

AUC Area Under the Curve

BDN Bayesian Decision Network

BN Bayesian Network

CPD Conditional Probability Distribution

CPT Conditional Probability Table

DAG Directed Acyclic Graph

DBN Dynamic Bayesian Network

DR Disaster Recovery

EA Enterprise Application

EM Expectation Maximisation

ERP Enterprise Resource Planning

EU Expected Utility

FMEA Failure Mode and Effects Analysis

HAC High Availability Cluster

HMM Hidden Markov Model

HA High Availability

HHAM Holistic High Availability Model

HMTHA Holistic Modelling Technique for High Availability

Nomenclature

JPD	Joint Probability Distribution
MAU	Multi Attribute utility
MEU	Maximum Expected Utility
MTBF	Mean Time Between Failures
MTTF	Mean Time to Failure
MTTR	Mean Time to Recover
OLTP	Online Transaction Processing
ROC	Receiver Operating Characteristics
SAN	Storage Area Network
SLA	Service Level Agreement
SPOF	Single Point of Failure
VIP	Virtual IP

Acknowledgements

I would like to express my deepest gratitude to my supervisor Professor Radu Calinescu for his indispensable support, valuable advice, insightful guidance and always being available throughout this research project. Without his support, this thesis would have never materialised.

Special thanks to Professor Dimitris Kolovos, my internal examiner, for his invaluable advice and support. I would also like to thank my external examiner Professor Marin Litoiu. Thank you to Dr Colin Paterson for the support and fruitful discussions. I am thankful to the Postgraduate administration of the Department of Computer Science for giving excellent support.

I want to thank Stacey Karlsson for her encouragement and support. I would also like to thank the late Roderick Hall for all the support I have received from him. I am grateful to Anders Haggren at Linnaeus University for his support.

I would like to express my warmest gratitude to my family, Sujatha Premathas and Clara Premathas, for their continuous support, without whom this would not have been possible. Finally, a special thanks to my mother Mangayarkarasy Somasekaram, Thulasi Jagadeeswaran, the rest of my family, and my friends for all their encouragement and support.

Declaration

I declare that this thesis is a presentation of original work, which I undertook at the University of York during 2017 – 2021, and I am its sole author. This work has not previously been presented for an award at this, or any other, University. All sources are acknowledged as References.

Parts of the research described in this thesis have appeared in the following research papers:

I. Premathas Somasekaram, Radu Calinescu, Rajkumar Buyya. **High-Availability Clusters: A Taxonomy, Survey, and Future Directions**. Journal of Systems and Software, Volume 187, 2022.

II. Premathas Somasekaram, Radu Calinescu. **Towards a Bayesian Prognostic Framework for High-Availability Clusters**. In 2021 IEEE/ACM 14th International Conference on Utility and Cloud Computing (UCC), 2021.

III. Premathas Somasekaram, Radu Calinescu. **Predicting Locally Manageable Resource Failures of High Availability Clusters**. Software: Practice and Experience, Under review, 2022.

IV. Premathas Somasekaram, Radu Calinescu. **Bayesian Prognostic Framework for High-Availability Clusters**. Unpublished manuscript, 2022.

V. Premathas Somasekaram, Radu Calinescu. **Bayesian Prognostic Framework for High-Availability Clusters: A Method for Failure Propagation and Prediction**. Draft, 2022.

Premathas Somasekaram

December 2021

Part I

Introduction and Background

Chapter 1

Introduction

1.1 Context and Motivation

The delivery of key services in domains ranging from finance and manufacturing to healthcare and transportation is underpinned by a rapidly growing number of business-critical enterprise applications (EAs) ¹. Continuous availability is a matter of significant concern for these EAs because disruptions in the services can negatively impact essential business processes [291, 194, 185, 32]. Therefore, the concept of high availability (HA) has emerged. The usual way to achieve HA is through the use of software-based solutions called *high-availability clusters* (HACs) [167, 239] or *failover clusters*. HACs are autonomous systems that support the reliable execution of complex EAs or their key layers and components. They comprise physical servers, storage, communication and other hardware infrastructure with sophisticated HAC management software.

A key requirement of HACs is to ensure the continuous operation of the single-point-of-failure (SPOF) components of protected EA layers. Such SPOF components may include databases, distributed transaction coordinators, software load balancers and storage. However, due to the diverse HA needs of business-critical EAs, HACs must satisfy a wide range of additional requirements, which differ significantly from one EA to another. For example, enterprise resource planning (ERP) applications facilitate transactions (e.g., online transaction processing). Therefore, they must be deployed on HACs capable of satisfying the atomicity, consistency, isolation and durability (ACID) requirements associated with transaction processing [167, 49].

In contrast, EAs such as online analytical processing solutions may not have the same stringent requirements for ACID properties because such solutions tend to be read-only in contrast to transactional EAs that are read-write intensive [165]. Moreover, the focus of analytical EAs is to handle analytical data in multiple steps, such as staging, transformation, processing and reporting, which are typically reflected in the EA architectural layers and their components [281, 57, 106]. Hence, the

¹This thesis adopts the definition of an *enterprise application* from [86], according to which “the term enterprise application applies to a large class of applications that perform important business functions, such as planning enterprise resource usage, automating key business functions, and managing supply chains and customer relationships”.

SPOF components of analytical EAs differ from those of transactional EAs. Therefore, the setup of HACs used to protect such solutions also varies significantly.

Further HAC requirements arise from the need to monitor and maintain the “health” of the cluster. An essential monitor called a *heartbeat* [223, 288] is required to periodically check the health of individual cluster nodes (i.e., servers) so that the appropriate failover procedure can be initiated when node failures are detected. At the cluster level, a *quorum* system [49, 26] is needed for scenarios in which the cluster is divided into cluster partitions that can no longer communicate with each other. A voting protocol is enacted to select a single partition that continues to run the EA in these scenarios. In this way, quorum systems prevent the occurrence of a *split-brain* [239, 167], a situation in which multiple partitions attempt to use EA *resources* (i.e., components)² simultaneously, potentially corrupting important EA data [223].

HACs are also responsible for continuously monitoring the protected EA layers (e.g., the application server and database layers) and seamlessly mitigating EA component failures. When a component fails, the HAC typically follows a threefold strategy to handle the failure. First, the HAC attempts to mitigate the failure by reinitialising the component (e.g., restarting the component) and its child components. Second, if the failure cannot be resolved, the failure is *propagated* to the resource group level where related resources are grouped. The HAC attempts to resolve the problem by failing over the resource group to another node. If no other nodes are available, the HAC can also try to reinitialise the resource group within the same node if the node is functional. Third, if dependencies exist between the resource groups or critical failures occur at the node level, a complete system failover to another node may be performed.

Despite the availability of multiple HAC solutions (including commercial solutions from leading technology companies such as PowerHA SystemMirror [220], Serviceguard [101] and Solaris Cluster [198]), HACs still face significant challenges due to the lack of standardisation, restrictions in IT environments (such as public clouds for monitoring, shared storage and failure mitigation) and limited capability to support EA-specific requirements. Furthermore, today’s HAC solutions operate suboptimally due to limited utilisation of multiple opportunities:

1. The components of an EA are organised hierarchically, but the effect on other components in the hierarchy is typically not evaluated when a component fails even though the failure of a component in the hierarchy affects all the components under the failed component.
2. The criticality of an EA resource is not considered in HAC decision making, which means that the failure of a noncritical resource can unnecessarily trigger a complete system failover.
3. The type of the component (e.g., local file system or global CPU) is not considered, though different component types can have different degrees of influence on the HAC.

²The terms ‘resource’ and ‘component’ of an EA are used interchangeably in this thesis to refer elements of an EA such as databases, disks and IP addresses.

4. Modern EAs provide self-healing capabilities to improve availability. However, HACs do not consider these capabilities and the exploitation of which has the potential to significantly reduce downtime as certain failures could be managed automatically and very efficiently by the EAs.
5. A HAC typically records all system events, including the failure of individual components and failovers, but such historical data are not exploited in HAC decision making.

This thesis introduces a *Bayesian prognostic framework* (BP framework) that leverages these underutilised EA capabilities and historical data to address multiple HAC challenges and to improve the overall effectiveness of HAC solutions. This BP framework comprises four modules:

1. a holistic modelling technique for high availability (HMTHA);
2. a BP framework preparation module (BPPF);
3. a Bayesian decision network (BDN) model for predicting locally manageable resource failures (BDN-HAC);
4. a Bayesian network (BN) model for failure propagation and prediction (BN-HAC).

The first module, HMTHA, provides a tool-supported technique for modelling the wide range of interrelated factors that affect the availability of an application. The second module, BPPF, is responsible for extracting EA and HAC log data, and for processing and preparing these data to be used by the third module. The third module, BDN-HAC, is used to predict which resource-level failures are locally manageable so that the unnecessary use of failure propagation and failover to mitigate these failures (as done by current HAC solutions) can be avoided. The fourth module, BN-HAC, propagates (information about) resource-level failures that cannot be managed locally to the resource group and system level. This enables HACs to predict and mitigate the potential failure of upper-level EA components earlier than currently possible.

1.2 Research Hypothesis

The research presented in this thesis is underpinned by the hypothesis that:

BN-based probabilistic reasoning can support:

1. *a more accurate prediction of locally manageable resource failures, and*
2. *the propagation of information about non-locally manageable resource failures to predict resource group and system failures, thereby enabling failure mitigation that achieves higher levels of EA availability compared to what is currently possible using today's HAC solutions.*

We further hypothesise that these considerable benefits can be achieved through leveraging the following capabilities of BNs:

- encoding complex dependencies between HAC components;
- incorporating HAC configuration and runtime characteristics that support assessing the hierarchy, component types, component criticality and application-provided self-healing capabilities;
- exploiting historical data.

1.3 Research Questions

To pursue the research hypothesis stated above, this thesis addresses the following research questions (RQs).

RQ1. How can a holistic HA modelling technique be created for systems comprising different types of components while also capturing HA-relevant concerns such as the interdependencies and criticality of components?

RQ2. How can failure data from the available HAC logs be used to develop a reusable BP framework that is able to predict future failures in other HACs? Answering this research question is particularly challenging for two reasons. First, the same failure is often recorded in HAC logs by multiple modules, and determining that these records correspond to a single failure is non-trivial [288, 198, 115]. Second, HAC logs tend to be HAC solution-specific [288, 198], and deriving HAC-independent data for a reusable BP framework is challenging.

RQ3. How can a BDN model be developed that incorporates both established availability-relevant characteristics of a component and currently unexploited characteristics to predict locally manageable failures at a component level? Answering this research question is challenging for several reasons. First, there are no current methods that can be used to determine which novel characteristics can improve the BDN detection capabilities. Second, the identified characteristics need to be reduced through the use of BN impact weights [131, 188, 244] to enable delivering data to the next BN model, though determining these weights is difficult. Last but not least, the BDN needs to cope with incomplete data, which is a known challenge in Bayesian reasoning [131, 188].

RQ4. How can a BN model be developed to assess a resource-level failure and propagate this information to predict potential failures at the resource group and system level? This question can be broken down into several sub-questions:

- How can a BN model with all the components of a HAC be constructed?
- How can the different types of components of an IT solution be represented in such a BN model?

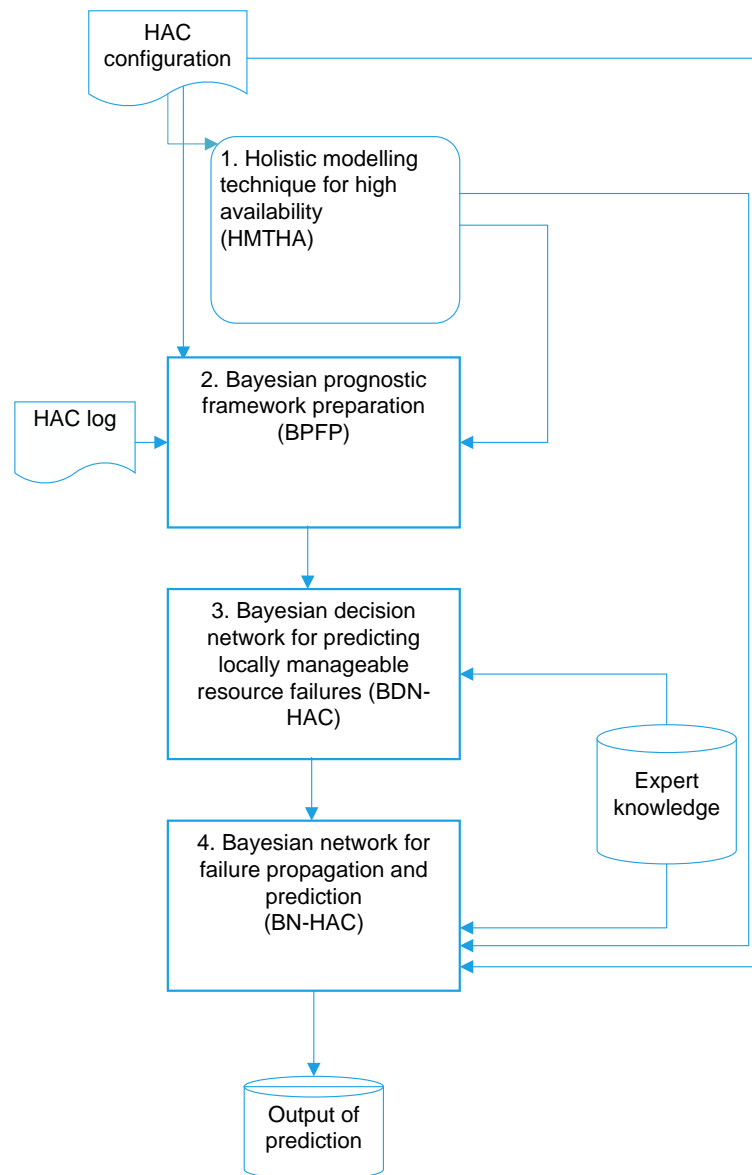


Figure 1.1 High-level view of the Bayesian prognostic framework.

- How can a noncritical or weak component and components with no physical representation be represented?
- How can the parameters of the BN be learnt? How does the data set size for learning parameter distributions affect inference outcomes?

1.4 Research Contributions

This thesis presents seven research contributions. These contributions consist of a new HAC taxonomy, a HAC survey and a set of five technical contributions (summarised in Sections 1.4.3–1.4.7) that are associated with the BP framework shown in Figure 1.1.

1.4.1 Taxonomy of High Availability Clusters

A comprehensive taxonomy of HACs has been devised and organised into eight top-level classes and numerous subclasses. The taxonomy introduces several new terms to capture key HAC features so that the elements of HACs can be addressed with more clarity. The new terms are *symmetric application-based topology*, *symmetric server-based topology*, *cluster-based replication*, *client-state synchronisation*, *cluster-state synchronisation* and *application-state synchronisation*. The HAC taxonomy is presented in Section 3.1.

1.4.2 Survey of High Availability Clusters

An extensive survey has been conducted on the current HAC solutions (focusing on EAs) using the taxonomy. Subsequently, we identified several limitations, open challenges and opportunities that require further research in the HAC domain. The survey and findings are presented in Section 3.2.

1.4.3 Holistic Modelling Technique for High Availability

We have developed a new holistic technique for modelling HA for IT systems. As shown in Figure 1.1, this technique represents the first key module of our BP framework. The new technique introduces a graphical notation that can be used to specify the range of components encountered within an IT system, as well as the different types of dependencies between them. The resulting models can be used to visualise EA component characteristics, such as the component type, dependency type, criticality and position in a hierarchy. The result is a special graph that we term a *holistic HA model*. An accompanying *mapping table* (M-table) and *translation rules* (T-rules) simplify the mapping of the HA model to a probabilistic model. The HA model is used to create and configure other modules in the BP framework. A process to create the HA model is also introduced to simplify creating such models. This contribution is presented in Chapter 5.

1.4.4 Bayesian Prognostic Framework Preparation

We have developed a Bayesian prognostic framework preparation (BPFP) module that has two objectives:

1. support the preparation of an environment ³ to deploy the BP framework;

³The activities associated with preparing an environment include installing the required database objects, libraries, the developed software programs, and populating the database objects with configuration data.

2. prepare HAC log data for the consumption by the BDN model of the BP framework.

These two objectives are delivered by four core components: configuration refinement, log interface, transformation and conversion, and a filter. The component configuration refinement updates the BPF tables with actual HAC details in the design phase. The log interface is crucial because it interfaces with the HAC logs through polling and extracts distinct failure information. Moreover, it is also responsible for parsing and enriching to enable the BDN-HAC module to consume the prepared data. The components transformation and conversion and the filter are used when data is prepared and preprocessed. This contribution is presented in Chapter 8.

1.4.5 Bayesian Decision Network for Predicting Locally Manageable Resource Failures

We have devised a new BDN-based model (BDN-HAC) that can be used to predict locally manageable resource failures. This is the third module in the BP framework, and because it represents a BDN model, it is also referred to as a model. The model is a HAC solution-independent module and uses a BDN to manage decision making under uncertainty. The model uses four groups of characteristics to evaluate a failure. The first group is used to identify and understand the behaviour of the HAC when a resource failure occurs. We introduce new characteristics for the remaining three groups to improve the detection capabilities. The second group extends the detection scope to include additional characteristics, such as the hierarchical position of a failed resource. The third group assesses the criticality of the failed resource, and the fourth group assesses the self-healing capabilities provided by the application. These characteristics are then translated into variables that are combined to detect component failures. The number of variables is reduced by employing a dimensional reduction approach, where weights are added to each variable to ensure that the correct impact factor is captured. Finally, a utility node is used to output a value that indicates whether a resource failure can be managed locally or not. Information about any locally unmanageable failure is sent to the BN-HAC model to propagate the failure and to predict the failure of higher-level components. This contribution is presented in Chapter 6.

1.4.6 Bayesian Network for Failure Propagation and Prediction

We have introduced a novel method for constructing BN models for HACs. This the fourth module (BN-HAC) in the BP framework and it enables failure propagation and prediction for HACs. The module is referred to as a model because it comprises a BN model. The model receives output from the BDN-HAC model to evaluate the probability of failure in high-level components, including at the level of the complete system.

The BN model is constructed to reflect the actual composition of a HAC, and the dependency connections are encoded in the model using conditional probabilities. A weak node (noncritical) concept is also presented along with a mapping procedure for all HAC resources. The model uses

an M-table (mapping table) to map the components discovered in the holistic availability model. Parameter learning is used as the primary means to update the probability distributions. When new failure information is received from the BDN-HAC model, the failure is assessed for the appropriate BN node considering prior failure distributions for the node and other related nodes. The failure is then propagated to high-level nodes to predict their potential failure using the joint probability distribution (JPD). The model also deals with an extreme form of latent nodes, implying that only one node (the node representing the failed resource) obtains data while the other nodes become latent. Moreover, failure can occur in any node, which means failure node variability is also an essential factor that the model addresses. The model outcome is a value that predicts the probability of failure for a resource group, multiple resource groups or the complete system. This contribution is presented in Chapter 7.

1.4.7 Bayesian Prognostic Framework for High-Availability Clusters

A final key contribution of this thesis is the BP framework itself, which integrates the components provided by all the technical contributions. Thus, the framework consists of four modules (Figure 1.1) aided by a set of conventions, rules and mapping procedures. Two modules are based on the BDN (BDN-HAC) and BN (BN-HAC). Three steps are associated with the BP framework: design time, implementation and runtime, and not all modules are part of all steps. For example, the runtime step does not have the HMTHA because it is only used when designing and implementing the BP framework. This contribution is presented in Chapter 9.

1.5 Thesis Organisation

This thesis is organised into three parts. Part I comprises Chapters 1 to 4 and provides the introduction and background for HACs and BNs. Chapter 2 describes the key concepts and terminology related to HA, how the different layers of EAs require protection, and the role of HACs in delivering HA for some of these layers. The reference architecture of a HAC and its components are also detailed in this chapter. Chapter 3 presents our comprehensive taxonomy and survey of HACs. Chapter 4 introduces BN and BDNs, and briefly describes BN-related concepts used in this thesis.

Part II consists of Chapters 5 to 9, which detail the main technical contributions of this research. Chapter 5 presents the HMTHA, which models HA for IT solutions. Once a model is created, it can be used to identify components that are part of an HA solution. The resulting model is used in subsequent chapters. Chapter 6 introduces the BDN-HAC model for predicting locally manageable HAC resource failures. The known characteristics of HACs are described, and then a set of new characteristics are presented. A description of translating these characteristics into a set of BDN variables and subsequently performing dimensionality reduction and weight assignment on those variables is also detailed. Chapter 7 presents the BN for the failure propagation and prediction model.

The steps for constructing a model using the outcomes of the HMTHA are described. The propagation of a failure and how propagation is performed for high-level components are also detailed.

Chapter 8 presents the BP framework preparation module and details the different components of the module responsible for preparing an environment to deploy the BP framework and process HAC log data. Chapter 9 describes the integrations of the modules presented in Chapters 5 to 8 into the complete BP framework.

Part III contains Chapters 10 and 11. Chapter 10 describes the testbed, evaluation metrics, test cases and the two-step approach used to evaluate the individual modules of the BP framework and to evaluate the complete framework itself. The evaluation results are also presented and discussed in this chapter. Finally, Chapter 11 concludes this thesis by summarising the findings and contributions of this research, and providing further research directions in the field.

Chapter 2

Uses and Architecture of High-Availability Clusters

In this chapter, we first describe the terminology related to availability and high availability, and then we describe the role of the HAC. We then provide a detailed definition of the threefold failure management strategy employed by HACs. Next, we present the layers of the EA and discuss the manifold uses of HACs to protect essential layers of a critical application and present the architecture of HACs. Finally, we discuss how HACs can support the realisation of disaster recovery (DR) requirements.

This chapter is organised as follows. Section 2.1 presents the key concepts and terminology related to availability and high availability. Section 2.2 explains how HACs are used to protect different layers of critical EAs. Section 2.2.1 presents an overview of HAC, and introduces a reference HAC architecture. Section 2.3 provides an overview of disaster recovery in the context of HACs.

2.1 Key Concepts and Terminology

The ISO/IEC 25010 standard defines availability as the *degree to which a system, product or component is operational and accessible when required for use* [119]. Availability is calculated as the ratio between the time for which a system is operational and the total time over which the system was observed. Equivalently, availability can be computed as the ratio between the *mean time between failures*, *MTBF*, and the sum of the mean time between failures and the mean time to recover after failures, i.e., the *mean time to repair*, *MTTR*: $availability = MTBF / (MTBF + MTTR)$ [135, 191].

Component failures lead to *downtime*, periods when the system is not operational or accessible, which results in a decrease in availability. As such, HACs are responsible for reducing both the frequency and the duration of failures, and thus their impact on the availability of the protected EAs. Discharging the first responsibility involves monitoring specific EA components to identify and resolve *faults* before they lead to *errors* and the identification of errors before they trigger *failures*, i.e., violations of requirements observable to EA users [135].

Uses and Architecture of High-Availability Clusters

A fault can occur in any resource (i.e., atomic component) of an EA, and the critical resources are usually combined into one or several SPOFs (or SPOF groups). If such a resource fails irrecoverably, it will lead to the failure of its associated SPOF as well. When a SPOF fails, it may bring down an entire application. Achieving high availability requires that the SPOFs of an application are entirely eliminated, partially eliminated or masked. Consequently, HACs discharge their second responsibility by relocating SPOF-related resources to a secondary server after irrecoverable failures. In this way, they mask the failures of resources and thus also of application SPOFs.

In this context, HACs employ a threefold strategy for failure management:

1. HACs avoid EA downtime, even in the presence of failures of individual resources. To achieve this, HACs reinitialise or restart resources after faults and errors (increasing MTBF) and after failures (reducing MTTR).
2. HACs promote the failure management to a resource group level if the failure at a resource-level cannot be resolved locally. This leads to a failover of the concerned resource group to another node. A resource group can also be reinitialised on the same node if there are no available secondary nodes. A resource group failover is faster than a complete system failover. Therefore, the likely outcome is that the failover does not cause downtime.
3. If there are dependencies between the resource groups and after critical failures, a complete system failure may occur. In this event, the complete system is failed over to another node.

In the first scenario, components are restarted, whereas in the other two scenarios components are first stopped and then started in a specific order determined by their interdependences.

As a simple example, consider a web application comprising *Service*, *IP*, *File system 1*, *File system 2*, where the resource *Database group* is a logical resource that groups all the underlying resources as depicted in Figure 2.1 (This application is used as a running example throughout this thesis). When the parent *Service* with three child resources (*IP*, *File system 1*, and *File system 1*) fails, all child components also fail. The HAC aims to resolve the failure by restarting the service (strategy 1). However, because the service component depends on two child components, they must be restarted before restarting the parent component. Similarly, when stopping a component, all related child components are stopped before the parent node can be stopped. If a failure is not resolved, it may lead to the failure of the related resource group or the entire application. For example, if *File system 1* fails and the failure is not resolved, it may bring down the entire resource group. In that case, the HAC stops all components in a specific order in the primary node, relocates the components to a secondary node, and starts the components in a specific order (strategy 2). The third strategy deals with a complete system failover. In such a case, all resource groups and the corresponding resources are stopped in a specific order, relocated to the secondary node, and started in a specific order. All three actions are automated, which reduces MTTR significantly by avoiding time-consuming manual detection, diagnosis, and recovery activities, thus improving availability.

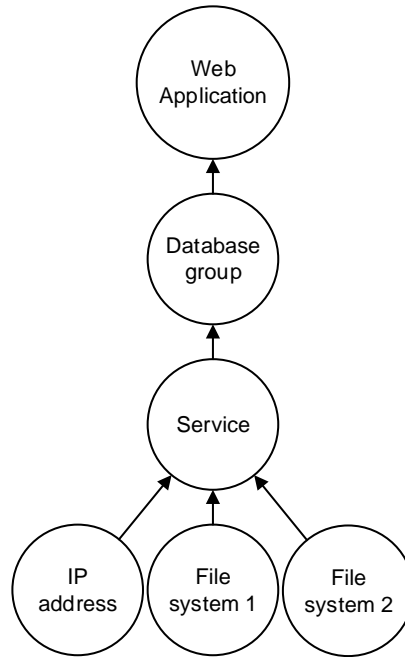


Figure 2.1 Structure of the running example application and its components.

2.2 Layers of Enterprise Application

EAs such as ERPs are transaction-intensive and require stateful communication. Moreover, data consistency and data integrity are vital for such applications. Additionally, modern EAs are highly integrated, which means that data corruption in one application may lead to data corruption in other integrated systems. Therefore, data corruption and data loss must be prevented even when failures occur. To identify and achieve HA holistically for an EA, it needs to be broken down into a set of essential building blocks that are referred to as *layers*. Critchley [49] proposes a layered architecture in describing an IT environment. Somasekaram [255] suggests a similar approach of separating the layers of an IT solution for outsourcing purposes.

When all layers of an EA are identified, an appropriate solution for ensuring HA of each layer can be devised. Multiple solutions are typically possible for each layer, including the use of a HAC. As such, different EA critical layers can each be protected by a separate HAC. Alternatively, a single HAC can be employed to protect several critical layers of an EA. In either case, any EA layer not protected by HAC(s) may require other types of HA solutions (e.g., redundancy or fault tolerance). In the special case of applications with only one critical layer (e.g., firewalls), HA can be ensured through using a *single-layer HAC* [10, 239, 39].

Based on the HA solutions that can ensure the availability of EAs [49, 255, 12, 304, 235, 69, 15, 63, 6, 290, 296], the components of an EA can be organised into nine layers, as shown in Table 2.1. For each layer, the table shows the typical role(s) that the layer can play within an EA, the solutions available for ensuring its availability, and whether an *application HAC* (i.e., a multi-layer HAC) is

among these solutions. As indicated in this table, an application HAC can protect the *application server*, *application core*, and *database* layers of an EA, as well as the client resources associated with the *EA network* and the *storage* layers.¹ In contrast, a HAC is not typically used to protect the *operating system*, and the *virtual machine* (VM) or *server* layers of the EA (i.e., layers 4–6 from Table 2.1), as a failover always involves relocating the application environment to a different VM or server, respectively. The protection of the *data centre* layer is also beyond the scope of a HAC. However, the HAC still needs to monitor critical elements from layers 4–6 in order to identify critical issues such unacceptably high levels of CPU utilisation for a server.

A few research initiatives have addressed the challenges of achieving HA solutions for multiple EA layers from Table 2.1. Bajohr et al. [12] have devised an HA framework for Springer Verlag’s Online Conference Service. Their framework combines different solutions for several layers of this multi-tier applications, including an N+M HAC (these terms are explained in the *topology* section of our taxonomy) for application servers (layer 1) and a master-slave configuration for the database (layer 3). Similarly, Sun et al. [262] present an HA architecture for a multi-tier application in which multiple HA solutions are combined to enable HA for the application. However, most research to date has focused on HACs for single EA layers. For instance, Cheng et al. [40] developed an application cluster service (APCS) scheme comprising separate methods that support state recovery and failure management, respectively. APCS assumes that the state of a shared-storage database layer does not change, and therefore focuses on the protection of the application layer of a three-tier architecture. In many other approaches, the layer protected by different types of HA solutions is the database layer [298], as in the case of Riley et al.’s HA cloud for research computing [228]. Built using the OpenNebula cloud computing platform, this solution employs an active-active HA MariaDB cluster (layer 3) to support the storage of cloud objects.

In summary, modern EAs require a combination of HA solutions to achieve the required levels of end-to-end availability. More often than not, the infrastructure components of EAs have their own HA setups, and thus HACs typically focus on ensuring the availability of the actual applications.

2.2.1 HAC Architecture

Figure 2.2 shows the high-level architecture of a generic HAC operating on $n \geq 2$ nodes distributed across one or multiple locations (i.e., data centres). The HAC is responsible for the management of an EA whose resources are organised into $m \geq 1$ *resource groups*, out of which only the resource groups on the *primary node* 1 are active. The HAC uses three dedicated private networks—a *cluster network* for communication across cluster nodes, a *quorum network* to connect all nodes to a quorum device (i.e., a facilitator of the quorum service), and a *heartbeat network* whose role is explained later in this section. The HAC modules deployed on each cluster node [223, 288, 139] are described below using the terminology summarised in Table 2.2.

¹The network and storage EA layers are part of the EA infrastructure, and present both a server view and a client view. As an example, a storage system in itself is part of the server view, while its individual disks associated with a server or with a virtual machine become part of the client view, and thus need to be protected by the application HAC.

Table 2.1 Enterprise application (EA) layers with possible high availability (HA) solutions

No	Layer	Typical Role(s) within the EA	Possible HA Solution(s)	HAC [†]
1	application (e.g., web servers [30, 237, 196])	server key tier in multi-tier EAs, e.g., presentation layer	use multiple instances with optional load balancing [223, 204]	optional
2	application (e.g., ERP [158, 200])	core central transactions, servers	coordination of distributed application	use application HAC [200, 239, 284]
3	database (e.g., Oracle, DB2, HANA [239, 16])	databases to support the main application	high-availability features provided by database, such as replication and mirroring [239, 176, 284] which can be used with application HAC	yes
4	operating system (e.g., Linux, UNIX)	operating environments	redundant server environment	no
5	virtual machine (VM)	VM (e.g., virtualisation platform)	VM cluster [50] (a HAC can be combined with a VM cluster [287])	no
6	server	server hardware	redundant servers and fault tolerance	no
7	network (e.g., private, public networks)	local area network (LAN), virtual LAN (VLAN)	redundant network devices, fault tolerance, hardware HACs (e.g., for routers and load balancers) [15, 207, 250]	only client resources
8	storage (e.g., the different type of storage systems)	storage area network (SAN), NAS, direct attached storage (DAS) [167]	redundant devices, fault tolerance, storage HAC [167, 304, 238]	only client resources
9	data centre (e.g., essential data centre components)	supporting utilities such as UPS, power distribution unit (PDU) [301], cloud operating systems [98], and backup infrastructure	redundancy by multiple sites, and redundant data centre equipment, such as UPS and fault tolerance for components, HA for the individual data centre components [15, 230]	no

[†] - High availability of layer can be ensured by an application HAC

I. **Cluster management:** This is the core HAC module responsible for overseeing the operation of the other modules and includes the following sub-modules:

- (a) **Cluster data:** It comprises the data stores managed by a cluster and shared by all nodes.
 - **Configuration:** It comprises static data (e.g., HAC configuration parameters).
 - **Runtime:** This consists of dynamic data (e.g., current status of the cluster components).
- (b) **Communication:** This module manages the communication between the HAC modules on the same node and between the cluster nodes, as well as the heartbeat communications.

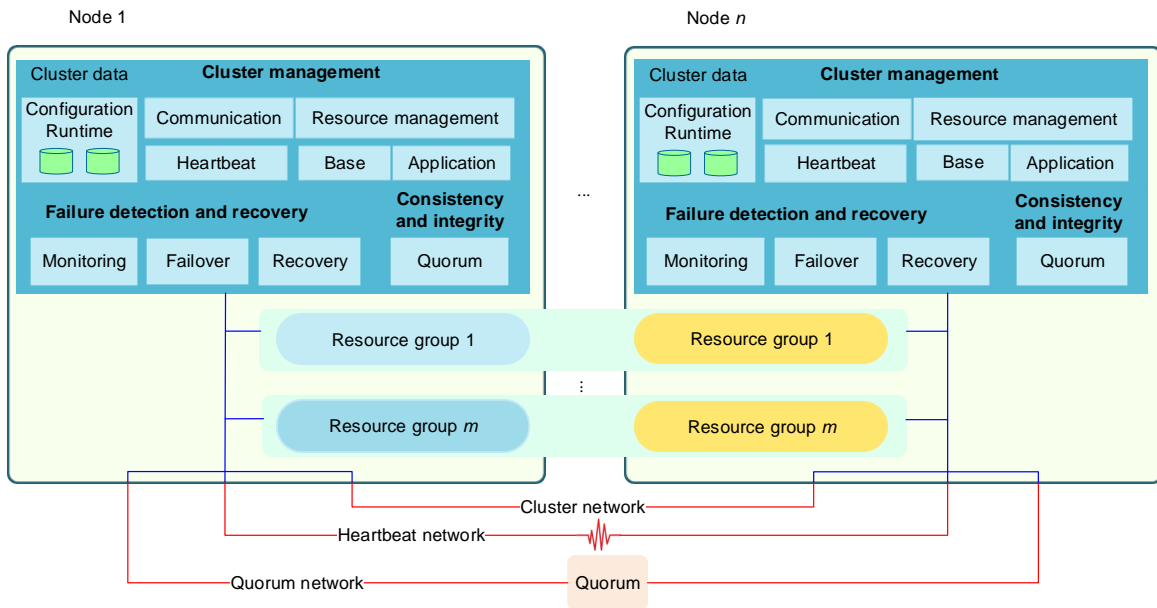


Figure 2.2 Architecture of a high availability cluster (HAC) with $n \geq 2$ nodes.

- **Cluster:** Cluster communication (also known as intra-cluster or inter-node communication) deals with communication between cluster nodes.
 - **Heartbeat:** This is an essential health monitor that checks the health of member nodes, notifying the HAC when the heartbeat of a particular node fails [167, 288]. During such an event, the HAC consults with a quorum to ensure that there are enough votes to continue to run the cluster. If it is the active node that has failed, this will result in a failover, provided that the cluster can reach a quorum [223].
 - **Node:** Communication (also known as intra-node communication) manages communication within a node.
- (c) **Resource management:** This module is responsible for managing two main groups of EA resources:
- **Base:** It manages base resources, which include key components such as CPUs and disks.
 - **Application:** It manages resources that are specific to the HAC-protected applications.

II. **Failure detection and recovery:** This module is responsible for managing failovers and recoveries, and includes the following sub-modules:

- (a) **Monitoring:** A monitoring mechanism to monitor the EA resources and notifies other HAC components (e.g., resource management) about any problems.
- (b) **Failover:** It is responsible for moving resource groups to a secondary node. Depending on the failure type, a failover can be at resource group or system (i.e., complete application)

Table 2.2 HAC terminology

Term	Description
Resource	A logical or physical component of an EA layer (e.g., an IP address used by a database, or an application component) that is managed as an atomic entity by a HAC, and is either fully operational or unavailable. Resources have interdependencies that can be described by a hierarchical map [223, 288, 167].
Resource group	A set of logically related resources that can be relocated to a secondary node as one entity. As such, each resource can only belong to one resource group. An EA may comprise numerous resource groups, each of which may represent a significant part of the EA, such as a database [239, 223, 288].
Split-brain	A condition that occurs when a cluster ends up divided into partitions that perform conflicting operations on the same resources, typically causing data corruption [167, 25].
Amnesia	A condition that occurs when cluster nodes operate with different configurations, e.g., because nodes that are rebooted resume operation with an older configuration. If such nodes are to become primary, a problem is created because they will run with an out-of-date configuration [197].
Switchover	The manual migration of resources from one node to another [49].
Shared storage	A HAC whose cluster members have access to the same storage. When it comes to EAs, typically only one node at a time can allocate the shared storage resources so that data integrity is not affected [223].
Dependency	Resources and resource groups have dependencies that must be taken into account during a failover and the subsequent restart of services. These dependencies can be modelled using an acyclic directed graph termed a <i>dependency configuration</i> [223].

level. The latter involves moving all resource groups that belong to an application [167, 49].

- (c) **Recovery:** This component decides whether failures need to be resolved at resource, resource group or node level by considering their criticality and resource dependencies. When a failure cannot be resolved, the failover sub-module is notified, so that failover can be initiated.

III. **Consistency and integrity:** This module ensures consistency and integrity across all cluster nodes through the following sub-modules:

- (a) **Fencing:** A protection mechanism that isolates a resource or node that experienced failures, removing its ability to connect to any of the critical EA resources [49, 265].

- (b) **Quorum:** A voting system for determining which partition is allowed to run a cluster when a split of the cluster occurs [49, 288, 220]. The partition that has the quorum is considered quorate, and can be used to run the cluster without causing a split-brain.

These terms are described in greater detail in the taxonomy in Chapter 3.

2.3 Disaster Recovery and High-Availability Clusters

The fundamental objective of disaster recovery (DR) is to protect an IT system from failures due to disasters such as earthquakes or flooding [49]. DR is considered part of Business Continuity (BC). However, the focus of BC is to ensure the operations of the business during a disaster, while DR aims to support the restoration of the required infrastructure to support redeploying or relocating an IT system.

A DR solution can be implemented in several ways [239], and one option is to use a HAC. In this case, the HAC employs a cluster type (Section 3.1) that can operate geographically dispersed nodes. For example, the cluster type continental cluster can manage cluster nodes over extended distances. A DR setup can be either active or passive, and in the case of an active setup, the primary infrastructure is mirrored to enable continuous synchronisation between the sites. In contrast, a passive DR solution only requires a minimal setup to facilitate data synchronisation. Nevertheless, a HAC can function as an orchestrator in both cases and then failover the complete application to a DR site in the event of a disaster. The difference is that the former setup supports instantaneous failover while the latter is associated with delays to ensure the operability of the infrastructure in the DR site first before a failover can occur.

Chapter 3

Taxonomy and Survey of High-availability Clusters

This chapter presents a taxonomy covering all key aspects of HACs. We use this taxonomy to provide a comprehensive survey of the end-to-end HAC software solutions available for the HAC deployment of EAs. Finally, we discuss the limitations and challenges of existing HAC solutions, and we present opportunities for future research in the area.

This chapter is organised as follows. Section 3.1 proposes the HAC taxonomy and presents the techniques underpinning core HAC operations such as monitoring, heartbeat, quorum, failure detection, and EA component failover. Section 3.2 applies the taxonomy to survey end-to-end and state-of-the-art EA HAC solutions, available commercially or from open-source projects. Section 3.2.6 discusses HAC limitations, open challenges, and research opportunities. Lastly, Section 3.3 concludes the chapter with a summary.

3.1 Taxonomy

Our taxonomy applies to single-layer HACs and multilayer HACs, which can provide application-specific services, such as understanding the internals of applications and managing those services accordingly upon the failure of one or more resources. Further, the focus is on HACs sharing common characteristics, such as monitoring, fencing, heartbeat or quorum.

The taxonomy is organised into eight top-level classes, as depicted in Figure 3.1. The first four classes capture how HACs are deployed (*deployment patterns*), which EA layers are protected by HACs (*application areas*), how this protection is achieved (*type of cluster*), and how the HAC nodes are structured and interconnected (*topology*). The next two classes reflect how HACs manage the resources of the protected EA (*cluster management*) and perform detection of and recovery from failures of these resources (*failure detection and recovery*). Finally, the last two classes indicate how the HACs preserve the consistency of the EA data and the integrity of the cluster (*consistency and integrity*), and how the EA data are synchronised across cluster nodes (*data synchronisation*).

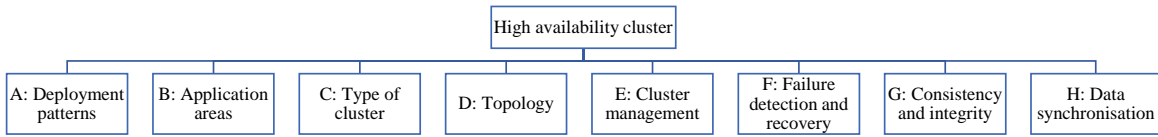


Figure 3.1 Top-level classes of the HAC taxonomy.

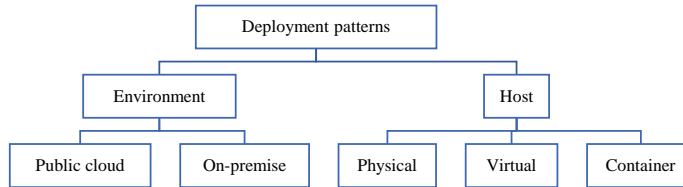


Figure 3.2 Deployment patterns.

3.1.1 A: Deployment Pattern

The *deployment pattern* of a HAC represents the platform **where** the HAC solution is deployed. As shown in Figure 3.2, we distinguish between the deployment *environment*—which can be a public cloud or on-premise IT infrastructure of the organisation using the HAC, and the type of *host* used for the cluster—which can be physical, virtual or container.

The deployment pattern decides, along with business requirements and technical capabilities, **what** *cluster type* can be implemented for an *application area*. Table 3.1 describes the relationship between deployment patterns, application areas, and the rest of the taxonomy. A *cluster type*, on the other hand, decides **how** an *application area* can be protected and the subsequent topology and related configuration. An example of this is as follows: if the deployment pattern is a set of virtual servers in a single data centre, it will not be possible to deploy topologies such as *metro* or *continental* (described under *Type of cluster*). Thus each HAC solution comes with deployment restrictions (e.g., whether it can be deployed in a public cloud or not).

Cloud environments impose restrictions that can cause problems for a HAC because many of the infrastructure elements that a HAC needs to monitor and manage may not be available for a cloud deployment. However, a distinction needs to be made between private (i.e., on-premises) clouds and public clouds because private clouds offer much more flexibility, and functionalities may be identical to an on-premises physical environment. Furthermore, the roles and responsibilities of different stakeholders play an essential role when deploying a HAC in a cloud. Table 3.2 describes the roles and responsibilities of customers and cloud providers for private clouds and for the service models available in public clouds [148, 173, 7], showing that multiple stakeholders may need to collaborate to support the different layers of a HAC in a public cloud.

Containerisation is facilitated through orchestration technologies to deploy and manage containerised microservices, and Kubernetes has become a de facto standard orchestration platform [211]. Kubernetes provides a range of services, including HA for containers. We investigate the HA capabilities of Kubernetes in this section, and further analyses are provided in Section 3.2.5. To study

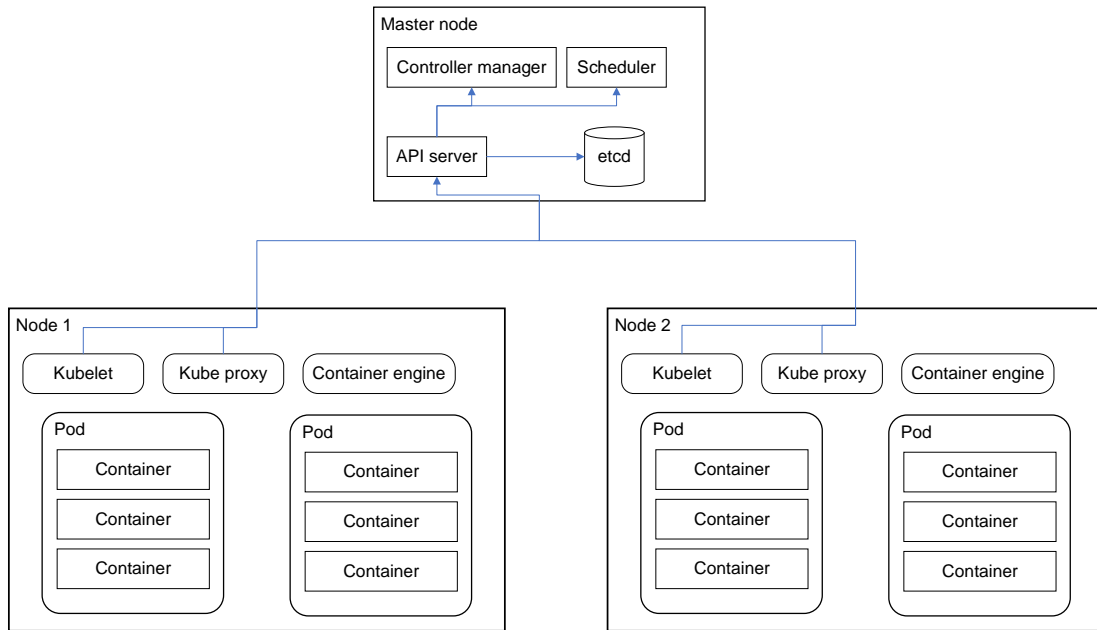


Figure 3.3 Kubernetes architecture and communication flow between key components.

the HA capabilities of Kubernetes, we first present the Kubernetes architecture and then analyse the HA capabilities. The key components of a Kubernetes cluster are illustrated in Figure 3.3, where the arrows mark the flow of communication [266]. There is at least one master node (control plane), one or more worker nodes, or just nodes managed by the master node [266]. A node can be a virtual or physical server.

The components of a Kubernetes cluster are described as follows [266]. A master node comprises multiple global components used to operate components in the nodes. The master node is also responsible for detecting and responding to cluster events. A **controller manager** is responsible for running controller processes which perform different tasks, such as monitoring the individual nodes and reporting back when a node stops (node controller). Similarly, there are controllers for watching jobs, endpoint controllers to populate endpoint objects, such as a pod, and service-account and token controllers to create default accounts and access tokens for the API. A **scheduler** is responsible for monitoring new pods without an assignment and assigning them to a node to run. An **API server** is responsible for presenting APIs for communication. An **etcd** is a key-value repository that stores data regarding the cluster, such as the current and desired states. When the state of the cluster changes, appropriate actions are initiated to ensure the desired state [266].

Each node has a set of node-level components to manage pods running on the nodes [266]. A **Kubelet** is an agent running on every node to support the Kubernetes services (ensuring that containers run in a pod) [266]. A **Kube proxy** is responsible for managing network rules in the nodes to enable communication between pods and network sessions originating from outside or inside the cluster. A **container engine** (container runtime) is responsible for maintaining the runtime environment for

Taxonomy and Survey of High-availability Clusters

Table 3.1 Connection between deployment patterns, application areas and the rest of the taxonomy

	Deployment Patterns	Application Areas	Rest of the taxonomy
Objectives	Where to deploy the solution?	What application or application components need to be protected?	How should the solution be set up to meet the requirements?
Examples	data centre locations, public cloud, virtual server	enterprise system, NAS, network appliance (e.g., firewall), storage system	cluster type, topology, replication, mirroring

Table 3.2 Roles and responsibilities for service models in a public cloud, and for on-premises deployment

No	Layers	On-premises	IaaS	PaaS	SaaS
1	Application server	C	C	C	AP
2	Application core	C	C	C	AP
3	Database	C	C	C	AP
4	Operating system	C	C	CP	CP
5	Virtual machine	C	CP	CP	CP
6	Server	C	CP	CP	CP
7	Network	C	CP	CP	CP
8	Storage	C	CP	CP	CP
9	Data centre	C	CP	CP	CP

Key: IaaS – Infrastructure as a Service, PaaS - Platform as a Service, SaaS – Software as a Service, C - Customer, AP - Application provider, CP - Cloud provider (who may also be application provider for the SaaS service model SaaS).

containers. A container is responsible for running the workload and is placed inside a pod, and the pod runs on nodes and groups containers with the same purpose. A pod is the smallest entity that Kubernetes can deploy, enabling inter-communication between the deployed containers.

Kubernetes has built-in HA capabilities to monitor and ensure that failed containers are restarted or replaced, and it can also terminate an unresponsive container [266]. However, ensuring that the critical components in the master node are also protected is critical for delivering HA. This outcome can be achieved by setting up multiple master nodes, with each master node hosting all components. A load balancer enables access to multiple master nodes. This topology is called a stacked control plane, and a second topology is called external etcd nodes, in which only the etcd nodes are separated into a set of redundant nodes, whereas the rest of the master node components run on a different node set. This topology improves the availability by separating the key component of etcd to facilitate distributed data storage.

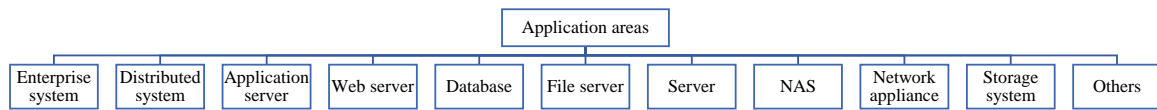


Figure 3.4 Application areas of HACs.

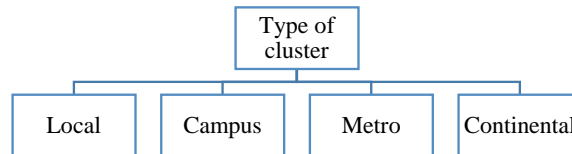


Figure 3.5 Type of cluster.

3.1.2 B: Application Areas

Application areas are the different IT solutions that can be protected by HACs, and a list of typical applications areas is presented in Figure 3.4. Considering the application area has dual purposes: (1) to identify if HACs can support the multiple layers that an application is composed of; and (2) to address all areas that are part of an IT solution, so that HA requirements for those areas can be achieved. For instance, the application area *enterprise system* may require other related areas, such as *application server*, *database*, *server*, *network*, and *storage* to be included to ensure that the *enterprise system* is protected across all critical layers. Some layers can be protected by an application HAC while others may require a different set of options which may include application area specific HACs (presented in Table 2.1) [296]. Moreover, application areas with fewer layers may need to protect fewer components [161]. For instance, a HAC in the context of a distributed system (e.g., high-performance computing—HPC) may need to protect fewer components than an EA HAC. In case of an HPC, a head node (principal node) is identified as a SPOF. Thus, a HAC can be deployed to protect the head node [272]. Therefore, the application areas of a solution are determined dynamically during an implementation phase, and the numbers of protected resources will change with the type of primary application to be protected.

Several recent projects have implemented HACs that support multiple application areas, as also discussed in Section 2.2. Xiong et al. [298] present a HAC for a relational database in a multi-cloud environment which supports the requirements of both HA and Disaster Recovery (DR). Engelmann et al. [66] have experimented with a HAC to protect the head nodes of an HPC environment. Addressing complex systems that consist of multiple layers, hence also several application areas, is a challenge. Wang et al. [290] address the challenge by proposing an HA solution for a comprehensive medical system which consisted of several layers hence also multiple application areas. The proposed solution used a multitude of HACs to enable HA across the different layers.

Table 3.3 Type of clusters and potential configurations

Type of HAC	Distance (in km)	Network Latency (ms)	Data Centres	Storage Systems	Disaster Recovery Support
Local	≤ 1	≤ 1	≥ 1	≥ 1	No
Campus	≤ 30	< 1	≥ 1	≥ 1	Limited due to short distance
Metro	≥ 30	< 5	≥ 2	≥ 2	Limited due to distance
Continental	≥ 300	> 5	≥ 2	≥ 2	Yes

3.1.3 C: Type of Cluster

The type of cluster plays a vital role in selecting the right topology and related configuration for a HAC. An important characteristic is the distance between nodes, and therefore the number of sites (e.g., data centres). The type could be chosen to meet business requirements, such as business continuity or DR. A DR solution requires at least two data centres with a sufficient distance between them and a related configuration. When a HAC solution is explicitly deployed to support DR, it has to comply with restrictions (e.g., low network latency between data centres). Moreover, supplementary mechanisms must be used to guarantee data integrity during failovers. Therefore, the type of cluster should be treated as the starting point for HAC selection, along with the two top-level taxonomy classes presented previously. There are four types of clusters, as shown in Figure 3.5 and Table 3.3. Based on a rule of thumb derived from [167, 239, 282, 99, 112], we assumed a communication speed of 3 ms per 160 km to calculate network latency. The distances described in the table may differ due to the use of different technologies. Moreover, the different HAC solutions can also come with specific recommendations.

C.1: Local. A local HAC is hosted in one data centre and uses one storage system, usually shared. When there are two data centres, the distance between data centres is often less than one km [265]. In such case, there exist two options. Option 1 is to distribute the HAC nodes across two data centres, with all nodes utilising shared storage from one of the data centres. Option 2, on the other hand, uses two storage systems in the two data centres, with the HAC becoming a shared-nothing cluster. However, because data integrity is crucial for EAs, either replication or mirroring must be enabled to synchronise data between the two data centres. The two-data centre setup with replication or mirroring is also a feasible solution for other types of cluster. Since there is usually one data centre associated with a local cluster, the setup is not compliant with DR requirements.

C.2: Campus. A campus cluster is usually deployed across two or more data centres, and the distance between the data centres is less than 30 km [249, 265]. Since a campus HAC has a redundant setup for data centres and related components, it can comply with DR requirements (e.g., it can handle DR scenarios such as a data centre failure). However, the distance requirement between data centres means that businesses may opt for other types of HACs which are optimised for longer distances. Nevertheless, campus clusters can support longer distances when combined with other HAC types,

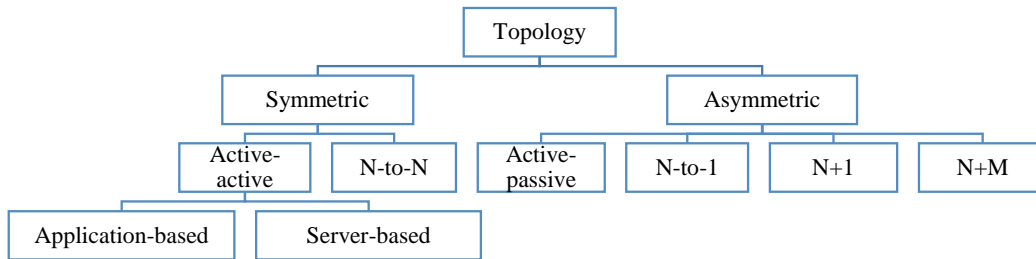


Figure 3.6 HAC topology.

becoming *hybrid* clusters — for instance, multiple interconnected campus clusters with one campus cluster functioning as the primary. This setup enables failover locally for most incidents but will trigger a failover to a different site only when a DR scenario takes place at the primary site.

C.3: Metro. In a metro cluster, the nodes are distributed across a distance of up to 300 km. Although there is no definite cut-off for this distance, the restrictions come from the techniques that are employed to synchronise data [112, 199]. For example, in some cases, the distance can be extended to 400 km by employing Wave Division Multiplexors (WDM) [195].

C.4: Continental. When cluster nodes are geographically dispersed, usually at a distance of more than 300 km, the cluster is characterised as a continental cluster. [100]. A continental cluster can also be referred to as a global cluster or geo-cluster.

3.1.4 D: Topology

The topology (or *redundancy model* [125]) of a HAC represents the way in which the HAC nodes are structured and linked. The topology of a HAC (Figure 3.6) depends on multiple characteristics of its nodes, on the roles of these nodes (primary or secondary), and on its communication devices, networks, storage systems, and supporting tools (e.g., quorum devices).

D.1: Symmetric. In a *symmetric* topology, all cluster nodes can be utilised concurrently: there is no standby node.

D.1.1: Active-active. While symmetric active-active describes that all nodes are utilised, there have been research efforts to implement variations of the topology to address the specific needs of distributed systems. Engelmann et al. [65, 97] implemented a prototype with a symmetric active-active topology that operated on more than two nodes to provide HA for an HPC. The prototype employed two replication mechanisms, internal and external, using reliable and totally ordered message delivery. The internal replication provided synchronisation for the HPC file system metadata service, while the external replication supported the same for the HPC job and resource manager [97]. The evaluation of the prototype showed that the availability could be improved significantly as more nodes were added to the cluster. Therefore, depending on how applications are hosted on such HACs, we distinguish between *symmetric application-based* and *symmetric server-based* topologies.

D.1.1.1: Application-based. In a symmetric application-based topology, an application is active on all available cluster nodes. This topology requires application support because managing transactions across multiple nodes is only possible by using additional mechanisms, such as distributed lock management. A component of an application, for instance, a database, may provide these mechanisms, which can then be combined with a HAC solution [203]. For example, IBM Purescale supports parallel access to IBM DB2 databases [16], and Oracle provides active-active concurrent access support for Oracle databases using the Oracle Real Application Clusters [262, 285, 201].

D.1.1.2: Server-based. A symmetric server-based topology is frequently referred to as an active-active topology, and this implies that multiple applications are hosted on all server nodes of a cluster; hence, the servers are fully utilised [284]. Since all servers are utilised, the topology is considered active-active. When a failover takes place for one or more applications, they failover to one or more of the available servers, implying that a standby node is not required.

D.1.2: N-to-N. In the *symmetric N-to-N* topology, multiple applications share the same set of N servers, like for the symmetric server-based topology. Upon failure of a primary node for an application, the application is failed over to one of the predefined member nodes of the cluster [59]. The new server will then host both the application that has failed over, and the previously running application [284]. The topology supports failing over multiple applications to multiple nodes.

D.2: Asymmetric. An asymmetric topology is an active-passive configuration in which one node is active while one or more nodes are in a passive or a standby mode [29].

D.2.1: Active-passive. An active-passive topology is the typical asymmetric topology consisting of a two-node cluster setup in which one node is active while the other node is passive or standby. This topology is sometimes referred to as 2N redundancy [246]. Today's HACs make a distinction between the different layers of an application. In protecting a layer 3 component (i.e., database), a HAC can either manage it by employing a database-specific extension (agent) or utilising replica or mirroring features that are offered natively by the database [160]. Most database vendors provide a replica or mirroring option to set up standby databases of primary databases [216], and this configuration can effectively be integrated with a HAC. The prerequisite in such a case is that the HAC has support for the specific feature so that the HAC can recognise and support it as part of its operations.

Several variants of the active-passive topology exist, depending on the set up for a standby database and for the secondary node [16, 49, 239], as shown in Table 3.4. While the standby modes from this table are often used with databases, other application layers may also employ a similar configuration. For example, a layer two component (i.e., application core), employs the active-warm (or warm) standby mode due to the limited need for data synchronisation. However, a common setup by a HAC is to employ either active-hot (or hot) standby or active-warm because otherwise failover time and MTTR will increase and, as a consequence, availability will go down. The primary reason for using the active-cold (or cold) standby or active-warm standby is cost, as using an active host node is associated with higher costs. The standby modes are usually not explicitly supported by modern HAC solutions; instead, the different standby modes of databases and related features that are

Table 3.4 Active-passive topology variants

Standby Mode	Recovery Time	Data Synchronisation Method	Description
Active-Cold	Hours	Backup/restore	A secondary node is installed and configured but brought up only when the primary node is down. Subsequently, the related services are started, as well [144].
Active-Warm	Minutes	Mirroring, shared storage	The secondary node is installed and configured and is running. Related application services are started upon failure of the primary node.
Active-Hot	Seconds	Mirroring, replication, shared storage	The secondary node is fully installed and configured, and services are also started. The secondary node takes over responsibilities immediately upon failure of the primary node.

supported are specified [16, 239]. Moreover, the standby modes are frequently used to refer to the modes of the data centres, particularly in the context of establishing DR for a system [186].

D.2.2: N-to-1. In an N-to-1 topology, multiple applications are supported by one dedicated standby node. Hence the name N-to-1 [59, 284]. If a node fails, the application is failed over to the standby node and made available there temporarily. However, while the application is active on the standby node, there will be no HA for that application until the primary node is back online. Another aspect of an N-to-1 topology is that such a standby node must be able to host all N applications simultaneously. Hence, sufficient capacity must be available on the standby node.

D.2.3: N+1. In an N+1 topology, one passive (spare) node supports multiple active applications, similarly to the N-to-1 topology. However, unlike the N-to-1 topology, the N+1 topology employs a *rotation scheme* for failovers [284]. This means that, during a failover, an application is failed over to the standby node, but the failed node, once the problems are resolved, effectively becomes the standby node. Hence, any node in the cluster can become a standby node. A variant of the N+1 topology that uses 2+1 nodes (with two active nodes and one node operating as a standby or backup) has been referred to as *asymmetric active-active* in the context of HACs for HPC [140].

D.2.4: N+M. The N+M topology refers to HACs that comprise N active nodes and M passive nodes in the cluster, and is called an *N+N* topology when the number of passive nodes equals the number of active nodes. The topology is employed when one passive node is not sufficient, and $M > 1$ passive nodes are required for failovers [125, 90].

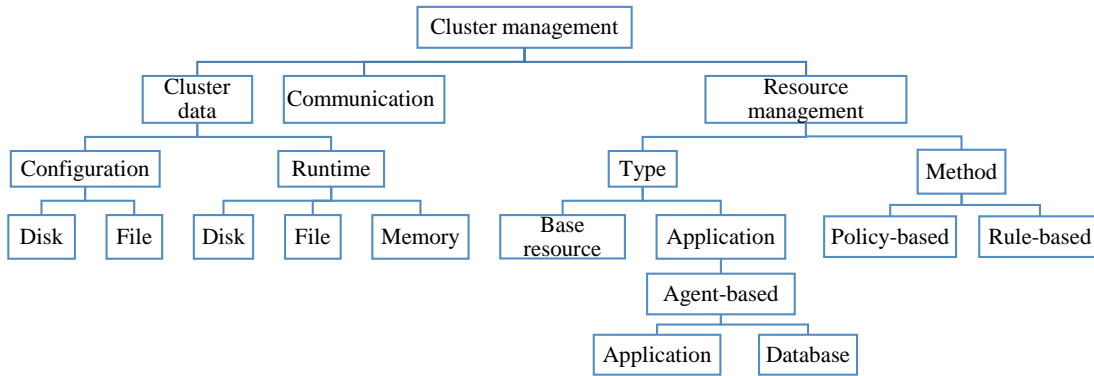


Figure 3.7 Cluster management.

3.1.5 E: Cluster Management

The cluster management module of a HAC is responsible for managing the resources, resource groups, nodes, heartbeats, cluster data, and failovers of a cluster, directly or through other modules. The characteristics used to distinguish between different types of HAC cluster management are shown in Figure 3.7 and described below.

E.1: Cluster data. Two types of cluster data are relevant to HACs: configuration and runtime. *Configuration* data contains configuration details of a HAC while *runtime* data stores status of the cluster components. Cluster data can be stored in three types of repositories: disk, file and memory.

A repository can be either local or shared. However, a prerequisite for a repository is that it is accessible by all cluster nodes. Hence, if a repository is local, a replication mechanism is used to replicate it between the nodes at regular intervals. However, in some cases, when persistent files are employed, the replication is a manual activity. Both in-memory and file repositories are local. However, there are differences in what cluster data type they support. Configuration data is static and is commonly stored in files, while an in-memory repository is generally used to store runtime data to capture changes in real-time. This means there is a rigorous requirement for in-memory repositories to replicate data to other nodes. Therefore, a designated process governs the synchronisation of the runtime data, for instance, Designated Coordinator (DC) in the case of a Pacemaker-based HAC [288, 265]. The coordinator ensures that one master repository exists in the primary node while a copy of it, a replica, is distributed across all the member nodes. A shared repository (e.g., disk or file share), on the other hand, stores both configuration and runtime data. In many cases, a quorum repository, which is shared, can support the requirements. Cluster data in a repository is organised using an information model. For instance, a Cluster Information Base (CIB) uses an XML-based object model to represent both configuration and runtime data. However, there are no standardised information models for dealing with cluster data, and, as such, HACs use different information models. Open Service Availability Framework (OpenSAF), for example, employs Information Model Management (IMM), and objects represent the two types of data: configuration and runtime [268].

E.2: Communication. HACs can use different communication types and methods (Figure 3.8).

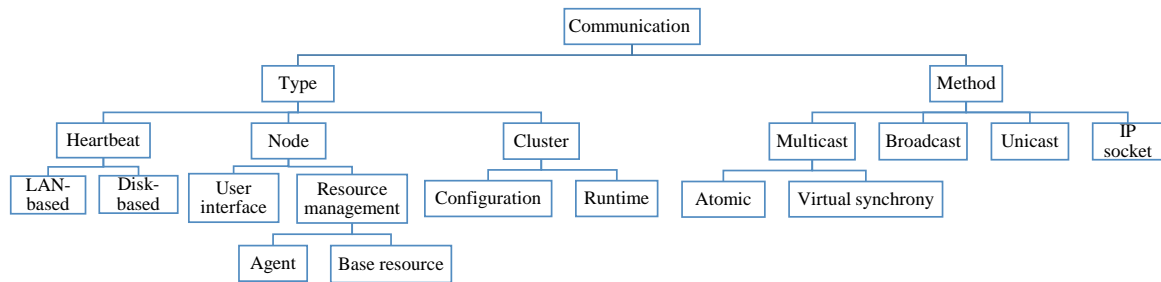


Figure 3.8 Cluster communication.

E.2.1: Type. A communication type describes the different kinds of communications that a HAC employs and is split further into three subclasses: heartbeat, node and cluster.

E.2.1.1: Heartbeat. A heartbeat is a form of intra-cluster communication. However, it is separated in the taxonomy to highlight its importance and use of additional resources, such as a dedicated network. The type and content of heartbeat messages differ from solution to solution. In some cases, a heartbeat message could be a simple ping or a keepalive to provide the status of a cluster node [103, 29]. Heartbeat communication use a LAN-based or a disk-based method [167, 284, 265, 111].

E.2.1.1.1: LAN-based heartbeat communication uses a Transmission Control Protocol/Internet Protocol (TCP/IP) network [167, 239]. Since heartbeat is a key component of a HAC, the recommendation for business-critical solutions is to set up a dedicated network, such as a virtual LAN, to facilitate heartbeat communication [167, 284, 111]. With this approach, the heartbeat traffic is not disturbed or delayed by other kinds of traffic in a network, which could be the case if the network is shared. Furthermore, adding redundancy to a heartbeat network by using multiple networks is also a good option so that a single network does not become a SPOF.

E.2.1.1.2: Disk-based heartbeat uses a shared disk and also the SAN fabric as a means to facilitate communication [111]. In some cases, LAN-based and disk-based heartbeat types can be combined to create a full heartbeat service. If a heartbeat mechanism is not employed, an alternative and robust mechanism is required to detect node failures. Cheng et al. [40] propose a HA solution that employs a module that can detect whether a node is sick or not and subsequently forecast the time of failure. This renders the heartbeat setup to be nonessential in such cases. However, there is no information regarding how such a solution works when there are many nodes in a cluster.

E.2.1.2: Node communication is referred to as *intra-node* and deals with communication within a cluster node. The node communication uses internal communication schemes, for instance, inter-process communication (IPC) within a server. Two types of such communication exist in HACs: user interface and resource management.

E.2.1.2.1: User interface communication refers to the different means to connect to the cluster on a particular node, including Graphical User Interfaces (GUIs) for cluster administration.

E.2.1.2.2: Resource management communication can belong to two subclasses: base resource and agent. *Base resource* describes the communication between the cluster resource management and

those resources that are available as a standard (e.g., IP, CPU of a server). An *agent* describes the communication between the cluster resource management and the agents that are responsible for managing application-specific resources (e.g., database, EA components) [51, 284].

E.2.1.3: Cluster communication, also termed *intra-cluster*, *inter-node* (or resource group) communication, describes communication between cluster nodes. For a HAC, internal cluster communication is crucial. It is required for continuous communication between nodes regarding changes in configuration, the health status of nodes, quorum status, and failure notification. Furthermore, since cluster communication is often a basis for making necessary decisions by a cluster, the requirement for cluster communication is that it is enabled using an atomic (ordered) and reliable messaging scheme. Even though several HAC solutions use different types of cluster communication, a strict definition can be used to distinguish the two main types: runtime and configuration. Thus, cluster communication deals primarily with the synchronisation of cluster *configuration* and cluster *runtime* data (e.g., the status of the nodes).

E.2.2: Method. The types of communications utilise different transmission methods, and these methods can employ different protocols, such as UDP and TCP. Some HAC solutions employ custom protocols to meet the HAC-specific requirements, and an example is the Transparent Inter-Process Communication (TIPC) protocol, used by OpenSAF [268, 164]. The methods are further divided into four subclasses: multicast, broadcast, unicast, and IP socket.

E.2.2.1: Multicast. Multicast enables transmission from one node to multiple nodes. Thus, it can be characterised as a one-to-many (1:m) method. The receivers are usually a group of nodes, which means that a subset of cluster nodes can also be addressed [74, 60, 167].

E.2.2.1.1: Atomic. Atomic multicast (or total order multicast) implies that all nodes receive the same message in their sent order [54].

E.2.2.1.2: Virtual synchrony. Virtual synchrony is an atomic multicast technology that supports reliable inter-process messaging. Corosync, the open-source communication protocol, employs the Totem Single-Ring Ordering and Membership (TOTEM) protocol, which is an example of implementation of virtual synchrony [51]. Engelmann et al. [67] present a multi-node HAC solution for HPC that employs virtual synchrony to support state machine replication between the nodes in a symmetric active-active topology.

E.2.2.2: Broadcast. This method supports one-to-all (1:n) transmissions. While multicast supports transmission to a group of nodes, broadcast transmits to all nodes [74, 239].

E.2.2.3: Unicast. This method facilitates transmission between two nodes, and it is characterised as a one-to-one (1:1) transmission [167, 74].

E.2.2.4: IP socket. An IP socket can also be used in some cases to facilitate communication between cluster nodes [74, 254]. However, the majority of the HACs do not support this method but rely on other methods.

Cluster communication employs either multicast or broadcast, but in some cases, unicast is also used. On the other hand, heartbeat communication employs either unicast or multicast [167].

E.3: Resource management. Resources are structured hierarchically to form a resource group, and links between the resources define the relationships between the resources [167].

E.3.1: Type. HACs can manage two types of resources: base resources and applications (Figure 3.7).

E.3.1.1: Base resource. A base resource is a standard building block (e.g., IP address, file system) [49, 284, 265, 203]. A HAC can manage base resources without requiring additional tools. Hence, a distinction is made between base and application resources. While managing base resources is supported by all HAC solutions to different degrees, application support must be provided explicitly.

E.3.1.2: Application. Application management is the capability to manage application-specific functionalities and features. Since each application must be handled individually, an extension to a HAC is usually required [49, 283, 114]. Such addition is provided in the form of either an extension or an agent.

E.3.1.2.1: Agent-based. Agents manage two main types of applications: application and database. *Application* agents deal with managing several application-specific layers (e.g., application core of an ERP as in layer 2). *Database* agents manage database-specific components (layer 3). The application agent functionality connects application-specific (this includes both types: database and application) configuration and procedures with the resource management module of a HAC and supports functionalities including [283, 114]:

- Monitoring application-specific components
- An application-specific configuration, which can recognise the architecture of the application components
- Complying with application-specific dependencies
- Logging
- Procedures – to stop and start related application components in a specific order
- Supporting Application Programming Interfaces (APIs) or specification by the application vendor

However, not all HAC solutions can support all applications, as each will require separate lifecycle management. When an application changes, for instance, when it is upgraded, the HAC application agent may also need to be updated to reflect the changes. Likewise, when the HAC solution is upgraded, the application agent may also need to be updated. Thus, supporting a large number of application agents could be connected to much effort. Furthermore, such support may be subject to licensing conditions, and HAC vendors could treat individual application support as an extension to license terms.

E.3.2: Method. Two main methods are used when managing resources: policy- and rule-based. *Policy-based* resource management uses policies to configure conditions, and, when a particular

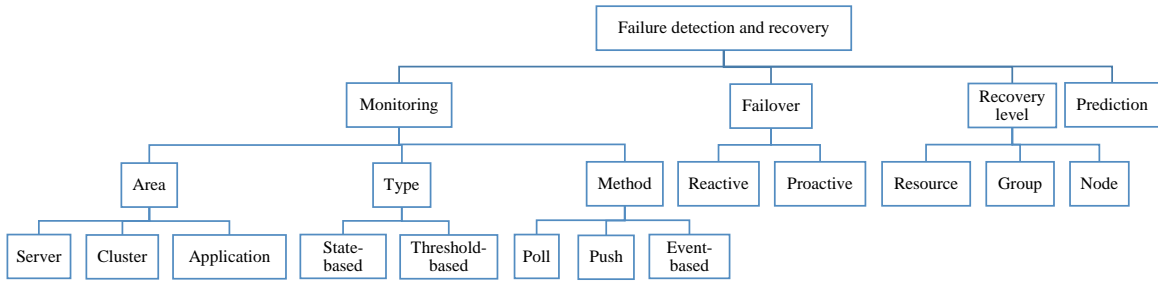


Figure 3.9 Failure detection and recovery.

condition is satisfied, appropriate action is triggered [139, 284, 114]. On the other hand, *rule-based* resource management uses one or more rules to make decisions and act upon them [167].

3.1.6 F: Failure Detection and Recovery

Failure detection implies detecting failures by monitoring and analysing monitoring output [300]. If the monitoring identifies a status change in a resource or a resource group, it invokes recovery management to initiate a recovery. If the recovery is not successful, the recovery manager may initiate a failover of a resource group or even a system; therefore, failover is part of recovery management. Figure 3.9 depicts the top-level class with its subclasses.

F.1: Monitoring. HAC failure detection and recovery monitoring can be further organised into subclasses depending on its area, type and method. The area describes the monitored domains, while the type of monitoring addresses monitoring from a configuration point of view. In most cases, HACs can provide support for specific monitoring metrics; however, if there is no support, a custom approach where HAC users define their own monitoring metrics is adopted. The monitoring scope may also vary and can range from the simple state monitoring of a resource to the monitoring of a resource in a detailed manner [167, 239, 284, 111, 77]. Several research initiatives refer to the monitoring aspect of HACs as means to detect failures. Cheng et al. [40] present a state-based internal monitoring approach for the experimental cluster APCS+PEV, while Leangsuksun et al. [139] employ threshold-based monitoring for the cluster HA-OSCAR.

F.1.1: Area. The monitoring areas that a HAC can support play an important role in the overall solution. This is because monitoring is the process that collects details regarding monitored elements from different areas and delivers that data to the cluster management to make appropriate decisions. The areas that a solution can support can roughly be split into three subclasses: server, cluster, and application.

F.1.1.1: Server. Server-specific metrics focus on critical and noncritical monitoring elements of an operating system and a server level. Examples of metrics are CPU utilisation and memory utilisation [284, 203, 111, 183].

F.1.1.2: Cluster. Cluster monitoring implies that monitoring is enabled, even for the internal components of a HAC, including cluster-related processes and objects [183, 184, 284]. This approach

enables a HAC to distinguish between failures of cluster and application elements, thus preventing making incorrect decisions.

F.1.1.3: Application. Application monitoring is usually administered by an application-specific agent or an extension that is specifically designed to support a particular application and its architecture. This implies that an application agent is aware of the internals of the application [201, 114].

F.1.2: Type. A monitoring type describes how the state of the resources is measured. There are two types of monitoring: state- and threshold-based.

F.1.2.1: State-based monitoring uses the state of a resource as a monitoring metric, and the states can be as simple as “up” and “down,” or the monitoring can be more elaborate and contain more states [284, 265, 111, 114].

F.1.2.2: Threshold-based monitoring uses a set of threshold values related to metrics [293]. As such, alerts with different severity levels can be generated, depending on which threshold is exceeded. While the threshold-based type gives the flexibility to configure monitoring at a granular level, it also adds complexity as the HAC must interpret all the different values and severity levels and act accordingly. One advantage is that a HAC will have more data that can be analysed, and decisions can be made at a granular level.

Even though state-based monitoring is the common type of monitoring, both monitoring types (and others) are sometimes combined. For example, OpenSAF HACs combine threshold-based monitoring with a type called *watermark* monitoring. The threshold-based monitoring is used to monitor system resources, while the watermark monitoring is employed to register the highest and lowest utilisation per configured resource [246].

F.1.3: Method. There are mainly three methods for monitoring, and they are push, poll, and event-based. *Polling* implies that the monitoring module of HAC and agents poll for state changes of resources periodically [293, 64]. On the other hand, *push* implies monitoring data is pushed to the monitoring module or agents [64, 293]. Such a setup will require additional enablers to interact with resources and push monitoring data to HAC agents. Polling is the most common method of monitoring [64], and it usually employs synchronous communication. However, this procedure is associated with a specific overhead. Therefore, other methods are studied by both industry and academia, and a technique that applies an *event-based* design is viewed as less resource-demanding. One type of event-based monitoring employs an intermediate module that interfaces with an operating system to capture instantaneous notifications relevant, for instance, the state change of a process. It passes that to an appropriate module of a HAC. An example of such a setup is the Intelligent Monitoring Framework (IMF) by Veritas [284]. The IMF has a monitoring feature integrated into an operating system for a particular resource so that state changes are captured instantaneously, and a relevant HAC agent is alerted. However, this approach requires specific development towards an operating system for a particular set of resources. Thus, IMF is not available for all types of resources but is being released gradually for different applications. Another recent development is to enable a HAC to interact with the monitoring feature of an operating system directly [111], which means that the HAC

needs only a slim variant of the monitoring module. The downside of this approach is that the HAC becomes highly dependent on the operating system and its developments.

F.2: Failover. Failover management includes procedures for failover and failback, and all such actions are usually policy-driven. *Policy-based* indicates that policies can be associated with events so that the appropriate policies are triggered whenever a related event occurs [102]. Policies can be used, for example, to determine the target node for failover. Furthermore, policies can encode application-specific requirements, such as the order for starting up or shutting down resources. Failover management can further be split into two subclasses: reactive and proactive [122].

F.2.1: Reactive. A reactive measure uses policies to ensure the correct failover actions. There are two types of policies: static and dynamic. A static policy is created during the implementation or when applying manual changes, while a dynamic policy is created automatically by HACs to enforce policies based on runtime failure cases [20].

F.2.2: Proactive. A proactive measure assumes that a predictive model is employed to ensure that a failover can be initiated based on predictions [122]. The predictions can, in turn, use policies to trigger the required actions [139]. However, the proactive approach could be a challenge in HAC environments that deal with complex EAs because all relevant layers must be addressed in such cases while evaluating the HAC behaviour. Therefore, all active HAC solutions employ only the reactive mechanism.

F.3: Recovery level. The threefold strategy to manage failures is implemented using three recovery procedures: resource, resource group, and node (system) level [239, 284].

F.3.1: Resource. A resource-level recovery deals with recovery attempts on a resource-level, implying reinitialisation of a failed resource while adhering to the dependency rules between resources. However, if this step fails, the failure is propagated to a resource group level [239].

F.3.2: Group. A group level recovery attempts to failover the entire resource group to a secondary node. However, if there are no available secondary nodes, an attempt to reinitialise the resource group within the same node can also be initiated. If a resource group have dependencies on other resource groups, it may lead to a node (system) recovery [239].

F.3.3: Node. A node-level (system) recovery deals with failing over the resource groups to a secondary node. Moreover, a resource or resource group failure can also have a cascading effect due to dependencies, and, in such cases, it might lead to recovery on a node level. Since the previous node is labelled as "failed," policies may prevent any resources from being started there until that node is repaired [239].

F.4: Prediction. Current HACs do not commonly employ prediction. However, some research initiatives explore the area of predicting failures, but often with a limited scope. An example of such an initiative is the HA-OSCAR project which assesses prediction by evaluating hardware component failures [139]. The research team used a Hardware Platform Interface (HPI), which the Service Availability Forum specifies, to identify hardware events and, subsequently, analyse such data to provide predictions [139]. Similarly, Lee et al. [142] propose a stochastic prediction model for node

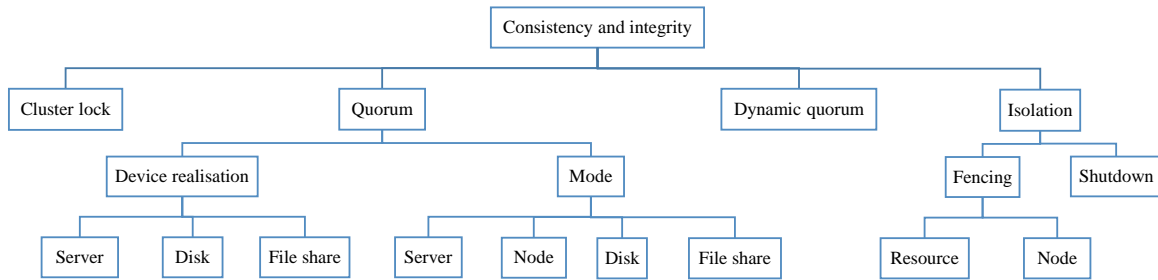


Figure 3.10 Consistency and integrity.

failure or a node-switch interconnected system failure of HA-OSCAR head nodes. Leangsuksun et al. [150] have also explored a failure-repair model for predicting the availability of HA-OSCAR cluster by using Stochastic Reward Nets (SRNs). While many of the prediction models use HA-OSCAR as the platform, some initiatives explore other platforms. For example, Cheng et al. [40] have used a module for a custom cluster solution that detects sick nodes and subsequently uses a prediction method to forecast the time-to-failure of nodes.

Both Veritas InfoScale Availability and Oracle Clusterware provide functionalities to simulate failures and observe potential paths to failovers [284, 202]. However, the objective of the subclass *prediction* is to ensure that the wealth of information that HACs produce can be incorporated to predict failures or optimise failovers. An example of such an approach could be using prediction to optimise the quorum voting process by dynamically evaluating scenarios.

3.1.7 G: Consistency and Integrity

Figure 3.10 presents the top-level class *consistency and integrity* and its subclasses. A HAC employs measures, such as a cluster lock or quorum, to preserve the data integrity of cluster resources and, most importantly, the clustered application by preventing harmful situations, for instance, *split-brain* and *amnesia*.

G.1: Cluster lock. Cluster lock is a technique used to lock cluster resources to a particular node, thus preventing other nodes from claiming the same resources. While a quorum-based approach could also be viewed as a cluster lock, a distinction is made to separate a quorum from a cluster lock. A cluster lock is a technique that does not employ a quorum-based approach but uses other means, such as a software-based lock mechanism. In the case of HACs, a distributed cluster lock is one such option, and an example is OpenSAF, which uses a global lock service to manage shared resources and ensure that only one node can access the resources at any given time [268]. Such configurations are deemed *quorum-less*.

G.2: Quorum. A HAC quorum serves two purposes: 1) maintaining cluster consistency by storing configuration and runtime data (e.g., cluster data) [88], and 2) managing a voting system required in the event of a cluster partition. For the latter purpose, the quorum hosts a voting mechanism in which every healthy and active node has a vote [181, 49]. Furthermore, the quorum also has a vote,

a potential decider, hence the alternative name *tiebreaker*. Other names that are used to refer to the quorum mechanism are *arbiter*, *witness* and *voting system* [239]. When a partition of a cluster occurs after the failure of one or more nodes, a quorum is gathered to decide which partition should have the quorum. To reach a quorum, a partition must have a majority of votes [292]. The quorum service casting its vote can ensure that one of the partitions achieves this majority. Ultimately, the majority cluster is allowed to run the cluster. If a quorum cannot be reached, the surviving nodes will shut down to ensure cluster consistency. The quorum collaborates closely with the heartbeat mechanism, as the heartbeat is the method used to identify unhealthy nodes. Additionally, the quorum or a similar service is required for fencing, as the two often collaborate to determine a quorum and subsequent fencing. A quorum consists of a device and a process [49, 288, 99]. A device describes where quorum elements are stored, and a device facilitates the process, which uses an algorithm to calculate votes dynamically to achieve a quorum. The process employs a mode to determine what policy to use when performing the quorum voting.

G.2.1: Device realisation. Three types of devices can be used by a HAC: server, disk, and file share. A quorum *server* is a service that runs on a server that is usually hosted outside a cluster configuration [49]. The cluster is subsequently configured to connect to the quorum server. A *disk-based quorum* is based on a disk, which can be either local or shared [239, 49]. A *file share* uses a shared file location, and it can be ideal for geographically distributed HACs since member nodes do not have access to a shared disk [171]. The prerequisite for all quorum devices is that they support concurrent access by all cluster members.

G.2.2: Mode. Four modes are possible: server, node, disk and file share. The modes and the devices are an integral part of the quorum solution. However, the supported combination of devices and modes are specific to the different HAC solutions. The mode **server** uses the device *server*, and the device *Disk* is used by the mode **Disk**. Similarly, the device *File share* is used by the mode **File share**. While the devices *disk* and *File share* imply that they are storage points that are managed by the quorum process, a *quorum server* indicates an advanced device type. The mode **node** is implemented implicitly. Hence, it does not require any additional devices but uses the number of available nodes to decide, and the arrangement is referred to as the ‘majority node’ mode.

A quorum can be set up in different ways, and, in some cases, the several modes of a quorum can be combined. For example, Windows Server Failover Clustering (WSFC) supports a combination of devices and modes, as detailed in Table 3.5 [171]. However, the same combination is not always supported by other HAC solutions, and an example of this is that the Serviceguard HAC does not recommend combining a quorum server with a quorum disk [99]. There are new quorum device types introduced to meet the advances in IT. For example, Microsoft has introduced recently a new quorum device called *cloud witness*, and the purpose is to support a server-based quorum in the Azure cloud, which could be ideal for cloud-based solutions [174].

The standard for all explicit quorum devices is that they are placed outside a HAC to avoid creating a quorum device as a SPOF. Moreover, redundancy of quorum is also preferred because

Table 3.5 Quorum implementation with Windows server failover cluster (WSFC)

Combination of Modes	Devices Realisation	Formula for Number of Nodes	Purpose
Majority Node	Node only (implicit device)	$n = 2k + 1$ (odd numbers)	Survive failures of $(n - 1)/2$ nodes.
Node and Disk Majority	Node and disk	$n = 2k$ (even numbers)	Survive failures of $n/2$ nodes when disk is available.
Node and File Share Majority	Node and file share	$n = 2k$ (even numbers)	Survive failures of $n/2$ nodes when file share is available.
No Majority: Disk Only	Disk only	-	Survive failures of $n - 1$ nodes when disk is available.

the quorum is a critical HAC functionality. For this reason, most current HAC solutions support a dynamic reconfiguration procedure for quorum devices, which enables adding or removing quorum devices without impacting the running clusters. While quorum is crucial for a two-node cluster, it can also be opted out using a different mechanism. Furthermore, when a cluster has more than two nodes, an explicit quorum device could become optional because that cluster can survive the failure of a single node. However, a configuration using the mode node is still required to achieve a quorum. The research in this area focuses on enabling probabilistic approaches. For instance, Malkhi et al. [163] have explored a probabilistic approach to address both benign server failures and arbitrary (Byzantine) ones.

G.3: Dynamic quorum. While a quorum deals with static votes, a dynamic quorum calculates the number of votes and adjusts the quorum dynamically upon the failure of one or more nodes [174]. Thus, if a node is unavailable, it will effectively be out of the quorum voting process. This gives more flexibility to continue running a cluster even when other nodes fail. For example, dynamic quorum enables WSFC to run a cluster when only one node and a quorum device are available [172].

G.4: Isolation. HACs may “isolate” a particular node from the rest of the cluster, i.e., prevent it from allocating any resources. The objective of node isolation is to preserve data integrity by employing several mechanisms, such as putting a fence around a node (fencing) or shutting down a node.

G.4.1: Fencing. There are two types of fencing, node-level and resource-level [49, 284, 265, 111]. The common implementation is to employ the node-level fencing [284, 265, 111].

G.4.1.1: Resource. Resource-level fencing isolates one or more critical resources and, by doing so, renders a node unusable because the node cannot allocate resources. Resource-level fencing can be based on a SAN switch, allowing only one node to connect to the SAN-based storage or SCSI. SCSI-based fencing often uses a SCSI-3 option called persistent reservation, which means there can be only one SCSI-3 persistent reservation per disk at any given time, making it an efficient method for

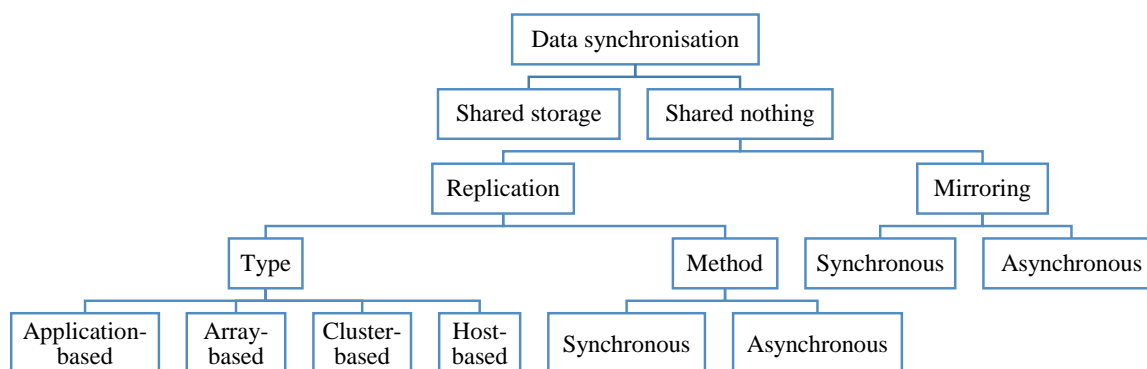


Figure 3.11 Data synchronisation.

isolating disks [284, 265, 111, 303]. Since resource-level fencing is based on storage input/output (I/O), it is sometimes called *I/O fencing* [218].

G.4.1.2: Node. On the other hand, node-level fencing acts at a node-level and isolates or quarantines the node completely [156]. In some cases, the node can be shut down instead, but the fencing functionality still manages the operation. Furthermore, the state of the fenced node is effectively changed so that it is no longer recognised as an active node by the cluster. Thus, the isolated node is not participating in any cluster operations.

G.4.2: Shutdown. A node shutdown is different from the shutdown procedure managed by the fencing functionality because it operates outside the fencing mechanism. This can be achieved by a HAC module that interacts with operating systems or servers using industry-standard specifications. Examples of APIs based on specifications are: Intelligent Platform Management Interface (IPMI) and vendor-specific embedded technology, such as Integrated Lights-Out (iLO) by HPE [265, 246, 183, 143].

3.1.8 H: Data Synchronisation.

Data synchronisation refers to the means, technologies and methods used to synchronise data between cluster nodes. The different layers of EAs require that data are synchronised to ensure consistency across all cluster nodes. Although a diverse range of synchronisation methods can be employed at the different layers, the overall responsibility for all layers managed by HACs lies with the HACs because they are responsible for failover management and ensuring data integrity. HACs may employ additional tools or features that come with the application components to facilitate data synchronisation. Hence, we identify three principal areas of data synchronisation:

1. *Client-state* (i.e., session state replication) deals mainly with client connectivity (e.g., sessions), which means the client state of an application running on a primary node is synchronised with other cluster nodes [231, 276, 178]. Subsequently, a failover can occur seamlessly and without losing any connection data or affecting any active connections. Hence, other nodes can continue

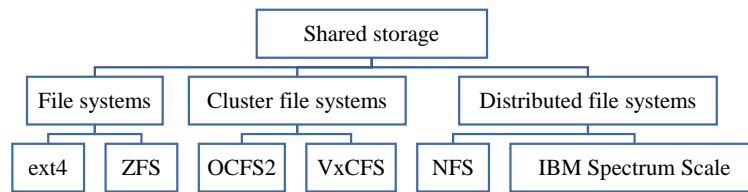


Figure 3.12 File systems related to shared storage. Key: ext4 – Fourth extended file system, ZFS – Z File System, OCFS2 – Oracle Cluster File System, vxVFS – Veritas Cluster File System, NFS – Network File System, IBM Spectrum Scale – A distributed file system, formerly called the General Parallel File System (GPFS)

to support the connections instead. Client-state synchronisation is widely employed in HAC for the application area *network appliances* (e.g., firewalls) [189, 39, 207, 73].

2. *Cluster-state* employs different methods to synchronise the state of a cluster, and it can be considered as an intra-cluster activity. For instance, OpenSAF uses a checkpoint service to record the state of an application or a service. Subsequently, states are replicated to a standby application or service that is hosted on the secondary node [268]. A more advanced approach is a *State Machine Replication* (SMR) which creates replicas of client and process states to one or more nodes deterministically [137], which can even support more comprehensive solutions such as databases [210, 160]. An example of SMR concerning a HAC is an implementation of a HAC for HPC, which employed SMR to synchronise states between nodes in a symmetric active-active topology [66]. A variation of SMR is *Replicated State Machine* (RSM) employed by Veritas Cluster Server (VCS) to synchronise the resource status across all nodes [284].
3. *Application-state*, on the other hand, implies that the data of an application that a HAC protects are synchronised to one or more nodes to support a possible failover. Hence, data synchronisation in this taxonomy refers implicitly to application-state synchronisation. Such synchronisation can occur at different levels, such as on an application or a file system level.

The top-level class *data synchronisation* is shown in Figure 3.11. It is further divided into two storage technologies, shared storage and shared-nothing. They both can be connected to a subclass of file systems which, in turn, can influence the configuration of a HAC. An example of file systems related to shared storage is presented in Figure 3.12. Both cluster and distributed file systems support concurrent access and are ideal for sharing data between multiple nodes [251]. A distributed file system can be deployed on the top of either shared storage or shared-nothing, and some file systems can be deployed on both. For example, IBM Spectrum Scale (formerly the General Parallel File System (GPFS)) file system can be deployed using both storage technologies [113].

H.1: Shared storage. HAC solutions use several forms of shared storage. However, requirements for such implementation usually come from business requirements, such as supporting DR or geographically dispersed user groups. While shared storage might be ideal for the cluster types local and

campus, metro and continental clusters require a different solution due to extended distances between nodes. Shared-nothing is an option in such cases. A hybrid approach is also possible, which means that shared storage and shared-nothing can support a combination of a local cluster (or campus) and continental cluster, as discussed in the topology section.

H.2: Shared-nothing. This setup assumes that there is no shared storage. Instead, each cluster node is connected to separate storage, which could either be SAN-based or based on local storage (e.g., Direct-attached storage (DAS)) [167, 239, 49]. However, EAs must explicitly support these kinds of setups. Moreover, there are also challenges with accessing shared storage in new and emerging technologies. For example, shared storage is limited in the public cloud; hence, it becomes difficult to set up a HAC using shared storage. In such cases, replication between the individual storage units is required, and this has led to the new term *SANless* (SAN-Less) [252]. There are two techniques associated with the synchronisation of data in a shared-nothing setup: replication and mirroring.

H.2.1: Replication. Replication describes the process of replicating from a primary node to other nodes so that data is synchronised and consistent across all participating nodes [167, 239, 49, 284]. Since there are different kinds of replications in HACs, we group them by type and method. The type describes the replication approaches, while the method specifies the execution technique.

H.2.1.1: Type. Four types of HAC replication are possible: application-based, array-based, cluster-based, and host-based.

Application-based replication is set up at an application level, and it uses replication features that are provided natively by an application [238]. One of the nodes will be active in such a setup, while other nodes will be either warm standby or hot standby. To include an application in a HAC, explicit support for the application-specific replication feature by the HAC is required. Databases employ application-based replication to synchronise with standby databases [298, 141, 203, 155]. *Array-based replication* is set up on a storage system level to enable synchronisation between two storage systems (e.g., SAN- or NAS-based) [49, 238]. Additional software may be required to facilitate array-based replication. In *cluster-based replication*, the replication functionality is within a HAC and entirely administrated by the HAC [183, 253]. It means that the solution is independent of the operating environment or any other tool; instead, it relies on a high-speed network connection. *Host-based replication* uses software tools on a host (server or nodes) to perform replication. An example is using a Linux logical volume manager (LVM) to set up replication between two logical volumes across two nodes [49, 112]. Tools that operate on an operating system level and are similar to host-based replication can also be included in this category [303]. For instance, Gómez et al. [89] use a software-based Distributed Replicated Block Device (DRBD) solution to enable replication between two volumes at a block-level in a virtual cluster setup.

H.2.1.2: Method. HACs can perform the replication synchronously or asynchronously.

Synchronous replication waits until a write is completed and an acknowledgement is received from the other replication end, guaranteeing consistency between the two replication points [49]. From a transaction viewpoint, synchronous replication can support all the ACID properties. Therefore,

no data loss is usually associated with it [123, 216]. However, synchronous replication is challenging with extended distances. Nevertheless, modern techniques may offer solutions. For instance, Schmidt [239] means that connections up to a distance of 100 km can be achieved by using dark fibre. This results in latencies of 0.5 μ s, which is adequate for synchronous replication. On the other hand, *asynchronous* replication does not wait until the writing is completed but gets an acknowledgement as soon as data is received at the second point [49, 155]. As such, it may not comply with the ACID properties entirely, which, in turn, may result in data loss.

Different factors influence the selection of a method, and some of the critical factors are [167, 49]: the distance between two nodes; the volume of data transported between nodes; type of data; frequency (continuous or burst); business requirements, such as DR. There is a network latency recommendation for synchronous replication, as it implies real-time mirroring, while asynchronous replication does not have the same kind of rigorous requirement [49, 216].

H.2.2: Mirroring. In some cases, the terms replication and mirroring are used interchangeably. For example, a host-based mirroring of a file system can also be referred to as file system replication. However, in other cases, a few differences can be observed; for instance, mirroring may differ by not having a running instance on the standby node [49, 239]. Mirroring can be performed synchronously or asynchronously [167, 49, 239].

Synchronous mirroring ensures that the mirroring process waits until a write is completed and committed on the standby node and an acknowledgement is sent back. This method secures consistency of data between two nodes. An *asynchronous* mirroring process, on the other hand, does not wait until a write is ended on the secondary node. This approach may result in data loss when the primary node fails abruptly.

3.2 Survey of High-availability Cluster Solutions

There are many different implementations of HAC across all layers. In this section, we focus on enterprise and large systems. This means the selected HACs can support typically a subset or all of the layers 1, 2 and 3 from Table 2.1.

3.2.1 Survey Methodology

We used a systematic approach detailed in Section 3.2.2 to select our survey's 17 end-to-end HAC solutions. For the analysis of these solutions, we combined the analysis of the HAC documentation, white papers, case studies, books and articles, and an online questionnaire sent to the vendors of the selected HACs. This hybrid analysis methodology is presented in Section 3.2.3.

The detailed results of the survey, which involved categorising the selected HAC solutions based on the HAC taxonomy from Section 3.1, are provided in Section 3.2.4. The following section, Section 3.2.5, presents an analysis of the survey results and discusses our key findings.

Table 3.6 Selection questions

No	Question	Evaluation Parameters
Q1	Support for enterprise class databases?	SAP ASE, DB2, HANA, Informix, MySQL, Oracle, PostgreSQL, SQL Server, Teradata
Q2	Support for EAs?	Oracle Siebel Customer relationship management (CRM), Oracle [†] , SAP [†] , Others [†] , WebSphere
Q3	Multi-tier support for EAs?	X-Yes, N-no, ?-no information
Q4	Enterprise support provided?	24x7x365
Q5	Application features can be supported by further developments?	X-Yes, N-no, ?-no information
Q6	Support for disaster recovery?	X-Yes, N-no, ?-no information
Q7	Support for virtualization?	X-Yes, N-no, ?-no information
Q8	Cloud support?	X-Yes, N-no, ?-no information
Q9	Support for enterprise operating environments?	AIX, HP-UX, IBM i, Linux, Solaris, Windows
Q10	Support for multiple platforms?	Power, SPARC, x86
Q11	Support for large-scale clusters?	Number of nodes
Q12	Support for multiple topologies?	Active-active, application-based, server-based, N-to-N, active-passive, N+1, N+M, N-to-1
Q13	Support for availability level?	Minimum 99.9%
Q14	Active lifecycle management?	X-Yes, N-no, ?-no information

[†] - any of the business suite EAs (e.g., ERP).

3.2.2 Approach for Selecting the Surveyed HACs

The systematic six-step approach we employed to select the relevant HACs for our survey is detailed below.

Step 1. Identification of HACs that support enterprise applications (EAs). Our survey focused on HACs that can protect EAs. However, only a limited number of HACs support EAs due to the complex composition of EAs, which are multi-tiered and multi-layered. We identified likely candidate HACs using comprehensive research reports [118, 219] and articles [139, 65, 221, 48], resulting in 23 candidate HAC solutions.

Step 2. Identification of relevant EAs and databases. In this step, we gathered information for assessing the applicability of each HAC solution to distinct layers of enterprise applications. To this end, we used relevant research and analysis reports, e.g. [81, 82, 80], to identify the databases and EAs listed next to questions Q1 and Q2 from Table 3.6.

Step 3. Elimination of HACs not supported by EA vendors. In this step, we used the lists of supported HACs released regularly by enterprise application vendors, e.g. [236, 116], to check which HAC

3.2 Survey of High-availability Cluster Solutions

solutions are supported (and sometimes certified) by these EA vendors. “Supported” HAC solutions are solutions that fulfil the requirements of the application vendor for a specific application, with the added implication that support channels have been established between the vendors.

We assessed the candidate HAC solutions using the following criteria to narrow down the list:

1. Does the HAC solution being assessed focus on only specific IT solutions (such as HPC or Hadoop)?
2. Is the HAC solution no longer active, implying that the product lifecycle has ended or the research project that developed it has ended?
3. Is the information available to analyse the HAC solution properly insufficient?
4. Are EAs supported by the HAC solution, or is it the case that the information available cannot be used to conclude whether EAs are supported or not?

We eliminated all the candidate solutions for which one or several of these questions were answered affirmatively. As a result, six HAC candidates were removed in this step, and we proceeded with the remaining 17 candidates. We made an exception for two of the candidate HAC solutions for the reasons described below:

- OpenSAF does not provide enterprise support directly. Nevertheless, we retained OpenSAF because of its stability as a HAC [124, 128]. Besides, application support can be developed individually with OpenSAF, meaning that an OpenSAF HAC can be used to support enterprise-class databases and applications.
- Similarly, the ClusterLabs stack does not support enterprise applications on its own. However, we retained the ClusterLabs stack because it provides the core components for two other selected solutions, SUSE Linux Enterprise High Availability Extension and Red Hat High Availability Add-On. This implies that customisation and further developments are possible using it.

Table 3.7 Evaluation of selected HAC solutions

Question No	ApplicationHA	Clusterware	EXPRESSCLUSTER X	InfoScale Availability	OpenSAF	ClusterLabs stack	PowerHA SystemMirror	PRIMECLUSTER	Red Hat High Availability Add-On	RSF-1	SafeKit	Serviceguard	SIOS Protection Suite	Solaris Cluster	SUSE Linux Enterprise High Availability Extension	Tivoli System Automation for Multiplatforms (SA MP)	Windows Server Failover Clustering (WSFC)
Q1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Q2	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Q3	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Q4	X	X	X	X	N	N	X	X	X	X	X	X	X	X	X	X	X
Q5	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Q6	?	?	X	X	X	X	X	?	X	?	?	X	X	X	X	X	?
Q7	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Q8	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Q9	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Q10	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Q11	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Q12	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Q13	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Q14	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

X - Yes, N - No, ? - No information

Step 4. Retention of only HACs that support automatic failover. We used this filter to retain only the HAC solutions that support automatic failover, which is crucial for an EA to minimise downtime. All 17 candidates support automatic failover; hence, all were retained.

Step 5. Design of additional questions for the selection and evaluation of HACs. In this step, we created the questions to evaluate the HACs. The queries reflected the typical requirements of EAs [75], and the objective was to select those HACs that could respond to most of the questions positively. The set of questions is listed in Table 3.6.

Step 6. Selection of the set of HAC solutions for the survey. We selected all the HAC solutions that can support EAs and fulfil the additional criteria from the questions Q1–Q14 in Table 3.6, where a positive response for any of the “evaluation parameters” from questions Q1, Q2, Q9, Q10 and Q12 was deemed sufficient to consider that a HAC solution met the criterion associated with that question. The

Table 3.8 Eliminated HAC solutions in the six-step approach for selecting HACs for survey

Product	Reason(s) for Elimination
Apache Mesos [56]	focus on specific IT solutions (HPC)
DxEnterprise [58]	lack of EA support; insufficient information available to evaluate the solution properly
everRun [259]	lack of EA support; insufficient information available to evaluate the solution properly
HA-OSCAR [91]	no longer active
Kimberlite [139]	no longer active
Linux FailSafe [139]	no longer active

result of the HAC selection is presented in Table 3.7, which comprises 17 HAC solutions for which we obtained positive responses to all queries and products, while noting the following exception:

- DR support (question Q6) for the following solution was unclear or not available: ApplicationHA, Clusterware 12c, Primecluster, RSF-1, SafeKit and WSFC.

For completeness, we also provide, in Table 3.8, a list of the six HAC solutions considered initially but eliminated in Step 3 of our selection approach. For each of these solutions, Table 3.8 also provides a summary of the reasons for its elimination from the survey.

3.2.3 HAC Analysis Methodology

As a first step, we created a comprehensive spreadsheet and an online questionnaire covering our entire HAC taxonomy, which we used as a basis for the survey. In the second step, we employed a hybrid methodology to survey the 17 selected HACs. We populated the spreadsheet ourselves for 17 HACs by analysing product documentation, technical white papers, case studies, books, and articles in the primary method. We noticed several inconsistencies between the different materials for the same edition and version of a HAC solution. To resolve these inconsistencies, we crosschecked the results by using a diverse set of materials (e.g., reference guides, technical manuals and documentation) whenever inconsistencies were observed. In the secondary method, we prepared an email that described what we were trying to achieve. We sent it to all the vendors of selected HACs, particularly to the experts responsible for the HAC products. After two weeks, we sent a reminder to those who did not reply to our original invitation; a second reminder was sent after an additional two weeks. After six weeks, we collected the data provided by the vendors and transferred it to a spreadsheet. Despite assurance from multiple vendors, we managed to obtain a response from only one vendor, High Availability for the HAC RSF-1. Subsequently, we transferred all collected data to the spreadsheet for conducting further analysis.

3.2.4 Survey Results

We used the taxonomy to establish the characteristics of the 17 end-to-end HAC solutions selected for the survey. The outcome of the survey is presented in Table 3.9, starting with general information about each HAC solution (i.e., version and vendor) in the second and third row. The remaining rows from the table present the main results of the survey, organised in the same way as our HAC taxonomy. The results are analysed in Section 3.2.5.

The surveyed HAC solutions usually consist of multiple editions with varying features, some of which are subject to additional licensing. Our survey covers only advanced editions that include most of the features. As even advanced editions do not support all the features when different operating systems and platforms are considered, we provide details about the limitations relating to the individual HACs where applicable (as footnotes at the end of Table 3.9).

As discussed, a HAC vendor may enforce further constraints by stating explicitly what version and edition of an EA are supported. Likewise, an EA vendor may also list what HACs are supported with a particular version and edition of an EA. Many combinations of EA version, database version, HAC version and edition, operating system version, and platform make it challenging to crosscheck every single combination. Therefore, only the relevant EAs and databases are included in Table 3.9.

Table 3.9 Outcome of the survey

Taxonomy	ApplicationHA ¹	Clusterware	EXPRESSCLUSTER X	InfoScale Availability	OpenSAF	ClusterLabs stack	PowerHA SystemMirror ¹	PRIMECLUSTER	Red Hat High Availability Add-On	RSF-1 ²³	SafeKit	Serviceguard	SIOS Protection Suite	Solaris Cluster	SUSE Linux Enterprise High Availability Extension	Tivoli System Automation for Multiplatforms (SA MP)	Windows Server Failover Clustering (WSFC)
Version	6.2	12c	3.3	7.3.1	5.17.07	2.3.2	7.2.1	4.5	7.0	3.9.10	7.2	A.12.20 ²⁴	9.2	4	12	4.1	2016
Vendor	Veritas	Oracle	NEC	Veritas	SA Forum	ClusterLabs	IBM	Fujitsu	Red hat	High-Availability	Evidian	HPE	SIOS	Oracle	SUSE	IBM	Microsoft
A: Deployment Patterns																	
OS and platform																	
AIX on Power	X ¹	X ⁴	NS	X	NS	NS	X	NS	NS	NS	NS ²²	NS	NS	NS	NS	X	NS
HP-UX on IA64	NS	X	NS	NS	NS	NS	NS	NS	NS	NS	NS	X	NS	NS	NS	NS	NS
IBM i on Power	NS	NS	NS	NS	NS	NS	X	NS	NS	NS	NS	NS	NS	NS	NS	NS	NS
Oracle Linux on SPARC	NS	NS	NS	NS	X ⁸	X	NS	NS	NS	NS	NS	NS	NS	NS	NS	NS	NS
Oracle Linux on x86_64	X ¹	X ⁴	X	X	X	X	NS	NS	NS	X	NS	NS	X	NS	NS	NS	NS
Red Hat Enterprise Linux on Power	NS	NS	X	NS	X ⁸	X	X	NS	X	X	NS	NS	NS	NS	NS	X	NS
Red Hat Enterprise Linux on x86_64	X ¹	X	X	X	X	X	NS	X	X	X	X	X	X	NS	NS	X ¹⁰	NS

Continued on next page

3.2 Survey of High-availability Cluster Solutions

Table 3.9 – Continued from previous page

Solaris on SPARC	X ₁	X	NS	X	NS	NS	NS	X	NS	X	NS	NS	NS	X	NS	X	NS
Solaris on x86_64	NS	X	NS	X	NS	NS	NS	NS	NS	X	NS	NS	NS	X ¹¹	NS	NS	NS
SUSE Linux Enterprise Server on Power	NS	NS	X	NS	X ⁸	X	X	NS	NS	X	NS	NS	NS	NS	X	X	NS
SUSE Linux Enterprise Server on x86_64	X ₁	X	X	X	X	X	NS	X	NS	X	NS ²²	X	X	NS	X	X ¹⁰	NS
Windows	X ₁	X	X	X	NS	NS	NS	NS	NS	NS	X	NS	X	NS	NS	NS	X
Support for virtualized environments	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Supported virtual solutions (E-Xen, H-Hyper-V, K-KVM, O-Others, V-VMware)	E, H, K, O, V	O	E, H, V	H, K, O, V	K, O, V	E, H, K, V	O ⁹	K, O, V	K	?	H	H, K, V	E, H, K, V	O	E, K	K, O, V	H, V
Maximum number of nodes per cluster	?	64 5	32	128	100	32	16	16	16	64	?	16/32 25	32	8/16 26	32	32/ 130 ¹⁴	64
B: Application Areas (EA category, B-Business-critical, T-telecom (carrier-grade))	B	B	B	B	T, B	B	B	B	B	B	B	B	B	B	B	B	B
C: Type of cluster																	
C.1: Local	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
C.2: Campus	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
C.3: Metro	?	X	X	X	x	X	X	NS?	X	X?	NS?	X	X	X	X	x	X
C.4: Continental	?	NS?	X	X	x	X	X	NS?	X	NS	NS?	X	X?	X	X	x	NS?
D: Topology																	
D.1: Symmetric																	
D.1.1: Active-active																	
D.1.1.1: Application-based	NS	X	X ⁷	X ⁷	X ⁷	X ⁷	X ⁷	X ⁷	X ⁷	X ⁷	X ⁷	X ⁷	X ⁷	X ⁷	X ⁷	?	X ⁷
D.1.1.2: Server-based	X	X	X	X	X?	X	X	X	X	X	X	X	X	X	X	X	X
D.1.2: N-to-N	X?	?	X	X	X	X	X?	X?	?	NS	?	X?	X	X	X	?	X
D.2: Asymmetric																	
D.2.1: Active-passive	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
D.2.2: N-to-1	X?	?	X	X	X	X	X	?	X	X	X	X	X	X	X	?	NS
D.2.3: N+1	?	?	X	X	X	X	X?	?	?	X	?	X?	X?	X	X	?	NS?
D.2.4: N+M	?	?	X	X	X	X	X?	?	?	X	X	X?	X?	X	X	?	NS?
E: Cluster management																	
E.1: Cluster data																	
E.1.1: Configuration (D-Disk or file share, F-File, M-memory)	D,F	F	F	F	F	F	D,F	F	F	D,F	F	F	D,F	F	F	F	F
E.1.2: Runtime (D-Disk or file share, F-File, M-memory)	D, M?	F, M?	F, M?	F, M?	F, M?	F, M?	D,F,M?	F, M?	F, M?	D, M?	F, M?	F, M?	D,M?	F, M?	F, M?	F, M?	F, M?
E.2: Communication																	
E.2.1: Type																	
E.2.1.1: Heartbeat	X ¹⁶	X	X	X	NS	X	X	X	X	X	X	X	X	X	X	X	X
E.2.1.1.1: LAN-based	X	X	X	X	NS	X	X	X	X	X	X	X	X	X	X	X	X
E.2.1.1.2: Disk-based	?	NS	X	NS	NS	NS	X	NS	NS	X	NS	NS?	NS?	X	NS	X	NS?
E.2.1.2: Node																	
E.2.1.2.1: User interface	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
E.2.1.2.2: Resource management																	
E.2.1.2.2.1: Agent	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	NS	NS
E.2.1.2.2.2: Base resource	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
E.2.1.3: Cluster																	
E.2.1.3.1: Configuration	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
E.2.1.3.2: Runtime	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
E.2.2: Method																	
E.2.2.1: Multicast	X?	X	?	NS?	X	X	X	?	X	X	NS?	X	NS?	X	X	?	X?
E.2.2.1.1: Atomic	?	?	?	NS?	X	X	?	?	X	?	?	?	?	?	X	?	?
E.2.2.1.2: Virtual synchrony	NS	NS	NS	NS	?	X	NS	NS	X	NS	NS	NS	NS	NS	X	NS	NS
E.2.2.2: Broadcast	?	?	?	X	X	X	X	X	X	?	X	X	X	X	X	X	X
E.2.2.3: Unicast	?	?	?	X	X	X	X	?	X	X	X	X	?	?	X	?	?
E.2.2.4: IP socket	NS	NS	NS	NS	NS	NS	NS	NS	NS	X	NS	NS	X	NS	NS	NS	NS
E.3: Resource management																	
E.3.1: Type																	

Continued on next page

Taxonomy and Survey of High-availability Clusters

Table 3.9 – Continued from previous page

E.3.1.1: Base resource	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
E.3.1.2: Application	X	X	X	X	x	X	X	X	X	X	X	X	X	X	X	X	x
E.3.1.2.1: Agent-based	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	NS	NS
E.3.1.2.1.1: Application (C-Siebel CRM, O-Oracle, S-SAP, T-Others W-WebSphere)	C, S, T	C, O, S, T	S, T, W	O, S	T ¹²	O, S	O, S, T, W	O, S, T, W	O, S, T, W	O, S, T, W	O, S, T, W	O, S, T, W	O, S, T, W	O, S, T, W	O, S, T, W	O, S, T, W	C, O, S, T, W ¹⁵
E.3.1.2.1.2: Database (A-SAP ASE, D-DB2, H-HANA, I-Informix, M-MySQL, O-Oracle, P-PostgreSQL, S-SQL Server, T-Teradata)	D, M, O, S	M, O	A, D, H, M, O, P, S	A, D, H, M, O, S	M ¹²	D, M, O, P	D, H, O	O	A, D, M, O, P	A, D, I, M, O, P	O, M, S	A, D, H, M, O, P, S	A, D, I, M, O, P, S	A, M, P, O	D, H, I, M, O, P	D, H, O	A, D, M, O, P, S ¹⁵
E.3.2: Method																	
E.3.2.1: Policy-based	X	X	X	X	X	X	X	X	X	NS	X	X	X	X	X	X	X
E.3.2.2: Rule-based	?	?	?	?	?	X	?	?	?	X	?	?	?	?	?	?	X?
F: Failure detection and recovery																	
F.1: Monitoring	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
F.1.1: Area																	
F.1.1.1: Server	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
F.1.1.2: Cluster	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
F.1.1.3: Application	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X ⁸
F.1.2: Type																	
F.1.2.1: State-based	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
F.1.2.2: Threshold-based	?	X	X	X	X	NS?	NS?	X	NS	X	NS?	?	X	?	NS?	NS?	X
F.1.3: Method																	
F.1.3.1: Poll	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
F.1.3.2: Push	NS	NS	?	NS	?	NS	?	?	NS?	NS	NS	?	NS?	NS	NS	NS?	?
F.1.3.3: Event-based	X ²	NS?	NS?	X	?	NS	NS	X	NS	NS	NS	NS	NS	NS	NS	NS	NS
E.2: Failover																	
E.2.1: Reactive	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
E.2.2: Proactive	NS	NS	NS	NS	NS	NS	NS	NS	NS	NS	NS	NS	NS	NS	NS	NS	NS
E.3: Recovery level																	
E.3.1: Resource	X	X	X	X	X	X	X	X	X	?	?	X	X	X	X	X	X
E.3.2: Group	X	X	X	X	X	X	X	X	X	?	X	X	X	X	X	X	X
E.3.3: Node	X	X?	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
E.4: Prediction	NS	NS	NS	NS	NS	NS	NS	NS	NS	NS	NS	NS	NS	NS	NS	NS	NS
G: Consistency and integrity																	
G.1: Cluster lock	NS	?	X	NS	X	NS	NS	X	X	?	?	X	X	NS	NS	X	NS
G.2: Quorum	?	X	NS	X	NS	X	X	X ²⁰	X	NS	NS	X	X	X	X	X	X
G.2.1: Device realisation																	
G.2.1.1: Server	?	NS	NS	X	NS	X	NS	X	X	NS	NS	X	X	X	X	X	X
G.2.1.2: Disk	?	X	NS	NS	NS	NS?	X	NS	X	NS	NS	NS	NS	X	X	X	X
G.2.1.3: File share	NS	NS	NS	NS	NS	NS	X ²¹	NS	NS	NS	NS	NS	NS	X	NS	X	X
G.2.2: Mode																	
G.2.2.1: Server	?	NS	NS	X	NS	X	NS	X	X	NS	NS	X	X	X	X	X	X
G.2.2.2: Node	NS	NS	NS	?	NS	?	?	?	?	NS	NS	?	?	?	?	?	X
G.2.2.3: Disk	?	X	NS	NS	NS	NS?	X	NS	X	NS	NS	NS	NS	X	X	X	X
G.2.2.4: File share	NS	NS	NS	NS	NS	NS	X	NS	NS	NS	NS	NS	NS	X	NS	X	X
G.3: Dynamic quorum	?	?	NS	X	NS	X	X	X	X	NS	NS	X	?	X	X	X	X
G.4: Isolation																	
G.4.1: Fencing	NS	X	NS	X	NS	X	X	X	X	NS	X	X	X	X	X	?	NS
G.4.1.1: Resource	NS	NS?	NS	X	NS	X	X	X	X	NS	X	X	X	X	X	?	NS?
G.4.1.2: Node	NS	X	NS	X	NS	X	?	?	X	X	X ¹⁹	?	X	?	X	?	NS
G.4.2: Shutdown	NS	X	X	NS	X	x ¹³	X	X	X	X	NS	X	X	X?	NS?	X	NS
H: Data synchronisation																	
H.1: Shared storage	X	X	X	X	X	X	X	X	X	X	?	X	X	X	X	X	X
H.2: Shared-nothing	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
H.2.1: Replication	X	X	x	X	X	X	X	X	X	X	X	X	X	X	X	X	x
H.2.1.1: Type																	
H.2.1.1.1: Application-based	X ⁶	X ⁶	X ⁶	X ⁶	?	X ⁶	X ⁶	X ⁶	X ⁶	X ⁶	X ⁶	X ⁶	X ⁶	X ⁶	X ⁶	X ⁶	X ⁶
H.2.1.1.2: Array-based	x	x	x	x	x	x	x	x	x	?	?	x	x	x	x	?	x
H.2.1.1.3: Cluster-based	NS	NS	X	NS	NS	NS	NS	NS	NS	NS	X	NS	X ¹⁸	NS	NS	NS	NS

Continued on next page

3.2 Survey of High-availability Cluster Solutions

Table 3.9 – Continued from previous page

H.2.1.1.4: Host-based	x ¹⁷	x	x	X	x	X	X	X	X	?	NS	X	X	X	X	?	x
H.2.1.2: Method																	
H.2.1.2.1: Synchronous	X	X	X	X	X	X	X	X	X	?	X	X	X	X	X	X	X
H.2.1.2.2: Asynchronous	X	X	X	X	X	X	X	X	X	?	X	X	X	X	X	X	X
H.2.2: Mirroring	x	x	x	x	?	x	x	x	x	x	x	x	x	x	x	?	x
H.2.2.1: Synchronous	x	x	x	x	?	x	x	x	x	x	x	x	x	x	x	?	x
H.2.2.2: Asynchronous	x	x	x	x	?	x	x	x	x	x	x	x	x	x	x	?	x

Rows that are blue-coloured indicate top-level classes and subclasses

?- No information

X - Supported

NS - Not supported

X? - Supported. Not explicitly stated in the documentation, but this interpretation has been made by analysing the documentation.

x - Supported together with additional components, and an example is replication support by the operating system volume manager.

NS? - Not supported. Not explicitly stated in the documentation, but this interpretation has been made by analysing the documentation.

¹ Supported only on virtualized environments.

² Intelligent monitoring framework.

³ Replication or mirroring support by additional tools is included.

⁴ Supported on both virtual and physical environments.

⁵ 64 nodes are supported for the hub, while leaf nodes can support many more.

⁶ Replication is provided natively by an application, but a HAC must support the feature.

⁷ If an application supports parallel deployments.

⁸ OpenSAF provides a generic development package; it can be ported to other UNIX and Linux flavours.

⁹ LPARs: 2 logical partitions (LPARs) on IBM PowerVM

¹⁰ Supported on System x hardware that is based on the x86 platform.

¹¹ Supported only on Oracle's x86 platforms.

¹² The implementer can develop application support.

¹³ Fencing by STONITH (Shoot the Other Node in the Head).

¹⁴ The maximum number of nodes on Linux is 32, and, for AIX, it is 130.

¹⁵ Application vendors provide application support for WSFC.

¹⁶ Usually, guest heartbeat is passed to a host.

Taxonomy and Survey of High-availability Clusters

Table 3.10 The surveyed HACs, versions and vendors

Surveyed HAC	Vendor	Surveyed HAC	Vendor
ApplicationHA 6.2*	Veritas	RSF-1 3.9.10*	High-Availability
ClusterLabs stack 2.3.2*	ClusterLabs	SafeKit 7.2*	Evidian
Clusterware 12c*	Oracle	Serviceguard A.12.20*	HPE
EXPRESSCLUSTER X 3.3*	NEC	SIOS Protection Suite 9.2*	SIOS
InfoScale Availability 7.3.1*	Veritas	Solaris Cluster 4	Oracle
OpenSAF 5.17.07*	SA Forum	SUSE Linux Enterprise High Availability Extension 12*	SUSE
PowerHA SystemMirror 7.2.1	IBM	Tivoli System Automation for Multiplatforms (SA MP) 4.1*	IBM
PRIMECLUSTER 4.5*	Fujitsu	Windows Server Failover Clustering (WSFC) 2016	Microsoft
Red Hat High Availability Add-On 7.0*	Red hat		

* = Solution that functions as middleware

¹⁷ Replication features of a virtual machine can also be used.

¹⁸ Replication feature is provided by the product DataKeeper, which is part of the SIOS Protection Suite.

¹⁹ Fencing as a concept is not employed, but, instead, the node with the problem is put into a waiting state..

²⁰ The solution uses a quorum technique called cluster integrity.

²¹ Implies repository disk.

²² Supported by SafeKit 7.1.3.

²³ The vendor provided most details.

²⁴ Version for Linux. Current version for HP-UX is A.11.20.

²⁵ The maximum number of supported nodes for Linux is 32, while for HP-UX, it is 16.

²⁶ The maximum number of supported nodes on Solaris on x86 is 8, and Solaris on SPARC supports 16.

Notes:

Although the operating system version is not stated, it is the most recent version at the time this survey was carried out.

3.2.5 Analysis of the Survey Results

The distribution of the operating system and platform support for the surveyed HACs is shown in Figure 3.13 grouped by the operating system. Linux is the dominating operating system, and 15 solutions support Linux, out of which 12 support SUSE Linux on an x86-based platform, seven support SUSE Linux on Power-based platforms. Similarly, Red Hat Linux supports 13 HACs on the

3.2 Survey of High-availability Cluster Solutions

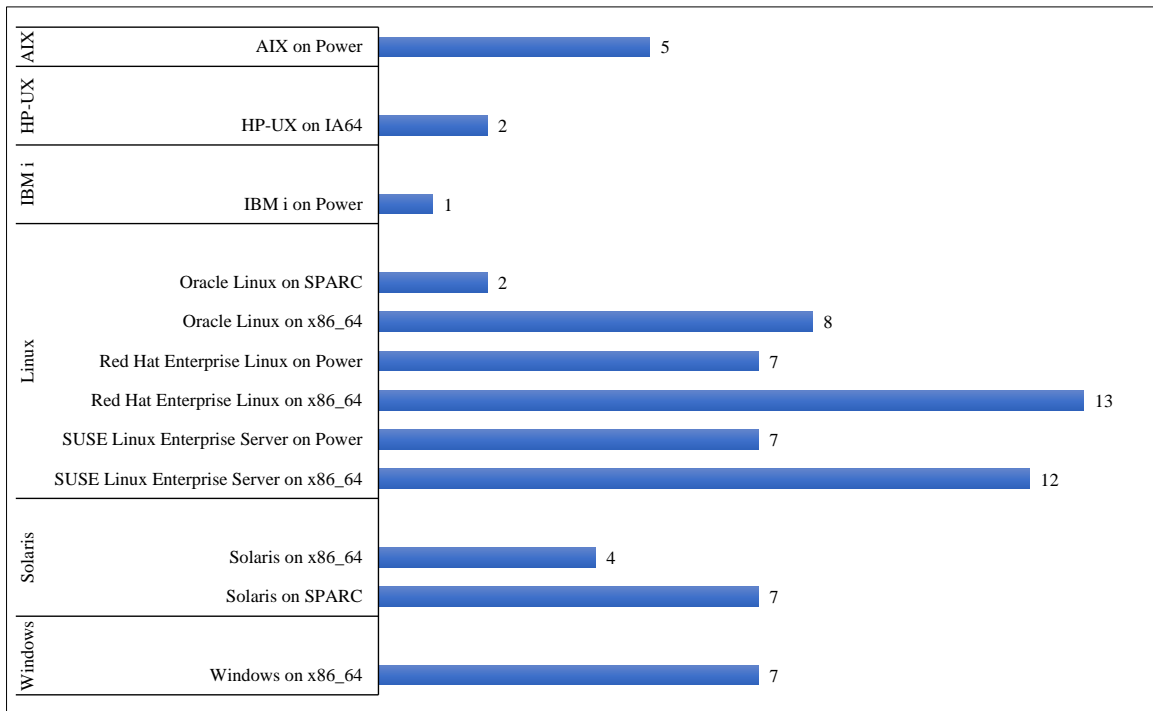


Figure 3.13 Platform and operating system support of the surveyed high availability clusters (HACs) grouped by operating system.

x86 platform and seven on the Power platforms. Oracle Linux is supported by 8 HAC solutions on x86 platforms, while only two support it on the SPARC platform. Solaris operating system is supported by seven HACs on the SPARC platform, while only four support Solaris on the x86 platform. Seven solutions support windows, and the platform is always x86. Five HACs support AIX on power, and only two HACs support HP-UX on the IA64 platform. Lastly, the rare environment is IBM i on the Power platform, which only one HAC supports.

The surveyed HAC solutions can be divided into two groups. The first group, comprising 14 of the 17 surveyed solutions, comprises the HACs marked with a star ‘*’ in Table 3.10. Each of these HACs functions as middleware, which means that it creates an additional layer on the top of an operating environment. The HACs from the second group, which comprises the remaining three solutions, are tightly integrated with an operating system and make use of the features that are offered by an operating environment. The latter type of HAC functions as part of an operating environment, operating in the kernel mode and directly interacting with operating system functionalities. While such features can make a HAC more efficient, they may also create problems with modularity and portability, and that is why, for example, such HAC solutions only support specific operating systems. Furthermore, the lifecycle management of such a HAC solution also becomes the operating system’s lifecycle management. WSFC has already embraced this approach, and it is entirely integrated with the Windows server operating system.

Major software and hardware vendors have their HAC solutions. However, some of them are supported only by the operating environment and platform from the vendor. An example of this is WFCS, which is only available on the Windows server enterprise edition. On the other hand, some independent vendors specialise in HAC products, and these vendors can support multiple operating systems and platform combinations. Typically, such HACs belong to the middleware group.

Cloud deployment has also come to play an important role. In the early days of cloud computing, a separate development of HAC was considered. This led to the development of specific HAC solutions, such as ApplicationHA by Veritas and vSphere App HA by VMware. However, a better approach is to port existing solutions to the cloud environment, which made developing cloud-specific HAC solutions unnecessary. As such, solutions like App HA by VMware were discontinued. However, HACs in the public cloud comes with limitations. For example, using shared storage is a challenge. On the other hand, this has contributed to developing enhancements to enable deploying HACs in the cloud. One such enhancement is the so-called *storage-less* or *SANless* HAC, which allows HACs to operate without shared storage. Moreover, the transition to cloud services models, such as SaaS, PaaS, and IaaS, changes the way HACs are deployed and managed. Likewise, roles and responsibilities for managing a HAC with the different service models also change.

Furthermore, the introduction of *multi-clouds* can also complicate a HAC deployment, not least from a roles and responsibilities perspective. Somasekaram highlights the issues with roles and responsibilities of HA and DR solutions in the context of outsourcing [255]. He argues that the issues are valid even for the cloud environment because the cloud is regarded as outsourcing, and cloud providers are usually responsible for multiple layers (e.g., network and storage). At the same time, other suppliers manage the rest of the layers.

The emerging deployment host container also faces challenges similar to those described for the deployment-environment public cloud and virtual host [1, 177, 170, 145]. In Table 3.11, we compare the key features for Kubernetes-provided HA and the HACs investigated in this thesis.

As presented in Section 3.1.1, Kubernetes provides efficient HA for workloads deployed in containers. However, a separate master node introduces new challenges because a master node is considered a SPOF, and HA must be ensured for the master node. Ensuring HA for stateful application has also been recognised as a challenge [1, 280]; however, this has changed recently, and Kubernetes now even supports stateful applications [266]. The main difference is the application-specific support provided by HACs. The HACs can monitor at the resource level, including application-specific components, whereas Kubernetes monitors at the container level. Application awareness is provided by out-of-the-box agents and extensions so that HACs understand the internals of an application and take appropriate measures based on these, for example, restarting application components in a certain order.

Furthermore, explicit support from EAs investigated in Section 3.2 was missing for Kubernetes at the time of investigation, which is perceived as critical for obtaining support for business-critical EAs. The primary reason might be associated with the effort required to transition applications to use microservice architecture [170]. However, this is rapidly changing, and some commercial vendors

3.2 Survey of High-availability Cluster Solutions

Table 3.11 Comparison of Kubernetes and high availability cluster features

Characteristics	Kubernetes	HAC
Deployment host	Containers	All (only experimental deployments for containers)
Node resource-level monitoring (CPU and memory)	Yes	Yes
Application resource-level monitoring	No	Yes
Failover management	Yes (only at container level)	Yes
Application resource-level failure detection	No	Yes
Heartbeat	Yes (between nodes)	Yes (between nodes)
Application-specific start up and shutdown sequence	No	Yes
Application support	No	Application-specific agents and extensions

have recently announced support for Kubernetes [206, 205], but it is unclear at this point how HA will be provided for applications at a component level.

Another major difference is that Kubernetes is limited to the host container in the deployment pattern, whereas HACs can be deployed using all host types, although only experimental deployment has been performed on containers. Both can monitor node-level resources, such as the CPU usage and memory. There are also differences in the way failures are handled. Kubernetes restarts failed containers, whereas HACs can reinitialise at the resource level [278] and support managing failures at several levels.

Both Kubernetes and HACs offer distinct features. Combining them can improve the availability of container-based applications. However, challenges occur when implementing HACs in a container-based environment to support multilayered EAs. For example, containers run as a process in the user space, which may restrict the implementation of HAC features that require running in the kernel space [222]. Moreover, containers typically support a single application or service in a container, which means the HAC cannot deploy agents in the same container to manage the application resources [286].

To overcome this limitation, the commonly implemented container orchestration system Kubernetes provides a sidecar option (i.e., a separate container), enabling deployments of HAC-related components. The sidecar container runs along with the container that hosts the application. Using this approach, commercial vendors have started providing HACs for containers. InfoScale availability (formerly Cluster Server—VCS) for containers from Veritas is one such HAC that works with Kuber-

netes. This HAC provides monitoring, integrated I/O fencing, arbitration and shared storage using a container storage interface plugin [286] to ensure that the HAC can deliver HA for applications. The solution requires at least two private networks to enable cluster communication and one public network to facilitate heartbeat communication. Research projects also explore the use of existing HACs, such as Pacemaker/Corosync [279] and OpenSAF [4] to support container-based applications.

Latency over long distances has traditionally been a major problem for HACs. However, the technology has evolved and techniques are currently available to reduce latency considerably, enabling the setting up of HACs across substantial distances. *Atomic broadcast* and *multicast* (total order messaging) are often associated with fault-tolerance in distributed systems; hence, there are persuasive arguments to employ it even for HAC communication [246]. However, it is only employed by some of the HACs today.

On the whole, *prediction* is absent from the surveyed HAC solutions. Most solutions employ a poll-based monitoring mechanism that is often state-based, meaning that only the states of the resources are monitored. Moving towards *industry-standards* has also been observed in some areas, such as when using the IPMI to shut down nodes as part of isolating a problematical node. The *SCSI-3* interface is widely employed to isolate on a resource-level, and often as part of *fencing*. The *quorum* concept is commonly employed so that a cluster can take action upon a situation that leads to the partitioning of a cluster.

In conclusion, the current HAC solutions for EAs are dominated by commercial vendors (15 out of the 17 surveyed solutions). This is unsurprising because customers look for HAC solutions for their business-critical applications, and, as such, proper support is paramount. However, this also means that the vendors conduct most of the research. There are, however, some open-source initiatives, and two active initiatives are OpenSAF and ClusterLabs stack (Pacemaker/Corosync). The open-source initiatives often focus on Linux, and there have been different projects to develop a consistent HAC solution for Linux. While such efforts have been split into other projects or discontinued, some of the components are still active, and the current open-source cluster solutions are a combination of various initiatives. The main components of the current setup of the ClusterLabs stack are Corosync, Pacemaker, DRBD, STONITH, and a diverse range of application agents, which are packaged under a ClusterLabs stack. OpenAIS was an initiative to support implementing Application Interface Specification (AIS) developed by the Service Availability Forum (SA Forum), and Corosync originated from that initiative. Pacemaker is a cluster resource manager (CRM) tool that originates from the Linux-HA project.

Application agents follow the standard API established by the Open Cluster Framework (OCF), which helps standardise the application resource management. Both SuSe Linux Enterprise HA Extension and Red Hat Enterprise Linux HA add-on use Pacemaker, Corosync, the OCF concept, and many other open-source components. OpenSAF, on the other hand, focuses on the telecommunications sector, where there is a need to support very high availability for carrier-grade servers that operate in the telecommunication infrastructure. However, there have been initiatives to deploy OpenSAF in a range of environments, such as the cloud. For example, Kanso et al. [124] proposed an OpenSAF

based deployment in the cloud, but it too focuses on telecom applications. The challenge with open-source initiatives is to secure proper support, which is crucial for EAs. On the other hand, Red Hat and SuSe provide such support even though they have developed their HACs using mainly open-source components. It must be noted that there have been several projects related to the development of HACs, both commercial and open-source, over the years. However, many of them are no longer active, and examples include FailSafe by Silicon Graphics (SG) in the commercial area, while HA-OSCAR represents an open-source equivalent.

3.2.6 Limitations, Challenges and Opportunities

We have identified several limitations, challenges and opportunities as part of constructing the HAC taxonomy from Section 3.1 and conducting the survey from Section 3.2. The limitations and challenges are from an implementation perspective and an operations viewpoint, while opportunities can improve the overall HAC solutions. Using the identified limitations, challenges and opportunities, we discuss future research directions.

3.2.7 Limitations

The HAC limitations presented in this section apply to a majority of the HAC solutions that we have studied, with limitations L1, L5, and L7 common for all solutions.

- L1. **Standardisation.** Standardisation of the HA domain, its components, and related processes is missing. For example, the terminology used by HAC solutions differs considerably. Standardisation could improve research approaches and could enable better discussions and research quality. Furthermore, the lack of standardisation makes it challenging to develop standard APIs that can function with multiple solutions to support specific functionalities, for instance, application-specific agent development. We have addressed this lack of standard terminology using consistent terminology while constructing the taxonomy and performing the subsequent survey.
- L2. **Virtual environments.** The separation between host and guest in virtualised environments complicates some of the functionalities of HACs, such as coordinated monitoring of two operating environments, guest and host, which must be correlated when hosting a critical application. If such a setup is not in place, a guest HAC may not be aware of the host at all. If there are problems in the host which impact all the guests hosted there, the guest HAC may not be able to recognise the problems [154], which could potentially impact the application. Likewise, if the guest application experiences problems, the host may not react since it is unaware of any issues except when hardware resource utilisation significantly increases. Kanso et al. [124] highlighted the problem with a guest HAC that is not aware of the host environment. Some HAC solutions promote a solution by running additional components on the host that also interact with the guest HAC. However, there is no uniformity for deploying such components

because they may differ based on the virtual environment, such as VMWare or kernel-based VM (KVM). In KVM, additional tools are typically required on the host, while VMWare comes with a set of accessories that can be used instead so that no additional means are required. Some HAC solutions, such as ApplicationHA on KVM, employ a separate HAC installation on a host machine. For example, ApplicationHA on the guest can interact with the host HAC. This setup can support monitoring of the host and enable the use of features that are not otherwise available in the guest environment due to restrictions. However, a heterogeneous virtual environment with different operating environments for hosts and guests may also complicate the cross-deployment of a HAC as each operating system, platform, and virtual environment comes with restrictions.

- L3. **Cloud environment limitations.** Both private and public clouds come with limitations. In such a cloud environment, particularly in an IaaS model, customers have access to a guest environment (e.g., VMs). To support an EA, a HAC will require access to some host elements well. In addition, the host environment must be monitored as well as part of a holistic HAC approach, which may mean deploying additional tools, as described in **L2**, on the host. The limitations of a cloud environment may require changes in the architecture of the HAC, hence also the protected application [179].
- L4. **Public cloud limitations.** In addition to what is described in **L2** and **L3**, the public cloud has some additional infrastructure-related restrictions, which are usually different from those of a private cloud. For example, shared storage is not typically supported [6]. Hence any shared-storage-based HAC must find an alternative solution that implies that shared-disk-based quorums cannot be employed. Further, there could be additional restrictions impacting the core functionality of a HAC, such as multicast or broadcast communication not being allowed [6], which would impact the HAC's ability to communicate. Again, this means alternative solutions must be identified and implemented by adding new tools and procedures, which may, in turn, add more complexity to a solution. Moreover, if an application is deployed in a virtual environment, additional restrictions, described in **L2**, apply. For instance, the deployment of additional HAC tools on a host, as explained in **L2**, is usually not possible as hosts are managed entirely by cloud providers in such settings (**L3**).
- L5. **Rating of errors.** Often a severity rating is not used for errors on a resource-level, which means that all errors are treated equally. Adding severity levels would help distinguish between the different types of errors and by the different modules of HACs (e.g., monitoring, failure management) so that actions can be taken accordingly. In addition, multilevel severity would help to improve the recovery process so that, in some cases, errors can be disregarded, indicating that such errors do not result in a complete failover.
- L6. **Standardisation of error, failure, and event message representation.** The current approach is very much individualised to different HAC solutions, implying no standard structure for log messages. This makes it hard to develop a general solution to analyse log messages (e.g.,

for analytical purposes). Furthermore, several modules of a HAC (e.g., monitoring, failure management) may write the same error messages with the same timestamp when a resource fails, making it challenging to mine the log for distinct error messages. Moreover, log sources can also vary as some HAC solutions may employ more than one log source. The difference presents a challenge in mining data from log sources, as it will require one or more data extraction interfaces for each HAC solution.

- L7. **Rating of resource and resource group dependencies.** Resource and resource group dependencies are not always rated, which means that the same failover and recovery policies are applied to all dependencies, regardless of the strength of the dependencies. The dependency rating describes how the failure of a resource can impact another resource through a dependency connection and on what level, which can ultimately influence the mitigation action.
- L8. **Application monitoring.** Even though many of the current HAC solutions employ application monitoring, application-specific errors (e.g., hang situations), are not usually captured. Furthermore, application-related errors are often difficult to monitor with a HAC. This may require additional modules and steps, such as logging in to an application, to detect such failure. Hence, the current situation is that an application may be completely unresponsive, yet it is still regarded as running by a HAC. Therefore, such errors do not trigger any action until the problem is reported by the users of the application. Consequently, this will also result in incorrect values for MTTR and MTBF since no accurate time of failure is available, thus providing unreliable figures for availability.

3.2.8 Challenges

Challenges are associated with functionalities or features that can be implemented to improve the effectiveness of HACs, but that are difficult to realise due to limitations and other constraints.

- C1. **Roles and responsibilities.** HACs work closely with operating environments and infrastructure components to provide the required HAC functionalities, such as heartbeat, monitoring, fencing, and quorum. While the roles and responsibilities of the experts in charge of setting up a HAC change with the different cloud service models [273], it is unlikely that one team can manage a complete HAC implementation and operations. Instead, multiple teams and even organisations must work together to support HAC implementation and operations. Moreover, a heterogeneous virtual environment further complicates the setup because at least two operating systems will be associated with host and guest, which means different teams are usually designated to support the host and guest environments. This means that there must be a support process that links all the different teams together according to a well-defined roles and responsibilities matrix. Moreover, the related support processes, for instance, change and incident management, must be designed accordingly.

- C2. Lifecycle management.** The combination of many application agents, HAC components, VMs, operating systems, and platforms complicates the lifecycle management of HACs. While having a standard across architecture components (including agents) can reduce the number of combinations, this is extremely difficult to achieve. In particular, in virtual environments, lifecycle management must take into account other elements, such as host and guest operating environments on various VMs, which adds further complexity, as described in **L2**. The number of combinations may prompt more threads of lifecycle management. For example, when an EA vendor releases an update to the application, a HAC vendor must also make sure to release an update of the HAC or agent to support the changes in the application.
- C3. Client-state synchronisation.** Client-state synchronisation for EAs is a difficulty. However, if achieved by a HAC, it can improve availability significantly because it can transfer user sessions in the event of a failover, which means that no user data is lost. When a failover takes place, all user input that is not saved is lost. When the failover is complete, users can log in again to establish new sessions and start their work from scratch. If an EA supports thousands of users, this means losing countless hours of work. On the other hand, if a client-state synchronisation can be achieved, it will preserve all connections and sessions, saving considerable time. It is also likely that, with faster failovers using client-state synchronisation, users will not even notice that a failover has taken place. Instead, they will be able to continue working as if nothing has happened. However, state synchronisation for an EA is a significant challenge because it requires replicating user connections, user sessions, user context, session context, user work, and global and local variables. While solutions with a limited scope, such as a firewall, widely employ client-state synchronisation, these are difficult to adopt for the much more complex settings of EAs. Since the problem is about preserving user sessions and related data, in many cases, an applications server layer may also need to be synchronised, as they are the front-ends for user communication in a multi-tier system. Furthermore, applications with a sizable workload require substantial time to stop and start application components in a specific order. Some portion of that time is consumed on ending user sessions gracefully during the stop and establishing non-user (e.g., batch) sessions at the start. However, client-state synchronisation may reduce that time significantly since user data would be already synchronised across the HAC members.

3.2.9 Opportunities

We have identified a set of opportunities that can improve HAC solutions considerably, typically by overcoming HAC limitations from Section 3.2.7. For instance, introducing probabilistic and statistical methods as detailed in opportunity **O5** below requires that ratings of errors and dependencies are in place. Hence, the exploitation of opportunity **O5** requires solutions to limitations **L5** and **L7**.

- O1. Architecture components.** HAC solutions employ different architecture components, and therefore having a standard and modular architecture will help standardise these components.

This assumes that such an architecture will consist of standard modules, and that a HAC solution can choose to implement only a subset of modules, but it can always refer such modules to the standard modules. A set of specifications can support defining the roles of the modules and even provide means to develop interfaces (e.g., APIs). Solutions that are developed using the APIs can potentially be used with multiple HAC solutions. Moreover, the approach would aid in simplifying and interpreting architecture components while enabling the development of approaches for new and emerging technologies (e.g., containerisation), standard testing, and benchmarking.

- O2. **Evaluation of historical data.** HACs produce a large volume of data, and such data can be invaluable when analysing past events and mitigations. These data are generated mainly through logging of events, failures, recoveries, and failovers. Historical data, together with current data, can be analysed to identify patterns, enabling proactive approaches to ensuring high availability. Therefore, evaluation of past data and current data can be used to predict failures of a repeating nature and other related failures.
- O3. **Reliable cluster communication protocols.** The reliability of cluster communication can be increased significantly by employing protocols with atomic features such as TOTEM [51]. These features are only supported by a few HAC solutions today. Employing a standard protocol will also enhance development in the areas, as more people can be involved in the development, which means that issues can be addressed quickly.
- O4. **Monitoring.** Most HAC solutions use a poll-based monitoring method, which is linked to performance problems [284]. If the polling frequency increases, it will improve the monitoring data quality because more up-to-date data will then be available. However, there is an additional overhead associated with frequent polling of many resources, which could be resource-demanding. Furthermore, detecting application-specific errors might also present some challenges as described in **L8**. The monitoring functionality of a HAC may not detect such cases since HACs often focus on monitoring state changes of a resource or a resource group. Therefore, relevant monitoring models should be evaluated to improve the data quality while reducing the performance overhead. The current monitoring type is mostly state-based. However, a different option might be to use a standard API to interact with the operating environments so that the enhanced monitoring features of the operating systems can be utilised. Though this approach may still require an application-specific development, it can be simplified by using standard APIs, as discussed in **L1** and **O1**.
- O5. **Incorporation of probabilistic and statistical methods.** Such methods are not employed currently, but they can improve effectiveness significantly and reduce downtime by analysing data, checking behaviours, and providing predictions. In addition, such methods will also improve the quality of the service for HACs and their components, in general, and promote a more robust proactive approach than currently employed by mostly reactive mechanisms. One

example of such an improvement is introducing statistical analysis to enable the management of quorum services more intelligently.

- O6. **Analytical services.** Analytical services will help identify patterns in the behaviour of HACs and their components while also providing a consolidated view of total downtime and causes. Analytical services can also incorporate data from multiple sources so that data can be combined to provide reliable analysis and even produce predictions on potential failures. An example is that if some HAC components manifest intermittent failures before complete failure, patterns can be analysed to estimate the subsequent failure or an eventual complete failure.
- O7. **Benchmark.** A standard benchmarking approach that can measure availability at a granular level will improve the performance measurements of HACs, while also enabling more a natural comparison between different solutions.
- O8. **Security.** *HAC security* is a rarely concern. However, unauthorised access to the services of a HAC means effectively that the protected application is also jeopardised because a HAC has typically complete control of the operations of the protected application. Security is of particular concern in cloud environments with shared responsibilities (**C1**), since multiple teams assume responsibility for the different layers, which may present new vulnerabilities without a proper security model in place. Moreover, operating a HAC solution in a public cloud may also introduce new vulnerabilities [42], mainly when new and alternative solutions must be introduced due to restrictions, as described in **L4**.
- O9. **Complete HA.** The notion of *complete HA* implies that HA for all SPOF components across all layers are identified and addressed. We have presented nine layers and discussed how HA could be achieved for each layer (Section 2.2). However, further research is required to streamline the various layers and provide a robust strategy to approach HA holistically. This is different from the current view, which focuses more on a system level. An example of a research purpose can be to develop a process to ensure HA across all layers.

3.3 Summary

This chapter has presented a novel taxonomy of high-availability clusters which classifies the key aspects of HACs such as deployment patterns, application areas and topology. The taxonomy is organised in eight top-level classes and many subclasses covering the vast domain of HACs. Subsequently, we applied the taxonomy to survey end-to-end HAC solutions that can support large-scale applications and EAs. Further, we analysed the outcome of the survey, and we presented limitations, challenges and opportunities of HACs.

Chapter 4

Bayesian Networks

This chapter presents an overview of Bayesian networks (BNs) and Bayesian decision network (BDNs), and the focus of the chapter is only on those areas relevant to this thesis. Sections 4.1, 4.2 and 4.3 describe BNs and their key characteristics. Section 4.4 describes BDNs and Section 4.6 summarises this chapter.

4.1 Bayes Theorem

Bayes theorem states that the conditional probability of events can be computed using prior and posterior distributions of the events. Mathematically, the Bayes theorem can be described in terms of probability given that X and Y are random variables [134]

$$P(X|Y) = \frac{P(X, Y)}{P(Y)} \quad (4.1)$$

and

$$P(Y|X) = \frac{P(X, Y)}{P(X)} \quad (4.2)$$

where $P(X)$ represents the prior distribution of X , $P(Y|X)$ is the likelihood that describes the probability of Y given X , $P(Y)$ is the marginal likelihood of Y or the normalising factor [134], and $P(X|Y)$ describes the posterior distribution of X given Y . Bayes theorem is then expressed as

$$P(X|Y) = P(X) \frac{P(Y|X)}{P(Y)}. \quad (4.3)$$

4.2 Bayesian Statistics

Bayesian statistics, which is based on Bayes theorem, differs from classical statistics (frequentist) in the way that it considers prior information. Moreover, the parameters of a model are also considered random, while data are considered fixed. This is opposite to how classical statistics deal with

parameters and data, which is where data are considered random while parameters are considered fixed [136].

4.3 Bayesian Networks

Pearl introduced BNs and related mathematical theory [188]. Since then, extensive research has been ongoing in this area. BNs play a crucial role in artificial intelligence (AI) and are typically used to represent interconnected data and perform inference under uncertainty. BNs are essentially a graph-based solution, and are frequently referred to as Bayesian belief networks, Bayes nets, belief networks, causal probabilistic network and probabilistic networks. BNs can be either continuous or discrete. The focus of the research presented in this thesis is on discrete models; hence, further descriptions are limited to discrete BNs.

BNs consists of two main parts, a qualitative part and a quantitative part. The qualitative part is a directed acyclic graph (DAG). The nodes of this DAG represent random variables, while the edges between nodes describe causal or statistical dependencies between the variables [261]. In contrast, the quantitative part deals with probability distributions representing the strength of dependency relationships stored in each node [261]. Hence, the two parts are combined to form BNs.

At the edges, the direction in the form of an arrow indicates the dependency or relation between nodes, and as such, the node to which an arrow points is referred to as the ‘child’ node, while the node on the other end of the arrow is called the ‘parent’. This relationship is described as a conditional probability, which implies that the probability of a node is computed while considering one or more of the other nodes [261]. Given a child node ‘X’ and a parent node ‘Y’, their relationship is described by the conditional probability

$$P(X|Y). \quad (4.4)$$

If the two nodes are independent, i.e., if there is no influence between them, this is denoted as

$$P(X, Y) = P(X)P(Y). \quad (4.5)$$

This can be written formally as

$$X \perp Y, \quad (4.6)$$

which effectively means

$$P(X|Y) = P(X). \quad (4.7)$$

Two nodes are *conditionally independent* if a third node Z exists so that X is independent given Z , and this is denoted as

$$P(X|Y, Z) = P(X|Z). \quad (4.8)$$

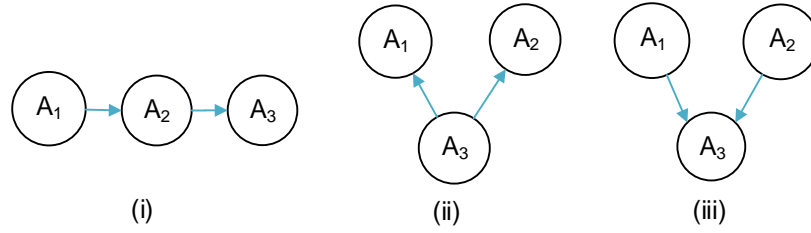


Figure 4.1 The causal connections: (i) Serial, (ii) Diverging, and (iii) Converging connections.

This can be written formally as

$$X \perp Y|Z. \quad (4.9)$$

As shown in Figure 4.1, there are three types of causal connections of the edges when constructing BNs: serial (sequential), diverging and converging.

In a serial connection (also called a chain), a message can flow from A₁ to A₃ unless A₂ is instantiated. If A₂ is instantiated, A₁ and A₂ become independent, which means A₂ blocks the flow between A₁ and A₃. This can be expressed as

$$P(A_1, A_2, A_3) = P(A_3|A_2)P(A_2|A_1)P(A_1). \quad (4.10)$$

This type of serial connection usually refers to a forward connection, which means there is also a backward serial connection where a message can flow from A₃ to A₁ unless A₂ is instantiated.

In a diverging connection (also called a fork), as shown in Figure 4.1 (ii), a connection is established between A₁ and A₂ through A₃. This means a message can flow between A₁ and A₂ unless A₃ is instantiated. If A₃ is instantiated, the message propagation is blocked, which implies that A₁ and A₂ become independent. This is written as

$$P(A_1, A_2, A_3) = P(A_2|A_3)P(A_1|A_3)P(A_3). \quad (4.11)$$

The converging connection (common effect or Collider), shown in Figure 4.1 (iii), allows the flow between A₁ and A₂ only if A₃ is initiated, but is blocked if A₃ is not instantiated, and this can be expressed as

$$P(A_1, A_2, A_3) = P(A_3|A_1, A_2)P(A_1)P(A_2). \quad (4.12)$$

A special case of the converging connection is explaining away. A₁ and A₂ (cause variable) influence A₃ (effect variable). If either A₁ or A₂ is conditioned on A₃, it reduces the probability of the other cause. If A₁ is conditioned on A₃, it reduces the posterior probability of A₂, which means A₁ has explained away A₂.

4.3.1 Representation of Probabilities

A probability distribution is associated with each node. This distribution is updated based on the probability distribution of other nodes, typically that of the parent nodes. The different combinations of the values of the node and its parents are stored in a conditional probability table (CPT) [134] and are presented as conditional probability distributions. The sum of all values in a column of a CPT must be precisely 1.0, which is the standard in computing probability distributions. The activity of ensuring that the sum of all values in a column equals 1 is called normalisation. The activity of complementation ensures that all cells in a column are automatically populated based on the existing values. For instance, when a column has three cells, and if two are populated, complementation is used to automatically populate the third cell and by using the probability distribution from the other two cells. If a node does not have any parents, the CPT contains only local distributions pertaining probability distributions for the available states in the node. If a node has parents, the CPT contains probability distributions that are conditioned on parents.

Each node represents a local probability distribution, and a joint probability distribution (JPD) is calculated from all the nodes that are connected through conditional dependencies. The chain rule is used to compute the JPD, and it is expressed as follows for the entire variable domain $P(U)$

$$\begin{aligned}
 P(U) &= P(X_1, \dots, X_n) \\
 &= P(X_1 | X_2, \dots, X_n) P(X_2 | X_3, \dots, X_n) \dots P(X_n) \\
 &= \prod_{i=1}^n P(X_i | X_1, \dots, X_{i-1}) \\
 &= \prod_{i=1}^n P(X_i | Pa(X_i))
 \end{aligned} \tag{4.13}$$

where $U = \{X_1, \dots, X_n\}$ is a set of n random variables, $P(U) = P(X_1, \dots, X_n)$ is the JPD of those random variables, $P(X_1 | X_2, \dots, X_n)$ is the probability of X_1 conditioned on X_2, \dots, X_n , and $Pa(X_i)$ denotes the parent node of X_i .

As an example of a simple CPT, Table 4.1 shows the CPT for the converging connection in Figure 4.1 (iii). This CPT describes the conditional probabilities of the child node A_3 , which is conditioned on the parent nodes.

4.3.2 Types of Bayesian Networks

There are different types of BNs and the most widely used ones are displayed in Figure 4.2, which was constructed from multiple sources [217, 133, 261] as part of the literature review that we conducted for this thesis.

Table 4.1 An example CPT

$P(A_3 A_1, A_2)$	t,t	t,f	f,t	f,f
t	0.8	0.6	0.5	0.2
f	0.2	0.4	0.5	0.8

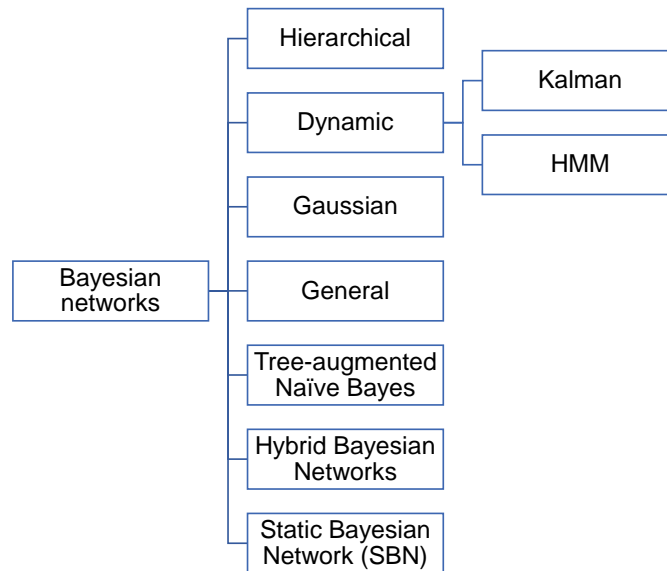


Figure 4.2 Types of Bayesian networks.

The naïve-Bayes BN has a simple network structure that only allows connections from a parent to one or more child nodes [41]. A tree-augmented naïve-Bayes (TAN) allows connections from one node to another, thus forming a tree-like structure. A BN augmented naïve-Bayes (BAN) can have nodes connecting to other nodes that are different from a TAN. The general BN (GBN) is an unrestricted BN type that treats a class node, typically a parent node, as a common node [41]. A hierarchical BN is typically formed in a hierarchical form with latent variables that can have aggregated data.

There are also dynamic variations of BNs, and the primary type are dynamic BNs (DBN). The DBN can be considered to be an expansion of BNs, although it incorporates a time series into BNs. Other similar instances include the temporal BN (TBN) and time series network. The difference between the TBN and DBN is that the structure of the TBN network does not change over time slices, while the structure can change in the DBN [83]. Since the differences are minor, the three terms are often used interchangeably to refer to the DBN. The DBN supports dynamic processes, which is different from BNs. The DBN supports two basic types of dynamic processes: state-based and event-based [260]. In a state-based model, each variable point represents the state of each variable at discrete time intervals. This effectively means that such a network consists of a set of time slices, and each slice represents the value of a variable at time t . On the other hand, the event-based model

represents changes in the state of variables where each variable points to the time of state change [260]. Dependencies between variables are defined for a specific time and are called a base network. The directions of the edges between variables follow the direction of time and are defined as a transition network. A variable in the DBN is also called a temporal variable, as it is related to time [217]. Since the DBN includes time series, it typically supports continuous variables. The Kalman filter and hidden Markov model (HMM) are also considered a variation of the DBN [188].

4.3.3 Learning in Bayesian Networks

There are two main learning approaches related to BNs: 1) structure learning, which is applied to create the structure of the DAG; and 2) parameter learning, which involves learning the parameter distributions of the nodes. Both supervised and unsupervised learning can be used for the two approaches.

4.3.3.1 Structure Learning

Structure learning in BNs implies that a structure can be constructed either manually or automatically. Manual construction assumes that extensive expert knowledge exists and can be used to construct a BN structure. This requires identifying and defining nodes and edges to build a BN structure. The automatic process may employ a constraint-based or a score-based approach, but a hybrid approach combines the constraint-based and score-based approaches. [14, 261, 133, 209].

Constraint-based Approach The constraint-based approach treats the BNs as a representation of dependencies. Thus, this approach focuses on finding a network that can describe the dependencies. However, it is susceptible to errors in single dependencies [134].

Score-based Approach The score-based approach sees learning as a model selection problem. Therefore, it focuses on defining a score function to observe how well a model fits the data. Thus, the objective is to look for a network structure with high scores [14]. A disadvantage of this approach is that a search space can be highly exponential.

4.3.3.2 Parameter Learning

Figure 4.3 summarises parameter learning approaches for discrete and continuous data, and it emphasises the fact that the different algorithms and methods may need to be employed to support the different approaches. For instance, a maximum likelihood algorithm is used when data are discrete, while either a logit distribution or a conditional Gaussian can be used when data are continuous [14].

Learning with Incomplete Data A challenge with BNs is that not all nodes can be supplied with data, which creates a situation in which some nodes will not receive any data [14]. This situation

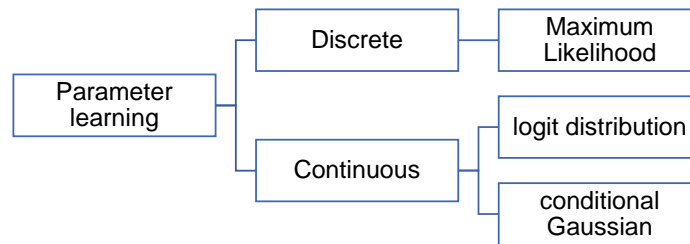


Figure 4.3 Parameter learning approaches.

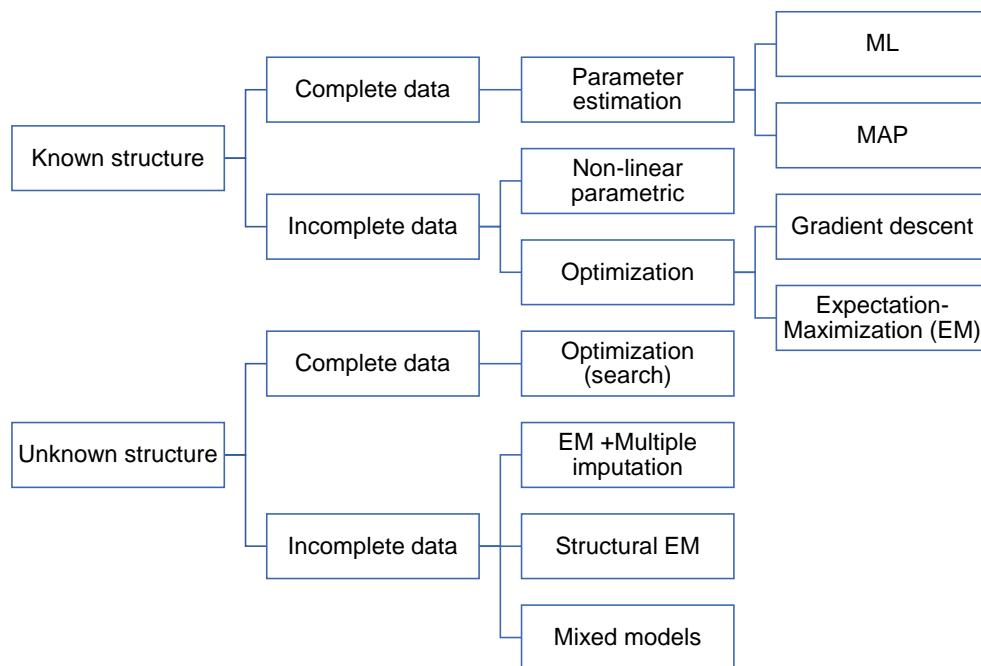


Figure 4.4 Learning scenarios and the corresponding learning methods.

changes which learning method can be employed, and this is presented in Figure 4.4, which was compiled from the literature review for this thesis [14, 261, 133, 209].

When data are complete and the network structure is known, learning can be performed using the Maximum Likelihood Estimation (MLE) or Maximum A posteriori Probability (MAP) [24]. If a structure is unknown, which implies that the structure must be learned as well, and the data are complete, techniques such as optimisation can be employed. Further limitations may apply depending on whether the data are discrete or continuous. When the network structure is known but data are incomplete, an optimisation approach can be used where either gradient descent or expectation-maximisation (EM) can be employed to estimate the parameters. EM consists of two main steps, and they are expectation (E) and maximisation (M). The E step uses the current model to compute the expectation over missing data while the M step maximises the probability of data and updates the model parameters. EM can work effectively even when the sample data is small or when a significant portion of data is incomplete.

As we show later in Chapter 7, the BN model representing a HAC is considered to have *known nodes* because such a model must have a known structure (i.e., the same as the HAC). However, data are considered *incomplete data* because only data for the node(s) corresponding to failed HAC component(s) are delivered to the BN model.

4.3.4 Learning Modes

The learning mode dictates the approach for learning the distributions of parameters in a model. There are three main modes: batch, online and incremental [188, 225]. In batch mode, the required data are assembled and sent in a batch to the model. Hence, batch mode is associated with historical data. Online mode updates the parameter distributions continuously and immediately. Incremental mode is hybrid of batch and online modes where batches of data are sent to the model periodically to update the distributions.

4.3.5 Inference

BNs can employ primarily three different approaches for inference: exact, approximate and hybrid. *Exact inference* implies that the conditional probability distributions are computed precisely [14]. *Approximate inference* performs only approximate inference, which means there is no guarantee that it will result in a correct response [14]. Both approaches can be combined to create a *hybrid* approach. Inference algorithms can further be classified as deterministic and non-deterministic. With deterministic algorithms, the same result is achieved each time, while non-deterministic algorithms may return different values [14].

4.3.6 Motivation for Using Bayesian Networks

HACs consist of interdependent elements (resources and resource groups applications) that present stochastic characteristics (e.g., failures and recoveries). As such, the ability of BNs to model both dependency relationships and stochasticity is particularly suited. Furthermore, we are interested in a model that supports prediction, which is the typical use of BNs. Finally, BNs have been successfully used in modelling and/or prediction in other application domains that present similar characteristics to HACs, including [215, 28].

We use the running example described in Section 2.1 to present a BN model (Figure 4.5). All resources are mapped to nodes in the model, and the dependencies between them are shown as edges. Hence the child node *Service* is conditionally dependent on the parent node *File system 1*. The nodes *IP address*, *File system 1* and *File system 2* have a local distribution representing failure distribution. The child nodes *Service*, *Database group* and *Web application* have conditional probability distributions set in the corresponding CPT. If there is a failure of a parent node (e.g., *File system 1*), a JPD can be computed to present the likelihood of failure in a child node (*database group*) or at a system level (web application).

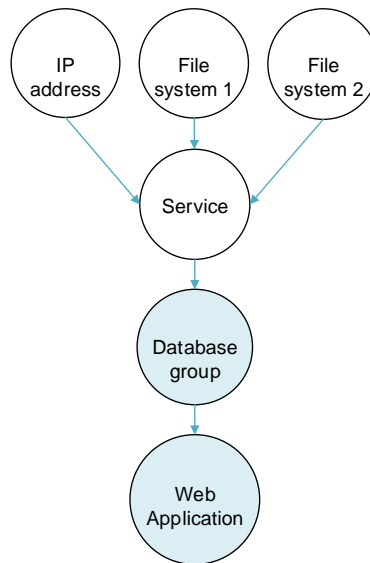


Figure 4.5 Bayesian network model for the HAC from the running example. Latent nodes are highlighted in blue.

There are additional (potential) benefits of using BNs to model HACs, which are as listed follows:

- Visualisation of complex structures in a BN graphical form makes it easier to understand and enables knowledge discovery by graph structure [78].
- Cause and effect relationships between the BN nodes can be clearly presented, and this means even complex relations can also be visualised and studied [78].
- A large number of data interconnected through causal relationships can be represented.
- Unlike “black-box” techniques, such as support vector machines or neural networks, the structure of BNs can be studied, offering insights into the relationships between HAC components.
- Several types of reasoning, such as prediction (causal) and diagnostics, anomaly detection and time series prediction are supported.
- BNs can work well with incomplete data, which is essential for decoding HAC failures since failure typically occurs on a resource-level.
- BNs can represent the complete structure of a HAC by mapping resources to nodes. In particular, a node will represent each low-level resource, and a resource group can be represented by a latent node. Additionally, the edge directions show the cause and effect in relationships, and, in the context of a HAC, this is ideal for describing a high degree of dependencies between HAC resources.

- Finally, conditional probabilities can be used to propagate a failure through the dependency path to a child node where the JPD can be computed to indicate a potential failure of the child node.

4.4 Bayesian Decision Networks

A Bayesian decision network (BDN) (or influence diagram) is an extension to a Bayesian network that combines BNs with decision-theory, enabling decision making under uncertainty [188]. To facilitate this, a BDN introduces two functionalities: utility and decision. A utility functionality (function) uses preferences as a means to define the desirability of a state; hence, the utility can be described as a measure of quantified preference of a state. A decision function describes the decision options while considering the information available at the time. To support these functionalities, BDNs introduce two new node types: utility and decision [131]. This means there can be four types of nodes in a BDN model: chance (or random), decision, deterministic and utility. A chance node represents a random variable, and it is similar to random nodes in a BN model. A decision node models the choices available for a decision, and the expectation is that such a choice is selected by a decision-maker interactively [188]. The selection of a choice, in turn, influences the entire network. When the selection process is automated, it can be referred to as decision automation. A deterministic node is discrete and represents a constant or a value that is calculated. While a random node deals with uncertainty, a deterministic node deals with certainty because the outcomes are known. A utility node is based on the utility theory and represents the utility function. Thus, a utility node presents preferences associated with all the possible outcomes of the parent nodes.

A BDN can have one or more utility nodes; however, there are specific rules that must be followed. For instance, a child utility node cannot have multiple parent utility nodes [17]. These restrictions, as well as further developments in the area, have led to the introduction of new types of utility nodes, which can be grouped into three types: regular utility (the normal utility), Additive-Linear Utility (ALU) and Multi-Attribute Utility (MAU) [17, 72]. A regular utility node is the utility type that is used if not stated explicitly otherwise. An ALU allows connecting multiple parent utility nodes to a single ALU node, thus summing the outcomes linearly into the ALU node. While an ALU is associated with a utility function, an MLU generalises it and allows any function to be defined at the node-level. Similar to BNs, BDNs also use edges to connect the different types of nodes, and there are three types of edges in a BDN: probabilistic, informational and functional. A probabilistic edge goes into a chance node and describes the probabilistic dependencies between such nodes. An informational edge describes the direction and the target, and it points to a decision node. Hence, a node that influences a decision node can be explained by the informational edge [134]. A functional edge describes the connection towards utility nodes hence representing the functional dependencies.

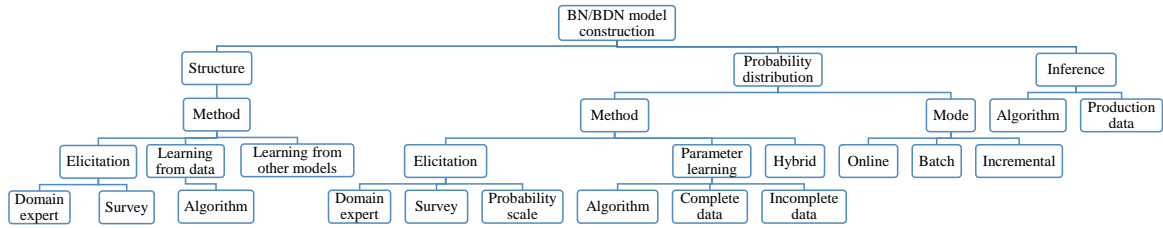


Figure 4.6 Components associated with Bayesian network model construction and inference.

4.4.1 Representation of Conditional Probabilities

The introduction of new nodes in BDNs requires changes in the way that conditional probabilities are handled. Chance nodes in BDNs have the corresponding CPTs that describe probability distributions. A decision node has a decision table that lists all the available decision options, which are called policies where different policies lead to different outcomes. A decision node has a set of policies for each decision, and all policies for a decision node are called a strategy [188]. Similarly, a utility node has a utility table that lists all the outcomes conditioned on its parents, and the values are numerical and expressed as a measure of preference [134].

4.4.2 Inference

The concept of expected utility (EU) is used to calculate the action that can provide the utility with most value in a wide range of decision-making problems [134]. Consider a finite set of actions A and a finite set of possible outcomes O of these actions, and suppose that $U(o_j|a_i)$ represents the utility of achieving outcome $o_j \in O$ having performed action $a_i \in A$. Then, the expected utility of action a_i is given by

$$EU(a_i) = \sum_{o_j \in O} P(o_j|a_i) \times U(o_j|a_i), \quad (4.14)$$

where $P(o_j|a_i)$ is the probability of achieving outcome o_j through executing action a_i .

The action $a^* \in A$ associated with the maximum expected utility (MEU) is then calculated as [131]

$$a^* = \operatorname{argmax}_{a_i \in A} EU(a_i) = \operatorname{argmax}_{a_i \in A} \sum_{o_j \in O} (P(o_j|a_i) \times U(o_j|a_i)). \quad (4.15)$$

4.5 Constructing and Using a Bayesian Network/Decision Network

This section describes the activities and components related to constructing and exploiting a BN or BDN model as summarised in Figure 4.6. The processes involved in the construction and use of a BN or BDN are complex, and heavily influence the effectiveness of the model. These processes typically comprise three main steps. The first step is to construct the model structure (as described in Section 4.3.3.1). The second step addresses the corresponding probability distributions, i.e., the

assignment of prior probabilities to each node in the network and the setting of appropriate conditional probabilities (as described in Section 4.3.3.2). The third step deals with inference such that the model is invoked to produce an output (as described in Section 4.3.5). We summarise these three steps below.

1. **Establishing the network structure.** There are three primary methods associated with constructing the structure: elicitation, learning from data and learning from other models [134]. Elicitation describes using expert knowledge, which can come from domain experts or through a survey. Learning from data implies that a structure is constructed using data as a source employing different algorithms. Additionally, machine learning can be employed to construct the structure of the model using data. Finally, existing models can be used to learn a new model structure by using a mapping procedure.
2. **Establishing the probability distributions.** A range of methods can be used in this step, alone or in conjunction. Similar to constructing the structure, elicitation can be based on a survey or on information provided by domain experts. Additionally, a method of using a probability scale can be employed to identify the probability distributions [131]. However, in such cases, expert knowledge or domain knowledge is required to assign the probabilities correctly. Parameter learning is an automated method; hence the different algorithms that can be employed play an important role in handling incomplete data. Furthermore, the distributions can be updated using an online learning mode; batch or incremental learning modes for this are discussed in Section 4.3.4. Online learning assumes data are delivered continuously, while batch mode implies that the learning can be done by providing batches of data. Finally, the incremental mode is essentially a hybrid of both methods where necessary data are provided incrementally to learn and update the probability distributions.
3. **Inference.** The third step is inference, in which the model computes the posterior probability distribution for a set of nodes when evidence is provided to one or more nodes.

We will detail how these three steps are carried out for each of the Bayesian networks developed later in this thesis.

4.6 Summary

This chapter presented the basic of BNs and BDNs, with a focus on the terminology and concepts used in the rest of this thesis. The probability-weighted utility can be used to capture the essential characteristics of a HAC, which is a prerequisite to predicting the manageability of a failure at a resource-level. As shown later in Chapter 6, a BDN model can therefore predict locally manageable resource failures while evaluating the characteristics of a HAC. A BN model can then be used to represent the dependencies in a HAC using the causal relationships of a BN model. The nodes in the BN model can represent the resources and other components of the HAC. As we will show in

Chapter 7, this means a complete HAC can be modelled using BNs to emulate failure propagation and failure prediction of the HAC.

Part II

A Bayesian Prognostic Framework for High-Availability Clusters

Chapter 5

Holistic Modelling Technique for High Availability

This chapter introduces a holistic modelling technique for high availability (HMTHA), which represents a key part of our Bayesian prognostic framework. The objective of the technique is to generate a formal model of the resources, resource groups and dependencies of an IT application to support the generation of probabilistic models (e.g., Bayesian networks) for the analysis of the application's HA properties. The input to the technique is information about the IT application's structure and components, the IT infrastructure on which the application is deployed and the set of HA requirements for the application, e.g., based on Service Level Agreements (SLAs). The technique produces:

1. A special graph comprising several types of vertices that correspond to different types of application elements and several types of both directed and nondirected arcs correspond to the different types of dependencies between application elements. We call this special graph the *holistic high availability model* (**HHAM**) of the application.
2. A *mapping table* (**M-table**) with information that simplifies the mapping of the HHAM to a probabilistic model such as a BN.
3. A set of *translation rules* (**T-rules**) that aids in the mapping of probabilities between an HHAM and a probabilistic model.

The technique enables the modelling of complex high-availability solutions at a granular level, which enables us to detail the dependencies between the components of the modelled system. The HMTHA is a crucial part of the BP framework, and modules including BN-HAC (Chapter 7) and BPPF (Chapter 8) use the outcomes of the HMTHA. Thus, the technique presented in this chapter answers research question RQ1 from Chapter 1, that being the development of a technique for modelling complex high-availability solutions and presenting dependencies between the HA components.

This chapter is organised as follows. Section 5.1 summarises the existing approaches to modelling the HA of IT applications. Next, Section 5.2 describes the related work. Section 5.3 presents the

proposed technique to modelling HA holistically and describes its outcomes. Section 5.4 presents a simplified process for applying the technique, and an example of this is provided using the running example. Section 5.5 describes the HMTHA software tool, which facilitates the use of the modelling technique. Lastly, Section 5.6 summarises the chapter.

5.1 Introduction

In an IT-driven world, numerous applications are required to be available around the clock. Achieving this requirement is typically facilitated by HA solutions. An example of such a solution is a HAC. However, IT applications and deployment environments are evolving, and the types and number of components that need to be managed by HA solutions are continually increasing. For instance, the wide adoption of cloud computing [64] and the emergence of technologies including fog computing, edge computing and containerisation have added more components to a system. Additionally, new architecture paradigms, such as microservices, have also introduced new types of components [61]. As a result of this diversification, correctly identifying and quantifying the dependencies among IT systems components have become a highly complex undertaking. This makes it challenging to address HA holistically. For instance, an application whose components span both cloud and fog computing infrastructure requires that the related resources in both environments are considered together to achieve HA [27].

Availability is, in some cases, discussed as part of the broader concept of dependability. In such cases, the terms *availability* and *dependability* are used interchangeably because availability is one of the characteristics of dependability [9]. Moreover, HA modelling typically employs the same techniques as availability modelling; therefore, it is often regarded as availability modelling. The current modelling approaches for availability can roughly be classified into two groups (Figure 5.1): state-based and structure-based. State-based approaches, also known as state-space-based, deal with states of components (available, unavailable) and with transitions between these states. Structure-based approaches, also called non-state-based or non-state-space-based, model both the structural relationships and the behaviours of components.

State-based modelling is typically stochastic and is used to examine different state transitions of components and their potential effects. Markov models represent a significant portion of the state-based modelling domain [270]. Markov models are based on stochastic processes, and the components of the models are statistically dependent. The Markov chain is an essential branch of Markov models, and it too can be grouped into two distinct approaches: continuous-time Markov chains (CTMC) and discrete-time Markov chains (DTMC) [270]. The primary difference between CTMC and DTMC is the time aspect of state changes. The DTMC resides precisely in a unit of time before changing the state, whereas CTMC allows a continuous length of time in any state while adhering to the Markov properties. Several extensions are also made to the Markov chains. For instance, when a reward rate is added to the Markov chain, it is known as a Markov reward model [270]. Petri nets are used as another modelling technique. Unlike Markov models, Petri nets are

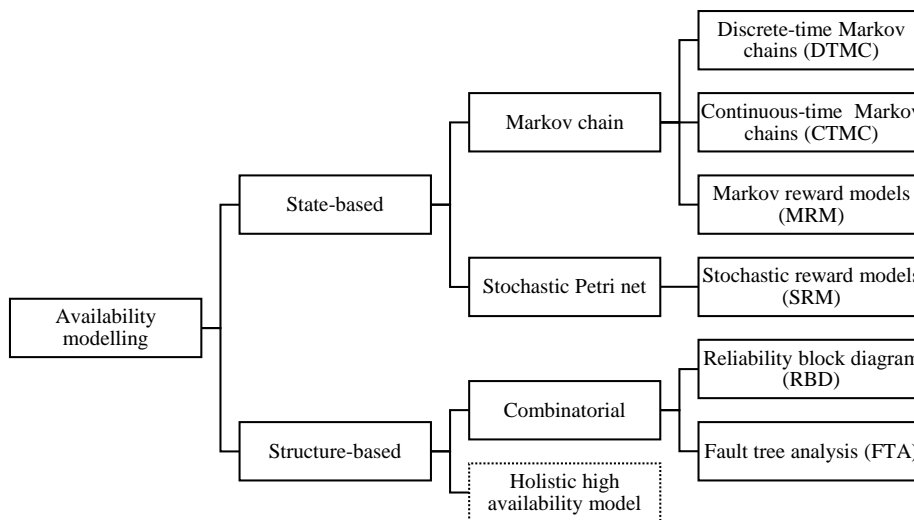


Figure 5.1 Classification of major availability modelling formalisms.

directed bipartite graphs with two disjoint sets of vertices: places and transitions [162]. The stochastic Petri net is a variant of the Petri net that has been extended with stochastic properties and, as such, incorporates a probabilistic approach. These techniques can also be combined to form new ones, and an example is the stochastic reward nets (SRNs), which combines Markov chains with a Petri net [153].

On the other hand, a structure-based approach captures the actual structure of the modelled system, although only a subset of the complete availability structure is typically used to perform a specific analysis. Structure-based approaches are typically presented graphically. Fault tree analysis is an event-driven graphical tree structure that is used to identify the basic events that cause a system failure [11]. The reliability block diagram is another technique that represents components of a system using block structures. Subsequently, the blocks are connected by dependencies in the context of availability [270]. However, structure-based models have a number of drawbacks. In particular, it is challenging to represent complex dependencies for this type of approach [270, 11]. This often leads to limitations in the encoding of complex dependencies.

The holistic high availability modelling technique introduced in this chapter is a structure-based approach (Figure 5.1) that can encode complex dependencies, and can therefore address multiple limitations of existing approaches when modelling availability. As detailed in the following sections, the technique supports the multilevel hierarchical composition of resources and multilevel dependencies between the resources, which are paramount for the development of HA solutions.

5.2 Related work

Modelling the availability and HA of IT applications has been the focus of intense research over the years. However, existing approaches often target narrow aspects of availability, such as observing

the state of a component or a system, which is often facilitated by state-based modelling approaches. Hunter et al. [108] employ an availability model using Markov modelling techniques to examine the potential availability of a two-node HAC. Petri nets and variations of Petri nets are also used frequently. Extended coloured stochastic Petri nets (ECSPN) are used to model production systems and their availability in Industry 4.0 [152]. SRNs are also generally employed to support the construction of availability models. Longo et al. [153] present a scalable method for availability analysis of large-scale infrastructure as a service cloud employing SRNs. Nguyen et al. deploy [186] a comprehensive availability model of a data centre using SRNs for disaster tolerance to facilitate availability assessment. SRNs are also employed by a component-based availability modelling framework used to quantify the availability of cloud services [159]. Another project that analyses the availability of cloud data centres using a monolithic SRNs is presented in [147]. Markov chain techniques are broadly employed to model availability while observing failures, and an example of this is the availability model for Cassandra, which deals with two types of failures, transient and memoryless [212].

It is also common for state-based and structure-based approaches to be combined. For instance, Dantas et al. [52] present a hierarchical heterogeneous modelling approach using reliability block diagrams (RBDs) and Markov reward model (MRM) in a Eucalyptus cloud computing environment. This modelling approach is used to represent a redundant architecture and to capture the availability measures so that results can be compared to a nonredundant architecture. Kim et al. [129] present an availability model for a virtualised system. The model organises system components in a two-level hierarchy where the upper level represents the complete system. In contrast, the lower level is used to capture the behaviour of the individual components. The model employs fault tree analysis for the upper-level and CTMC for the lower-level. Another research initiative combines RBDs and Markov chains to create an availability model for an HA platform using three-level hierarchical decomposition [271].

In summary, numerous methods of modelling availability exist. Nevertheless, an approach for encoding the actual system requirements (both software and hardware) from an HA viewpoint is missing. Furthermore, existing models often focus only on specific cases of live IT application architectures. Hence, to the best of our knowledge, no holistic approach to modelling a system exists to represent a multilayered IT application with intricate dependency patterns. Another significant difference between the existing modelling approaches and our HMTHA is that our modelling technique can be used during the design phase. In contrast, most other modelling approaches rely on existing deployment environments or facilitators.

5.3 Holistic Modelling Technique for High Availability

5.3.1 Terminology

Before presenting our HMTHA technique, we revisit the definitions of several HAC terms:

- *Application*: An application represents a complete IT application, which is further broken down into resources, resource groups and arcs.
- *Resource*: A resource represents an element of an IT application and it is often the smallest entity of an application. A resource can represent a software-based or a hardware-based element. The typical properties of a resource are monitorable, invokable and initiable. Examples of resources include processes (e.g., database), IP addresses, file systems and network interface cards.
- *Resource group*: A collection of related resources that are grouped to provide specific functionality.
- *Arc*: An arc in an IT application represents a dependency relationship between two or more resources or resource groups.
- *Facilitator*: We introduce the term ‘facilitator’ to indicate a solution that enables the delivery of HA services. For instance, a HAC is typically implemented to provide HA for applications. However, many such solutions can be employed to deliver HA across the multiple layers of a complex system.
- *Enabler*: We introduce the term ‘enabler’ to refer to a probabilistic model (e.g., a Bayesian network) that can make use of an HHAM.

5.3.2 HHAM Model Definition

The HHAMs generated by our technique are graph-like structures that have vertices corresponding to resources, resource groups and the application, and arcs corresponding to dependencies between these. To distinguish between different classes of IT application resources and between different types of dependencies, HHAM uses six types of vertices and five types of arcs, respectively. The definition of the HHAMs is, therefore, provided as follows.

Definition 1 (Holistic high availability model). A holistic high availability model of an IT application is a tuple

$$HHAM = (V_R, V_W, V_G, v_A, V_{LD}, V_{GD}, A_R, A_G, A_A, A_{LD}, A_{GD}), \quad (5.1)$$

where:

- V_R is a finite set of *simple resource vertices*, i.e., vertices corresponding to the simple resources of the application (e.g., IP address, file system). Hence, the majority of the resources are identified as simple vertices.
- V_W is a finite set of *weak resource vertices* that are associated with resources whose failure does not lead to the failure of a resource group or an application. Therefore, these resources are not required to ensure high availability but are still required to satisfy business requirements.

Holistic Modelling Technique for High Availability

- V_G is a finite set of *resource group vertices*. For example, all resources required to maintain a database instance are grouped into a database resource group.
- V_A is the *application vertex* and models the top-level application. The modelling technique is typically applied to one application, and therefore a model represents one application. Hence, there is only one such vertex for a model.
- V_{LD} is a finite set of *shared dependency vertices*, i.e., vertices corresponding to resources that are shared by resources such as simple resources, weak resources and resource group. Hence, resources that can affect individual resources or resource groups are created as shared dependency vertices.
- V_{GD} is a finite set of *global dependency vertices*, i.e., vertices corresponding to resources that are essential for the functionality of other types of resources. Failure of such a resource can affect the entire system.
- $A_R \subset (V_R \cup V_W) \times (V_R \cup V_W)$ is a finite set of *resource arcs* that connect resource vertices (simple or weak). Each arc $(v_1, v_2) \in A_R$ models a dependency between a simple or weak resource v_1 and another simple or weak resource $v_2 \neq v_1$.
- $A_G \subset (V_R \cup V_W) \times V_G$ is a finite set of *resource group arcs*, i.e., arcs that model dependencies between (simple or weak) resource vertices and resource group vertices.
- $A_A \subset (V_R \cup V_W \cup V_G) \times V_A$ is a finite set of *application arcs*, i.e., arcs that model dependencies between simple/weak resource vertices or resource group vertices and the application vertex.
- $A_{LD} \subset (V_R \cup V_W \cup V_G) \times V_{LD}$ is a finite set of *shared dependency arcs*, i.e., arcs modelling dependencies between simple/weak resource vertices, resource group vertices and shared dependency vertices.
- $A_{GD} \subset (V_A \times V_{GD})$ is a finite set of *global dependency arcs*, i.e., arcs representing connections between global dependency vertices and the application vertex.

The corresponding metamodel of the HHAM is depicted in Figure 5.2 using a UML class diagram. The different vertex types can also be combined. For example, a global dependency vertex that exhibits weak resource characteristics can be designated as a *global dependency and weak resource vertex*. Furthermore, an application vertex and resource group vertices are logical without direct representation of resources such as IP address, while simple resource and weak resource vertices always represent a corresponding HA resource. To ensure that the models are created accurately, we provide a set of modelling rules in Table 5.1.

Additionally, the modelling technique provides a naming convention and an addressing scheme to uniquely identify a resource, a resource group, a top-level application and the different types of arcs. Moreover, the position of each element can also be identified, which can assist in evaluating a

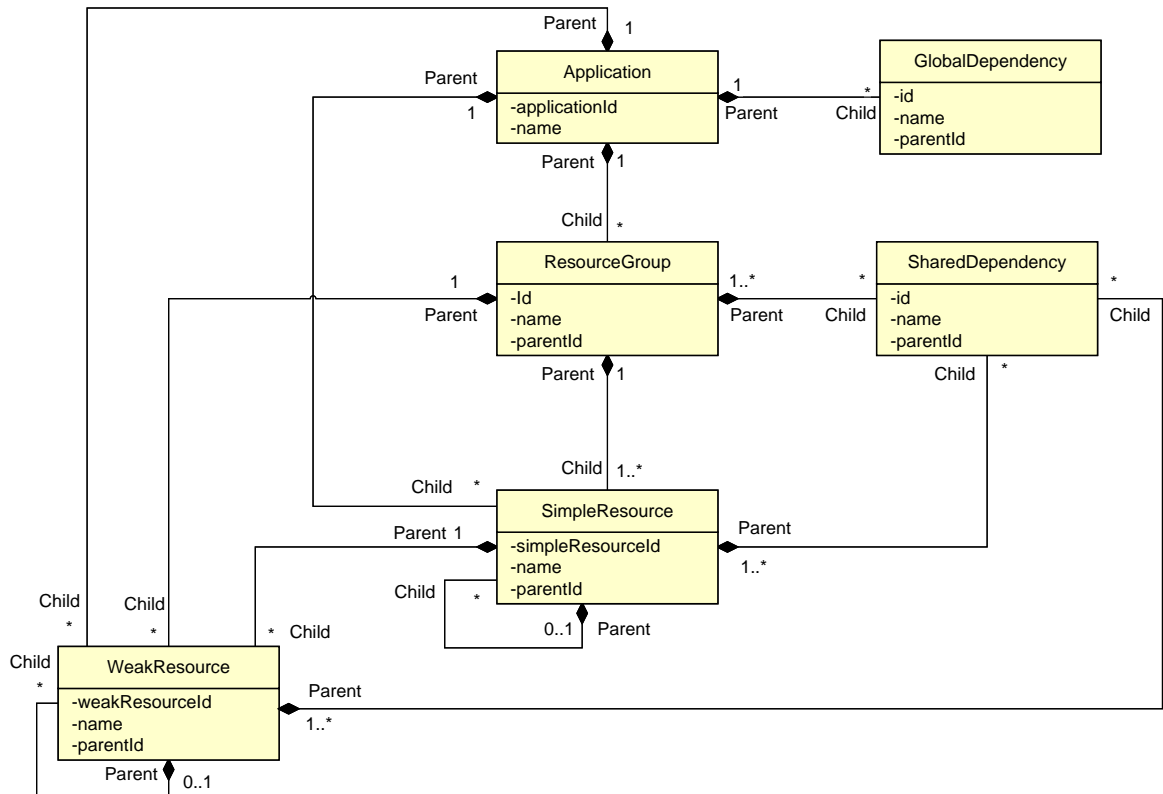


Figure 5.2 Metamodel of the holistic availability modelling technique presented as a UML class diagram.

potential effect. For instance, if the state of the resource changes, the position can be evaluated to assess the effect on the underlying resources in the hierarchical tree. Similarly, the upwards effect can also be evaluated. As dependencies are presented in multiple levels, they present opportunities to perform in-depth analysis on dependencies [33].

5.3.3 Mapping Table (M-table)

A table is constructed immediately after creating the HHAM, and the objective of the table is to assist in mapping the HHAM to a probabilistic model, hence the name mapping table (M-table). The table consists of two parts, and the first part consists of four columns and presents resources, vertex types, arc types and layers of the system being modelled. The second part consists of three columns added when a probabilistic approach is identified. A six-step mapping approach (Table 5.2) is used to map an M-table to a probabilistic model. However, because the focus of this thesis is on BNs, we use BNs as the probabilistic approach. Hence, the M-table and the T-rules are created accordingly.

The first part of the M-table for the running example is provided in Table 5.4. The creation of the second part of the M-table and mapping to a BN structure is presented in Section 7.2.2. Similarly, the

Table 5.1 Modelling rules

No	Modelling Rules
1	A vertex cannot be simple, global dependency or shared dependency simultaneously, and only one such type can be associated with a vertex.
2	A weak resource vertex can be a global dependency vertex or shared dependency vertex to indicate the noncriticality of a resource.
3	One application vertex per model is allowed because a model is created to represent an IT application.
4	Global dependency vertices can only be connected to an application vertex to indicate that a global dependency vertex affects the entire application.
5	Shared dependency vertices can be connected to simple/weak or resource group vertices to indicate the dependency to one or more vertices.
6	If there are n resource group vertices and $n - 1$ resource group vertices are connected to the same shared dependency vertex, the shared dependency vertex can be defined as a global dependency vertex. When a majority of the resource group vertices depend on the same shared dependency vertex, it can affect the entire application; hence, it is defined as a global dependency vertex.
7	An application vertex cannot be a weak resource vertex because if it is defined as a weak, the application is not considered critical; hence, it does not require high availability.
8	If there are n resource group vertices, a maximum of $n - 1$ can be defined as weak because if all are designated as weak, this can indirectly label the application as weak; hence the need to set up high availability is no longer present.
9	Arcs point from child vertices to parent vertices to show the dependency. This means an application resource is always the top-level parent.

construction of the first part of the M-table for the testbed application is presented in Section 10.3.1. The construction of the second part and mapping to a BN structure is described in Section 10.3.3.1.

5.3.4 Translation Rules (T-rules)

Transformation rules (T-rules) enable a transition from an HHAM to an enabler (a probability model). Therefore, the T-rules are specific to the probabilistic approach because different approaches differ in how the models are constructed and probabilities are assigned. A T-rule is defined for each vertex type, providing instructions on calculating and assigning the initial probabilities. We list the T-rules created for BNs as the probabilistic approach and three approaches to calculate the probabilities. The approaches use the BN structure to describe the relationship between a parent and child component because HHAM and BN employ different orientations for the model structure. The three approaches are presented below.

5.3 Holistic Modelling Technique for High Availability

Table 5.2 Six-step approach to mapping an M-table to a BN model

Step	M-table Objects	Description
1	Simple	Map each simple vertex with a physical representation to a corresponding node in the BN model using a reversed layer approach
2	Resource group and application	Map vertex types that do not have a physical representation to latent or hidden nodes
3	Weak	Map a weak vertex type to a node and assign the conditional probability to a child node based on the effect on the child node
4	Dependency	Map the dependency nodes (shared and global) to nodes and assign the conditional probability based on the effect on the child nodes
5	Layers	Map the layers of HHAM to the reversed layers of a Bayesian network model to organise the structure
6	N/A	Assign probabilities to nodes using the T-rules

N/A - Not applicable to a specific M-table object.

Table 5.3 Translation rules (T-rules) for each vertex type

No	Vertex Type	T-rules
1	Application	Assign conditional probabilities using an impact analysis for all resource groups (Approach 3). If global dependency vertices exist, perform an impact analysis using Approach 2 to determine and assign probability distributions
2	Resource group	Assign probabilities conditioned on parent vertices, including weak vertices, using an impact analysis (Approach 2). If one or more shared dependency vertices are connected, estimate the probability distribution using an impact analysis (Approach 3)
3	Simple	Assign probabilities according to the SLA (Approach 1). If the simple vertex is a child to several parent vertices, use an impact analysis to assign probabilities (Approach 2). If the simple resource is connected to one or more shared dependency vertices, perform an impact analysis using Approach 3
4	Weak	Assign probabilities using the SLA (Approach 1)
5	Shared dependency	Use SLA to assign probabilities (Approach 1)
6	Global dependency	Assign probabilities using the SLA (Approach 1)

Approach 1: The SLA defined for the availability of the application under analysis is used because availability requirements defined by SLAs guide the implementation of the HAC solution that ensures

Holistic Modelling Technique for High Availability

HA for the application [245, 45]. Therefore, the SLA value is translated directly into probabilities as follows

$$C(\text{no failure}) = \frac{SLA}{100}, \quad (5.2)$$

$$C(\text{failure}) = 1 - \frac{SLA}{100}. \quad (5.3)$$

Assuming that the SLA is 99%, this yields:

Failure = .01

No failure = .99

Approach 2: The failure of a parent component and its effect on the child node for each potential failure mode is calculated using the failure mode and effects analysis (FMEA), which is a systematic analysis technique to identify potential failures and their effects [213, 149]. FMEA is well suited for a system comprising multiple interconnected components, and uses three factors to calculate the risk priority number (RPN): the severity of the consequences (severity - S), the probability of failure (occurrence - O), and the probability of undetected failure (detectability - D). The RPN is given by the product of these factors:

$$RPN = S \times O \times D. \quad (5.4)$$

Throughout this thesis, we used a scale of 1 to 10 for each of these factors, where 10 is associated with a high impact (severity - most likely, likelihood - most likely, and detectability - least detectable). Approach 2 calculates the effect of the failure of the individual parent components on a common child component for each potential failure mode as

$$C_i = \frac{RPN_i}{RPN_{max}}, \quad (5.5)$$

where RPN_{max} is the maximum obtainable value for RPN, and C_i is the component under analysis.

Approach 3: The effect of one or more component (parent) failures on a common component (child) for potential failure mode is calculated using FMEA. Hence, the relative effect of all parent components connected to a common child component is calculated as

$$C_i = \frac{RPN_i}{\sum_{j=1}^n RPN_j}, \quad (5.6)$$

where RPN_j denotes the RPN for the j -th parent component, and n indicates the total number of components connected to the child (common or target) component C_i .

If C is an application, the result is adjusted to reflect the fact that an application fails when at least $n - 1$ of its n resource groups fail:

$$A = \begin{cases} \text{failure,} & \text{if } RG \geq n - 1 \\ \text{no failure,} & \text{otherwise,} \end{cases} \quad (5.7)$$

5.4 Building the HHAM model and M-table of an IT application

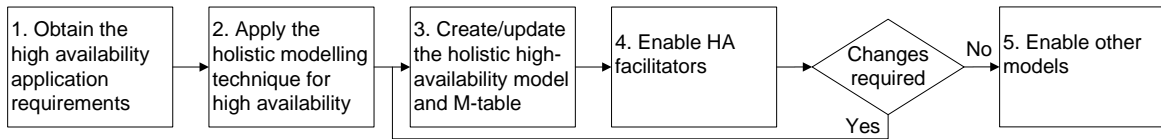


Figure 5.3 Simplified process for creating a holistic high availability model for high availability.

where RG denotes the number of resource group failures. The calculation and assignment of probabilities are provided in Section 7.2.6 for the running example and Section 10.3.3.1 for the testbed application.

5.4 Building the HHAM model and M-table of an IT application

Since the modelling technique is dependent on other activities such as gathering requirements of an IT application, we present a simplified model creation process that consists of five steps, as illustrated in Figure 5.3. Moreover, now that we have described the process for building an HHAM model and other outcomes, we use the running example application (presented first in Section 2.1) to illustrate its steps.

1. Obtain the high availability application requirements

Three different categories are considered when combining the requirements that function as inputs to the model, and they are:

1. Requirements based on availability (i.e., SLA)
2. Application architecture (i.e., SPOF and non-SPOF components)
3. Deployment environment (i.e., network, storage solution)

The first category represents the established SLA for an application, and the assumption is that the HHAM model's depth and complexity increase when the availability SLA increases. For example, an availability SLA of 99.9% expects an annual downtime of 8 hours and 46 minutes. This means the HA solution must be designed and implemented to reduce the MTTR to achieve that goal. The second category focuses on the application's requirements. Hence, all SPOF components (e.g., service, file system) required for the operations of the application are considered first. Next, all the non-SPOF components required to support the business functionalities are added. For example, a different file system to store temporary files even though the file system is not considered a SPOF because the application can continue to function even without this file system for a few hours. Nevertheless, the component is added to the HA facilitator to satisfy business requirements. The existing deployment environment and its capabilities, such as an SAN solution, are considered in the third category.

Example 1. Identification of the availability requirements of an application is the first step in the five-step modelling process. We assume categories 1 and 2 as described above for the running example. This means assuming an availability SLA of 99.9% while also considering application-specific requirements.

2. Apply the holistic modelling technique for high availability

The model creation process is iterative, and uses dependency patterns to identify and add the relevant components. The application is the top-level vertex (parent to all the other vertices) added first, and there is only one such vertex for each model. The layer two vertices are added next using the dependency relationships that they have with the parent vertex. These are typically the resource group vertices and the arcs used to link are application arcs. The subsequent vertices are added for each resource group vertex, and this iteration continues until all SPOF components and business-required components are identified and added. A relevant arc is added for each dependency to form a child-parent relationship. For example, if a simple vertex points to its parent resource group vertex, the corresponding arc type is a resource group arc. Finally, the dependency vertices are added. The shared dependency vertices are added first and then global dependency vertices are added. The shared dependency vertices point to either a resource group vertex or an individual simple/weak resource vertex. The global dependency vertex always points to the application vertex.

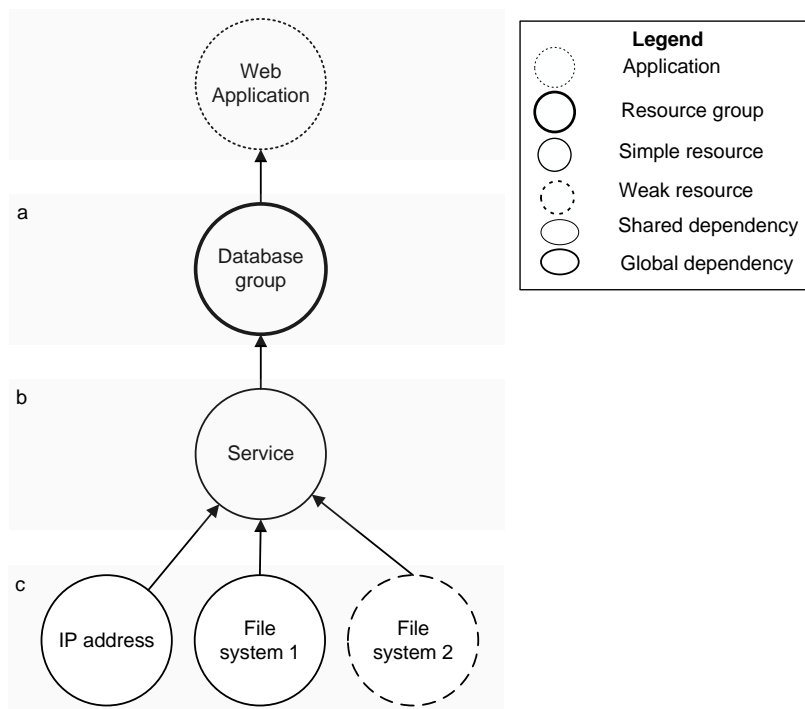


Figure 5.4 High availability model for an example IT application using a graphical notation to represent the different types of vertices and arcs from Definition 1. The different graphical elements are shown in the legend.

5.4 Building the HHAM model and M-table of an IT application

To depict an HHAM model graphically, we use a different graphical notation for each type of HHAM vertex. Figure 5.4 presents an HHAM model that shows this notation. As illustrated in this figure, the HHAM vertices are organised hierarchically into layers labelled as a, b, \dots . The components of the *HHAM* from eq. (5.1) are as follows:

- The set of simple resource vertices is given by

$$V_R = \{\text{Service}, \text{FileSys1}, \text{IPAddr}\},$$

where *Service* is the service of the Web application, *FileSys1* is a file system for the *Service*, and *IPAddr* is the related IP address.

- The weak resource vertex is given by

$$V_W = \{\text{FileSys2}\},$$

where *FileSys2* is a temporary file system.

- The resource group vertex is given by

$$V_G = \{\text{DatabaseGr}\},$$

where *DatabaseGr* represents a resource group.

- The application resource vertex is given by

$$V_A = \{\text{WebApp}\},$$

where *WebApp* represents the top-level application.

Example 2. A set of instructions used to create the example in Figure 5.4 are listed as follows:

1. Add the first vertex that represents the main application (*Web Application*).
2. Add the second vertex, a resource group, to represent the database group.
3. Add the simple vertex representing the resource *Service 1* and link it to the vertex *database group*.
4. Add the three vertices that represent *IP address*, *File system 1* and *File system 2*. All are connected to the parent vertex *Service1* using resource links. *File system 2* is not a SPOF component but is required in the HA solution to satisfy business requirements for storing temporary files and the label *weak* indicates this. This means that the application can survive without the weak resource for a few hours.

Table 5.4 M-table of the running example

Resource	Vertex Type	Arc Type	HHAM Layer
Application	Application	Application	
Database	Resource group	Resource group	a
Service1	Simple	Resource	b
IP address	Simple	Resource	c
File system 1	Simple	Resource	c
File system 2	Weak	Resource	c

3. Create/update the holistic high-availability model (HHAM) and M-table

While the previous step describes the input aspects, this step describes the outcomes of the modelling technique. The primary outcome is the finalised HHAM (the model) presented in Figure 5.4. Subsequently, a corresponding M-table is created using the following steps:

1. Create a table with four columns: *resource*, *vertex type*, *arc type* and *HHAM layer*. The *resource* is the name of the resource, and *vertex type* represents the vertex type that the resource belongs to, while *arc type* indicates the type of the arc used to connect to this resource. The *HHAM layer* presents the layer that the resource is located. Table 5.4 presents the M-table for the running example model shown in Figure 5.4.
2. Populate the M-table with the resources, related vertex type, arc type and the layer.
3. In the last step, as part of constructing a BN model, three more columns are added to represent the BN model components *BN node identifier*, *node type* and the corresponding *layer*. A six-step mapping approach (presented in Table 5.2) is used to complete the table. For example, the vertex types application and resource groups are mapped to latent nodes in a BN model. Similarly, the layers are also mapped, though they are reversed because the HHAM has the parent vertex at the top level while a BN model has a child node at the top level.

The accompanying **T-rules** are presented in Table 5.3, which aid in setting probability distributions based on the vertex type. For example, they recommend setting a reduced conditional probability for a weak node in the child node of a BN model to indicate that the failure of such a node will not have an impact on the child node.

Example 3. The HHAM in Figure 5.4 shows the essential components that require protection when HA is considered. The model consists of four layers and there are six vertices in the model. A dotted circle represents a weak vertex (*File system 2*) and a bold circle indicates the associated resource group vertex. The top-level vertex represents the web application. Two vertices (application and resource group) are identified as logical without any physical representation. The corresponding M-table is constructed as presented in Table 5.4.

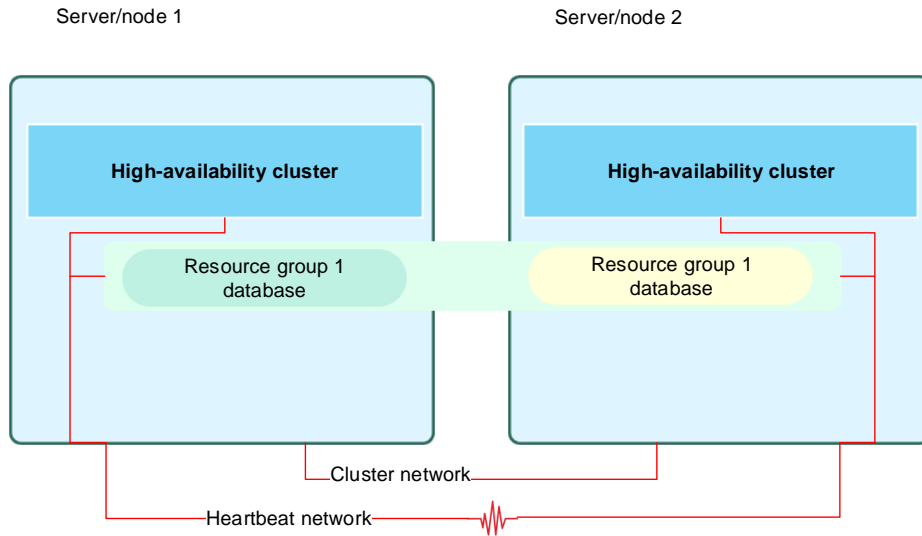


Figure 5.5 High availability cluster (HAC) of the running example.

Table 5.5 HAC configuration for the running example

Vertex ID/resource ID	Resource	HAC Name
C1	Application	WEB
C1G1A1	database	grp_WEB_database
C1G1A1B1	Service1	rsc_WEB_database
C1G1A1B1C1	IP address	vip_WEB_database
C1G1A1B1C2	File system 1	fs_WEB_database
C1G1A1B1C3	File system 2	fs_2_WEB_database

4. Enable HA Facilitators (e.g., HAC)

In this step, the relevant HA facilitator is added. Suppose the facilitator requires additional resources to operate. The model needs to be updated with that information. This is indicated by an arrow between the output of step 4 leading to step 3.

Example 4. We present a facilitator (HAC) in Figure 5.5. The HAC is set up in a two-node cluster, and it consists of one resource group and four resources. Table 5.5 lists all the related resources, the vertex ID/resource ID and the HAC names.

5. Enable other models (e.g., Bayesian networks)

The model and the accompanying M-table and T-rules are used to construct a BN model for the HAC of the running example. The detailed steps are provided in Chapter 7.

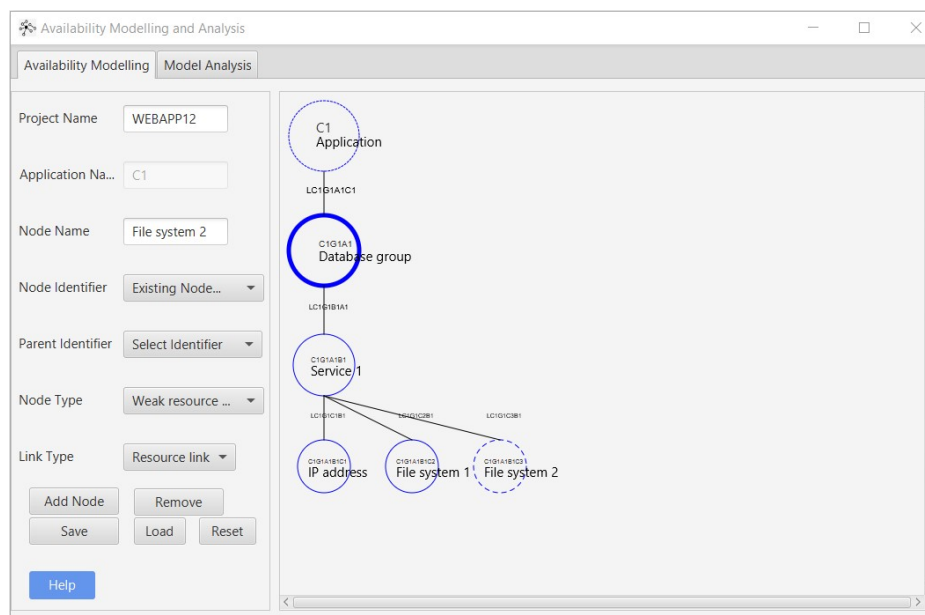


Figure 5.6 The HMTHA tool showing the high availability model (HHAM) for the running example using the graphical notation for the different types of vertices and arcs from Definition 1.

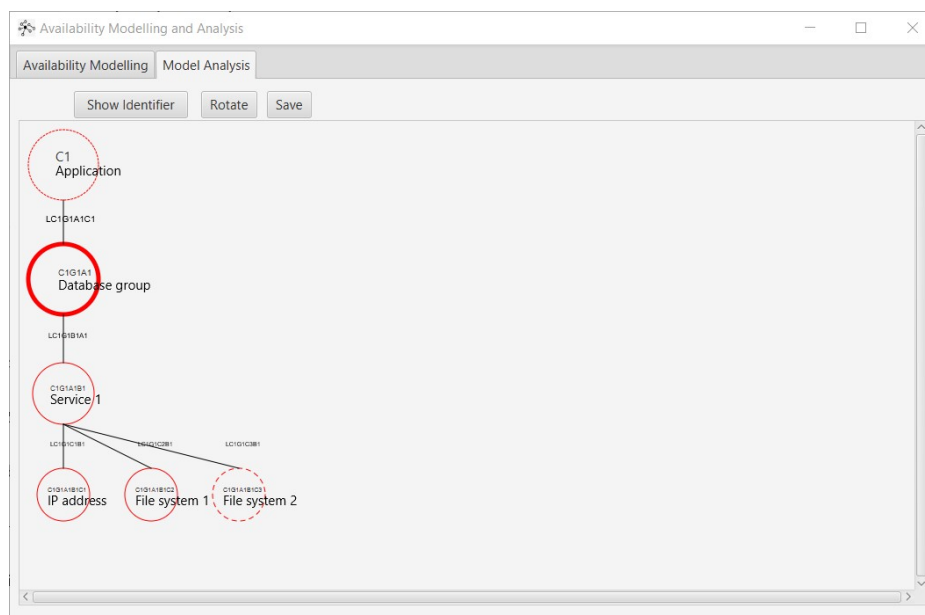


Figure 5.7 HMTHA tool showing the second view to analyse the model.

5.5 Tool Support for the Holistic Modelling Technique for High Availability

The HMTHA tool is a software solution to facilitate HMTHA¹. The tool has two views presented as tabs, and Figure 5.6 and Figure 5.7 show screenshots of the two tabs. The model visualised in these

¹The corresponding software is available on GitHub [257].

figures is the model of the running example. There are two sections in tab one, where the left-side presents a selection screen and the right-side displays the vertices and arcs. Each is labelled using a unique identifier to connect them to other models (enabler or facilitator). All types of vertices and arcs, as described in Section 5.3.2, can be visualised. An application vertex is always the first to be created, and subsequently, other types of vertices are added. Moreover, the different properties of the resources represented by vertex types and the modelling rules (described in Table 5.1) are incorporated into the tool to simplify the modelling activities. For example, the tool will not allow multiple application resources in one model because only one such resource per model can be created. The completed model (HHAM) can be saved in a database and retrieved later. The second tab provides options to analyse the model. For example, the model can be rotated by 180 degrees to represent a BN model structure. All or some vertices can be selected, and the resulting structure and the corresponding relationships can be downloaded to a text file, which can then be used by a probabilistic modelling tool to create a corresponding graphical model automatically.

The vertices are also uniquely identified, and the unique name comprises indicators of an application vertex, a resource group vertex and a parent vertex. All arcs are depicted as directed arrows. The arcs are also uniquely identified, and the name provides details, such as an arc indicator, and indicators for the application, resource group and flow (from child to parent).

The HMTHA tool is developed in Java and requires a Java runtime environment and a database to run. The models can be saved in the database to be loaded at a later time to make additional changes or display. The corresponding database table is created using an SQL script, and a configuration file is provided with the tool that allows changing the database uniform resource identifier. For the evaluation, we used JDK 1.8 and MySQL 8.0 on a Windows 10 PC.

5.6 Summary

In this chapter, we have presented a holistic modelling technique for high availability. The key advantage of our new modelling technique is its support for holistically representing complex IT applications, which implies that core components and other dependencies of an application or a system are addressed. Hence, the HMTHA improves the overall HA solution by presenting the related components and dependencies at multiple levels, and it facilitates attaching the means to perform further analyses. We presented the primary output, a model (HHAM), and a complementary mapping table (M-table) to simplify mapping the model to a probabilistic model structure. Moreover, a set of translation rules (T-rules) that accompanies the technique is presented, and these rules assist in assigning probabilities to a probabilistic model. We focus on Bayesian networks in this thesis and present the translation rules accordingly. Finally, we present a software tool that incorporates all the modelling rules to enable the application of the technique.

Chapter 6

Predicting Locally Manageable Resource Failures

This chapter presents a novel BDN model (BDN-HAC) that represents the third module of the BP framework and that improves the first part of the threefold strategy employed by HACs to deal with individual resource failures, as described in Chapter 1. The BDN-HAC model is used to predict whether a resource failure can be managed locally (with no or minimal disruption) or its resolution requires recovery actions at the resource group or system level (with potentially very significant disruption, including application downtime). To make this important prediction, BDN-HAC uses a comprehensive set of characteristics for the analysed resource (i.e., the resource that failed).

To carry out all the activities associated with the construction of a BDN (Section 4.5), this chapter covers these activities as summarised in Table 6.1 and details in the following sections. Section 6.1 provides an overview of our locally manageable resource failure prediction and related work is discussed in Section 6.2. Section 6.3 presents a formalised, general approach to identifying and capturing the behaviour of HACs and improving their detection and decision capabilities using a set of characteristics. Section 6.4 describes the general variables and state definitions associated with properties representing these characteristics. Section 6.5 discusses the relative weight assignment and dimensionality reduction for the variables. Two alternative BDN models are then proposed in the chapter. Section 6.6 introduces the first BDN model and covers such topics as the variable and state definitions, transformation into the Bayesian decision network, conditional probability and model inferences. Similarly, Section 6.7 details the second BDN model under the same subsections as for the first model. In Section 6.8, inference examples and outcomes are presented, followed by a discussion of the causality, decision network and reasoning with incomplete data. Section 6.9 summarises the chapter.

Predicting Locally Manageable Resource Failures

Table 6.1 BDN-HAC model construction steps and step components

No	Component	BDN-HAC Model Considerations
Step 1: Establish the network structure		
1	Method	The structure is constructed using the identified HAC characteristics, and cause and effect are identified by employing FMEA (Section 6.3). States are identified using FMEA (Section 6.4).
2	Algorithm	BDN data do not support structure learning using data.
3	Training data	N/A
Step 2: Establish the probability distributions		
4	Method	Combination of two approaches: probability scale (Section 6.6.3) and HAC taxonomy as a repository of domain expertise (Section 3.1).
5	Algorithm	N/A
6	Training data	Training is not applicable to the BDN model.
7	Production data	HAC log
8	Mode	N/A
Step 3: Inference		
9	Algorithm	Policy evaluation
10	Data	Inference data are prepared from HAC logs using BFPF (Section 6.6.4).

N/A - Not applicable

6.1 Overview

Understanding how HACs behave upon failure is required to predict whether a resource-level failure is locally manageable or not. To capture that information, our model uses a set of characteristics (or “properties”) that comprises both (i) established characteristics extracted from the research literature and the current practice and (ii) new characteristics identified by our project. The properties associated with these characteristics include Boolean-value properties, indicating whether certain failure recovery mechanisms are present, and integer-value properties specifying the number of times that such a mechanism was activated within a given time window. As such, we organise these characteristics into four groups based on their objectives.

1. The objective of the first group is to understand and interpret the runtime behaviour of HACs for a specific resource failure. As the BDN-HAC model operates as a standalone solution, it must consider the behaviour of the HAC upon failure. Hence, our characteristics indicate whether the HAC failure management modules can automatically reinitialise a resource.
2. The second group improves the failure detection capability by extending the detection scope to include additional characteristics (e.g., the position of a resource in a hierarchy and resource

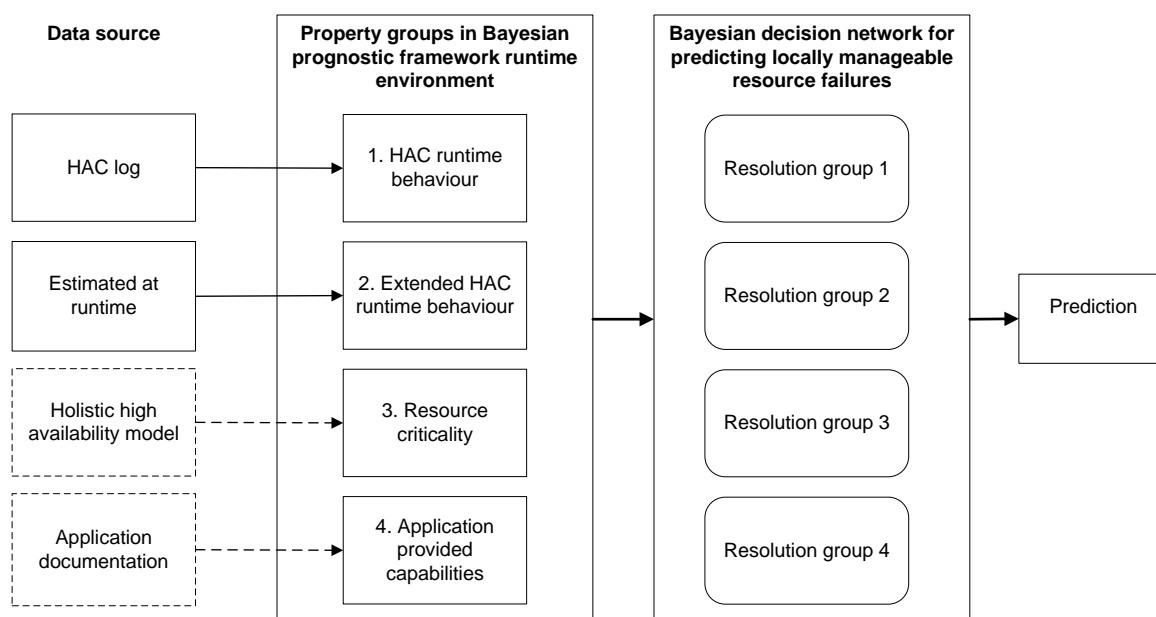


Figure 6.1 Overview of locally manageable resource failure prediction. Dashed boxes and arrows indicate that the data are already available in the runtime environment and are retrieved from sources in the design phase.

types). The objective is to improve the detection capabilities and to detect failure behaviour at a low level of granularity.

3. The objective of the third group is to assess the criticality of a resource. If a resource is identified as noncritical, the implication could be that the failure does not affect the operation of the EA or lead to the failure of any other interconnected resources. Hence, such failure can be masked, and the outcome could be treated as a ‘manageable’ failure.
4. The fourth group is associated with failures that are resolved by events triggered outside the control of the HAC, e.g., failures managed by the protected EAs as part of the application-provided self-healing capabilities [107].

Out of the four groups, only the first group of characteristics is taken into account by the current HACs. The characteristics of the second group are usually not considered by HACs, but the values can be obtained from HACs at runtime. Groups 3 and 4 have new characteristics; hence, Groups 2, 3 and 4 are introduced for the first time in this thesis.

As shown in Figure 6.1, the information associated with the four groups of characteristics used by the BDN-HAC model are obtained from multiple sources. First, the data structures required to support the four property groups are set up in the BFPF module during the design phase. These structures are populated with static values obtained from multiple sources (e.g., HAC configuration, HAC logs, application capabilities and the corresponding HHAM model). When a HAC resource fails at runtime, the failure data are captured from HAC logs and processed by the BFPF module. Hence, some

Predicting Locally Manageable Resource Failures

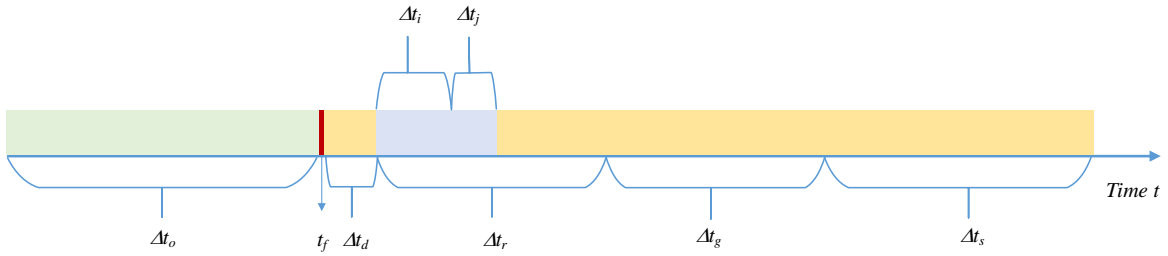


Figure 6.2 Timeline of resource failure events showing the HAC activities in the lower level, and the related activities of the Bayesian decision network model for predicting locally manageable failures in the upper level.

property values come from the HAC log whereas others come from the static information already available in the runtime environment. Additional property values (e.g., the frequency of resource failures) are calculated at runtime. Hence, data for all four property groups are prepared and included into the runtime environment of the BP framework by the BFPF module, as detailed in Figure 6.1.

Values associated with the four property groups are linked to four resolution groups in the BDN-HAC model. These resolution groups and the relationships between the characteristics are encoded in the BDN-HAC model using conditional probabilities and utility preferences. Therefore, the BDN-HAC model assesses characteristics in all four groups, and the model output is a binary value that indicates whether resource failure can be managed locally or not. Hence, our BP framework does not pass this information to its next component (i.e., the BD-HAC) if the output indicates a high probability of managing the failure locally. In contrast, if the value indicates a low probability of managing the failure locally, the failure is classified as locally unmanageable. Subsequently, the information is passed to the next model, the BN-HAC, to propagate and predict a resource group or system level failure.

To work with the characteristics from the four groups in our BDN-HAC model, we introduce techniques that: (i) capture the characteristics as variables, (ii) add relative weights to the variables, and (iii) use these weights to reduce the number of variables (*dimensionality reduction*).

In this chapter, we devise two alternative variants of the BDN model with distinct decision-making techniques, which we label BDN-HAC-1 and BDN-HAC-2. The objective is to include the best-performing model in the BDN-HAC module of the BP framework (as presented in Figure 1.1).

We illustrate the envisaged benefit of the BDN-HAC in Figure 6.2, which shows how, after a period of regular operation Δt_o , a resource (e.g., a file system or an IP address) fails at time t_f .

1. With current HAC solutions, the failure is detected by the HAC during a detection window Δt_d and log entries are generated to indicate this resource failure. The same information is then passed to other modules of the HAC for failure management. Assume, however, the failed resource has a self-healing capability provided by the application. Therefore, when the resource fails, the application initiates the self-healing action (e.g., a restart of the resource) to resolve the problem. Because the HAC is not aware of such capabilities, it proceeds with its threefold

strategy of (1) reinitialising the resource during the window Δt_r , (2) initiating a resource group failover in the Δt_g window if the reinitialisation was unsuccessful, and (3) triggering a system failover (Δt_j) if both the reinitialisation and the resource group failover were unsuccessful.

2. In contrast, the BDN-HAC model (and thus a HAC solution using it) considers additional characteristics related to the self-healing capability of an EA (Group 4), and it therefore predicts that the failure is manageable locally by the self-healing capabilities of the application, indicating that no further action is required. This BDN-HAC prediction is completed within a brief time window $\Delta t_i + \Delta t_j$, where Δt_i is the time for extracting and preparing the relevant log entries to generate BDN-HAC inputs, and Δt_j is the prediction window for BDN-HAC.

Therefore, the most significant improvement of using the BDN-HAC model is the avoidance of the failover of a resource group or of the entire system (and thus the avoidance of potentially significant downtime), as demonstrated by our evaluation of the BP framework in Chapter 10. This improvement can make a significant difference because it can reduce the application MTTR, which improves the overall availability of the application protected by the HAC. Thus, the BDN-HAC model can improve the detection and decision capabilities in HACs by identifying locally manageable resource failures through evaluating a combination of established and newly proposed resource characteristics in a novel way. Therefore, the objective of the model is sixfold and can be described as follows:

1. To capture the behaviour of a HAC upon resource failure using a set of existing (known) characteristics (Group 1).
2. To improve detection and resolution capabilities by evaluating a set of new characteristics (Group 2).
3. To improve detection and resolution capabilities by evaluating a new characteristic related to the criticality of a resource (Group 3).
4. To improve detection and resolution capabilities by introducing a new characteristic to assess the application-provided self-healing capability for a resource (Group 4).
5. To introduce a BDN-based approach that considers all key HAC resource characteristics (both existing and new) and the relationships between the characteristics to predict whether the failure can be managed locally by assessing objectives 1 to 4.
6. To pass only information about unmanageable failures to the BN-HAC model.

Hence, the two variants of the BDN-HAC models presented in this chapter address research question RQ3 from Chapter 1, that being the development of a probabilistic model that captures the essential resource characteristics, reduces dimensions, adds weight and predicts whether the resource failure is manageable. We present the evaluation of the models in Chapter 10.

6.2 Related Work

BDNs are frequently used to optimise the decision-making process under uncertainty in disciplines including engineering [138, 295, 187, 127], medicine [70, 126, 247, 47], biology [157, 35], and information technology [229, 193, 121]. BDNs are also ideal for reducing risk in the decision-making process because risk is associated with uncertainty. For instance, risk can be expressed as a product of likelihood and consequence, which BDNs can represent. The expected utility (EU) of the BDN can then be calculated to maximise the utility, which consequently reduces the risk. Using eq. (4.14), we can express this relationship as:

$$\text{Risk} = P(\text{undesired_outcome}|\text{Action}) \times \text{Consequence}(\text{undesired_outcome}|\text{Action})$$

In economics, BDNs are used to reduce risk (e.g., invest in a new product with minimum risk) [104]. Similarly, reducing risks in projects and other implementation initiatives allow one to choose the option that reduces the risk significantly [151]. BDNs are also widely studied to support complex decision-making processes where there is a need to evaluate a large volume of interconnected data [242]. For instance, Seixas et al. [243] propose a BDN model to support diagnosing Alzheimer's disease (AD) and mild cognitive impairment (MCI). The models show better results for diagnosing MCI when compared to many of the well-known classifiers and competitive results for dementia and AD. Similarly, Neapolitan et al. [182] developed a BDN model to determine whether kidney transplants are beneficial for a particular group of patients by evaluating the likelihood of treatment success.

BDNs are also used in combination with other techniques. For instance, Seixas et al. [36] combined a BDN-based clinical guideline-based system with a rule-based system. Another discipline that employs BDNs to address complex decision-making processes is engineering. For example, Rashid et al. [224] use a BDN model to investigate oil system failure analysis in helicopters, focusing on random failure probabilities. In an example from IT, Christoforou et al. [43] use a BDN model to investigate and determine cloud adaptability for IT services. The research team stated that 'the model provided highly capable results for predicting the right decision'.

To summarise, many studies have explored BDN to support the complex decision-making process under uncertainty. However, none of our surveyed HAC solutions (Section 3.2.4) uses any form of BDNs. Moreover, we could not find any HAC-related research initiatives that employ BDNs or other stochastic decision models. Hence, to the best of our knowledge, the approach proposed in this chapter is the first to construct a BDN model to predict whether resource-level HAC failures are manageable or not based on evaluations of key characteristics of a HAC resource.

6.3 HAC Characteristics for Predicting Locally Manageable Resource Failures

We used a two-stage systematic process to identify HAC characteristics and related properties relevant to detecting and resolving a resource failure and to improving the detection and decision capabilities. In the first stage, we analysed and identified a set of characteristics by using research studies, technical manuals, HAC and EA documentation, as well as the taxonomy and survey presented in Chapter 3. We then organised the identified characteristics into two categories. The first category consists of key HAC characteristics already known to impact resource failure analysis (established characteristics). For example, the characteristic to enable reinitialisation by a HAC assumes that the HAC understands how the resource works to proceed with the correct initialisation (e.g., remount a file system). The second group consists of characteristics mentioned in the literature but not related to the HAC resolution of resource failures. Therefore, we deem these as new characteristics when used alongside HACs for the first time in this thesis. These new characteristics can significantly improve detection and resolution capabilities. For example, some EAs have in-built fault tolerance capabilities to resolve resource failures using self-healing or rejuvenation (e.g., restarting a service or a process) [84]. In such a case, the first attempt to resolve the failure is managed by the application. Therefore, using this characteristic improves failure management. The two categories are defined as follows:

1. **Established characteristics (EC).** *EC is a set of established HAC characteristics that influence how a HAC behaves upon a resource's failure.*
2. **New characteristics (NC).** *NC is a set of new characteristics that extend the EC category to improve the detection and resolution capabilities by capturing more details to make accurate detections upon a resource's failure.*

In the second stage, we performed FMEA (Section 5.3.4), which aims to capture the characteristics that significantly influence a resource upon failure. FMEA was performed in two steps:

- (i) each characteristic was treated as a component, and the different states of a characteristic were considered potential failure modes. Characteristics with a high RPN to rank system failures were selected. Further, we narrowed the list of characteristics by ranking them using RPNs and their applicability. Applicability implies whether it is possible to obtain information related to a characteristic given the conditions of the testbed environment. We identified nine critical properties (four ECs and five NCs) out of the six ECs and 10 NCs initially considered. A complete list of the characteristics that were eventually dismissed as insufficiently relevant is provided in Appendix A.
- (ii) We used three common components to connect to the nine characteristics [85]. For example, error-related characteristics were connected to an error-related common component. Thus, we used these three common components as component failures and the nine characteristics as failure modes to identify the effect of the individual characteristics on common component failures.

Predicting Locally Manageable Resource Failures

Table 6.2 Resource properties of high-availability clusters grouped by sets

HAC Resource Property	Description	Category	Values	RG
Error-related properties set E				
Failure repetition (fr)	Number of failures that occurred in the last $n > 0$ minutes	EC	{0,...,n}	1
Redundancy factor (rf)	Application provides in-built self-healing capabilities	NC	{0,1}	4
Aggregated failure count (afc)	Distinct failures of the resource within the last $m > 0$ hours	EC	{0,...,n}	1
Reinitialisation factor (rc)	Resource reinitialisation possible	EC	{1,2}	1
Dependency-related properties set D				
Dependency type (dt)	The type of a resource	NC	{1,2,3}	2
Dependency levels down (dld)	The number of lower-level resources	NC	{0,...,n}	2
Dependency levels up (dlu)	The number of upper-level resources	NC	{0,...,n}	2
Criticality-related property set C				
Critical factor (cf)	Indicates the criticality of a resource	NC	{0,1}	3
Current status-related property set S				
Current state (cs)	Current status of a resource	EC	{On,Off}	1

On - Online, Off - Offline, PG - Resolution group, EC - Established characteristics, NC - New characteristics

This approach helped identify the cause-and-effect relationship between the characteristics and common components. The common components are described further as derived variables in the next section.

Table 6.2 lists the nine retained properties used as inputs for our BDN-HAC models, as well as the descriptions and categories for each one of them. The values column specifies their value ranges, and the column RG (Resolution group) lists the related resolution groups (cf. Section 6.1). The motivation for the selection of each property, and its goal in the proposed model are as follows:

1. Failure repetition

Motivation. When a HAC reinitialises a failed resource, it uses a local timeout value for the reinitialisation to complete [21, 128, 168]. If the reinitialisation does not complete successfully within the time limit imposed by this timeout, the HAC will retry the reinitialisation procedure (up to several times). If none of these reinitialisation attempts succeeds, the HAC will eventually reclassify the resource as not reinitialisable. Therefore, the number of such attempts that have been performed within the last n minutes is an important indicator of how likely the resource is to incur a more severe failure. The value for n is estimated from the number of failures allowed

in a HAC for a resource type and the resource's start time. However, the resource types have different values for start time, and the number of allowed failures per node could be different per resource. Therefore an average value is calculated from multiple resource types.

The role in the BDN-HAC model. The model considers the number of failures while also assessing other properties (for example, the *aggregated failure count* or the *reinitialisation factor*) to determine whether the likelihood of unmanageable failure increases or not.

2. Redundancy factor

Motivation. An application that a HAC protects may have in-built self-healing capabilities (e.g., software rejuvenation [297, 84]) for key resources, enabling the application to automatically initiate the first mitigation action upon the failure of a resource [93, 274, 107]. HACs must recognise these features to avoid initiating any mitigating actions that could conflict with the application's actions. However, this property is not used by HACs.

The role in the BDN-HAC model. This property indicates how the application-provided self-healing capabilities can be used to reinitialise a resource, thus increasing the likelihood of managing the resource failure locally.

3. Aggregated failure count

Motivation. The number of failures of a resource during a period is aggregated to indicate a potential persistent failure. It can also show that a global threshold value for the timeout to manage resource failures is reached [21, 168, 284]. The implication is that a HAC classifies the resource as more error-prone in a specific node and may ban the resource from getting started in that node. A high number of failures of a resource within the last n hours is an indicator for a potential persistent failure. The aggregated failure count is calculated using a global timeout value [21, 168, 284], the number of failures allowed in a node for a resource type, and the average start time per resource type. When a resource cannot be started on a node after exceeding the number of allowed starts on the node, this may be because: (1) policies are set automatically by the HAC to prevent the resource from starting on the node; (2) policies are set automatically by the HAC to prevent the resource from starting on other nodes; or (3) policies are set not to allow the resource to start on any node. An example of the third case is file system corruption. Suppose the corruption is on a block level. In that case, it affects shared storage or replication, showing the same failure in all nodes, which results in setting a policy to prevent the resource from being brought up in any node, affecting all the related resources.

The role in the BDN-HAC model. This property identifies a persistent failure pattern and whether the mitigation actions have been successful or not. For example, a high value indicates a more persistent failure. Moreover, the high value could also indicate that the mitigation actions may not have been successful, and the HAC may have set one of the three policies to prevent the resource from getting started. A low value indicates a low probability of failure, and when combined with other positive outcomes, the result may indicate a high likelihood of managing the failure locally.

4. Reinitialisation factor

Motivation. This refers to a HAC's ability to reinitialise a resource [21, 239]¹. The reinitialisation procedures are different for the different types of resources. For example, if the resource type is a service, the procedure is to restart the service. If the resource type is a file system, the mitigation action is to remount the file system. There are multiple steps associated with these procedures, such as checking the resource status and shutting it down gracefully before restarting.

The role in the BDN-HAC model. This property evaluates whether a HAC can reinitialise a resource or not. If the property is set to true, the probability of managing failure for the resource increases significantly.

5. Dependency type

Motivation. There are three types of resource dependencies (local, shared and global), and they have different impact factors [239] when the related resources fail. Hence, each resource is assessed based on the impact factor. A local dependency type can only impact other resources in the same group, while a shared resource may impact one or more related resource groups. On the other hand, a global resource is likely to impact all resource groups and the entire system.

The role in the BDN-HAC model. This property evaluates the impact factor associated with each resource type and combines the outcome with the results from evaluating other properties to enable accurate predictions.

6. Dependency levels down

Motivation. The HAC resources have a hierarchical organisation [167, 239], which means start and stop procedures follow a particular sequence to start or stop all the related resources. For example, when a resource fails, an attempt to reinitialise the resource is started, including the resources at the lower level within the hierarchy. If the number of such lower-level resources is high, it may impact the overall start or stop time which can, in turn, decrease the likelihood of managing failure of any of those resources.

The role in the BDN-HAC model. Considers the impact of losing lower-level resources when a resource fails or when a resource is reinitialised. A lower value indicates an impact on fewer resources; the evaluation can be combined with other properties such as the *critical factor* and *reinitialisation factor*.

7. Dependency levels up

Motivation. Similar to the dependency level down, a high number of upper-level resources decreases the likelihood of managing failure [167, 239]. When a low-level resource fails, it may impact all the related upper-level resources within the hierarchy. For example, if a low-level resource "disk" crashes, it may terminate all the processes using that disk, causing those resources' failures. These, in turn, can cause failures of other resources to adhere to the

¹The ability to reinitialise failed resources or the failed components of a resource automatically by the HAC.

start and stop dependencies. If a sufficient number of resources fail, it can be interpreted by the HAC as critical, thus initiating a failover either for a resource group or for the entire system.

The role in the BDN-HAC model. This property assesses the impact on upper-level resources. A higher value results in a low likelihood of managing failure.

8. Critical factor

Motivation. If a resource is critical, the probability of causing a resource group failure or a system failure is estimated to be high [49]. The objective is to evaluate whether such a resource has an immediate impact on the system's operations. For example, if a system can survive without a particular resource for a short period, it can be rendered as noncritical. Failure of such a resource does not need to be propagated to other resources; hence, there is no impact at the resource group or system level.

The role in the BDN-HAC model. Considers the critical factor of a resource; if a resource is not critical, the likelihood of the propagated failure is reduced significantly.

9. Current state

Motivation. This property captures the current status of a resource [49]. If the status is offline and the property failure repetition has also recorded a high number, this may indicate that the failure's resolution was unsuccessful. If the resource is online after recording a momentary failure, it may indicate that the procedures associated with either the reinitialisation factor or the redundancy factor may have resolved the problem.

The role in the BDN-HAC model. The model considers the current state of a resource. For example, when the state is online, it significantly increases the likelihood of managing failure.

6.4 General Variable and State Definitions

Before we can use the HAC resource properties from Table 6.2 with the BDN introduced in this chapter, they need to be mapped to basic variables.² This mapping is described in Table 6.3, which shows the symbols, values, and groups for these variables. Four groups of variables are identified based on the change they have to undergo before being part of a model (Table 6.3). Group 1 variables require categorising the value of their corresponding property as either 'low' when this value is not larger than a threshold specified later in the thesis or 'high' when this value exceeds the threshold. This group includes the variables *failure repetition*, *aggregated failure count*, *dependency levels down*, and *dependency levels up*. Group 2 variables are converted from integer to Boolean, and the variables in the scope are *redundancy factor*, *reinitialisation factor*, and *critical factor*. The third group of variables do not change but are transferred directly to the model, and they are *current state* and *dependency type*. The fourth group represents the *derived variables* (i.e., variables obtained from the set properties in Table 6.2); the variables in this group are *error rating*, *dependency factor* and

²Properties have multiple values, and mapping them to variables allows them to be processed (e.g., by transforming them into categorical variables).

Predicting Locally Manageable Resource Failures

Table 6.3 Basic variables representing the properties of high-availability clusters, related symbols, values, and variable groups (including group 4 of *derived variables*)

Property/Variable name	Symbol	Values	Group
Failure repetition	<i>fr</i>	{low, high}	1
Redundancy factor	<i>rf</i>	{true, false}	2
Aggregated failure count	<i>afc</i>	{low, high}	1
Reinitialisation factor	<i>rc</i>	{true, false}	2
Dependency type	<i>dt</i>	{local, shared, global}	3
Dependency levels down	<i>dld</i>	{low, high}	1
Dependency levels up	<i>dlu</i>	{low, high}	1
Critical factor	<i>cf</i>	{true, false}	2
Current state	<i>cs</i>	{online, offline}	3
Error rating	<i>e</i>	{failure, no_failure}	4
Dependency factor	<i>d</i>	{low, high}	4
Resource state	<i>r</i>	{failure, no_failure}	4

resource state. The steps for our variable transformation and conversion are described further in Section 8.5 .

All variables are binary except for the *dependency type*, which is multivalued to represent the three types of dependency. The decision to use such binary variables comes from our FMEA performed in Section 6.3 (step i), where possible states of each property were included to investigate the different states of resources, potential failures, and effects [213, 149].

6.5 Relative Weight Assignment and Dimensionality Reduction

The different variables from Table 6.3 have different impacts on the outcome of the BDN-HAC model. Hence, a weighting factor is encoded in the model to reflect these different levels of impact. For example, the variable *reinitialisation factor* (*rf*) has more weight than *dependency levels down* (*dld*) because, if the *reinitialisation factor* is set to true, the implication is that the HAC can reinitialise the resource. If that activity succeeds, the resource is no longer considered a failed resource; hence, the BDN-HAC model does not interpret the failure as a failure that needs to be managed locally. This means that the *reinitialisation factor* is more important than *dependency levels down*, and more weight is associated with it. Furthermore, the number of variables must also be reduced (dimensionality reduction) in order for the BDN-HAC model to process and compute an outcome for the next model in the BP framework. This section presents our approach for applying relative weights to each property and reducing dimensionality.

Dimensionality reduction and weight assignment are performed in two steps. In the first step, the variables are grouped based on the defined sets in Table 6.2. There is a causal relationship between the variables within a set. For example, when the value of a variable changes, this also influences other

6.5 Relative Weight Assignment and Dimensionality Reduction

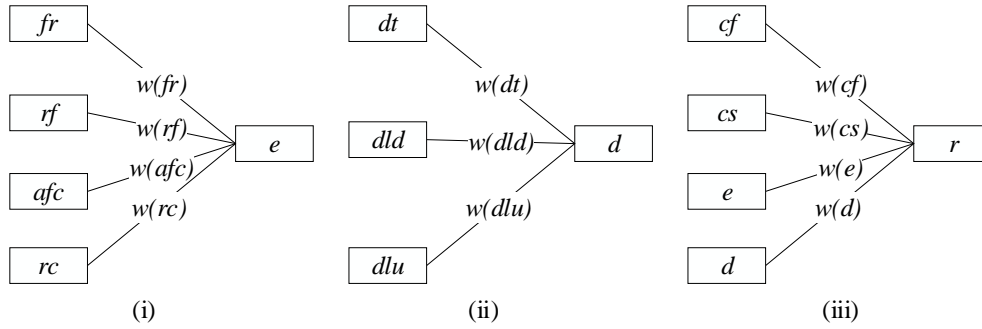


Figure 6.3 Reducing dimensionality and adding relative weights to the variables in (i) the error-related properties set E , (ii) the dependency-related properties set D , and (iii) combining the outcomes of (i) and (ii) with criticality-related property set C and current status-related property set S .

variables in the same set, as described in stage 2 of the FMEA (Section 6.3). The relative weights within a set are used to derive a target variable reflecting the impacts of the variables from that set. This target variable represents all the variables in the set, hence reducing the dimensionality effectively. Two main variable sets are defined: *error rating* (E) and *dependency factor* (D). The former comprises all variables related to errors in a resource, while the latter comprises the dependency-related variables. Figure 6.3 (i) shows the variable set *error rating* (E) and the four variables that are part of this set. The four variables undergo dimensionality reduction where they are consolidated into the target variable e with a relative weight $w(v)$ assigned to each variable $v \in E$. Similarly, Figure 6.3 (ii) shows the set *dependency factor* (D), which has three variables, and its dimensionality is reduced to one target variable d through associating relative weights to its elements.

In the second step (Figure 6.3 (iii)), the dimensionality is reduced further. To this end, both the target variables e and d derived from the sets E and D , respectively, and the two variables from sets C and S from Table 6.2 are consolidated into a target variable r . Thus, eleven variables are reduced to one variable. The value of the target variable r is an indicator of whether the resource failure in question can be managed or not.

The relative weights were calculated from the RPNs obtained using FMEA (Section 6.3) as follows

$$W_i = \frac{w_i}{\sum_{i=1}^n w_i}, \quad (6.1)$$

where w_i represents the value obtained from the i -th RPN, and n denotes the total number of components connected to a target variable.

In the remainder of this section, we demonstrate the two steps used to perform dimensionality reduction and weight assignment.

All variables are assumed to be random, and therefore a probabilistic approach when adding weights for a given state of a child (target) variable is formalised. For any combination of values fr_0 , rf_0 , afc_0 and rc_0 of the four input variables from Figure 6.3 (i), the conditional probability that the

error e has a specific value e_0 is given by

$$P(e = e_0 | fr = fr_0, rf = rf_0, afc = afc_0, rc = rc_0) = \sum_{v \in E} [w(v)P(e = e_0 | v = v_0)], \quad (6.2)$$

where $w(v) \in (0, 1]$ is a weight associated with the variable $v \in E$ such that $\sum_{v \in E} w(v) = 1$.

Similarly, for any combination of values dt_0 , dtd_0 and dlu_0 of the three input variables from Figure 6.3 (ii), the conditional probability that the dependency d has a specific value d_0 is given by

$$P(d = d_0 | dt = dt_0, dld = dld_0, dlu = dlu_0) = \sum_{v \in D} [w(v)P(d = d_0 | v = v_0)], \quad (6.3)$$

where $w(v) \in (0, 1]$ is a weight associated with the variable $v \in D$ such that $\sum_{v \in D} w(v) = 1$.

In the second step, the outcomes of equations (6.2) and (6.3) are combined with the two variables from the property sets C and S in Table 6.2. For any combination of values cf_0 , cs_0 , e_0 and d_0 of the input variables from Figure 6.3 (iii), the conditional probability that the resource state r has a specific value r_0 is given by

$$P(r = r_0 | cf = cf_0, cs = cs_0, e = e_0, d = d_0) = \sum_{v \in R} [w(v)P(r = r_0 | v = v_0)], \quad (6.4)$$

where $R = \{cf, cs, e, d\}$, $w(v) \in (0, 1]$ is a weight associated with the variable $v \in R$ such that $\sum_{v \in R} w(v) = 1$. This equation provides the likelihood of resource failure after reducing dimensionality and adding weights.

Our two proposed BDN-HAC model variants (BDN-HAC-1 and BDN-HAC-2) use different variants of the equations (6.2), (6.3) and (6.4). Weight assignment and dimensionality reduction are performed as part of constructing the model and defining CPT, and this is described in Section 6.6 (for model BDN-HAC-1) and in Section 6.7 (for model BDN-HAC-2).

6.6 BDN-HAC-1

In this section, we present our first variant of the BDN model. The model uses the variables from Section 6.3, but the types are changed to represent a BDN model. Additionally, a standard utility function is employed to output the outcome in the model. The model construction is described in the next sections.

6.6.1 Variable and State Definition

The BDN-HAC-1 model includes one utility node and one decision node. Table 6.4 lists the node identifiers, the nodes representing the variables (node name), the node type, state and the relative weights associated with each node.

Table 6.4 Description of all the nodes in the BDN-HAC-1 model

Node ID	Node Name	Node Type	State	Relative Weights
A_1	Error rating	Chance	{failure, no_failure}	{ ≈ 0.15 }
A_2	Failure repetition	Chance	{low, high}	{ ≈ 0.1 }
A_3	Redundancy factor	Chance	{true, false}	{ ≈ 0.4 }
A_4	Aggregated failure count	Chance	{low, high}	{ ≈ 0.1 }
A_5	Reinitialisation factor	Chance	{true, false}	{ ≈ 0.4 }
B_1	Dependency factor	Chance	{low, high}	{ ≈ 0.15 }
B_2	Dependency type	Chance	{local, shared, global}	{ ≈ 0.3 }
B_3	Dependency levels down	Chance	{low, high}	{ ≈ 0.4 }
B_4	Dependency levels up	Chance	{low, high}	{ ≈ 0.3 }
C_1	Critical factor	Chance	{true, false}	{ ≈ 0.35 }
D_1	Current state	Decision	{online, offline}	{ ≈ 0.35 }
U_1	Resource state	Utility	{ $-100 < U_1 < 100$ }	Target

The node ID column shows the technical names of the nodes. The node names are the same as in Section 6.3. Node type refers to the type of the node in the model, and it also shows how the variable types in Section 6.3 are mapped onto node types in the model. The column 'state' describes the states associated with each node, and the column 'relative weights' lists the relative weights associated with each node. The model mainly uses the variables defined in Section 6.3. However, two variables are changed to reflect the BDN nature of the model. Node D_1 (variable *current state*) is changed to a decision node with two states, *online* and *offline*, while node U_1 represents the variable *resource state*. Node U_1 is a utility node responsible for assembling the outcome; hence, it does not follow a probabilistic approach but instead uses a utility function. The node uses a scale between -100 and 100, where 0 is used as a boundary to interpret the outcome. The utility node is continuous, while all the other nodes are discrete.

The relative weights were obtained using eq. (6.1). First, we calculated the weights for each variable connected to a target variable, and an example is provided for the target variable A_1 and connected variable A_2 below. Assuming that the variable A_2 has the RPN of 100, we obtain the following

$$A_2 = \frac{100}{1000} = 0.1, \quad (6.5)$$

where 100 represents the RPN obtained from the FMEA, and 1000 denotes the sum of all RPNs for the connected variables A_2 , A_3 , A_4 and A_5 .

Second, we calculated the relative weights of the nodes A_2 , D_1 , B_1 and C_1 using eq. (6.1), and the resulting relative weights were added to the utility table of U_1 . Further, if the outcome of a common node is negative, a negative weight is added, and when the outcome of a common variable is positive, a positive weight is used, which can be expressed as follows

Predicting Locally Manageable Resource Failures

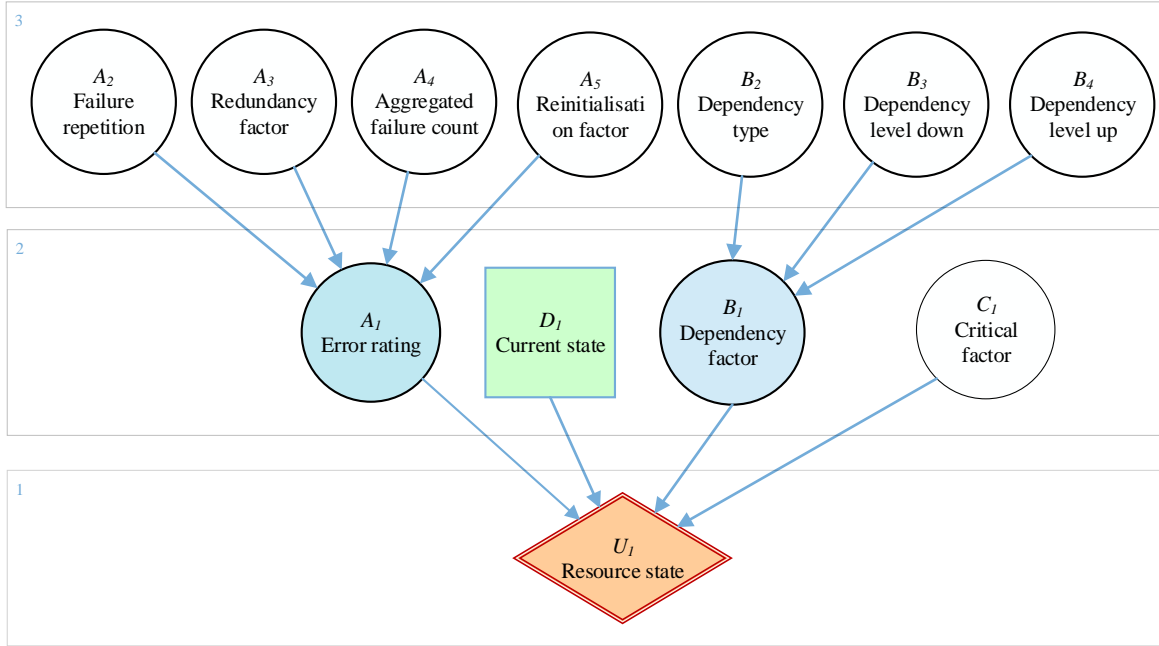


Figure 6.4 Bayesian decision network model BDN-HAC-1. Random nodes are depicted with a white background, blue shading indicates latent nodes, green shading indicates the decision node, and the utility node is shaded orange.

$$W_i = \begin{cases} -w_i, & \text{if } c_i = \text{negative outcome} \\ w_i, & \text{otherwise,} \end{cases} \quad (6.6)$$

where c_i represents a negative outcome in a child node, such as that obtained for $A_1 = \text{failure}$, $B_1 = \text{low}$, $C_1 = \text{true}$ and $D_1 = \text{offline}$.

The calculation of utility values using both weights and conditional probabilities is described in Section 6.6.2, and examples are provided in Section 6.6.4. We validated the BDN models and the corresponding numbers using two example applications constructed from HAC log files obtained from a company. Further validations were done using the running example (Section 2.1) and the testbed application (Section 10.1).

6.6.2 Transformation into the Bayesian Decision Network

As shown in Figure 6.4, the BDN-HAC-1 model is organised into three layers and twelve nodes. The third layer consists of nodes that represent most of the HAC properties. The causality is maintained by ensuring that the nodes belonging to the same set build a causal relationship. Subsequently, conditional probabilities are used to quantify the causal relationships.

Two nodes (A_1 and B_1) in layer two are responsible for reducing the parent nodes' dimensionality from layer three and adding relative weights. This is possible due to conditional probabilities in the two child nodes. These two nodes are *latent nodes* in the BDN; hence, they are unobservable.

In addition, two nodes (D_1 and C_1) in layer two capture the crucial properties of HACs, which can significantly influence the overall outcome. All four nodes in layer two converge in the first layer.

The only node in layer one is a utility node (U_1), which is responsible for adding weights and reducing the dimensionality further by associating each parent node with a preference. Both chance nodes and the decision node in layer two influence the utility node, while the nodes in layer three have an indirect influence on the utility node. Hence, D_1 has a significant influence on U_1 because U_1 aims to maximise the expected utility of the decisions by D_1 . This relationship is represented by a functional edge from D_1 to U_1 , which indicates a functional dependency. The decisions made by D_1 also influence the rest of the network. However, the model considers all nodes, which means that the decision from D_1 may not reflect the actual outcome. For example, an unfavourable decision by D_1 may not lead to a negative outcome by the model.

Following the two-step approach defined in Section 6.5, and using equations (6.2), (6.3) and (6.4), we present the following equations to represent the model.

First, given any combination of values $A_{2,0}$, $A_{3,0}$, $A_{4,0}$ and $A_{5,0}$ for nodes A_2 to A_5 , the conditional probability that the *error rating* node A_1 has a specific value $A_{1,0}$ is given by

$$P(A_1 = A_{1,0} | A_2 = A_{2,0}, A_3 = A_{3,0}, A_4 = A_{4,0}, A_5 = A_{5,0}) = \sum_{v \in \{A_2, A_3, A_4, A_5\}} [w(v)P(A_1 = A_{1,0} | v = v_0)], \quad (6.7)$$

where $w(v) \in (0, 1]$ is a weight associated with the variable $v \in \{A_2, A_3, A_4, A_5\}$ such that

$$\sum_{v \in \{A_2, A_3, A_4, A_5\}} w(v) = 1.$$

Similarly, given any combination of values $B_{2,0}$, $B_{3,0}$ and $B_{4,0}$ for nodes B_2 , B_3 and B_4 , respectively, the conditional probability that the *dependency factor* node B_1 has a specific value $B_{1,0}$ is given by

$$P(B_1 = B_{1,0} | B_2 = B_{2,0}, B_3 = B_{3,0}, B_4 = B_{4,0}) = \sum_{v \in \{B_2, B_3, B_4\}} [w(v)P(B_1 = B_{1,0} | v = v_0)], \quad (6.8)$$

where $w(v) \in (0, 1]$ is a weight associated with the variable $v \in \{B_2, B_3, B_4\}$ such that

$$\sum_{v \in \{B_2, B_3, B_4\}} w(v) = 1.$$

In a second step, the outcomes of equations (6.7) and (6.8) are combined with nodes C_1 and D_1 . For any combination of values $A_{1,0}$, $B_{1,0}$, $C_{1,0}$, $D_{1,0}$ of nodes A_1 , B_1 , C_1 and D_1 , respectively, the

Predicting Locally Manageable Resource Failures

preference that the utility node *resource state* $U_1 \in [-100, 100]$ has a specific value $U_{1,0}$ is given by:

$$U(U_1 = U_{1,0} \mid A_1 = A_{1,0}, B_1 = B_{1,0}, C_1 = C_{1,0}, D_1 = D_{1,0}) = \sum_{v \in \{A_1, B_1, C_1, D_1\}} [w(v)U(U_1 = U_{1,0} \mid v = v_0)], \quad (6.9)$$

where $w(v) \in (0, 1]$ is a weight associated with the variable $v \in \{A_1, B_1, C_1, D_1\}$ such that

$$\sum_{v \in \{A_1, B_1, C_1, D_1\}} w(v) = 1.$$

The outcome of the utility node U_1 (and thus of our BDN-HAC-1 model) is interpreted as

$$P(\text{locally_manageable_resource_failure}) = \begin{cases} \text{low}, & \text{if } U_1 < 0 \\ \text{high}, & \text{otherwise} \end{cases} \quad (6.10)$$

Hence, a utility $U_1 = 0$ functions as the cutoff value for the BDN-HAC-1 decision making.

6.6.3 Conditional Probability Tables

All or most non-utility and non-latent nodes must be instantiated to predict the outcomes as accurately as possible. When nodes are instantiated, only the posterior distributions are evaluated. However, if a situation with incomplete data arises, the prior distribution is also evaluated. Hence, the model's objective is to instantiate as many nodes as possible to improve the prediction quality.

This section describes the conditional probabilities associated with each node, as well as the reasons and justifications for the way in which we set their prior distributions. The probability distributions are estimated using a method of elicitation [226, 131]. The method uses a numerical probability scale [227, 275], and elicitation is performed at a single probability level [131]. As input, we used extensive literature reviews and the expert knowledge developed in this research. However, it was challenging to obtain statistics concerning HAC failures, and in particular, failures at the resource-level. Therefore, we used studies of several non-HAC systems and configurations (e.g., HPC [241, 240]) to estimate the probability distributions.

Table 6.5 lists the probability distributions obtained by using the above method for all relevant nodes in the model BDN-HAC-1. While chance nodes have CPTs, there are no probability distributions associated with the utility (U_1) and decision (D_1) nodes. The decision table D_1 lists two possible states, *online* and *offline*. The utility table U_1 represents weight assignments and dimensionality reductions as a numerical measure of preferences over the entire network. Hence, there are ten CPTs, one decision table, and one utility table associated with the BDN-HAC-1 model. The CPTs for all the nodes in layer three and the layer two node, C_1 , have local distributions. The CPTs for (A_1 and B_1) are specified as conditional probabilities over their parent nodes. The weight assignments and dimensionality reductions are encoded into those conditional probabilities and are presented in the CPTs.

Table 6.5 Probability distributions for all nodes in the model BDN-HAC-1 and their states

Node	Probability Distribution
A_1	$P(A_1 A_2, A_3, A_4, A_5)$
A_2	low =.75 high =.25
A_3	true =.3 false=.7
A_4	low=.9 high=.1
A_5	true =.75 false=.25
B_1	$P(B_1 B_2, B_3, B_4)$
B_2	local=.8 shared=.15 global=.05
B_3	low =.5 high =.5
B_4	low =.3 high =.7
C_1	true =.8 false=.2
D_1	online offline
U_1	utility node

The distribution of A_2 reflects the fact that the probability of repeated failures is relatively low because the HAC has the responsibility of mitigating the failures promptly, and a failover is triggered otherwise [284]. Therefore, the state *low* is set to a higher probability distribution than the state *high*.

The distribution of A_3 represents the in-built redundancy factor provided by the applications. An EA include self-healing capabilities on a resource-level, such as restarting a crucial process quickly [146, 194]. However, such systems may only deal with issues associated with specific components, such as processes; this means that such capabilities cannot be extended to include a platform or infrastructure-related components because the applications do not manage those. When applications provide these capabilities, the HAC must still evaluate the impact on other linked resources. Therefore, the probability distributions are set while considering all these factors.

The distribution for A_4 is set to reflect the fact that a HAC can mitigate repeated errors in most cases [223, 194]. The value for A_5 is derived from the fact that most HACs provide a mechanism to

reinitialise a resource. Therefore, A_5 is considered to be a typical property of HACs. However, some resources cannot be reinitialised by HACs, such as a CPU, memory and operating system-related errors [120].

B_2 captures the type of dependency on a resource [223, 284]. The assumption here is that the failure of a resource with local dependency is restricted to the resource group. In contrast, the failure of a shared resource may impact multiple resource groups. Similarly, a global dependency can have an impact at a system level. Therefore, the distribution is set based on the distributions of these in typical HAC configurations.

The distribution of B_3 describes the impact on all the lower-level nodes [223]. Similarly, B_4 presents the distribution based on the impact on all the upper-level resources.

Finally, the distribution of C_1 reflects the criticality of a resource. Most of the resources are considered to be critical [223]. However, some resources can be classified as noncritical because they do not pose an immediate threat to the application's operation.

6.6.4 Model Inference Example

In this section, we provide an example of model inference. However, the BDN-HAC model only expects input values to output a value. Therefore, the example calculation does not reflect how the model computes the outcome. To provide an example of model inference, we assume that all parent nodes in layer three ($A_2, A_3, A_4, A_5, B_2, B_3$ and B_4) and the nodes C_1 and D_1 in layer two are instantiated. First, we assume that nodes A_2 to A_5 are instantiated as follows:

- A_2 is instantiated as having a low value, which indicates that there is no or negligible failure repetition of the resource.
- $A_3 = true$ indicates that the resource has in-built self-healing capability that reduces the likelihood of failure and increases the likelihood of managing failure locally.
- A_4 is set to *low*, which means that the aggregated failure count is *low* (none or negligible).
- A_5 , which is a key property of HAC, has the state *true*, which means the HAC can reinitialise the resource, and this reduces the likelihood of failure significantly.

Assuming that the four weights from eq. (6.7) are $w(A_2) = 0.1$, $w(A_3) = 0.4$, $w(A_4) = 0.1$ and $w(A_5) = 0.4$, we obtain

$$\begin{aligned}
 P(A_1 = A_{1,0} \mid A_2 = low, A_3 = true, A_4 = low, A_5 = true) &= \\
 &= 0.1 \cdot P(A_1 = A_{1,0} \mid A_2 = low) + 0.4 \cdot P(A_1 = A_{1,0} \mid A_3 = true) + \\
 &\quad + 0.1 \cdot P(A_1 = A_{1,0} \mid A_4 = low) + 0.4 \cdot P(A_1 = A_{1,0} \mid A_5 = true) = \\
 &= 0.99
 \end{aligned} \tag{6.11}$$

where A_1 represents a target node, and $w(v) \in (0, 1]$ is a weight associated with the variable $v \in A_2, A_3, A_4$ and A_5 such that $\sum_{v \in A_2, A_3, A_4, A_5} w(v) = 1$. Hence, when we instantiate the BDN-HAC model with these values, it outputs *no_failure*, and the probability of that state is calculated as 99%.

Similarly, we assume that B_2, B_3 and B_4 are instantiated as follows:

- B_2 is instantiated as having a *local* value, which indicates low impact upon failure.
- $B_3 = low$ indicates that the number of lower-level resources is low within the hierarchy that reduces the likelihood of failure and increases the likelihood of managing failure locally.
- B_4 is set to *low*, which means the number of upper-level resources is low within the hierarchy, and thus reduces the likelihood of failure and increases the likelihood of managing failure locally.

Assuming that the three weights from eq. (6.8) are $w(B_2) = 0.3$, $w(B_3) = 0.4$ and $w(B_4) = 0.3$, we obtain

$$\begin{aligned}
 P(B_1 = B_{1,0} \mid B_2 = local, B_3 = low, B_4 = low) &= \\
 &= 0.3 \cdot P(B_1 = B_{1,0} \mid B_2 = low) + 0.4 \cdot P(B_1 = B_{1,0} \mid B_3 = true) + \\
 &\quad + 0.3 \cdot P(B_1 = B_{1,0} \mid B_4 = low) = \\
 &= 0.90
 \end{aligned} \tag{6.12}$$

where B_1 represents a target node, and $w(v) \in (0, 1]$ is a weight associated with the variable $v \in B_2, B_3, B_4$ such that $\sum_{v \in B_2, B_3, B_4} w(v) = 1$. The outcome is *low*, and the probability of that state is calculated as 90%, which effectively lowers the risk of failure due to dependencies.

In the second step, we use the output from the layer three nodes to consolidate into layer two nodes, and instantiate two additional nodes, as follows:

- C_1 is set to *true* to indicate that the resource is a critical resource which increases the likelihood of failure while reducing the likelihood of managing the failure locally.
- the decision node D_1 is instantiated with the state *offline*, which means the resource is offline.
- A_1 has the state *no_failure* with a probability of 99% as computed in eq. (6.11).
- B_1 has the state *low* and probability of that state is calculated as 90% from eq. (6.12).

All four nodes are then consolidated into the layer one utility node as follows:

Assuming that the four weights from eq. (6.9) are $w(C_1) = 0.35$, $w(D_1) = 0.35$, $w(A_1) = 0.15$ and $w(B_1) = 0.15$, we obtain

$$\begin{aligned}
 U(U_1 = U_{1,0} \mid C_1 = true, D_1 = offline, A_1 = no_failure, B_1 = low) &= \\
 &\approx 0.35 \cdot U(U_1 = U_{1,0} \mid C_1 = true) + \approx 0.35 \cdot U(U_1 = U_{1,0} \mid D_1 = offline) + \\
 &\quad + \approx 0.15 \cdot U(U_1 = U_{1,0} \mid A_1 = no_failure) + \approx 0.15 \cdot U(U_1 = U_{1,0} \mid B_1 = low) = \\
 &= \approx 56
 \end{aligned} \tag{6.13}$$

where $U_1 \in [-100, 100]$ represents a target node, and $w(v) \in (0, 1]$ is a weight associated with the variable $v \in C_1, D_1, A_1, B_1$ such that $\sum_{v \in C_1, D_1, A_1, B_1} w(v) = 1$.

Revisiting eq. (6.10) and considering that the output of the utility node is 56, the model predicts that the resource failure can be managed locally (no failure) with a high degree of confidence. This is despite having the current status *offline* for the resource. Although it is the evaluation of all nodes and their combinations that dictate the outcome, three nodes play a significant role in this case. Node C_1 is set to true, which means the resource is a critical resource. The second node A_3 has the value *true*, which means there is an in-built self-healing capability by the protected application that enables the application to reinitialise the resource. The third node A_5 is set to *true*, which means that HAC can also reinitialise the resource.

6.7 BDN-HAC-2

In this section, we present our second variant of the BDN model, BDN-HAC-2. This model variant uses a variation of utility known as Additive-Linear Utility (ALU) [71]. An ALU approach enables the linear addition of multiple utility nodes to an ALU node.

6.7.1 Variable and State Definition

The BDN-HAC-2 model has three utility and two decision nodes. Two parent utility nodes are linked to a child utility node (the ALU node). Table 6.6 lists the nodes, descriptive node names (variable name), node types, and the states and relative weights associated with each node.

The model uses variables that represent the different properties of HACs, as explained in Section 6.4. However, four variables are changed: (1) *error rating* is a utility node represented by U_2 ; (2) *dependency factor* is also a utility node (U_3); (3) both (1) and (2) converge as a third ALU utility node U_1 ; and (4) the variable *critical factor* is changed into a decision node with two states: *true* and *false* represented by D_2 . The variable *current state* is a decision node, similar to the BDN-HAC-1 model. The resource state is represented by the ALU node U_1 , which outputs the prediction. U_2 uses a scale between 1 and 20 while U_3 uses a scale between 2 and 14. U_1 uses a scale between 0 and 400 where 200 is used as a boundary to separate the two states.

Table 6.6 Description of the BDN-HAC-2 model

Node	Node Name	Node Type	State	Relative Weights
U_2	Error rating	Utility	$\{1 < U_2 < 20\}$	$\{\approx 0.15\}$
A_2	Failure repetition	Chance	$\{\text{low, high}\}$	$\{\approx 0.1\}$
A_3	Redundancy factor	Chance	$\{\text{low, high}\}$	$\{\approx 0.4\}$
A_4	Aggregated failure count	Chance	$\{\text{low, high}\}$	$\{\approx 0.1\}$
A_5	reinitialisation factor	Chance	$\{\text{true, false}\}$	$\{\approx 0.4\}$
U_3	Dependency factor	Utility	$\{2 < U_3 < 14\}$	$\{\approx 0.15\}$
B_2	Dependency type	Chance	local,shared, global	$\{\approx 0.3\}$
B_3	Dependency levels down	Chance	$\{\text{low, high}\}$	$\{\approx 0.4\}$
B_4	Dependency levels up	Chance	$\{\text{low, high}\}$	$\{\approx 0.3\}$
D_2	Critical factor	Decision	$\{\text{true, false}\}$	$\{\approx 0.35\}$
D_1	Current state	Decision	$\{\text{online, offline}\}$	$\{\approx 0.35\}$
U_1	Resource state	Utility (ALU)	$\{0 < U_1 < 400\}$	Target node

The relative weights were calculated using eq. (6.1), and we followed the same calculation method as described in Section 6.6.1. However, eq. (6.6) was not used because the BDN-HAC-2 model does not deal with negative numbers. Further, in Section 6.7.2, we describe how the utility values are calculated and provide examples in Section 6.7.4.

6.7.2 Transformation into the Bayesian Decision Network

Like the BDN-HAC-1 model, the BDN-HAC-2 model also consists of twelve nodes across three layers (Figure 6.5). The chance nodes representing most of the HAC properties are in layer three, but (unlike in the BDN-HAC-1 model) they converge as utility nodes in this model. The weight assignment and dimensionality reduction are performed using a measure of preference over the parent nodes. All the nodes related to error (set E) are consolidated in U_2 , and all dependency-related nodes (set D) are consolidated in U_3 . Hence, the first step in dimensionality reduction and weight assignment occurs in layer two and is performed by the two utility nodes. Furthermore, the four nodes in layer two, U_2 , U_3 , D_1 and D_2 , are consolidated in U_1 . U_1 is responsible for dimensionality reduction and the assignment of weights in step 2. The outcome is interpreted as a prediction of failure of the resource. This model does not have latent nodes because it takes a different decision path than the BDN-HAC-1 model.

Following the two-step approach defined in Section 6.5, and using equations (6.2), (6.3) and (6.4), we present the following equations to represent the model.

First, given any combination of values $A_{2,0}$, $A_{3,0}$, $A_{4,0}$ and $A_{5,0}$ for nodes A_2 to A_5 , the preference that the *error rating* utility node U_2 has a specific value $U_{2,0}$ is given by

Predicting Locally Manageable Resource Failures

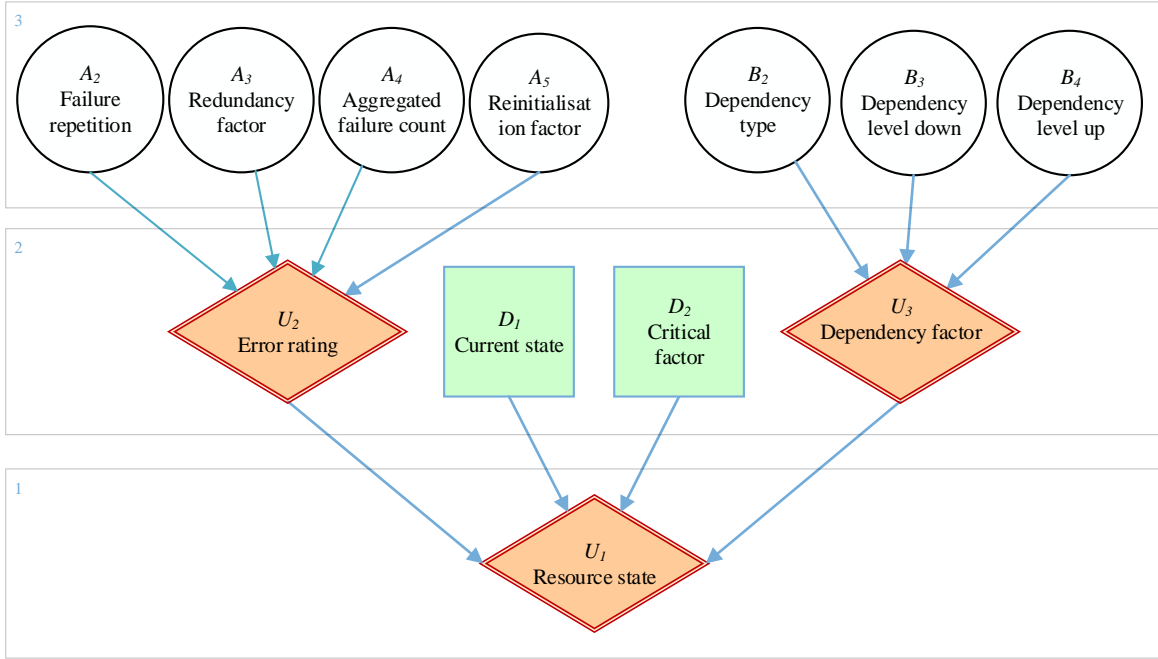


Figure 6.5 The Bayesian decision network model, BDN-HAC-2, showing the nodes and the related edges. White represents random nodes, green indicates decision nodes, and utility nodes are represented by red.

$$U(U_2 = U_{2,0} | A_2 = A_{2,0}, A_3 = A_{3,0}, A_4 = A_{4,0}, A_5 = A_{5,0}) = \sum_{v \in \{A_2, A_3, A_4, A_5\}} [w(v)U(U_2 = U_{2,0} | v = v_0)], \quad (6.14)$$

where $U(v)$ denotes a utility function, and $w(v) \in (0, 1]$ is a weight associated with the variable $v \in \{A_2, A_3, A_4, A_5\}$ such that

$$\sum_{v \in \{A_2, A_3, A_4, A_5\}} w(v) = 1.$$

Similarly, given any combination of values $B_{2,0}$, $B_{3,0}$ and $B_{4,0}$ for nodes B_2 , B_3 and B_4 , respectively, the preference that the *dependency factor* utility node U_3 has a specific value $U_{3,0}$ is given by

$$U(U_3 = U_{3,0} | B_2 = B_{2,0}, B_3 = B_{3,0}, B_4 = B_{4,0}) = \sum_{v \in \{B_2, B_3, B_4\}} [w(v)U(U_3 = U_{3,0} | v = v_0)], \quad (6.15)$$

where $U(v)$ denotes a utility function, and $w(v) \in (0, 1]$ is a weight associated with the variable $v \in \{B_2, B_3, B_4\}$ such that

$$\sum_{v \in \{B_2, B_3, B_4\}} w(v) = 1.$$

In a second step, the outcomes of equations (6.14) and (6.15) are combined with the nodes D_1 and D_2 . For any combination of values $U_{2,0}$, $U_{3,0}$, $D_{1,0}$ and $D_{2,0}$ of nodes U_2 , U_3 , D_1 and D_2 , respectively, the preference that the ALU node *resource state* $U_1 \in [0,400]$ has a specific value $U_{1,0}$ is given by

$$U(U_1 = U_{1,0} \mid U_2 = U_{2,0}, U_3 = U_{3,0}, D_1 = D_{1,0}, D_2 = D_{2,0}) = \sum_{v \in \{U_2, U_3, D_1, D_2\}} [w(v)U(U_1 = U_{1,0} \mid v = v_0)], \quad (6.16)$$

where $U(v)$ denotes a utility function, and $w(v) \in (0,1]$ is a weight associated with the variable $v \in \{U_2, U_3, D_1, D_2\}$ such that

$$\sum_{v \in \{U_2, U_3, D_1, D_2\}} w(v) = 1.$$

The outcome of the utility node U_1 (and thus of our BDN-HAC-2 model) is interpreted as

$$P(\text{locally_manageable_resource_failure}) = \begin{cases} \text{low}, & \text{if } U_1 < 200 \\ \text{high}, & \text{otherwise} \end{cases} \quad (6.17)$$

In this model, a utility value of 200 functions as the cutoff value.

6.7.3 Conditional Probability Tables

There are seven CPTs, three utility tables and two decision tables in the BDN-HAC-2 model. Table 6.7 lists the probability distributions for each state. The probability distributions for nodes A_2 - A_5 and B_2 - B_4 are the same as those described in Section 6.6.3. However, compared to the BDN-HAC-1 model, this model does not have chance nodes as the target (consolidation target) but has utility nodes instead.

6.7.4 Model Inference Example

Comparable to the BDN-HAC-1 model, this model also processes the outcome in two steps. However, there are differences in the network structure and in the way in which inference takes place. The model is more inclined towards a deterministic estimation due to the introduction of two additional utility nodes, which profoundly affects inference. We provide an example model inference, although the example does not illustrate the actual model inference because the model computation takes place at runtime and considers all nodes and dependencies. We instantiate the nodes in layer three ($A_2, A_3, A_4, A_5, B_2, B_3$ and B_4) as follows:

- A_2 is set to *low* to indicate that the failure repetition has a negligible effect on the target node U_2 .
- A_3 is instantiated to *true* to indicate that the resource has self-healing capability, which reduces the likelihood of failure.

Predicting Locally Manageable Resource Failures

Table 6.7 Probability distributions associated with the nodes in the BDN-HAC-2 model

Node	Probability Distribution
U_2	Utility node
A_2	low =.75 high =.25
A_3	true =.3 false=.7
A_4	low=.9 high=.1
A_5	true =.75 false=.25
U_3	Utility node
B_2	local=.8 shared=.15 global=.05
B_3	low =.5 high =.5
B_4	low =.3 high =.7
D_2	true false
D_1	online offline
U_1	utility node

- A_4 is set to *low*, implying that the potential effect by the aggregated failure count is negligible.
- A_5 is instantiated to *true*, which shows that the resource can be reinitialised, reducing the likelihood of failure.

Suppose the four weights from eq. (6.14) are $w(A_2) = 0.1$, $w(A_3) = 0.4$, $w(A_4) = 0.1$ and $w(A_5) = 0.4$, we obtain

$$\begin{aligned}
 U(U_2 = U_{2,0} \mid A_2 = low, A_3 = true, A_4 = low, A_5 = true) &= \\
 &= 0.1 \cdot U(U_2 = U_{2,0} \mid A_2 = low) + 0.4 \cdot U(U_2 = U_{2,0} \mid A_3 = true) + \\
 &\quad + 0.1 \cdot U(U_2 = U_{2,0} \mid A_4 = low) + 0.4 \cdot U(U_2 = U_{2,0} \mid A_5 = true) = \\
 &= \approx 20
 \end{aligned} \tag{6.18}$$

where $U_2 \in [0, 20]$ represents a target node, and $w(v) \in (0, 1]$ is a weight associated with the variable $v \in A_2, A_3, A_4, A_5$ such that $\sum_{v \in A_2, A_3, A_4, A_5} w(v) = 1$. The utility outcome is estimated as a numerical value of 20, indicating a high probability that the resource will not fail.

Similarly, we instantiate the nodes B_2, B_3 and B_4 as follows:

- B_2 is set to *local* to indicate that the resource is a local resource and that the failure of the resource has a low effect on the target node U_3 .
- B_3 is instantiated to *low* to indicate that number of lower-level resources are low within the hierarchy, which increases the likelihood of managing failure locally.
- B_4 is instantiated to *low* to indicate that the number of upper-level resources is low within the hierarchy, which reduces the likelihood of failure and increases the likelihood of managing the failure locally.

We assume the three weights from eq. (6.15) are $w(v) \in (0, 1]$, $w(B_3) = 0.4$ and $w(B_4) = 0.3$, and we obtain

$$\begin{aligned}
 U(U_3 = U_{3,0} \mid B_2 = \text{local}, B_3 = \text{low}, B_4 = \text{low}) &= \\
 &= 0.3 \cdot U(U_3 = U_{3,0} \mid B_2 = \text{low}) + 0.4 \cdot U(U_3 = U_{3,0} \mid B_3 = \text{true}) + \\
 &\quad + 0.3 \cdot U(U_3 = U_{3,0} \mid B_4 = \text{low}) = \\
 &= \approx 14
 \end{aligned} \tag{6.19}$$

where $U_3 \in [0, 14]$ represents a target node, and $w(v) \in (0, 1]$ is a weight associated with the variable $v \in B_2, B_3, B_4$ such that $\sum_{v \in B_2, B_3, B_4} w(v) = 1$. The outcome, a numerical value of 14, indicates a high probability that the combined outcome of U_3 has a low affect on the failure of the resource.

In the second step, the outcome of the layer two utility nodes and the other two nodes are consolidated into the ALU node. These nodes are instantiated as follows:

- D_1 gets the value *offline*, implying that the resource is offline.
- D_2 is instantiated to *true*, which indicates that the resource is a critical resource that increases the likelihood of failure and reduces the likelihood of managing the failure locally.
- U_2 has the value 20 to indicate a low probability of failure.
- U_3 has the value of 14 to indicate a low probability of failure caused by dependencies.

Considering the the four weights from eq. (6.16) are $w(D_2) = 0.35$, $w(D_1) = 0.35$, $w(U_2) = 0.15$ and $w(U_3) = 0.15$, we obtain

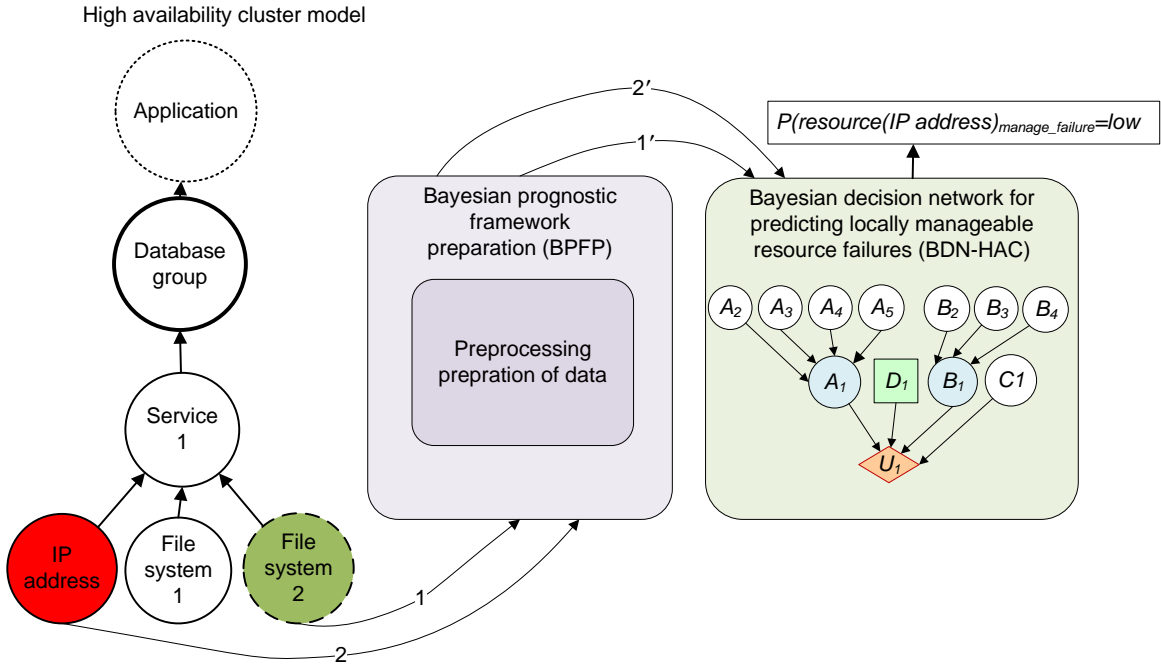


Figure 6.6 An illustrative example of the BDN-HAC model inference for the running example.

$$\begin{aligned}
 U(U_1 = U_{1,0} \mid D_2 = true, D_1 = offline, U_2 = 20, U_3 = 14) &= \\
 &\approx 0.35 \cdot U(U_1 = U_{1,0} \mid D_2 = true) + \approx 0.35 \cdot U(U_1 = U_{1,0} \mid D_1 = offline) + \\
 &\quad + \approx 0.15 \cdot U(U_1 = U_{1,0} \mid U_2 = 20) + \approx 0.15 \cdot U(U_1 = U_{1,0} \mid U_3 = 14) = \\
 &\approx 270
 \end{aligned} \tag{6.20}$$

where $U_1 \in [0, 400]$ represents a target node, and $w(v) \in (0, 1]$ is a weight associated with the variable $v \in D_2, D_1, U_2, U_3$ such that $\sum_{v \in D_2, D_1, U_2, U_3} w(v) = 1$.

Considering eq. (6.17), the outcome indicates that the resource failure can be managed locally. The prediction is identical to that of the BDN-HAC-1 model.

6.8 Causality and Decision Network

The inference of our BDN-HAC model (in both its variants) assumes that one or more nodes are instantiated. The decision node is usually set to *offline* because “failure” is a keyword used to extract failure events from log data in the BPFPP module (i.e., the BDN-HAC model is used when a resource failure is detected). Therefore, the model is set to consider an automated decision in this case. Nodes that are not instantiated rely on both prior and posterior probability distributions. Figure 6.6 illustrates the model inference. We use the running example and the related HAC setup (described in Chapter 5) to present the complete flow, and subsequently to describe the reasoning. To simplify the illustration,

Table 6.8 The impact of incomplete data on the prediction outcomes for model BDN-HAC-1

No	A_2	A_3	A_4	A_5	B_2	B_3	B_4	C_1	D_1	U_1
1	low	true	low	true	local	low	low	true	offline	54.62
2	NI	true	low	true	local	NI	NI	true	offline	31.94
3	low	NI	low	NI	local	low	low	NI	offline	18.58
4	low	NI	low	NI	local	low	low	NI	online	87.4

NI - not instantiated

we use only one of the model variants, BDN-HAC-1, to describe the flow. The reasoning is similar for BDN-HAC-2.

Example 5. Suppose that a first failure event occurred in the resource "File system 2". This event is depicted by the arrow labelled 1 in Figure 6.6 where the dotted circle around the failed resource indicates that it is a noncritical resource (in the HHAM of the HAC). When event 1 occurs in the resource, the failure information is extracted by the BFPF module, which adds all the values obtained from the runtime environment of the BP framework. This means data for all four property groups are added, which are then passed to the BDN-HAC model. Our BDN-HAC-1 model is initiated and considers the resource failure while incorporating the properties and their causal relationships in two steps. The error-related and dependency-related nodes are consolidated in the two latent nodes (shown as blue circles). In the second step, the outcomes from the latent nodes and the two nodes (C_1 and D_1) are consolidated into the utility node (U_1). The utility node U_1 outputs a value that shows a low failure probability that the resource related to 1' will fail, and hence the failure can be managed locally. Although the model considers all values, the deciding factor is that resolution group 3 promotes masking the failure for a noncritical failure.

Suppose now that a new failure event 2 occurs in resource 'IP address', which is shown as a red circle in Figure 6.6. Our BDN-HAC-1 model is initiated and considers the resource failure while also incorporating the properties and their causal relationships in two steps. Consequently, the utility node U_1 outputs a value that shows a high probability that the failure of the resource related to 2' cannot be managed locally.

6.8.1 Reasoning with Incomplete Data

Incomplete data influence the prediction outcome. If different subsets of nodes are instantiated for the same resource's failure, the outcome may differ each time. Further, if data are missing for those nodes that add significant weight to the outcome, it will also significantly impact the BDN-HAC utility calculation. It may even increase the likelihood of mispredicting whether a resource failure can be handled locally or not. Similarly, in some cases, nodes with lesser weights can take precedence over nodes with significant weights.

Example 6. To demonstrate these behaviours, four sets of data (including a complete set and three incomplete sets) are used to initiate our BDN-HAC-1 model, and the outcomes are listed in Table 6.8. Latent nodes (A_1 and B_1) are not listed because they do not receive any data. However, the conditional probability distributions of the latent nodes are updated automatically when the probability distributions of the parent nodes are updated. The weights, as presented in Section 6.6.1, are implicitly managed because they are part of the conditional probability construction. The sample data are constructed to emulate a nonfailure outcome and are performed for the same resource to provide a consistent view.

The first data set in Table 6.8 is fully instantiated, while the second data set has some incomplete data. The outcome shows that in both scenarios the BDN-HAC model yields an optimistic prediction that the resource failure will be locally manageable (since $U_1 > 0$, see eq. (6.10)). When some nodes are not instantiated, both prior and posterior (instantiated nodes) probability distributions are computed, impacting the outcome. There are also differences between nodes with significant weights and those with lesser weights. For example, two of the critical nodes (C_1 and A_5) are not instantiated in Data Set 3, and the prediction (i.e., the utility U_1) differs significantly compared to Data Sets 1 and 2.

The prediction for Data Set 3 is still optimistic, which means that the likelihood of the resource failure not being locally manageable is low. However, the outcome has a lower utility value than for the other scenarios. This is because A_5 has a favourable prior probability distribution, which assumes that the resource can be reinitialised and decreases the likelihood of failure. Hence, the prior probability distribution of node A_5 has precedence over the prior probability of node C_1 even though C_1 has a high weight factor. Suppose A_5 is instantiated with a *false* value. In that case, the prior probability of C_1 takes precedence because it favours the state *true* (most of the HAC resources are considered critical), increasing the likelihood of failure.

6.8.2 Influence of the Decision Node ‘Current State’

The decision node D_1 has a significant impact on the overall outcome. When none of the other nodes is instantiated, the default decision is *online*, and the expected utilities for the policies *online* and *offline* are obtained from the model as online 83.7 and offline 2.45, respectively. This means that the combined prior and posterior probability distributions, weights and preferences favour the decision policy *online*, and the decision outcome only changes when other nodes are considered.

Example 7. D_1 is set to *offline* in Table 6.8 for Data Sets 1–3, but when other nodes are considered, the final prediction in all cases is that the resource failure in question will be manageable locally. If the decision changes to *online* (Data Set 4 in Table 6.8), it changes the prediction significantly. The reason for this is that, when the decision is set to *online*, it adds significant weight to the overall outcome. Therefore, the prediction indicates that the failure of the resource is manageable locally.

6.9 Summary

This chapter presented the BDN-HAC model of the BP framework. The model's objective is to predict locally manageable failures at a HAC resource-level using property values corresponding to key characteristics as the BDN model inputs. Firstly, we identified four groups of characteristics that are representative of any HAC solution. We then mapped the corresponding properties onto variables and identified the relevant states related to each variable. We then introduced a method for reducing the dimensionality of these variables, and we added weights to influence the decision making process. Subsequently, we introduced two model variants, BDN-HAC-1 and BDN-HAC-2, with two distinct decision-making techniques and with the idea that the most suitable one is included in the BP framework. The BDN-HAC-1 model uses a standard utility function to add preferences to the outcomes. In contrast, the BDN-HAC-2 model uses ALU, enabling the addition of multiple utility nodes and connecting them to an ALU utility node. We also described how to transform the variables to construct the models. An example of inference was also provided for each model where the decision-making process was emulated using an automated decision-making process. Finally, we illustrated reasoning under uncertainty using one of the proposed models. We studied reasoning with incomplete data and provided several examples to emphasise that the prediction accuracy improves when more nodes are instantiated. Thus, this chapter answered research question RQ3 from Section 1.3.

Chapter 7

Bayesian Network for Failure Propagation and Prediction

This chapter presents the method for constructing a Bayesian network for the failure propagation and prediction (BN-HAC) model, representing the fourth module in the BP framework (Figure 1.1). The method is used to construct a BN-HAC model that represents the underlying HAC. Hence, when a HAC resource fails, the BDN-HAC model (Chapter 6) predicts whether the resource failure can be managed locally or not using the key HAC characteristics described in the previous chapter. The model outputs a numerical value as a prediction, and this value is interpreted as a binary value to indicate the state of the resource (failure manageable or unmanageable locally). Only unmanageable failures are input into the BN-HAC model, which performs inference in three steps. First, the model considers the corresponding node value while assessing the potential influence on other related resources (e.g., children and parents). Second, the model propagates the node-level failure along with the structure of the Bayesian network. Third, the model predicts the outcome for high-level nodes (e.g., the resource group or system). The outcome of the model indicates whether the HAC components modelled by these high-level nodes will fail or not. The relationships between resource failure and models BDN-HAC and BN-HAC are illustrated in Figure 7.1, which is an extension of, and reuses the notation from, the diagram in Figure 6.2. As in the previous diagram, the failure of a

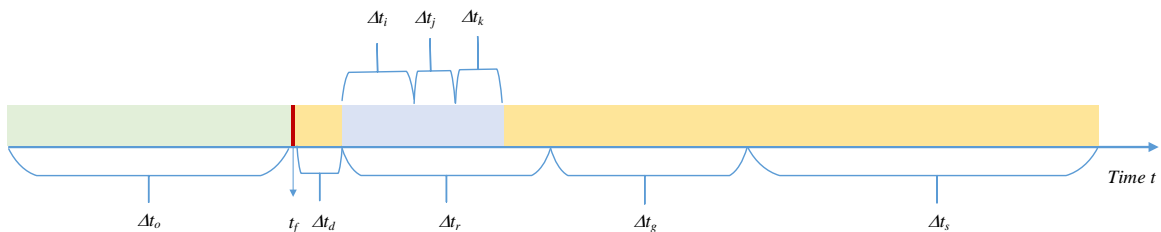


Figure 7.1 Timeline of resource failure events where HAC activities are shown in the lower level, and the activities of the Bayesian network model for failure propagation and prediction are displayed in the upper level.

HAC resource (indicated by its time of failure t_f) is followed by a period Δt_d required for the HAC to detect the problem and generate log entries about it and the BDN-HAC prediction time $\Delta t_i + \Delta t_j$. In addition, the new diagram shows Δt_k , the time required for the BN-HAC model to compute its prediction.

To propagate a resource-level failure and predict a resource group or system level failure in a HAC, the BN model must consider the complete structure of a HAC, including all the resources, resource types, and dependencies between them. Therefore, a BN model should directly represent a HAC, which requires a systematic method to map all HAC resources to a BN model. Several challenges must be addressed as part of constructing such a BN model because of the specific properties of a HAC. For example, when a resource fails, the event data related to only that failure are captured. Thus, only data for the node of the BN model that represents the failed resource are available for the analysis, which represents an extreme form of dealing with incomplete data. Another challenge is that any resource can fail at any time. This behaviour is interpreted as a random node failure in the BN model, and each time a different node is initiated.

Furthermore, some HAC resources do not have a physical representation but are only logical, and examples include the main application and resource groups. Further challenges arise from considering the criticality of a resource. For example, a resource must be part of the HAC, although such a resource failure does not present an immediate concern for the entire HAC. However, a HAC may still treat it as any other resource. Thus, it becomes subject to failure propagation and failover if the HAC does not handle the failure.

To address these challenges, we introduce a general approach for constructing a BN model from an underlying HAC using the outcomes of our HMTHA method (Chapter 5) and the configuration details of the HAC. This approach involves identifying and transforming the structure and related variables of the HHAM model built by the HMTHA method in order to construct a BN model. The approach enables modelling each resource within a HAC as a node in the BN model, with the logical resources of the HAC modelled as the latent nodes of the BN model. Moreover, a set of *transformation rules* (T-rules) is used to simplify the prior probability distribution assignments to each BN node. The dependencies in a HAC are encoded through conditional dependencies in the BN model. Furthermore, we propose the concept of a *weak node* for modelling a noncritical resource. The challenges of incomplete data are dealt with by substituting the missing data and employing the EM algorithm presented in Section 4.3.3.2.

Moreover, although prior probability distributions are initially assigned, parameter learning is used to learn and update the distributions, ensuring that the prior probability distributions are updated to reflect the actual HAC situation. However, the latent nodes are excluded in the learning process to ensure that only the probabilities of the representative nodes are updated. The propagation of the failure along the network is performed by updating the posterior probability when evidence is observed. This allows the high-level nodes to use posterior probabilities to provide predictions. Thus, the model presented in this chapter addresses research question RQ4 from Chapter 1, i.e., developing a probabilistic model to propagate the node-level failure to provide prediction at high-level nodes.

Table 7.1 BN-HAC model construction steps and step components

No	Component	BN-HAC Model Considerations
Step 1: Establish the network structure		
1	Method	The method "learning from other models" is used, and the other model is the HHAM model and the accompanying M-table (Section 7.2). States of the nodes are identified using the state of the underlying HAC resources. Therefore, only two states were considered (Section 7.2.3).
2	Algorithm	N/A
3	Training data	N/A
Step 2: Establish the probability distributions		
4	Method	Combination of two approaches: prior probability distribution is established using the availability SLA and FMEA (Section 5.3.4) —T-rules aid in setting conditional probabilities (Sections 7.2.5,7.2.6), and parameter learning (Section 7.3.3) to update the distributions incrementally.
5	Algorithm	EM
6	Training data	Data sets containing only unmanaged failures, output by the BDN-HAC model
7	Production data	Same as the training data but using a different group of failures.
8	Mode	Incremental
Step 3: Inference		
9	Algorithm	Clustering
10	Data	Only unmanageable failure information output by the BDN-HAC model (Section 7.3.5).

N/A - Not applicable

To carry out all the activities associated with the construction of a BN (Section 4.5), this chapter covers these activities as summarised in Table 7.1. We discuss related work in Section 7.1. Section 7.2 presents our general approach to constructing a BN model using the outcome from the holistic modelling technique for high availability (HHAM and M-table) and the configuration data from the HAC. The probability assignment is described in Section 7.2.6, and the related T-rules are described in Section 7.2.5. Section 7.3 elaborates on inference under a range of conditions and how these conditions are handled (e.g., latent variables, incomplete data, and joint probability distributions). Section 7.3.3 describes the model parameter learning approach, and Section 7.3.5 presents inference using production data. Section 7.3.6 provides a discussion on reasoning with the model using the causal relationship between the nodes. Finally, Section 7.4 summarises the chapter.

7.1 Related Work

Evaluating the impact of an individual resource failure within a system with many interconnected components needs to be done at the resource group or system level. Here, we investigate related work that addresses this from a BN perspective. Additionally, the focus of the related work summarised in this section is on 1) failure evaluation and detection at the component level and 2) failure propagation and prediction at the system level.

The use of BNs to predict failures of individual components is widely studied. For example, Chaves et al. [38] propose a BN-based method to predict failure in hard disk drives (HDD). The method uses the deterioration over the time of an HDD to predict failure. The method provides better results than a baseline model.

The characteristics of interconnecting multiple nodes in BNs are useful to present systems comprising many components while encoding the dependencies between them. Thus, failure detection and prediction can be provided while considering all components in the BNs. Hence, BNs are used in a broad range of disciplines, including biology [76] and medicine [23, 208]. For example, Agrahari et al. [2] developed a BN model to predict types of haematological malignancies. The model has an accuracy of 93%, a precision of 98%, and a recall of 90% on the training data set. This result outperforms the results reported by other researchers on the same data set. Park et al. [208] proposed a BN model to predict post-stroke outcomes and the results showed that the model had high prediction accuracy.

BNs are often effective even for safety-critical systems. For instance, Baldoni et al. [13] tested the BN model that uses HMM to predict failures. The results reveal that failures could be accurately predicted within a few hundred seconds before the failure. Wang et al. [289] developed a BN model to predict weather-related failures in railway turnout systems, and the model performed well with high prediction accuracy. Similarly, Codetta-Raiteri et al. [46] present an approach for fault detection, identification, and recovery of autonomous spacecraft by employing a DBN method.

When a method exists for detecting or predicting failures at the component level in a system with many components, the next step is usually to propagate the failure so that its influence on high-level components (e.g., a system) can be modelled. Pitakrat et al. [215] proposed a hierarchical online failure prediction approach, called HORA, which combines failure prediction at a component level with a failure propagation model. BNs are used to model the failure propagation part of the HORA. The HORA approach is compared to a monolithic approach by its authors who show that HORA improves the area under the ROC curve by 9.9%. Bottone et al. [28] proposed a BN model to predict failures of a satellite earth terminal. The system consists of multiple components, and the BN model enables propagating probabilities and provides predictions at the system level.

To summarise, many studies on BN areas support failure evaluation, failure detection, failure propagation, and prediction. However, no such approach exists for HACs. Currently, no HAC products we surveyed (Section 3.2) employ BNs or any stochastic models. Therefore, to the best of our knowledge, the approach introduced in this chapter is the first to construct a model that

considers failures at the component level of the HAC and propagates this component failure to predict system-level failure.

7.2 Method for Constructing the BN Model

The HAC structure is different for each HAC implementation. All such components that make up the HAC must be represented in a BN model to enable failure propagation and subsequent prediction. Therefore, a systematic approach to creating a BN-HAC model is presented in this section. The approach enables creating the network structure of the BN model and assigning the conditional probabilities to it. The outcomes of the HMTHA method (described in Chapter 5) are used as the primary sources for performing the transformation. In addition, the HAC configuration data are used as a reference to ensure that the BN model reflects the HAC structure.

7.2.1 Transformation Methodology

The three outcomes of the HMTHA are used to transform the HAC to a BN model. The two outcomes (HHAM and M-table) are used directly in the transformation process, whereas the T-rules are explicitly designed for the target BN model to simplify assigning probabilities. The outcomes of HMTHA and their roles in the transformation are presented in this section.

1. **HHAM**

Analyse the HHAM model to identify the node types, edge types, dependencies, and layers.

2. **M-table**

Perform the six-step mapping approach (Table 5.2) to complete the M-table to map between the HMTHA components and BN components.

3. **T-rules**

Create a set of T-rules specifically for the BN model using the guidelines described in Section 5.3.4.

7.2.2 Transformation from HMTHA to BN-HAC

We use the running example to explain the transformation steps. The first step is to complete the M-table mapping to decide how vertices, vertex types, and layers will be represented in the BN model. Hence, as described in Section 5.3.3, the six-step mapping approach is used to map and complete the M-table.

Example 8. The first part of the M-table, which we presented in Table 5.4, is completed with a second part, as shown in Table 7.2. This table shows how the logical resources are mapped to latent nodes and how the layers between the models are mapped. In the second step, both the M-table and HHAM are used to construct the BN model structure.

Bayesian Network for Failure Propagation and Prediction

Table 7.2 Completed M-table

HHAM				BN		
Resource	Vertex Type	Edge Type	Layer	BN Node	BN Node Type	Layer
Application	Application	Application	-	C_1	Latent	1
Database	Resource group	Resource group	a	A_1	Latent	2
Service 1	Simple	Resource	b	A_2	Node	3
IP address	Simple	Resource	c	A_3	Node	4
File system 1	Simple	Resource	c	A_4	Node	4
File system 2	Weak	Resource	c	A_5	Node	4

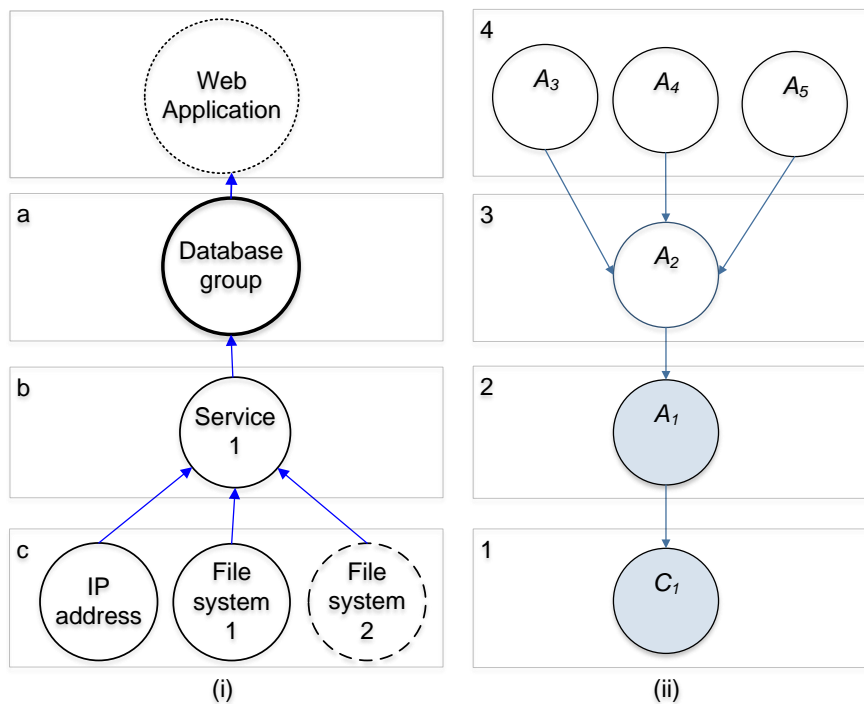


Figure 7.2 Mapping the HHAM to the BN-HAC model: i) HHAM model; ii) mapped BN-HAC model. In the BN-HAC model, the nodes shaded in blue are latent nodes; all the other nodes are random nodes.

Example 9. The direct mapping between the HHAM and the BN-HAC model is depicted in Figure 7.2. Figure 7.2 (i) displays the HHAM, which was first presented in Section 5.4. The layers of the HHAM are mapped in reverse order, and six resources from the HMTA are represented in the equal number of nodes in the BN model. The logical resources *application* and *resource group* are created as latent nodes (shaded in blue). The accompanying T-rules are described in Section 7.2.5.

7.2.3 Variables and State Definition

The variables in a BN-HAC model represent the HAC resources and these are obtained from the M-table (Table 7.2). The column ‘resource’ in the table lists all the HAC resources, and the column ‘BN node’ lists the related BN nodes. All variables are treated as discrete random variables with two states: 1) failure and 2) no failure. Keeping the same type and state ensures a smooth mapping of HAC resources to the BN-HAC model.

7.2.4 Incorporating the Weak Node Concept

The weak node concept is realised by assigning conditional probabilities to child nodes. However, if a weak node experiences repeated failure, the posterior probabilities are updated to reflect this. The implication is that the model may promote the ‘failure’ state while considering historical failures.

Example 10. Node A_5 is identified as a weak node in the running example (Figure 7.2 (ii)). Hence, the conditional probability in child node A_2 is updated according to the T-rule number 4 . This condition reflects that, even in the event of the failure of A_5 , A_2 will not promote the ‘failure’ state, and the posterior probabilities of the network are updated accordingly.

7.2.5 T-rules

There are two steps associated with updating the probability distributions in a BN-HAC model: (i) assigning the distributions initially; and (ii) updating the distributions using a learning approach (Section 4.3.4). T-rules are used in the first step and only aid in assigning the prior probability distributions. The second step is to apply parameter learning, which updates the parameter distributions using data from the HAC environment.

A basic set of T-rules is provided in Table 5.3. These T-rules are applicable when constructing BN models. The SLA of the application may play an important role when interpreting the T-rules to assign probabilities because HACs are often deployed to ensure that SLA requirements are met.

Example 11. Consider an SLA of 99.9% availability, which represents a monthly downtime of 43.2 minutes for the running example, and which can be translated and represented in prior probabilities. Hence, the values are based on the business requirements. However, these values are later updated using parameter learning, which reflects the current situation in a HAC solution. Therefore, nodes A_3 , A_4 and A_5 can be assigned a probability distribution of 0.1% for the state failure and 99.9% for no failure. Additionally, conditional probabilities are set in the child node A_2 to reflect the same values except for the weak node. In that case, the conditional probability distribution is set to indicate the noncritical characteristic of the node.

7.2.6 Assigning Prior Probabilities

Table 7.3 Prior probability distributions of the BN-HAC model

Node	Probability Distribution
C_1	$P(C_1 A_1)$
A_1	$P(A_1 A_2)$
A_2	$P(A_2 A_3, A_4, A_5)$
A_3	Failure = .001; no failure = .999
A_4	Failure = .001; no failure = .999
A_5	Failure = .001; no failure = .999

We used the T-rules and three approaches from Section 5.3.4 to calculate and assign probabilities. For example, all low-level parent nodes (Layer 4 in the BN model) with local probabilities are assigned a uniform probability distribution (i.e., using SLA). The child nodes in Layer 3 and the latent nodes in Layer 2 were assigned conditional probabilities. Table 7.3 lists the prior probability distributions for the model from the running example, and the example below describes how these probabilities were calculated and assigned.

Example 12. First, we used Approach 1 (eqs. (5.2) and (5.3)) to assign uniform probabilities to nodes A_3 , A_4 and A_5 . The probabilities were calculated directly from the availability SLA of the application as Failure = .001 and No failure = .999. We used Approach 2 for nodes A_2 and A_1 . FMEA and eq. (5.5) produced a low value for node A_5 (weak node), which was then assigned as a conditional probability to node A_2 , as Failure = .03 and No failure = .97. Thus, if the parent node A_5 fails, it does not result in the failure of the child node A_2 . However, the parent nodes A_3 and A_4 have a high impact factor, which means if any of them fail, it results in the failure of A_2 , and the conditional probability was assigned as Failure = .999 and No failure = .001. Similarly, the failure of the parent node A_2 results in the failure of the latent node A_1 , and the conditional probability was assigned accordingly. We used eq. (5.6) to calculate the conditional probability for the main application C_1 . The application has only one resource group; therefore, we did not use eq. (5.7). The outcome of the calculation is that if the parent node A_1 fails, the application node also fails. Table 7.3 lists the prior probability distributions for the model from the running example.

While the prior probability distributions are set initially, as a one-time activity, the primary focus is on parameter learning to ensure that the model distributions are learned from the actual data (described in Section 7.3.3).

7.3 Failure Propagation and Prediction

The model inference consists of three steps. The first step considers the failure at a node level, and the second step propagates the failure to upper levels. The third step predicts the potential failure at high-level nodes.

Table 7.4 Data set with incomplete data

No	C_1	A_1	A_2	A_3	A_4	A_5
1	Latent	Latent	Failure	?	?	?
2	Latent	Latent	$P(A_2 A_3, A_4, A_5)$	Failure	?	?
3	Latent	Latent	$P(A_2 A_3, A_4, A_5)$?	Failure	?
4	Latent	Latent	$P(A_2 A_3, A_4, A_5)$?	?	Failure
5	Latent	Latent	$P(A_2 A_3, A_4, A_5)$	Failure	?	Failure

Table 7.5 Data set with substituted values

No	C_1	A_1	A_2	A_3	A_4	A_5
1	Latent	Latent	Failure	No failure	No failure	No failure
2	Latent	Latent	$P(A_2 A_3, A_4, A_5)$	Failure	No failure	No failure
3	Latent	Latent	$P(A_2 A_3, A_4, A_5)$	No failure	Failure	No failure
4	Latent	Latent	$P(A_2 A_3, A_4, A_5)$	No failure	No failure	Failure
5	Latent	Latent	$P(A_2 A_3, A_4, A_5)$	Failure	No failure	Failure

Section 7.3.1 provides a brief discussion about working with latent nodes, and Section 7.3.2 describes working with incomplete data. Section 7.3.3 describes the model parameter learning approach, and Section 7.3.5 presents inference using production data. Section 7.3.4 describes the JPD, and Section 7.3.6 describes the causal reasoning with the model.

7.3.1 Working with Latent Nodes

Figure 7.2 (ii) presents the latent nodes in blue. These nodes are always unobserved when the model receives data. Hence, they do not participate in the parameter learning process but instead are associated with conditional probabilities. During inference, the conditional probabilities are updated due to the conditioned parent node influence. Based on this, the high-level latent nodes can provide a prediction. For example, $P(C_1|A_3)$ predicts the outcome of node C_1 (latent) given the failure of node A_3 (Figure 7.2) (ii).

7.3.2 Working with Incomplete Data

Incomplete data are expected from the HAC because the related BN model usually receives evidence for a single resource (node) failure. More than one resource may also fail, but this can be considered a rarity because the HAC tries to resolve individual failures continuously.

In Figure 7.3, we assume that node A_3 is failed, which means only data about that node are captured and passed to the model. All other nodes do not receive any data. Five examples of data sets for the model are presented in the rows of Table 7.4 showing that only one node receives data (Records 1 to 4), whereas Record 5 indicates a two-node failure, and thus two nodes receive data in

Bayesian Network for Failure Propagation and Prediction

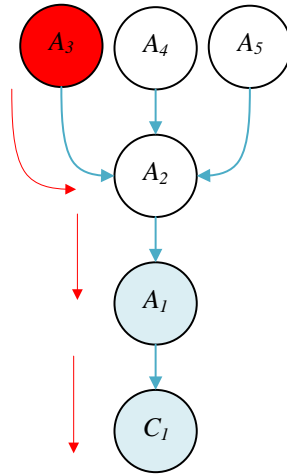


Figure 7.3 Failure propagation in the BN-HAC model when node A_3 fails. Red indicates a failed node, white represents running random nodes and blue indicates latent nodes.

Table 7.6 Three data sets used during parameter learning experimentation

	#Records	Failures (A_3)	Failures (A_4)	Failures (A_5)
Data set 1	8	3	3	2
Data set 2	16	3	3	2
Data set 3	340	15	21	12

this last case. We perceive this as only one or two resources failing, and all other resources continue to run. Therefore, we propose substituting the incomplete data with a running state (no failure) before the inference. Table 7.5 displays the same data set that has undergone substitution.

7.3.3 Parameter Learning

The model requires up-to-date probability distributions to provide accurate predictions. An updated probability distribution considers the historical failures of related nodes, which improves the overall prediction accuracy. Therefore, the model employs a parameter learning approach to learn distributions from data. However, the BDN-HAC model, which supplies data to the BN-HAC model, only delivers data related to failure events. Thus, learning the distributions only from failures would result in these distributions being updated incorrectly (i.e., failure state is preferred for the failed nodes). Moreover, to capture the resolution features of HAC, the learning should be reset at regular intervals to reflect the failure distributions correctly. To address these, we experimented with three data sets, and Table 7.6 describes the records and failure distribution of the parent nodes.

The first data set contains only failure events (standard output from the BDN-HAC model), the second data set has 50% failures, and the third data set has only 14% failures. We employ parameter learning to learn from the three data sets independently, and the results are presented for each node in Figure 7.4. As shown by these results, the probability distributions differ considerably. Data Set 1

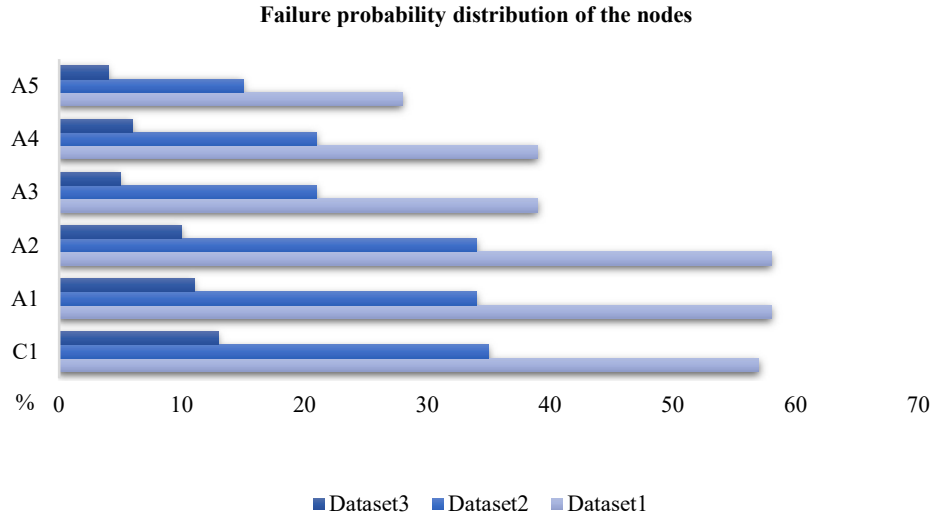


Figure 7.4 Node failure probability distribution after parameter learning using the three data sets.

Table 7.7 Prediction that node C_1 will fail (shown as a percentage) when each of nodes A_3 , A_4 and A_5 fails (individually) after parameter learning using the three data sets from Table 7.6

	A_3 fails (%)	A_4 fails (%)	A_5 fails (%)
Data set 1	91	90	59
Data set 2	89	88	36
Data set 3	87	87	14

exhibits high values for failure probability distributions, which significantly increases the probability of predicting failures. The second data set also results in a high value, whereas the third data set has significantly reduced the distributions. To observe the prediction in node C_1 , we perform inference after parameter learning by providing evidence to nodes A_3 , A_4 and A_5 . Table 7.7 presents the prediction results in C_1 when nodes A_3 , A_4 and A_5 fail independently.

The first data set provides an incorrect prediction when node A_5 fails by predicting that the application node C_1 will also fail (59% probability) when A_5 fails. However, because A_5 is a weak node, the failure of such a node should not result in application failure. Data Set 2 predicts this correctly, but the probability value is still high. Further, Data Set 3 provides predictions with high accuracy. Therefore, the conclusion is that the data set for parameter learning should be based on the time series to represent the failure distributions correctly.

However, because the baseline data sets are outputs of the BDN-HAC model, we propose augmenting the data sets. The augmentation technique is to increase the number of records to reflect the behaviour of the HAC. For example, nonfailure events are added at regular intervals (as illustrated in Data Set 3). Furthermore, the parameter learning process should be initiated regularly to correctly capture the HAC failure pattern while excluding the latent nodes. Consequently, we propose initiating

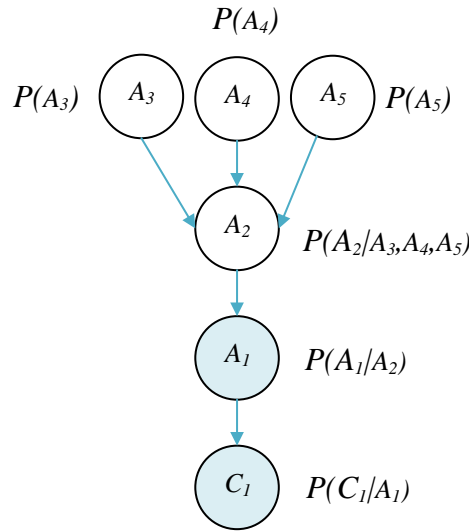


Figure 7.5 Joint probability distribution of the model for the running example.

the process daily to update the distributions while resetting the previous distributions. This approach improves the accuracy for identifying the failure behaviour of the individual nodes of the model.

Moreover, parameter learning should be initiated when structural changes to the HAC occur, which results in changes to the BN model. An example of a structural change is adding a new resource to the HAC. The proposed model can use different algorithms to achieve parameter learning. However, the proposed algorithm is EM to facilitate the maximum likelihood estimation.

7.3.4 Failure Propagation

The model uses JPDs to calculate the conditional probabilities of the nodes from the corresponding CPT tables. Furthermore, the JPDs are computed at several levels, such as the node representing a resource group and the main application.

The calculation of the JPD for the model can be performed as follows

$$P(A_5, \dots, C_1) = P(A_3)P(A_4)P(A_5)P(A_2|A_3, A_4, A_5)P(A_1|A_2)P(C_1|A_1). \quad (7.1)$$

The JPD of the model is presented in Figure 7.4. The local probability distributions are presented for nodes A_3 , A_4 and A_5 , whereas the conditional probability distributions are associated with nodes C_1 , A_1 and A_2 . Both A_1 and C_1 are latent nodes, which implies that the JPD computation in these nodes provides predictions.

Example 13. $P(C_1|A_1)$ provides the probability of a system (C_1) failure given the resource group (A_1) failure. Similarly, $P(A_1|A_2)$ provides the probability of the resource group (A_1) failure given the service failure (A_2). Hence, $P(C_1|A_3)$ provides the probability of a system (C_1) failure given a resource (A_3) failure.

Table 7.8 BD-HAC inference for the running example in five scenarios involving the individual failures of resources A_2 , A_3 , A_4 and A_5 (scenarios No 1–4), and the combined failure of A_3 and A_5 (scenario No 5)

No	C_1	A_1	A_2	A_3	A_4	A_5
1	Failure	Failure	Failure	No failure	Failure	No failure
2	Failure	Failure	Failure	Failure	No failure	No failure
3	Failure	Failure	Failure	No failure	Failure	No failure
4	No failure	No failure	No failure	No failure	No failure	Failure
5	Failure	Failure	Failure	Failure	No failure	Failure

7.3.5 Inference Using Production Data

The inference using production data is the last step in the BP framework, which occurs after parameter learning. Whenever a resource failure that cannot be managed locally occurs within the HAC, the BN-HAC model is initiated to perform the inference. A correlation exists between the parameter learning process and inference, as detailed in Section 7.3.3, and updating the distributions through parameter learning directly affects the prediction quality.

Example 14. The model inference and outcomes using Data Set 3 for five failure scenarios (Section 7.3.3) are presented in Table 7.8. Red indicates a failed node, whereas blue indicates latent nodes.

In the first scenario, node A_2 fails and the prediction for nodes A_1 and C_1 is that a resource group failure (A_1) and system failure (C_1) will occur. In contrast, the prediction is no failure for scenario A_4 because the probability of child node failure is estimated to be low.

7.3.6 Causal Reasoning with the BN-HAC Failure Propagation and Prediction Model

An illustrative example of the model inference is shown in Figure 7.6, which presents the complete flow using the running example. The figure is an extension of Figure 6.6, in which the BDN-HAC module first predicted that the failure event labelled 1' could be managed locally. Hence, the failure information was not passed further to the BN-HAC model. We then described the failure of the resource "IP address", which was predicted by the BDN-HAC to be unmanageable. Now, we continue with the same resource failure to describe the complete flow. The failure of the resource is marked in red in the HHAM of the HAC. Subsequently, the corresponding failure event 2 is captured by the BPF module, which enriches, transforms, converts and filters the data before passing the results to the BDN-HAC model. The event is depicted by the arrow labelled 2'. The BDN-HAC model predicts that the resource failure is unmanageable, then passed to the BN-HAC model as shown by the event 2''. As demonstrated in the figure, the BN-HAC model directly represents the HAC, and the failed resource is mapped to a corresponding node (A_3). The red node A_3 indicates that the resource failed, which means the network posterior probability is updated to reflect that. The failure is propagated to

Bayesian Network for Failure Propagation and Prediction

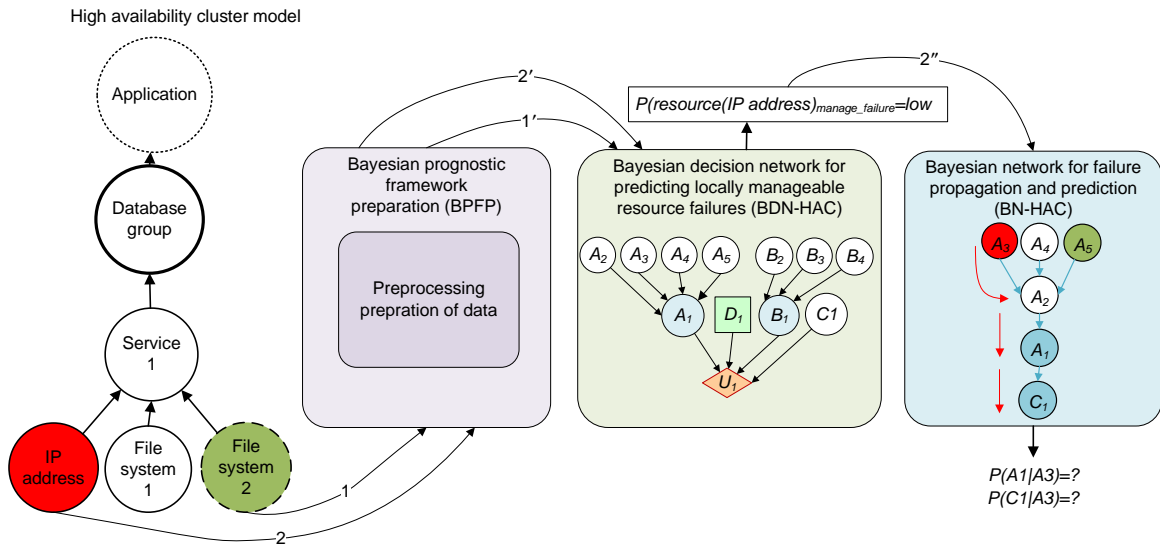


Figure 7.6 An illustrative example of the BN-HAC model inference for the running example.

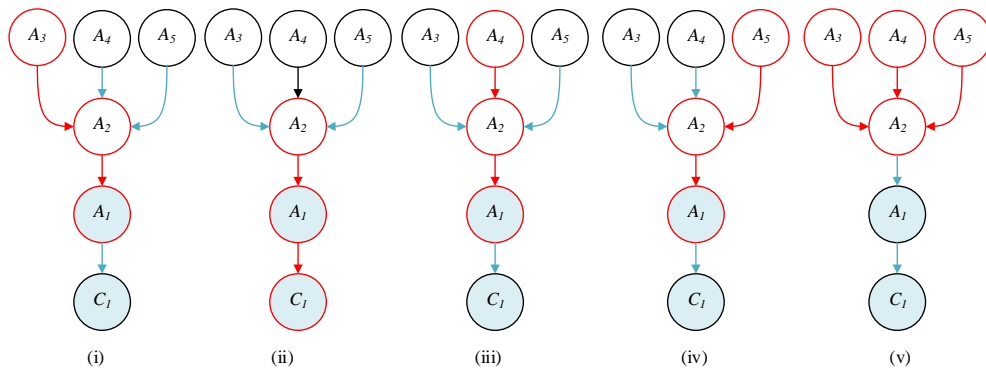


Figure 7.7 Identified causal connections in the BN-HAC model for the running example.

the node in the next level (indicated in Figure 7.3 with red arrows), A_2 , and then to the next levels, A_1 and C_1 , which are latent nodes. The outcome (prediction) of these nodes indicates whether the high-level nodes (C_1 and A_1) will fail or not. However, no direct influence exists between A_3 and A_1 and between A_3 and C_1 , but the influence occurs through A_2 . Similarly, the influence between A_3 and C_1 is routed through both A_2 and A_1 .

Two types of causal connections can be observed in the example BN-HAC model: serial and converging. Figure 7.7 (i, ii, iii and iv) identifies the four serial connections. In the first connection (Figure 7.7 (i)), the failure is propagated from the source (A_3) to A_1 through A_2 to provide predictions in A_1 . If the failure occurs in A_2 (A_2 is instantiated), the flow is blocked in A_2 , and A_3 becomes independent. In this case, the failure is propagated to the high-level node (A_1), which is denoted as follows

$$\begin{aligned}
 P(A_3, A_2, A_1) &= P(A_3) P(A_2 | A_3) P(A_1 | A_2) \\
 P(A_1 | A_3) &= \sum_{A_2} P(A_2 | A_3) P(A_1 | A_2).
 \end{aligned} \tag{7.2}$$

In the second serial connection, the failure is propagated from A_2 to C_1 through A_1 (Figure 7.7 (ii)), which is denoted as follows

$$\begin{aligned}
 P(A_2, A_1, C_1) &= P(A_2) P(A_1 | A_2) P(C_1 | A_1) \\
 P(C_1 | A_2) &= \sum_{A_1} P(A_1 | A_2) P(C_1 | A_1).
 \end{aligned} \tag{7.3}$$

In this case, the parent nodes (A_3 , A_4 , and A_5) become independent (orphans), which means they do not participate in failure propagation. However, to avoid creating situations with independent parents, the BDN-HAC model considers several crucial characteristics of HAC to ensure this, for example, only those that cannot be reinitialised are predicted to fail. The third serial connection (Figure 7.7 (iii)) indicates that a failure is propagated from A_4 to A_1 and is denoted as follows

$$\begin{aligned}
 P(A_4, A_2, A_1) &= P(A_4) P(A_2 | A_4) P(A_1 | A_2) \\
 P(A_1 | A_4) &= \sum_{A_2} P(A_2 | A_4) P(A_1 | A_2).
 \end{aligned} \tag{7.4}$$

The fourth serial connection describes the failure propagation from A_5 to A_1 (Figure 7.7 (iv)) and is denoted as follows

$$\begin{aligned}
 P(A_5, A_2, A_1) &= P(A_5) P(A_2 | A_5) P(A_1 | A_2) \\
 P(A_1 | A_5) &= \sum_{A_2} P(A_2 | A_5) P(A_1 | A_2).
 \end{aligned} \tag{7.5}$$

Both A_1 and C_1 are latent nodes, which suggests that if a failure occurs in A_2 , it is propagated to C_1 through A_1 . A latent node fails only when a critical parent node or a combination of parent nodes fail, and when the posterior probabilities are changed accordingly. The corresponding converging connection is represented by A_3 , A_4 , A_5 and A_2 (Figure 7.7 (v)), which is denoted as follows

$$P(A_2, A_3, A_4, A_5) = P(A_3) P(A_4) P(A_5) P(A_2 | A_3, A_4, A_5), \tag{7.6}$$

where A_4 is independent of A_3 , A_3 is independent of A_5 , and A_5 is independent of A_4 .

This indicates that the propagation between A_3 and A_4 is allowed when A_2 is initiated only. Although A_2 is not explicitly instantiated, it is regarded as initiated due to the conditional probabilities. Thus, when A_3 fails and the failure is propagated to A_2 , which also fails, the flow can go to A_4 (Figure 7.8). Moreover, the intercausal inference 'explaining away' can also be observed because all parents are conditioned on child nodes. When A_3 and A_4 fail, the first node (A_3) failure is propagated to A_2 , which implies the failure of A_4 is not considered. Moreover, the posterior probability of A_4 is reduced in child node A_2 , which can be interpreted as A_3 explaining away A_4 .

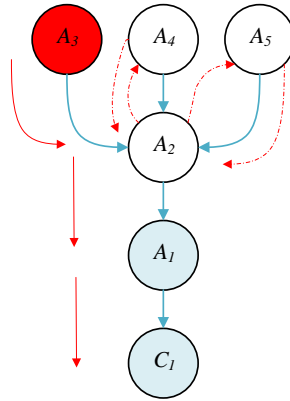


Figure 7.8 Failure propagation and the impact on the nodes in the same layer when node A_3 fails.

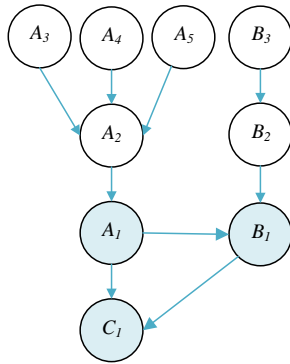


Figure 7.9 Diverging connection depicting nodes representing two resource groups. White represent random nodes and blue indicates latent nodes.

A diverging connection is not present in the BN model for the running example. However, it may exist in a complex BN-HAC model with numerous nodes. In such a case, a diverging connection may indicate the existence of a dependency between two nodes representing either resource groups, or between a shared resource and two or more simple resources. Consider a second resource group B_1 , as illustrated in Figure 7.9, such that A_1 forms a diverging connection with C_1 and B_1 . Thus, if A_1 fails (instantiated), B_1 and C_1 become conditionally independent, which can be denoted as follows

$$\begin{aligned}
 P(C_1, A_1, B_1) &= P(A_1)P(C_1|A_1)P(B_1|A_1) \\
 P(B_1|C_1) &= \sum_{A_1} P(A_1)P(C_1|A_1)P(B_1|A_1).
 \end{aligned}
 \tag{7.7}$$

However, A_1 is a latent node, and it may fail when the parent nodes fail based on the conditional probabilities. When A_1 fails, it influences both C_1 and B_1 while maintaining both C_1 and B_1 as conditionally independent. Nonetheless, if A_1 does not fail, but C_1 fails, it changes the failure probability of A_1 and B_1 . The serial connections allow the failure propagation from the child (C_1) to A_2 and then to B_2 if A_1 and B_1 are not observed. If the resource group A_1 fails, the HAC perspective

means that it is likely to cause C_1 and B_1 failures based on the conditional probabilities. Similarly, if the system (C_1) fails, it affects both resource groups (A_1 and B_1). Alternatively, if only B_1 fails, it influences system C_1 but may not result in a system failure, which means A_1 and C_1 continue to function. Furthermore, only B_1 and the parent resources are subject to a potential failover.

In conclusion, all three basic connections (serial, converging and diverging) can exist in a BN-HAC model and can influence how failure is propagated. However, a diverging connection is a rarity and can only be observed when establishing dependencies between two resource groups or when a shared dependency resource is present. A BN-HAC model representing the HAC with many resources and resource groups can have multiple basic connections.

7.4 Summary

This chapter presented the steps involved in building a BN model for failure propagation and prediction (BN-HAC) and answers research question RQ4. We first described the transformation steps to construct a BN-HAC model using the outcomes of the HMTHA (HHAM and the accompanying M-table and T-rules) and to assign prior probabilities. Then, we present an approach for parameter learning to update the probability distributions using log data (which is processed by the BDN-HAC model). Finally, a brief discussion is provided on the causal reasoning. Examples are provided on how a node failure is propagated to child nodes to predict high-level child nodes' potential failure.

Chapter 8

Bayesian Prognostic Framework Preparation

This chapter presents the BPF module, which represents the second module in the BP framework (Figure 1.1). The objective of the module is twofold: (1) to prepare an environment to deploy the BP framework and (2) to prepare log data for the BP framework models (BDN-HAC and BN-HAC). Hence, the former facilitates the latter, and this is shown in Figure 8.1. The environment's preparation is attributed to the following components: (1) configuration refinement, (2) log interface, (3) transformation and conversion and (4) filter. These components are supported by additional components: (5) database models and (6) database structures. Components 1 and 5 are one-off activities that do not need to be repeated once implemented. Thus, the module presented in this chapter addresses the research question RQ2 from Chapter 1, i.e., creating a module to prepare the environment and log data and to facilitate the deployment of the BP framework.

The chapter is organised as follows. Section 8.1 presents related work. Next, in Section 8.2, the database tables required to manage configuration and runtime data are described. Section 8.3 presents the configuration refinement component that is responsible for entering configuration data into the relevant tables. The log interface is introduced in Section 8.4. This component is responsible for extracting, parsing, enriching and updating log data. Subsequently, log data is transformed and

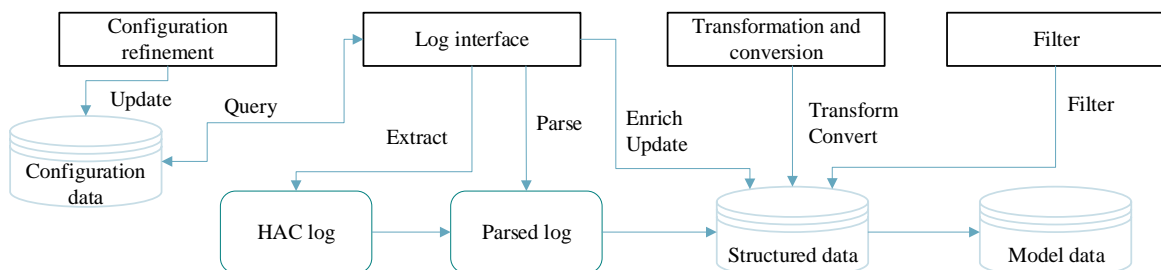


Figure 8.1 Components of the preparation environment are depicted, which show the log processing steps.

converted by the transformation and conversion component described in Section 8.5. Section 8.6 presents the database table for storing model data for the BDN-HAC model. Section 8.7 presents the filter applied to the log data before they are stored in the database table described in the previous section. Finally, Section 8.8 summaries this chapter.

8.1 Related Work

The preprocessing of data is an integral part of developing probabilistic models to ensure that the source data are prepared correctly to improve the prediction quality. Preprocessing may consist of multiple steps such as transformation, discretisation and conversion, all of which are applied to the source data. The need for such preprocessing also emphasises the importance of securing reliable data sources, and one of the widely employed data sources is log files [109]. Log files descriptively record key system events, and therefore represent a rich source for obtaining data. The event recording details can often be adjusted to capture more data (e.g., enabling logging in multiple levels).

Because log data typically represent a reliable source for obtaining data, log management and processing have evolved into a broad research area that includes log analysis, log management, parsing and pattern search. However, this also means that the focus is often on the subdisciplines. For example, Zheng et al. [302] presented a log preprocessing method to improve failure prediction and observed that the method improved failure prediction by 174%. He et al. [96] focused on log analysis and proposed an online log parser using DAG for distributed systems that can parse log entries in streaming data. The volume of data and type (e.g., streaming) have changed significantly nowadays, which has promoted log analysis using machine learning (ML) and AI to facilitate automated analysis [79, 294]. For example, Du et al. [62] proposed a deep neural-based approach to model system log as a natural language sequence. The solution learns log patterns that are then used to detect anomalies. Similarly, Xu et al. [299] proposed an anomaly detection method to mining console logs using principal component analysis (PCA).

In contrast to these solutions, our BFPF module aims mainly to provide a range of services to enable the deployment of the BP framework and to prepare the HAC log data. Therefore, to the best of our knowledge, the proposed BFPF module is the first to facilitate pattern search, log parsing, transformation, conversion, enrichment and filtering in the context of HAC logs.

8.2 Database for Storing Log and Configuration Data

The BFPF module tables can be categorised into two types based on what data they store: configuration and runtime. The content of configuration tables is provided initially as part of the configuration refinement step (Section 8.3). The runtime tables are updated at runtime, and they store the extracted, parsed and enriched event messages in a structured form. There are two kinds of runtime tables: (1) tables used to store the initial structured data, and (2) tables used to store data prepared specifically for the BDN-HAC model (as described in Section 8.6).

8.2 Database for Storing Log and Configuration Data

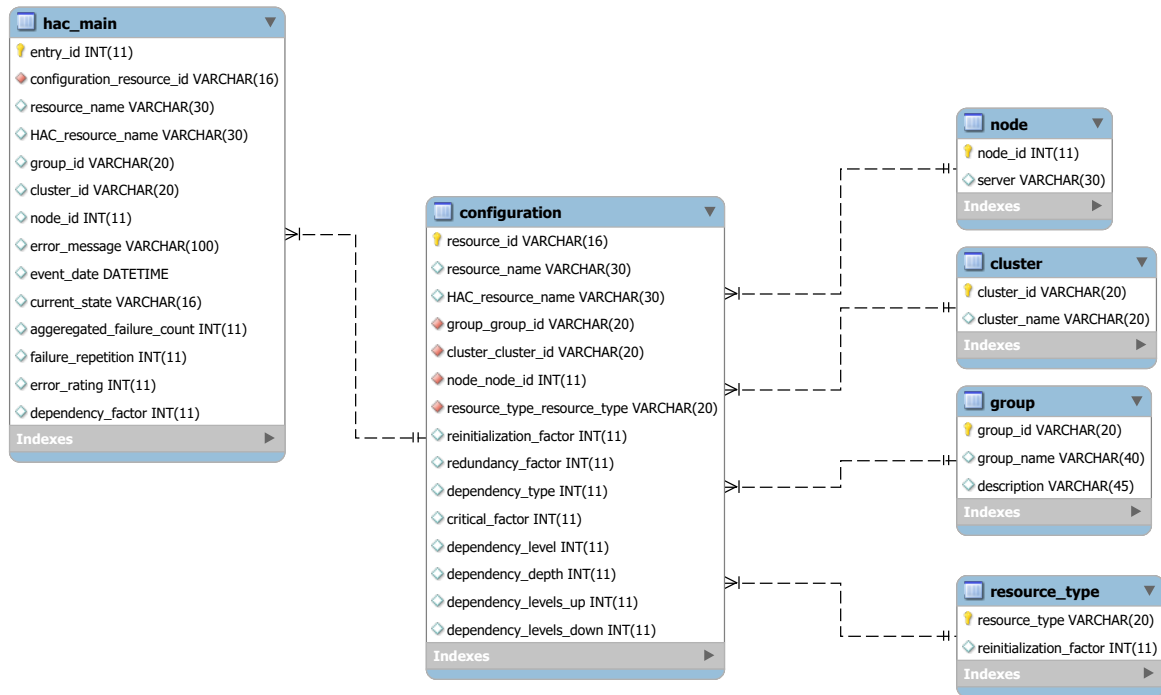


Figure 8.2 Database structures for storing configuration and runtime data.

Table 8.1 Description of the table resource_type

Attributes	Description	Source for Values
resource_type	Type of a resource	Literature study, HAC and the application in scope
reinitialization_factor	Reinitialisation factor	HAC and the application in scope

The BFPF module uses five configuration tables and two runtime tables. Figure 8.2 shows the main runtime table (`hac_main`) and all the configuration tables (`configuration`, `resource type`, `group`, `cluster` and `node`) and their relationships. The second runtime table (`model_data`) is described in Section 8.6. Most of the table attributes are described in the previous chapters. For example, Chapter 6 identifies the characteristics and their associated properties of HACs. The other attributes are explicitly designed to capture the common properties of a HAC (e.g., cluster name, node identifier, group name). Each table is described briefly below, while Tables 8.1, 8.2, 8.3, 8.4, 8.5 and 8.6 list their attributes, attribute description and the source for obtaining the attribute values.

1. Table **resource_type**: This table stores data about the HAC resource types and their reinitialisation values (Table 8.1). The reinitialisation value (attribute `reinitialization_factor`) is linked to a resource type. This combination indicates whether resources belonging to that resource type can be reinitialised by a HAC or not. Nineteen resource types (e.g., disk, process and service) are identified, and additional resource types can be added if and when needed.
2. Table **cluster**: This table stores the cluster identifier and the cluster name (Table 8.2).

Table 8.2 Description of the table cluster

Attributes	Description	Source for Values
cluster_id	Cluster identifier	Primary key (auto generated)
cluster_name	Name of the cluster	HAC

Table 8.3 Description of the table node

Attributes	Description	Source for Values
node_id	Node identifier	Primary key (auto generated)
server	Server on which a node is hosted	HAC

3. Table **node**: This table stores the server names associated with the cluster nodes and their identifiers (Table 8.3).
4. Table **group**: This table (Table 8.4) stores resource group identifiers, names and descriptions.
5. Table **configuration**: The main configuration table (Table 8.5) has all the details about the HAC configuration, including the specific properties. The configuration table has a foreign key relationship to all the other four configuration tables.
6. Table **hac_main**: The main runtime table (Table 8.6) stores log entries that are parsed, enriched and structured. Whenever there is a failure event recorded in the HAC log, this table is updated by the log interface. The table has a foreign key relationship to the main configuration table *configuration*. The attributes *error_rating* and *dependency_factor* are used only for the internal validations to ensure that the values match the corresponding latent nodes' output in the BDN-HAC model.

8.3 Configuration Refinement

The configuration refinement BDFP component ensures that all configuration details are entered correctly into the corresponding tables (described in Section 8.2). This component uses three primary sources to obtain the required data: (1) the HHAM model, (2) the M-table and (3) the HAC configuration. HHAM and M-table provide granular details about resources, such as the position of resources in the HAC hierarchy and the number of levels up and down in this hierarchy. Moreover, it

Table 8.4 Description of the table group

Attributes	Description	Source for Values
group_id	Resource group identifier	Primary key (auto generated)
group_name	Name of the group	HAC

Table 8.5 Description of the table configuration

Attributes	Description	Source for Values
resource_id	Unique resource identifier	HHAM
resource_name	Resource name	HHAM
HAC_resource_name	The HAC resource name	HAC
group_id	Resource group identifier	Table group
cluster_id	Cluster identifier	Table cluster
node_id	Node identifier	Table node
resource_type	Resource type	Table resource_type
reinitialization_factor	Reinitialisation factor	Table resource_type
redundancy_factor	Redundancy measures by the application	Application in scope
dependency_type	The type of a resource	HAC
critical_factor	Criticality of a resource	HAC, application in scope
dependency_level	Depth of a resource	HHAM, M-table
dependency_depth	Depth of the structure	HHAM, M-table
dependency_levels_down	The number of lower levels	HHAM, M-table
dependency_levels_up	The number of upper levels	HHAM, M-table

Table 8.6 Description of the table hac_main

Attributes	Description	Source for Values
entry_id	Unique id	Primary key (auto generated)
resource_id	Unique resource identifier	config. (HHAM)
resource_name	The generic resource name	config.
HAC_resource_name	The HAC resource name	config.
group_id	Resource group identifier	config.
cluster_id	Cluster identifier	config.
node_id	Node identifier	config.
error_message	Error message	HAC log
event_date	Time stamp	HAC log
current_state	Current status of the resource	HAC log
aggregated_failure_count	Aggregates distinct number of failures	Runtime1
failure_repetition	Counts the failures	Runtime1
error_rating	Error rating of a resource	Runtime2
dependency_factor	Dependency factor of the resource	Runtime3

Config. - table *configuration*, Log - HAC log, Runtime1 - computed at run-time using table *HAC_main* and HAC log, Runtime2 - computed at run-time using table *configuration* and HAC log, Runtime3 - computed at run-time using the dependency attributes from table *configuration*

updates details about specific features of HAC and the protected application (e.g., ability to reinitialise an application resource). One important attribute of this component is a unique identifier (*resource_id* in both the runtime and configuration tables) generated by the HMTHA. This identifier is used to associate HAC resources with the BN-HAC model.

Table 8.7 A record entered in the table *configuration* as part of the configuration refinement

Attributes	Example Values
resource_id	C1G1A1B1
resource_name	database
HAC_resource_name	rsc_WEB_database
group_id	C1G1A1
cluster_id	cluster1
node_id	1
resource_type	service
reinitialization_factor	1
redundancy_factor	2
dependency_type	1
critical_factor	1
dependency_level	2
dependency_depth	3
dependency_levels_up	1
dependency_levels_down	1

Figure 8.3 shows two log entries from a high availability cluster. The first entry is: "Jun 22 07:54:30 [1619] vmi243500 crmd: notice: process_lrm_event: Result of probe operation for rsc_DEV_database on vmi243500: 7 (not running) | call=17 key=rsc_DEV_database_monitor_0 confirmed=true cib-update=38". The second entry is: "Jun 22 08:38:53 [1619] vmi243500 crmd: info: update_failcount: Updating failcount for rsc_DEV_CI on vmi243500 after failed start: rc=1 (update=INFINITY time=1592807933)". Key elements are highlighted with boxes and numbered: 1 (timestamp), 2 (server), 3 (process), 4 (resource), 5 (server), 6 (failcount), and 7 (status).

Figure 8.3 High availability cluster log extract and the key elements.

Example 15. Table 8.7 shows one record in the configuration table for the system from the running example, as created during the configuration refinement.

8.4 Log Interface

As described in Section 3.2.6, the log format and structure differ between HACs and there is no standard. Some logs record more details, while others have minimum information. A log extract from the HAC Pacemaker/Corosync is presented in Figure 8.3 where key elements are highlighted. A short description of the key elements is as follows.

1. **timestamp:** Jun 22 2020 07:54:30 (event timestamp)
2. **server:** vmi243500 (Server (node) where the log entry is recorded)
3. **process:** crmd (the HAC process that is responsible for this entry)

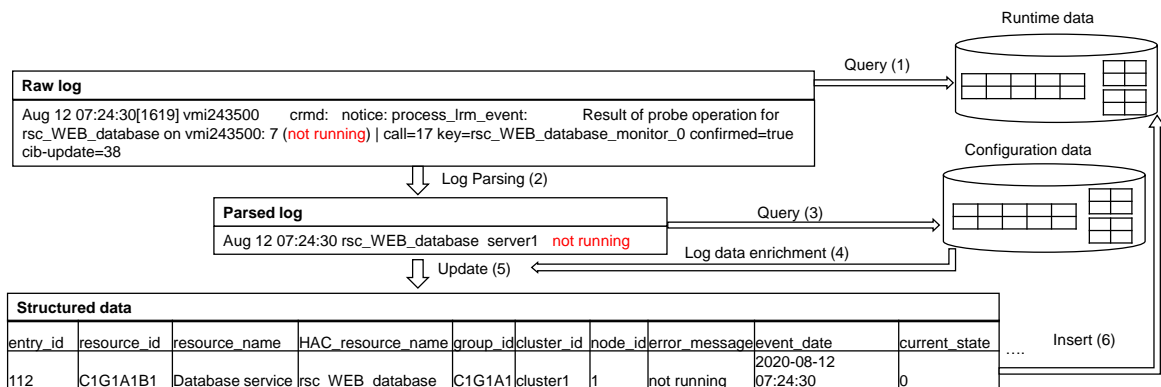


Figure 8.4 An illustration of the log management approach by the log interface.

4. **HAC resource name:** rsc_DEV_database (A resource name that is used by the HAC)
5. **status:** not running (Current status of the resource)
6. **failure count details:** Whenever a resource fails, the cluster increases the failcount for the resource
7. **a score-based calculation per resource and node:** If the attempt to start or stop a resource fails, the resource's failure count is set accordingly (here +INFINITY) to influence the HAC's behaviour, for example, a resource is banned from running in a node

The first five elements are typically present in HAC logs because they are essential for the HAC to take actions. However, the last two elements are specific to the HAC that is used as an example here. Element 6 indicates that the HAC counts the number of failures that have occurred in a particular resource. After reaching a timeout value, the HAC may automatically take actions or enforce policies such as relocating the resource. Element 7 describes a policy update that is set after experiencing multiple failures. When set to a particular value (for example, INFINITY for the resource on a specific node), it implies that the HAC will stop trying to start the resource. Instead, it can move the resource immediately to a secondary node when certain conditions are met. The log interface introduced in this section recognises log elements between one and five; however, it can be extended to extract and interpret other log elements [256].

The log interface employs a pattern search method to identify a specific event related to resource failures, and it uses positions in a log entry to identify the different elements.¹ However, the log files of different HACs have different structure and positioning, and therefore the position of the elements and the search pattern must be updated during the implementation.

As shown in Figure 8.4, when the log interface is initiated, it queries the runtime database to retrieve the latest timestamp 1, which is then used as a separator to ensure that only those log entries after the timestamp are extracted. Those entries are sorted to identify only the distinct log entries,

¹The corresponding software is available on the GitHub repository for the project [256].

Bayesian Prognostic Framework Preparation

Table 8.8 Basic variables that represent the properties of high-availability clusters, related symbols, values, types, value description and sources for obtaining the values

Variable	Values	Value Description	Group	Source
Failure repetition	{low, high}	Low - >4, high-<4	1	Runtime
Redundancy factor	{true, false}	Redundancy factor	2	Runtime
Aggregated failure count	{low, high}	Low - >8, high-<8	1	Runtime
Reinitialisation factor	{true, false}	Resource Reinitialisation	2	Runtime
Dependency type	{local, shared, global}	Three types of dependencies	3	Runtime
Dependency levels down	{low, high}	Low = >2, high= <3	1	Runtime
Dependency levels up	{low, high}	Low = >2, high= <3	1	Runtime
Critical factor	{true, false}	Criticality of a resource	2	Runtime
Current state	{online, offline}	Current state of a resource	3	HAC log
Error rating	{failure, no_failure}	Derived	4	CP
Dependency factor	{low, high}	Derived	4	CP
Resource state	{failure, no_failure}	Derived	4	CP

Log - HAC log, CP - Conditional probabilities.

which are then passed for parsing at time 2. The log interface then uses key elements to query (3), match and fetch (4) additional data from the configuration database. The new data is appended (5) to the structured log data as part of the enrichment. The enriched entries are written (6) to the runtime database (HAC_Main table in Section 8.2). The log interface faces a challenge when extracting the log entries, and that is, multiple entries with the same event message at the same time are created. For instance, multiple modules of a HAC (e.g., ‘crmd’ in Figure 8.3) may attempt to write the same entry at the same time because events are passed between the HAC modules. However, only a distinct entry must be captured and inserted into the database for further processing. Therefore, the log interface ensures that only one distinct entry is extracted using sorted timestamps.

8.5 Transformation and Conversion

We described the four groups of variables in Section 6.4, and the group indicates what kind of changes the group variables must undergo before the BDN-HAC model can use them. The changes associated with each group are listed as follows.

1. Transformation
2. Conversion
3. No change
4. Derived variables

We present these four groups of variables, their values, value descriptions, groups and sources to provide the required data at runtime in Table 8.8. Hence, group 1 variables are transformed while group 2 variables are converted. In contrast, group 3 variables undergo no changes and group 4 variables are derived variables, implying that they are associated with conditional probabilities. We describe how the changes are applied to the variables in the groups in the remainder of this section.

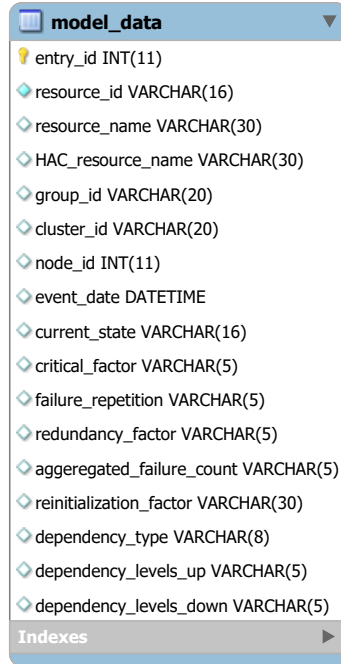
1. **Transformation.** We followed a three-step approach to applying a transformation to the group 1 variables.

- (a) Identify the potential variables to be included
- (b) Identify the split interval domains to represent the states
- (c) Create the transformation condition

In this group, the variables are transformed, i.e., changed to categorical (ordinal) variables. Four variables were identified already as part of identifying the variables associated with the characteristics in Section 6.4: *failure repetition*, *aggregated failure count*, *dependency levels down* and *dependency level up*. Two states are identified using the split interval domains: low and high.

The transformation conditions for the variables *dependency levels down* and *dependency level up* are defined using FMEA assessment to indicate a potential impact on the related resources (upper or lower) in the hierarchical representation of a HAC upon the resource's failure [223]. Hence, a low value indicates the impact on a smaller number of resources, while a high value indicates an impact on many resources. The transformation conditions for the variables *failure repetition* and *aggregated failure count* are defined as follows. Suppose a particular resource failure is repeated and captured by the variable *failure repetition*, and this repetition is greater than four in the last two minutes, it may indicate that any attempt to reinitialise the resource by the HAC has reached a timeout threshold [277, 22, 20]. It may, in turn, imply that the probability of resource failure is increased. Therefore, a value greater than four is categorised as *high* to indicate a high probability of failure. Similarly, the variable *aggregated failure count* aggregates all failures for a resource in the last four hours. If the counter is greater than eight, the value is set to high, which indicates that a global timeout is reached or the resource experiences a permanent error, and hence the probability of failure is increased.

2. **Conversion.** Variables in this group are converted to Boolean, where true indicates that the related properties are active and where false indicates related properties are not active. Three variables are part of this group: *redundancy factor*, *reinitialisation factor* and *critical factor*.
3. **No change.** The variables in this group do not undergo any changes, and two variables are part of this group: *dependency type* and *current state*. The variable *current state* presents the actual status of a resource, which can be either online or offline. The variable *dependency*



Attribute	Data Type
entry_id	INT(11)
resource_id	VARCHAR(16)
resource_name	VARCHAR(30)
HAC_resource_name	VARCHAR(30)
group_id	VARCHAR(20)
cluster_id	VARCHAR(20)
node_id	INT(11)
event_date	DATETIME
current_state	VARCHAR(16)
critical_factor	VARCHAR(5)
failure_repetition	VARCHAR(5)
redundancy_factor	VARCHAR(5)
aggregated_failure_count	VARCHAR(5)
reinitialization_factor	VARCHAR(30)
dependency_type	VARCHAR(8)
dependency_levels_up	VARCHAR(5)
dependency_levels_down	VARCHAR(5)

Figure 8.5 Database table for storing data for the BDN-HAC model.

type represents the three types of dependencies. Each can increase the probability of failure in various degrees (local-low, shared-medium and global-high).

- Derived variables.** The fourth group consists of three derived variables to aid in assigning weights and reducing dimensionality. These variables have two states, *failure* and *no failure*. The state no failure indicates that the failure can be managed locally, and the state failure implies that the failure cannot be managed locally.

8.6 Database Table for Storing Model Data

The structured log data (described in Section 8.2) undergoes the process of transformation, conversion and filtration before it is stored in the table for storing model data. The table update is triggered at runtime as soon as there is a new entry in the main table. The structure is presented in Figure 8.5.

After applying the filter, 17 attributes are extracted. *entry_id* is a primary key and generates sequential numeric values automatically when new records are created. *resource_id* is the resource identifier that is generated by the HMTHA tool (Chapter 6). *resource_name* represents the name of a resource, while *HAC_resource_name* is the HAC name of the resource. The descriptions of the attributes *group_id*, *cluster_id*, *node_id* and *event_date* are provided in Section 8.2. The remaining nine attributes are passed to the BDN-HAC model.

8.7 Filter

The filter is the final step in the BFPF module, and the objective of the filter is to prepare the data for the BDN-HAC model. Therefore, the filter is applied to the log data that has undergone transformation and conversion to select the required data for the BDN-HAC model. The filter is implemented together with the transformation and conversion in the same software tool.

8.8 Summary

This chapter presents the BFPF module, the second module in the BP framework, and answers research question RQ2. The module prepares an environment that facilitates processing the log data so that they can be used by our BDN-HAC model. To this end, the BFPF module consists of components for configuration refinement, transformation and conversion, the log interface, filtration and the related database objects. These components ensure that the log data undergo parsing, enrichment, transformation and conversion and filtration.

Chapter 9

Bayesian Prognostic Framework

This chapter presents the BP framework that integrates the four modules introduced in Chapters 5, 6, 7 and 8. The chapter is organised as follows. Section 9.1 presents related work, and Section 9.2 describes the BP framework. Section 9.3 lists the steps associated with the implementation of the BP framework. Finally, Section 9.4 summarises the chapter.

9.1 Related Work

It is not uncommon that a series of probabilistic models are used in conjunction [233, 8] to deliver specific functionality. For example, one model supports detection or diagnosis. In contrast, the second model predicts, e.g., the potential failure of another component, based on the outcome of the analysis performed using the previous model. However, to deliver an accurate diagnosis, detection or prediction, knowledge is required about a component or all interconnected components through dependencies. Obtaining such information could also be part of the modelling framework. Pitakrat et al. [214] proposed an online failure prediction framework for component-based software systems. This framework captures the dependency information between components from the system architecture models and uses this information for failure prediction at runtime. Cai et al. [31] proposed a prognostic model based on BNs to identify the key factors that influence the survival of patients who underwent surgery for gallbladder cancer. The study combined the BN model with essential measures such as age and sex.

Another frequent application area where one or more BN models are used is the industrial sector. Such models can support a range of steps, such as identification, isolation and prediction of failures in industrial equipment. One area where these steps can be combined is to predict the remaining useful life (RUL) of a machine. For example, Medjaher et al. [169] used a DBN-based failure prognostic model to estimate the value of the RUL of industrial machines before a failure. The model could also be used to study the propagation of the effect of one state on other states. Similarly, Tobon-Mejia et al. [267] constructed a model to estimate the RUL of a computer numerical control (CNC) tool machine by capturing the degradation behaviour of the machine.

However, to the best of our knowledge, our proposed BP approach is the first to provide a complete framework to extract log entries, preprocess data through a series of components, deliver data to a BDN-based detection model and BN-based prediction model, and then process the outcomes. The outcome of a prediction indicates whether the higher-level components (resource group or system) will fail or not. The framework also provides an approach for constructing a BN model aided by the HHAM model and accompanying M-table.

9.2 Bayesian Prognostic Framework

The four modules of our BP framework are shown in Figure 9.1. Three steps associated with the framework are also depicted in the figure. The use of the components is indicated by 'DIR', which stands for 'design', 'implementation' and 'runtime' while '-' means that a component is not used within a particular step. As shown in Figure 9.1, the module HMTHA is a standalone module that is used mainly at design time. For example, the primary outcome of the HMTHA, HHAM, and the corresponding M-table and T-rules are used further with the BPF and BN-HAC modules. However, the HMTHA can also be revisited to make changes during the implementation phase in order to reflect any changes in the HAC environment. Thus, the use of the model is annotated with 'implementation' as well.

The BPF module (described in detail in Chapter 8) prepares an IT environment for deploying the BP framework and enables preprocessing using six module components as shown in Figure 9.1. Hence, the step environment preparation deals with creating the required database structures and implementing the module components. For example, the relevant database objects are updated to incorporate the information related to the HAC configuration using the naming convention introduced by the HMTHA. One key component is the log interface that identifies unique failure entries from HAC logs. It initiates a series of steps, such as query, extract, enrich, update, transform, convert and filter. However, the preparation components, such as configuration refinement, are not used at runtime. Instead, only the preprocessing of the HAC failure information is conducted. The preprocessed information is then stored in a database table before the BDN-HAC model is initiated.

The BDN-HAC model is a HAC solution-independent model. The model is employed to detect whether a resource failures can be managed at the resource-level. To that end, the model uses a set of characteristics to understand and interpret the failures. In addition to the set of established characteristics typically employed by HAC solutions, we added new characteristics to improve the overall detection capabilities (e.g., the self-healing capability of an application). In the model, conditional probability adds weight to parent nodes, whereas preferences add weight to the top-level child nodes in the utility node. After variable consolidation and dimensionality reduction, the result is a utility value that indicates whether a resource failure can be managed locally or not. This utility value is interpreted, and only those individual failures predicted as not manageable locally are sent to the BN-HAC model.

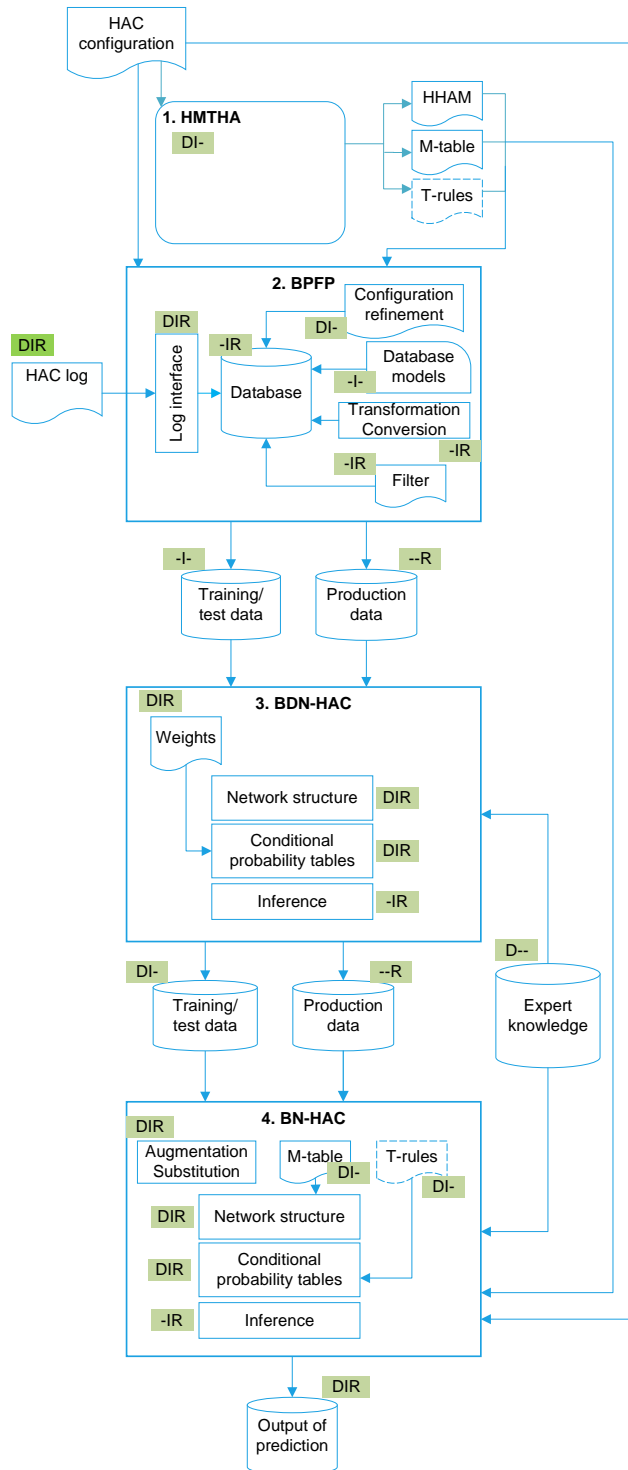


Figure 9.1 Modules and components of the Bayesian prognostic framework, annotated to indicate whether they are used during D(esign), I(mplementation) and/or at R(untime), e.g., 'DI-' indicates that a component is used during design and implementation and is not used at runtime.

Notes: HMTHA - Holistic modelling technique for high availability, HHAM -Holistic high availability model, M-table - Mapping table, T-rules - Translation rules, BPPF - Bayesian prognostic framework preparation, BDN-HAC - Bayesian decision network for predicting locally manageable resource failures, BN-HAC - Bayesian network for failure propagation and prediction.

Bayesian Prognostic Framework

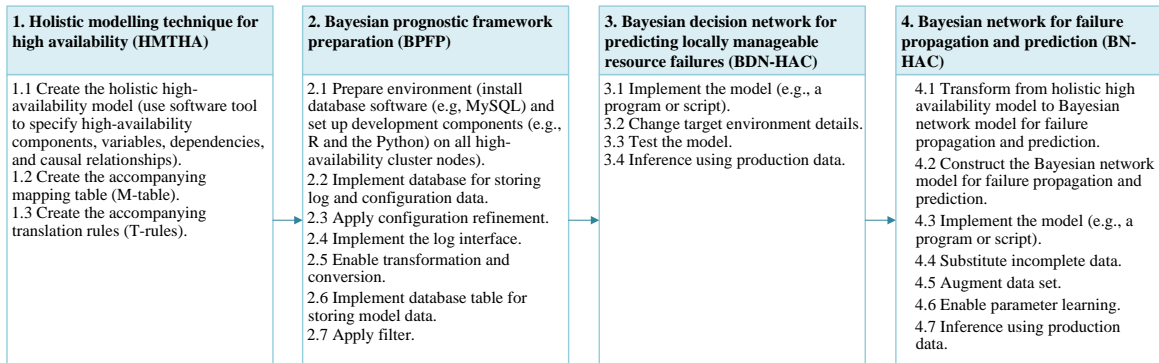


Figure 9.2 Bayesian prognostic framework development process.

The BN-HAC provides a method for constructing a BN model for the corresponding HAC. Thus, the resulting model mirrors the setup of the HAC, meaning that all HAC components are represented in the BN-HAC model. The HMTHA (HHAM, M-table and T-rules) outcomes are used to construct the BN model and to map the HAC resources to the appropriate BN nodes. For example, weak resources are mapped to ordinary nodes, but conditional probabilities are assigned to the child node to indicate a weak influence. Additionally, all resources without physical representations are mapped to latent nodes. Apart from assigning conditional probabilities to child nodes, uniform prior distributions are given to all other nodes. Parameter learning is employed in the implementation phase to train the BN model using a data set where incomplete data are substituted. During the runtime phase, the inference of the model occurs using the learnt parameter distributions to predict the potential failure of the related resource group or system.

To summarise, the HMTHA module is only used during the design phase. Similarly, the expert knowledge (Figure 9.1) is also used to construct the BDN-HAC and BN-HAC models during the design phase. All components are implemented during the implementation phase, and failure information from the HAC is collected, prepared and processed by the BDN-HAC model to create a training data set. The training data set is then processed by the BN-HAC model to learn the parameter distributions. In the third phase (runtime), the BPPF module extracts failure information whenever a failure occurs in the production HAC and prepares data before improved detection by the BDN-HAC can occur. Only the confirmed failure information is passed to the BN-HAC model to propagate the failure and to predict the potential failure at the resource group or system level.

9.3 Implementation Steps

Figure 9.2 summarises the process for developing our complete BP framework. The first step is to create an HHAM model using the holistic HA modelling technique. The resulting model and accompanying M-table and T-rules are required to construct the BN model in Step 4. The second step is to prepare the environment to deploy the BPF and to prepare the data to be processed by the model

Table 9.1 Reusability of framework components

Framework Component	Reusability
Holistic modelling technique for high availability (HMTHA)	
HMTHA tool (Java program and database script)	Reusable as is
Bayesian prognostic framework preparation (BPF)	
Database objects for storing log and configuration data	Reusable as is (scripts to create the objects)
Log interface	Reusable as is (positions for extracting log records must be changed)
Script for transformation, conversion and filtering	Reusable as is
Database object for storing model data	Reusable as is (script to create the object)
Bayesian decision network for predicting locally manageable resource failures (BDN-HAC)	
BDN model	Development required (the model used in the thesis is provided but a script is required to automate the inference)
Bayesian network for failure propagation and prediction (BN-HAC)	
BN model	Development required (a sample model for the testbed application and HAC is provided but a model must be generated for the target application and HAC specifically, and a script is required to automate the inference)

in Step 3. In the third step, the BDN-based model is implemented and configured to accept data from Step 2. Finally, the inference output is prepared and sent to the model in Step 4, which details the construction of the BN model using the outcome from Step 1. The model uses parameter learning to learn the distributions. Subsequently, the model is used for inference with runtime data from the BDN-HAC model.

We summarise the framework components and whether they are reusable as-is, are customisable or require development from scratch in Table 9.1. All framework components require minor changes to accommodate the target deployment environment, such as the OS and database.

9.4 Summary

This chapter presented the BP framework, which brings together the four modules from in Chapters 5, 6, 7 and 8. The four modules are combined to deliver a prognostic framework for a HAC. All four modules are integrated so that the first module, HMTHA, provides input to both the BPF and BN-HAC modules. The BDN-HAC is a HAC solution-independent module that improves the detection of failures at the resource-level by exploiting data associated with established HAC characteristics and

Bayesian Prognostic Framework

new characteristics introduced in this thesis. In contrast, the BN-HAC module must be constructed to reflect an underlying HAC solution. Therefore, we present a method for constructing and mapping the HAC components to a BN model. The parameters of the constructed model are then learnt using data from the HAC under analysis. Using this information, the BN-HAC can then assess failures confirmed as locally unmanageable and propagate them to the resource group or system level to predict a potential failure at these levels.

Part III

Implementation and Evaluation

Chapter 10

Evaluation

This chapter presents the evaluation of the individual modules of our BP framework and the framework as a whole by comparing them to a widely used open-source HAC solution. To this end, we performed extensive experiments by subjecting an ERP application deployed in a production-like environment to a broad range of injected failures over a period of 8 months (between 6 July 2020 and 12 March 2021). This testbed is presented in detail in Section 10.1, followed by a description of our evaluation methodology in Section 10.2. This chapter then presents the evaluation of the four BP framework modules in Section 10.3 and the evaluation of the end-to-end framework in Section 10.4. We conclude with a discussion of threats to validity in Section 10.6 and a brief summary in Section 10.7.

10.1 Testbed

The testbed was established in the public cloud ¹, and the open-source HAC we deployed in the testbed was HAC ClusterLabs stack ² (Pacemaker/Corosync) [128, 20]. Subsequently, we deployed the ERP solution and included it in the HAC. The testbed was running continuously for more than two years and three months between February 2019 and May 2021 to facilitate the development, test, optimisation and evaluation of the individual modules of the framework and the complete framework. To identify all the required components for a HAC deployment, we applied the taxonomy introduced in Section 3.1, and an extract of the result is described in Appendix B. Using this information, we present the deployment of the infrastructure components (virtual machines, network and storage.) for the HAC in the testbed. In the following sections, we also describe the selected ERP and how the ERP components were integrated into the HAC.

Evaluation

Table 10.1 Virtual machines used to enable high availability in the testbed

	Server 1	Server 2	Server 3
CPU (vCPU Cores)	8 vCPU Cores 2.20 GHz	8 vCPU Cores 2.20 GHz	6 vCPU Cores 2.40 GHz
Memory (GB)	30	30	20
Operating system (64-bit)	openSUSE Leap 15.0	openSUSE Leap 15.0	openSUSE Leap 15.0
Role	Primary node	Secondary node	Storage server
IP-address	IP address 1	IP address 2	IP address 3
network	Network 1	Network 2	Network 3

VCPU – Virtual CPU

Table 10.2 List of virtual IP addresses and associated resource groups

IP	Network	HAC Resource group	Virtual Host Name
IP address 4	Network 4	Message and lock instance group	Virtual name 1
IP address 5	Network 5	Database group	Virtual name 2
IP address 6	Network 6	Main instance group	Virtual name 3
IP address 7	Network 7	Backup lock server group	Virtual name 4

10.1.1 Virtual Machines

We configured three virtual machines for the testbed environment, and the details are presented in Table 10.1. Servers 1 and 2 functioned as the primary and secondary nodes of the HAC, respectively. Server 3 was employed as the shared storage provider.

The three virtual servers were placed in the same physical location (data centre); thus, the only type of cluster possible is ‘local’. We selected the open-source operating system OpenSUSE Leap 15.0 for all virtual machines. In addition, the software packages (e.g., libraries) that were prerequisites for the ERP solution were installed.

10.1.2 Network Configuration

The HAC requires multiple networks to enable independent communication between different components (e.g., heartbeat and cluster) [239]. However, setting up multiple dedicated networks is expensive and time-consuming. We therefore chose the setup available in the selected public cloud environment. Thus, we shared the same network for some of the communications, for example. Table 10.2 lists the different subnetworks used in the configuration. When using multiple networks, multiple network interfaces (NICs) are required to ensure redundancy. However, we opted to proceed with only one NIC per virtual server in the standard virtual machine setup. Furthermore, the HAC employs multiple VIP addresses (called relocatable or floating) to ensure that each resource group can be addressed individually and relocated independently. Table 10.2 lists the virtual hostnames associated with the VIPs for the resource groups. The virtual hostnames were used when implementing the application.

¹<https://contabo.com/en/>

²<https://www.clusterlabs.org/>

Table 10.3 Storage configuration of the testbed

Device	Mount Point	Resource Group Name	Size (GB)	File System Type [†]	LVG	LV
/dev/sdc	/usr/sap/DEV/ASCS00	Message and lock instance group	9.8	ext4	vg_ASCS00	lv_ASCS00
/dev/sdc	/usr/sap/trans	FS transport	3.9	ext4	vg_trans	lv_trans
/dev/sdh	N/A	SBD	0.5	N/A	vg_sbd	lv_sbd
/dev/sdi	/sapmnt	Distributed lock manager (DLM) group	19.5	OCFS2	vg_sapmnt	lv_sapmnt
/dev/sdd	/sapdb	Database	781.3	ext4	Vg_sapdb	lv_sapdb
/dev/sdj	/interface	FS interface	2	ext4	vg_interface	lv_interface
/dev/sdf	/usr/sap/DEV/DVEBMGS01	FS main instance group	19.5	ext4	vg_DVEBMGS01	lv_DVEBMGS01

[†] All file systems had EXT4 as the file system type except disk sdi, which had a shared cluster file system, and the file system type was OCFS2.

LVG - Logical Volume Group, LV - Logical Volume, FS - File system, SBD - Storage-Based Death, DLM - Distributed lock manager, N/A - Not applicable.

10.1.3 Storage Configuration

We implemented an iSCSI-based storage solution to enable shared storage for the HAC. There are two terms associated with the iSCSI domain: the iSCSI server (target) and clients (initiator) [3]. The communication between the iSCSI target and initiators occurs using an IP network; therefore, a dedicated network and related NICs are typically required. However, we used the single NIC available in the servers and the shared network to facilitate storage communication. Thus, Server 3 functioned as the iSCSI target and provisioned the required storage to the initiator, Servers 1 and 2. We partitioned the available disks in Server 3 and presented them as raw disks to the iSCSI initiator servers.

The presented raw disks were then configured using logical volume management [94] and created as logical volumes (LVs) in the primary node. Each LV was then formatted using a specific file system type to be included in the resource groups. The complete list of the raw partitions devices, corresponding mount points, related resource group name, size, file system type, related LVG and LV are listed in Table 10.3. The shared storage signifies that the same file systems are available to both servers, Servers 1 and 2. However, only the active node has full access to the shared storage at any given time in an active-passive topology like the one we used.

In the case of a failover to the secondary node, the HAC ensures that the shared file systems are available in the active node. However, there was an application requirement to allow a specific file system to be distributed and made available on both nodes simultaneously. For this purpose, a cluster file system was set up using OCFS2 as the file system type. Then, we set up a DLM service to coordinate the read-write activities from both servers. Moreover, an SBD disk was also presented on

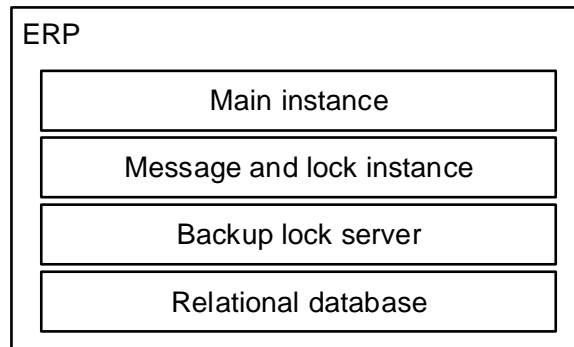


Figure 10.1 Main layers of the enterprise resource planning (ERP) solution in the testbed.

both servers to provide a fencing mechanism. The disk was explicitly formatted to store SBD-specific data.

10.1.4 HAC Solution

The ClusterLabs stack HAC (Pacemaker/Corosync) that we selected was also part of the survey presented in Section 3.2, and it is a comprehensive solution that can support a broad range of features (Section 3.2). Although the clusterLab stack is an open-source HAC, it has been commercialised and included in commercial HACs such as SUSE Linux Enterprise High Availability Extension and Red Hat High Availability Add-On; both of which were part of our survey. We selected this HAC for three main reasons: (1) it is based on an open-source license and is free of charge, (2) it supports many EAs, including the selected ERP solution, and (3) the HAC could be deployed in the public cloud where we established our testbed. This means the HAC provides ERP-specific agents to manage the ERP resources and that it understands the internals of the ERP solution.

10.1.5 Enterprise Application

We used the above infrastructure to deploy, configure and run a fully fledged commercial ERP application that is widely used in industry to manage business functions such as accounting, procurement and logistics. The licence to use this application was secured with the help of a project collaborator. The actual ERP application used and the details of our collaborator cannot be listed here due to confidentiality reasons. However, the high-level architecture of the ERP application is presented in Figure 10.1.

The selected ERP solution is a multilayered and multitiered solution that can be implemented across several servers. To show how different layers are addressed in an HA setup, we present the layers of the ERP solution in Figure 10.1. However, there is also a layer for application servers to distribute and manage the workload of the solution. We did not include this layer in Figure 10.1 because it was not used. Instead, we placed the application layer in the main instance because it provides the same services. Furthermore, application layers are usually not part of the HAC setup

because HA is enabled by a load distribution mechanism (e.g., load balancer). The message and lock instance enables the distributed transaction management for the EA. In contrast, the backup lock instance ensures that it can take over some responsibilities from the message and lock instance when the latter fails. The relational database stores the complete data related to the application. All four layers and their components are treated as SPOF in this implementation.

10.1.6 Enterprise Application Deployment in the Testbed

Deploying the ERP application in an HA setup consists of multiple steps and requires that the environment is also prepared to support the implementation. For example, virtual hostnames pointing to VIPs are required to enable relocation for the resource groups. Hence, the installation requires that the VIPs and related hostnames are available before starting the installation. Hence, the preparation activities are listed as follows:

- The required virtual servers are prepared (e.g., the operating system is installed).
- The network configuration is completed, including the allocation of the new IPs and VIPs. The related virtual hostnames are also defined.
- Storage is prepared, and the required file systems are implemented.
- The cluster file system is prepared and added to the HAC.

When all preparation activities are completed, the installation of the application can start, which must be performed sequentially. Moreover, because the ERP application must be included in the HAC, the installation must consider the target node for the installation. Most components are installed in the primary node, but some may be installed in the secondary node. Hence, we installed the following application components:

1. the message and lock instance using the virtual hostname in the primary node,
2. the backup lock server instance using the virtual hostname in the secondary node,
3. the database instance using the hostname in the primary node,
4. the main instance in the primary node, and
5. the front-end component.

The first four components (which are described as layers) were identified as SPOFs; hence, they were included as resource groups in the HAC. The fifth component is a graphical user interface tool installed on a computer to access the application.

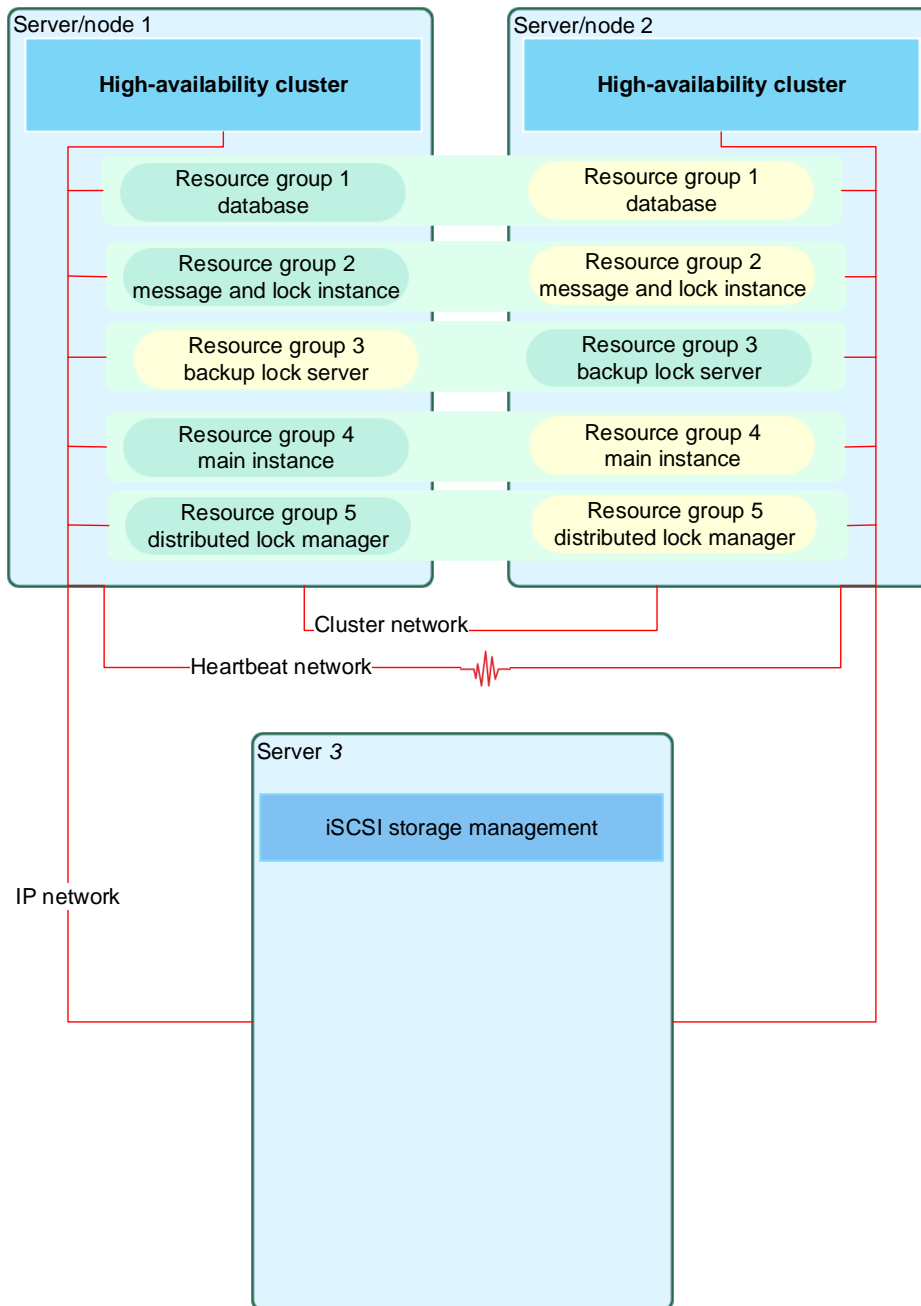


Figure 10.2 Architecture of the high availability cluster (HAC) in the testbed application.

10.1.7 HAC Architecture

The different aspects of the HAC discussed in the previous sections were brought together to present the HAC architecture illustrated in Figure 10.2. The implemented HAC software, ClusterLabs stack, is an open-source package comprising two primary components: Pacemaker (cluster management) and Corosync (cluster communication). Further, we also deployed multiple resource agents to manage


```
quorum {
  provider: corosync_votequorum
  two_node: 1
  expected_votes: 2
  wait_for_all: 1
  last_man_standing: 1
  last_man_standing_window: 10000
}
```

Figure 10.3 Extract from the quorum configuration file.

the ERP application, and some of the agents were standard (e.g., to manage a relocatable IP file system). In contrast, others were specific to the ERP components (e.g., the message and lock instance and database). Hence, the HAC consisted of five resource groups to provide complete protection for the application. Four resource groups represented the SPOF components of the application. In contrast, the fifth resource group, the cluster file system, hosted a cluster fail system. A DLM service was also included in the resource group because it was a prerequisite for the cluster file system.

We established a heartbeat network using the shared network for the heartbeat communication between servers. Moreover, we also set up a cluster network to enable node (inter-node) communication. Resource groups represent all four instances of the ERP application. All running resource groups are on Node 1, the primary node, whereas resource Group 3 runs on Node 2 because the backup lock server runs on the secondary node. Hence, active resource groups are displayed in green and passive resource groups are in yellow. Further details of the configuration are provided in the next section.

10.1.8 HAC Configuration

The complete configuration of the HAC is provided in Table 10.4, which presents the resource names, related HAC resource names, resource types, corresponding resource groups and short descriptions. The five resource groups provide complete protection for the ERP application. All resource groups, except the resource group 'distributed lock manager', had dedicated VIPs to enable relocation. In contrast, the resource group 'distributed lock manager' hosted a cluster file system. A VIP is not required because the services are available on both nodes simultaneously, managed by the DLM service.

There are also three global dependency resources, and two are simple monitors without any governing function on the underlying resource. The third resource, SBD, is a critical resource to enable fencing. The resource represents an SBD disk and SBD daemon, presented on both nodes. The SBD disk stores messages monitored by the SBD daemon. The SBD service is included in the HAC by configuring an SBD STONITH resource.

Evaluation

Table 10.4 High availability cluster (HAC) configuration listing all resources, HAC names, resource types, resource groups, and short service descriptions

Resource Name	HAC Resource Name	Resource Type	Resource Group
Message and lock instance group	grp_DEV_ascs	Resource group	N/A
Message and lock service	rsc_DEV_ASCS00	Service	MLI
FS message and lock instance	fs_DEV_ASCS	File system	MLI
FS trans	fs_2_DEV_ASCS	File system	MLI
FS interface	fs_3_DEV_ASCS	File system	MLI
VIP	vip_DEV_ASCS	VIP	MLI
Database group	grp_DEV_database	Resource group	N/A
Database	rsc_DEV_database	Service	DB
FS database	fs_DEV_database	File system	DB
VIP	vip_DEV_database	VIP	DB
Main instance group	grp_DEV_ci	Resource group	N/A
Main instance	rsc_DEV_CI	Service	MI
FS main instance	fs_DEV_CI	File system	MI
VIP	vip_DEV_CI	VIP	MI
Backup lock server group	grp_DEV_ers	Resource group	N/A
lock system	rsc_DEV_ERS10	Service	BLS
VIP	vip_DEV_ERS	VIP	BLS
Distributed lock manager (DLM) group	grp_DEV_storage_dlm	Resource group	N/A
DLM	dlm_DEV	DLM	DLM
FS DLM	fs_DEV_sapmnt	File system	DLM
CPU monitor	global_rsc_DEV_CPU	Monitor	N/A
NIC monitor	global_rsc_DEV_NIC	Monitor	N/A
SBD	stonith-sbd	SBD	N/A

Resource Group: N/A - Not applicable because it is a resource group, MLI - Message and lock instance group, DB - Database, MI - Main instance group, BLS - Backup lock server group, DLM - Distributed lock manager, FS - File system, SBD - Storage-Based Death, VIP - Virtual IP

10.1.9 Quorum Configuration

We implemented a quorum service using the device realisation ‘node only’, implying no separate quorum device was required. Instead, the available nodes were used to provide the quorum service. The quorum mode was a ‘majority node’, which allows running a cluster with only one node. An extract from the configuration is provided in Figure 10.3.

The service `corosync_votequorum` runs on all available nodes. The `two_node` parameter indicates that the cluster consists of only two nodes, whereas the parameter `expected_votes` refers to the number of votes when all nodes are active. The feature `wait_for_all` enables a quorate when all nodes are online. The feature `last_man_standing` enables the cluster to recalculate the `expected_votes` dynamically, whereas the `last_man_standing_window` (in ms) indicates a timer, and when the timer expires, the `expected_votes` and quorum are recalculated.

Table 10.5 Applied policies for the testbed high availability cluster (HAC)

Policies	Description
Failover	Fails over to the secondary node when failures occur and when local reinitialisation fails
Failback	Policies that enforce a node to be preferred are not considered
Start-up	Policies to manage the resource start-up order
Automatic failover	Automatic failover is enabled
Automatic policy	Automatic policy setting is not considered (e.g., after multiple failovers on the primary node, the HAC may decide to favour the second node)

Table 10.6 Resource-level configuration parameters for the testbed high availability cluster (HAC)

HAC Resource	Start (s)	Stop(s)	Monitor(s)	Monitor Interval(s)
Message and lock service	180	240	120	60
VIP	20	20	20	10
FS message and lock instance	60	60	40	20
FS trans	60	60	40	20
FS interface	60	60	40	20
Database service	1800	1800	60	120
VIP	20	20	20	10
File system for the database	60	60	20	40
Main instance service	180	240	60	120
VIP	20	20	20	10
FS main instance	60	60	40	20
Backup lock server service	190	240	60	120
VIP	20	20	20	10
DLM service	90	100	60	60
FS DLM	60	60	40	20
CPU monitor	10	10	10	10
NIC monitor	60	20	60	10
SBD	20	15	20	3600

DLM - Distributed lock manager, FS - File system, SBD - Storage-Based Death, VIP - Virtual IP

10.1.10 Conditions of the HAC

The HACs have different policies and rules that influence their failure management behaviour. Therefore, to evaluate the test cases consistently, the HA policies and configuration parameters were set to ensure that the different HACs can be represented accurately within the testbed setup. Moreover, to promote solution independence, no HAC-specific features were used. Table 10.5 presents the policies set in the testbed environment.

The configuration parameters can be set either on a resource or global level, or both. The resource-level parameters are specific to a particular resource type, whereas a global-level parameter can be

Evaluation

valid for all resources. Hence, each resource is associated with parameter values for ‘start’, ‘stop’, ‘monitor’ and ‘interval’ [128]. The start is the allowed start time, and the stop is the allowed stop time. The time to complete the resource status check is the monitor time, whereas the interval for such a check is specified as the interval time. Thus, a timeout may occur after the specified value. We chose to use the default values listed in Table 10.6.

Furthermore, the Pacemaker/Corosync HAC has a multitude of configuration options. Three distinct configurations were considered to allow the experiments to be as accurate as possible:

1. The HAC automatically sets the policies, rules and constraints upon failure of a resource to influence the failback, resource group failover or system failover. This ensures that the resource group or the system stays in the secondary node, thus preventing the failback to the primary node. However, because we executed many test cases in the primary node, this affected the testing. Therefore, we did not consider all those policies, rules and constraints created after a failure. Furthermore, we also validated that these changes did not affect the functionality of the HAC by injecting multiple failures, observing the results and then comparing these with the default configuration when all such policies were enabled.
2. A stickiness policy was applied to ensure that all resources remain in the same node as they were running to avoid failovers even when only one resource fails. If one resource fails, it may affect all the related resources resulting in either a resource group failover or a system failover. However, to capture the behaviour of the HAC, our test cases were executed under two conditions: with stickiness policy applied and without the policy. The latter enables the standard behaviour of the HAC to be observed.
3. All the other Pacemaker/Corosync constraints and policies were set to default.

10.2 Evaluation Methodology

As described earlier in Section 10.1, the HAC-protected ERP application from our testbed was run for approximately two years. During the first nearly 14 months, the testbed was monitored to collect information used to develop the BP framework and its modules. Afterwards, the ERP application was subjected to numerous injected failures for the final eight months. Our objective was to induce faults in two ways: first, by simulating high application usage, creating a substantial workload and forcing resources to fail, and second, by injecting faults. We selected a prepackaged ERP application (Section 10.1.5) with a model company configured with considerable batch processes to simulate the workload. The batch processes performed data extraction for reporting and writing, and collected system and performance statistics. Some of the batch jobs were long-running and required significant system resources. However, when the workload did not result in underlying resource failures, we proceeded with injecting the failures, as described in Section 10.2.1.1.

The logs generated by the application and the HAC were used to create multiple data sets, which were used both to train the Bayesian networks from our BP framework and for evaluation purposes.

The BP framework was not integrated with the HAC, so the evaluation of its modules and of the end-to-end framework were conducted by passing inputs from these data sets to the BP framework modules. This enabled us to establish the effect of using the BP framework as part of a future HAC.

We executed the test cases (Section 10.2.1.2) in the testbed and observed the results, and when required, repeated the test cases. The HAC supporting the ERP required multiple checks such as data consistency checks after execution of each test case, which took between 10 and 240 minutes. Hence, completing all test cases took more than eight months between 6 July 2020 and 12 March 2021. Because the execution of each test case took time to complete, we followed two approaches to executing test cases. The first approach scheduled scripts to run in the background to initiate failures without supervision. In the second approach, we executed the test cases interactively. Most of the test cases were executed using this second approach. We established the test protocols to document each step and the outcome when executing each test case. All the relevant data sets, log files, test protocols, calculations and graphs are available on our Github repository³.

10.2.1 Test Cases

HACs experience failure on multiple levels, and these failures are handled using the threefold strategy detailed earlier in the thesis. As the size and complexity of a HAC increases, so does the combination of failures that can occur. Testing all conceivable kinds of failures is, in general, not possible. As such, we chose to induce the critical failures using fault injection methods [105, 232].

The HAC solution (Pacemaker/Corosync) uses scores to determine the target for relocation. For example, if a node with a high score is available upon a resource failure, the HAC solution relocates the service to that node. Moreover, the HAC solution can also automatically distribute the resources across all available nodes (Pacemaker-specific feature). This behaviour could incur downtime because some resources might be relocated automatically after a failover or failback. However, to ensure that we covered most of the critical combinations of the configuration parameters of HACs, our test cases were repeated under two conditions: with the stickiness set and without stickiness, meaning that the standard behaviour of the HAC under analysis could be observed. The stickiness parameter prefers the node where the resources are currently running.

10.2.1.1 Fault Injection Methodology

The objective of using fault injection was to induce many of the real-life failures related to EAs and HACs. Hence, we identified the corresponding failures by analysing the research literature, case studies and documentation [21, 110, 117, 264]. Moreover, we also identified several fault injections recommended by vendors of both the EA and HAC that we implemented in the testbed [190, 55, 264]. A typical failure in the HAC is at a single resource-level, but multiple failures can also occur across two or more resource groups. Further, we created additional fault injections to capture failures related to the characteristics introduced in Chapter 6, e.g., application-provided self-healing capabilities or a

³<https://github.com/ps234/Project>

Evaluation

weak (noncritical) resource. Hence, the following types of failures are included in our evaluation of the BP framework:

1. A critical resource fails.
2. A resource with application-provided self-healing capabilities fails.
3. A noncritical resource fails.
4. Two resources in the same resource group fail.
5. Two resources in two different resource groups fail.
6. A resource repeatedly fails.
7. A shared-dependency resource fails.

Moreover, with each such failure, we also considered the hierarchical position of the failed resource and the resource type. Consequently, fault injection approaches were designed to simulate failures either manually or automatically. Finally, the identified fault injections were included in test cases to capture comprehensive information related to conditions, rules and policies associated with the HAC, protected EA and deployment environment.

10.2.1.2 Test Cases

Two groups of test cases were created. The first group consists of nine test cases, and the second group has 10 test cases. The data sets generated by the first group were used to train and test the model (BN-HAC), and the data sets from the second group were used to supply online testing data to the models in the framework evaluation experiments. Each test case consists of one or more fault injections. Usually, only one fault injection is associated with a test case. However, when two faults are injected on two different resources, the test case consists of two fault injections. The resources in such a case could come from the same resource group or two resource groups (scope). Table 10.7 lists all test cases used in our evaluation. The table shows the criticality, dependency level (D-level), scope, whether the test case was reiterated multiple times or not (R), the resource type (RT) affected by the failure, and whether the test case is a direct measurement of learning performance (LP) or not. Each test case was repeated multiple times to draw a statistically significant conclusion.

The test case T1 was performed on a single critical resource, and its scope was within the same group. Test case T2 was performed on a weak resource. The resource was noncritical, had a medium dependency level, and the scope was within the resource group. There was no reiteration of the test case, meaning that the test case was not repeated several times consecutively on the same resource. In contrast, Test Case T7 injected a fault on two different resources across two resource groups, which is indicated in the table by the scope, and which is set to two. Test Case T3 indicates that the test case was repeated five times to observe the outcome. For example, if the time-out threshold was reached

Table 10.7 Overview of test cases

TC ID	Fault Injection	Scope	R	RF	DT	DLD	DLU	CF	RT	LP
T1	Kill a key process of a service	1	Y	Y	L	L	L	Y	S	Y
T2	Simulate disk crash	1	N	N	L	L	H	N	F	Y
T3	Stop a service in OS-level	1	Y	N	L	L	L	Y	S	N
T4	Kill the main database processes	1	N	N	L	L	L	Y	S	N
T5	Simulate disk crash	1	N	N	L	L	H	Y	F	N
	Kill a key process of a service		N	Y	L	L	L	Y	S	N
T6	Kill a key process of a service	1	N	Y	L	L	L	Y	S	N
	Simulate disk crash		N	N	L	L	H	N	F	N
T7	Simulate disk crash	1	N	N	L	L	H	N	F	N
	Simulate disk crash		N	N	L	L	H	N	F	N
T8	Simulate disk crash	2	N	N	L	L	H	Y	F	N
	Stop a service in OS-level		N	N	L	L	L	Y	S	N
T9	Kill a key process of a shared service	1	N	N	S	L	L	Y	S	N
T10 (T1)	Kill a key process of a service	1	Y	Y	L	L	L	Y	S	Y
T11 (T2)	Simulate disk crash	1	N	N	L	L	H	N	F	Y
T12	Simulate disk crash	1	N	N	L	L	H	Y	F	N
T13	Simulate disk crash	1	Y	N	L	L	H	Y	F	N
T14	Simulate disk crash of the database	1	N	N	L	L	H	Y	F	N
T15	Stop a service in OS-level	1	N	N	L	L	L	Y	S	N
	Simulate disk crash		N	N	L	L	H	Y	F	N
T16	Simulate disk crash	1	N	N	L	L	H	Y	F	N
	Simulate disk crash		N	N	L	L	H	N	F	N
T17	Simulate disk crash	1	N	N	L	L	H	N	F	N
	Simulate disk crash		N	N	L	L	H	N	F	N
T18	Simulate disk crash	2	N	N	L	L	H	Y	F	N
	Kill a key process of a service		N	Y	L	L	L	Y	S	N
T19	Simulate disk crash of a shared resource	1	N	N	L	L	S	Y	F	N

TC - Test case.

Fault injection methods: Kill a key process of a service - kill forcibly a key process of an application service. Simulate disk crash - unmount the file system forcibly while killing all the processes using the disk. Kill the main database server processes (kernel) - kill the kernel processes of the database at OS level. Stop a service in OS-level - a service with a large number of processes is stopped. Kill a key process of a shared service - a key process of the shared service is terminated. Simulate disk crash of the database - unmount the file system forcibly while killing all the processes using the disk. Simulate disk crash of a shared resource - unmount the file system forcibly while killing all the processes using the disk.

Scope: 1-within a resource group, 2-within two resource groups. Reiteration (R): Y - the test case is executed consecutively five times, N - one-off execution. Redundancy factor (RF): T - true, F -false. Dependency type (DT): L - local, S - shared, G - global. Dependency levels down (DLD): L - low, H - high. Dependency levels up (DLU): L - low, H - high. Critical factor (CF): T - true, F - false. Resource type (RT): S - Service, FS - File system. Learning performance (LP): whether part of the direct measurement of learning performance Y - Yes, N - No.

Table 10.8 Full details of test case T1

Parameter	Value
Test ID	T1
Test Case	A single resource fails
Scope	Within a resource group
Category	Recoverable, no failover
Reiteration	No
Constraints	Policies associated with failover preferences are removed
Prerequisites	The HAC is running
Expected outcome	The process is restarted automatically by the service
Resource id	C1G1A1B1
Resource name	Message and lock service
HAC Resource name	rsc_DEV_ASCS00
Group ID	C1G1A1
Cluster id	cluster1
Node id	1
Resource type	Service
Reinitialization factor	Reinitialisable
Redundancy factor	No
Dependency type	Local
Critical factor	2
Dependency type	2
Critical factor	3
Dependency level	1
Dependency depth	1
Dependency levels up	1
Dependency levels down	1
Fault injection method	Execute the script to kill the key process of the service
Script	script_01

for reinitialisation, the HAC could take a different mitigation action than for a single failure. The complete list of all details captured in a test case is illustrated, for test case T1, in Table 10.8.

10.2.2 Data Sets

We recorded the following information during the execution of the tests cases mentioned in the previous section: name of the failed resource, event timestamp, the related resource group and the characteristics that were introduced in Chapter 6. The characteristics are: reinitialisation factor, location of the resource in the hierarchy and the number of resources above and below the failed resource, type of the resource, failure repetition count, criticality and self-healing capabilities provided by the application.

The execution of the BP framework uses two data sets at any given time. For example, the BDN-HAC model uses Data Set 1 as an input into the model, and the BFPF prepares this data set

after extracting it from the HAC log. The output of the BDN-HAC model becomes Data Set 3, which is then used as an input to the BN-HAC model. We created 12 data sets and organised them into four groups: (1) training; (2) production; (3) derived; and (4) availability. We further associate the data sets in groups 1 and 2 with a stickiness policy of the HAC, as described in Section 10.2.1.2, to ensure that HAC failover preferences are considered in the evaluation. Therefore, we used two cases when creating the data sets, stickiness policy enabled and stickiness policy disabled.

The first group of data sets focuses on training by parameter learning and consists of Data Sets 1, 2, 3 and 4, which were obtained by running test cases T1–T9. Data Sets 2 and 4 were created with the stickiness policy enabled, while Data Sets 1 and 3 were created with the stickiness policy disabled. The second group supports inference using production data and comprises Data Sets 5, 6, 7 and 8, which were obtained by executing test cases T10–T19. Data Sets 7 and 8 were created using the stickiness policy enabled, and Data Sets 5 and 6 were obtained with the stickiness policy disabled. The objective of the third group is to test the correlation between parameter learning and data sizes. Hence, Data Sets 9 and 10 were created by inputting Data Set 1 into the BDN-HAC model and adding a different number of instances with the state nonfailure. The fourth group was created to measure MTTR and availability over time. Hence, the Data Sets 11 and 12 obtained failure instances from Data sets 6 and 8, respectively.

The complete list of data sets produced by the experiments is presented in Table 10.9, and we detail each data set as follows:

1. Data Set 1 was obtained by executing test cases T1–T9 twice in the HAC with the stickiness policy disabled. The resulting HAC log file was extracted, processed and prepared by the BPFPP because the primary objective of this data set was to provide data to the BDN-HAC model. The HAC failures identified and prepared by the BPFPP were duplicated to test the prediction quality of the BDN-model and the BN-HAC model.
2. Data Set 2 was acquired by executing test cases T1–T9 twice in the HAC with the stickiness policy enabled. The resulting HAC log file was extracted, processed and prepared by the BPFPP to be inputted into the BDN-HAC model. We duplicated the output from the BPFPP to validate the prediction quality of the BDN-HAC and BN-HAC models.
3. Data Set 3 was created by retaining those instances from Data Set 1 that the BDN-HAC model predicted to be associated with locally unmanageable failures. These data samples were replicated 15 times to obtain sufficient failures for training the BN-HAC model. Moreover, to ensure that the failures were distributed evenly, we inserted additional nonfailure data samples, one for each 20s to emulate data for a week with 20s interval.

Data Set 4 was created the same way as Data Set 3 but by obtaining unmanageable failures from Data Set 2. We replicated data samples 15 times and used the same configuration as Data Set 4 to obtain additional data samples.

Evaluation

Table 10.9 Overview of the data sets

Data Set	Model	#Nodes	#Instances	#Failures	Test Cases	Scope	Dependency	S [†]	Group
1	BDN-HAC	12	72	48	T1-T9	Tr	None	N	1
2	BDN-HAC	12	68	48	T1-T9	Tr	None	Y	1
3	BN-HAC	24	30240	180	T1-T9	Tr	Data Set 1	N	1
4	BN-HAC	24	30240	180	T1-T9	Tr	Data Set 2	Y	1
5	BDN-HAC	12	36	26	T10-T19	Pd	None	N	2
6	BN-HAC	24	10	10	T10-T19	Pd	Data Set 5	N	2
7	BDN-HAC	12	37	28	T10-T19	Pd	None	Y	2
8	BN-HAC	24	10	10	T10-T19	Pd	Data Set 7	Y	2
9	BN-HAC	24	672	180	T1-T9	Tr	Data Set 1	N	3
10	BN-HAC	24	10080	180	T1-T9	Tr	Data Set 1	N	3
11	BPF	24	10080	10	T10-T19	Pd	Data Set 6	N	4
12	BPF	24	10080	10	T10-T19	Pd	Data Set 8	Y	4

#Nodes: number of nodes in the model, #Instances: total number of obtained records, #Failures: number of failures for further analysis, Test cases: T1-T9 - part of producing training (learning), 10-T19 part of producing production data, Scope: Tr - Test/training, Pd - Production data, Dependency: dependency to another data set, [†] (Stickiness): Y - used, N - not used, Group: 1 - Training, 2 - Production, 3 - Derived, 4 - Availability.

Dependency implies that the previous data set was used to create a new data set.

For Data Sets 3, 4, 9 and 10, the number of instances reflects the frequency of the collection in seconds, k = 20s for Data Sets 3 and 4, k = 60s for Data Set 10 and k = 900s for Data Set 9.

For Data Sets 11 and 12, the number of instances reflects the frequency of the collection in seconds, k = 60s.

- Data Set 5 was acquired by executing test cases T10-T19 twice in the HAC with the stickiness policy disabled. The resulting HAC log file was extracted, processed and prepared by the BPF to deliver the data to the BDN-HAC model.

Data Set 6 was obtained by obtaining those instances from Data Set 5 that the BDN-HAC model predicted to be locally unmanageable failures.

- Data Set 7 was obtained by executing test cases T10-T19 twice in the HAC with the stickiness policy enabled. The resulting HAC log file was extracted, processed and prepared by the BPF to be inputted into the BDN-HAC model.

- Data Set 8 was created from the outcome of inputting Data Set 7 to the BDN-HAC model. Therefore, it has only unmanageable failures.

- Data Set 9 was created from the outcome of Data Set 1 from the BDN-HAC model, and as such, it has only unmanageable failures. The failure information was replicated by a factor of 15. We then inserted a new instance for every 900s to produce data for a week with a 900s interval.

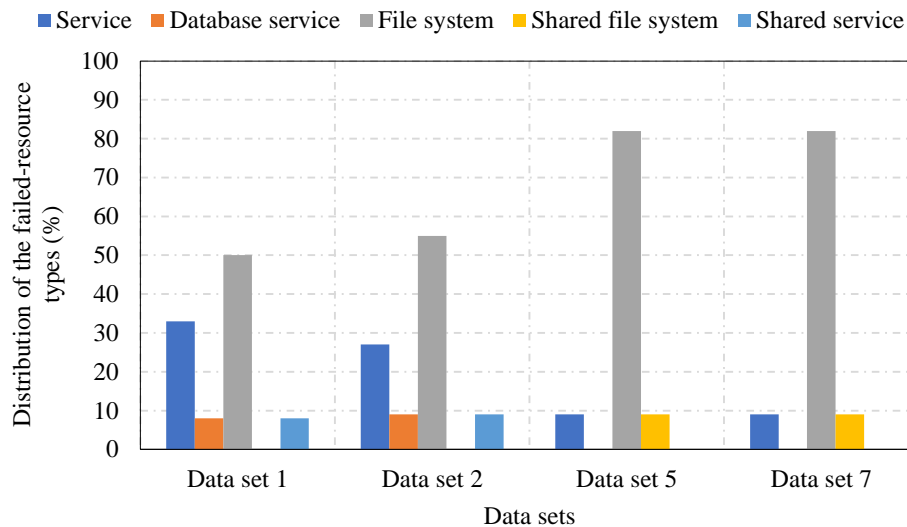


Figure 10.4 Distribution of the failed-resource types in data sets.

8. Data Set 10 was created similarly to Data Set 9 and using the outcome of Data Set 1 from the BDN-HAC model. However, the nonfailure instances were inserted for every 60s instead.
9. Data Set 11 was created using failure data from Data Set 6, which was updated with new instances to represent nonfailure data. The total number of instances was 10080 to emulate continuous data for a week with a one-minute interval to represent availability over time. The instances were updated with the percentage of availability that the system experienced during a failure. Hence, the failure instances received 0% while those without failures received 100%.
10. Data Set 12 was created in the same manner as Data Set 11, but it obtained failure data from Data Set 8 instead.

10.2.3 Data Set Analysis

We analysed Data Sets 1, 2, 5 and 7 because they are the source data sets obtained directly from the HAC logs. The types of the failed resources from the HAC determine what mitigation action can be initiated. Other characteristics that can influence the mitigation actions are as follows:

- position in a resource hierarchy,
- dependency relationships with other resources,
- whether the resource can be reinitialised,
- criticality,
- an application providing the self-healing capability.

Evaluation

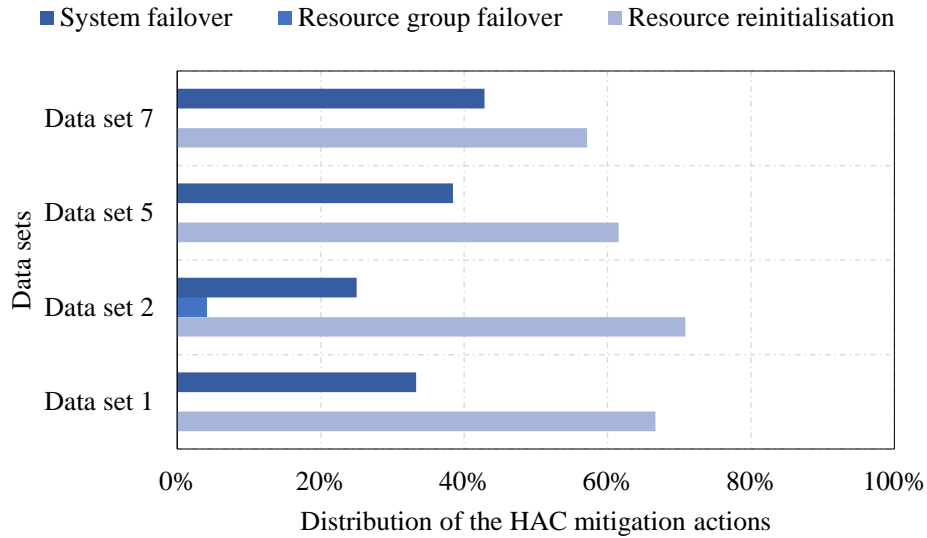


Figure 10.5 Distribution of mitigation actions based on the threefold strategy to handle failures by the HAC.

Table 10.10 Mean execution time in seconds for three mitigation steps of the HAC in the testbed

	Resource Reinitialisation	Resource Group Failover *	System Failover
Data Set 1	26	836	927
Data Set 2	97	107	2498
Data Set 5	15	no	824
Data Set 7	11	no	1142
Mean	37	236	1348

no - no occurrences of resource group failover, * the failure of the distributed lock manager (DLM) group E_1 always resulted in a system failure; hence, it is not presented in the table.

In the case of a resource group, the dependency on other resource groups is considered. Figure 10.4 illustrates the distribution of the resource types in Data Sets 1, 2, 6 and 7.

The mitigation actions taken by the HAC to resolve different failures are presented in Figure 10.5. In most cases, the HAC could reinitialise the resources, avoiding any downtime for the EA.

There are significant differences in the time required for reinitialisation, resource group failover or system failover. For example, a file system requires 15s to reinitialise; however, the same resource can require 30s in other circumstances. Some complex resources (e.g., a database service) take more than two minutes to complete reinitialisation. Similarly, the time required to complete a resource group failure also differs based on the type of resources in the group. A system failover is always the most expensive mitigation action because it requires all resource groups to be stopped in the source node and restarted in the target node. Therefore, the system failover is the least preferred. We analysed the

		Actual outcome	
		Positive	Negative
Prediction outcome	Positive	True positive (TP)	False positive (FP)
	Negative	False negative (FN)	True negative (TN)

Figure 10.6 Contingency table for the basic metrics.

execution time for each mitigation action from the data sets and presented the mean values for each action in Table 10.10.

10.2.4 Evaluation Metrics

Several metrics are used to evaluate the BP framework. Of these, some are relevant to individual modules, whereas others are appropriate for the complete BP framework. Achieving high prediction quality for the individual models, BDN-HAC and BN-HAC, is the basis for ensuring high prediction quality for the complete framework. When measuring the prediction quality, four outcomes are possible, and we refer to them as *basic metrics*, which are used to define *derived metrics* [234]. Figure 10.6 presents the contingency table with the basic metrics, and a concise description of each outcome is provided as follows:

- True positive (TP)
Both the prediction and the actual result are positive.
- False positive (FP)
The prediction is positive, whereas the actual result is negative.
- False negative (FN)
The prediction is negative, whereas the actual result is positive.
- True negative (TN)
Both prediction and the actual result are negative.

The derived metrics used in our evaluation and defined in terms of the basic metrics are presented in Table 10.11. We also present several other relevant metrics below that we use to evaluate the BP framework from additional perspectives:

Evaluation

Table 10.11 Metrics derived from the basic metrics, respective symbols, formulas and descriptions

Metric	Symbol	Formula	Description
Recall or sensitivity or true-positive rate (TPR)	r, tpr	$\frac{TP}{(TP+FN)}$	Correctly predicted failures/all true failures
False positive rate (FPR)	fpr	$\frac{FP}{(FP+TN)}$	Incorrectly predicted failures/all non failure
Accuracy	ac	$\frac{(TP+TN)}{(TP+TN+FP+FN)}$	Correctly predicted failures/all predictions
Precision	pr	$\frac{TP}{(TP+FP)}$	Correctly predicted failures/all predicted failures
Specificity	$1 - fpr$	$\frac{TN}{(TN+FP)}$	Correctly predicted nonfailures/all non failures
F-measure	fm	$\frac{2 \times pr \times tpr}{pr + tpr} \in [0, 1]$	Weighted harmonic mean of precision and recall
MCC [†]	mcc	$\frac{(TP \times TN - FP \times FN)}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$	Provides a balanced measure to measure the quality of binary classifications

[†]MCC - Matthews correlation coefficient

M1. Receiver operating characteristics (ROC) curve. The ROC analysis and resulting ROC curves are used to evaluate the prediction quality of a binary model or compare the prediction quality of multiple binary models [166]. The ROC curve depicts the trade-offs between the *TPR* (sensitivity) and *FPR* (1– specificity). Hence, the curve presents the *TPR* on the vertical axis and the *FPR* on the horizontal axis, and the corresponding area under the curve (AUC) represents the area underneath the ROC curve. The AUC ranks randomly chosen positive predictions higher than randomly chosen negative predictions. If $TPR \approx 1$ and $FPR \approx 0$, it indicates the model performs well [68]. The following paragraph describes the experimental design for ROC analysis to measure the prediction quality of the BDN and BN models.

We assumed a 95% confidence interval for measuring the prediction quality using the ROC analysis. Further, we used a state variable to indicate the binary outcome and used 0 to indicate failure and 1 for no failure using the expected outcome (Section 10.2.5). Thus, the binary value was given as the "value of the state variable" when plotting the ROC curves, which means 0 represents both resource group and system failures. Subsequently, we mapped the outcomes from the HAC and BDN models using a mapping table derived from the threefold strategy that HACs use to deal with failures. These values were entered as test variables for HAC,

Table 10.12 Area under the curve intervals and their interpretations

AUC Interval	Interpretation
0.9-1.0	Excellent (A)
0.8-0.9	Good (B)
0.7-0.8	Fair (C)
0.6-0.7	Poor (D)
0.5-0.6	Fail (F)

Table 10.13 Mapping values used in plotting the receiver operating characteristic (ROC) curves

Mapping Value	Description
1	System failover
2	Resource group failover
3	No failure [†]

[†] This implies that reinitialisation is enabled; thus, HAC can reinitialise the resource.

BDN-HAC and BN-HAC. Therefore, if the HAC performs a system failover, this outcome is mapped to 1. Similarly, if the HAC performs a resource group failover, the mapping value becomes 2, and then 3 for no failure, indicating that the resource in question is successfully reinitialised. The BDN-HAC outcomes are mapped to either 1 or 3 to indicate failure and no failure, as it deals only with binary outcomes. For example, if the HAC performs a system failover and the BDN-HAC predicts no failure, these values are mapped to 1 and 3, respectively. On the other hand, the BN-HAC output can be mapped to all three values as listed in table Table 10.13. If the BN-HAC predicts a resource group failover is required, the test variable is set to 2. Using this procedure, we used the actual outcomes to map and plot the ROC curves using SPSS Statistics.

- M2. **Utility analysis.** The objective of the utility analysis is to compare utility outcomes between the two variants of our BDN model. The resulting curve presents utility outcomes on the vertical axis as a function of test cases on the horizontal axis.
- M3. **Strength of influence.** The strength of influence is computed using the CPTs of the child nodes, and therefore it shows the distance between the conditional probabilities conditioned on the state of the parent nodes [192]. The objective of the measurement is to compare our two BDN models to ensure that the conditional probabilities are accurately represented even when weights are considered in the child nodes. Furthermore, we employ the measurement to ensure that the conditional probabilities reflect the expected influence between child and parent nodes (resources and resource groups) in the BN model. Two types of measurements are used: Euclidean and Hellinger. The Euclidean distance measures the absolute difference between probability distributions, whereas the Hellinger distance focuses on relative differences [132].

Evaluation

Table 10.14 Execution time metrics and the related notation associated with the high-availability cluster (HAC), Bayesian decision network (BDN-HAC) model and Bayesian network (BN-HAC) model

Notation	Scope	Description
Δt_o	Common	Period of normal operation
t_f	Common	Time instant of suspected failure
Δt_d	Common	HAC detection window
Δt_r	Baseline	HAC resource reinitialisation window
Δt_g	Baseline	HAC resource group failover window
Δt_s	Baseline	HAC system failover window
Δt_i	BPF	BP framework preparation window for resource
Δt_j	BDN-HAC	BDN-HAC prediction window for locally manageable resource
Δt_k	BN-HAC	BN-HAC failure propagation and prediction window

Table 10.15 Runtime metrics and the related notation for the individual steps of the Bayesian decision network (BDN-HAC) model and Bayesian network (BN-HAC) model

Notation	Scope	Description
Δt_{ja}	BDN-HAC	BDN input time
Δt_{jb}	BDN-HAC	BDN inference time
Δt_{jc}	BDN-HAC	BDN output time
Δt_{ka}	BN-HAC	BN input time
Δt_{kb}	BN-HAC	BN inference time
Δt_{kc}	BN-HAC	BN output time

10.2.4.1 Evaluation Metrics for Runtime Overhead and Execution Time

We measure the execution times for the each step of the activities carried out by the baseline HAC solution and by the modules of our BP framework as shown in Figure 9.1 earlier in the thesis and summarised in Table 10.14. The scope ‘common’ indicates that such a metric is used by both the HAC and BP framework modules. For example, the HAC detection window is used by both, and it is therefore described as ‘common’. The scope ‘baseline’ shows metrics associated with the HAC runtime, whereas the name of a module from our BP framework indicates a module-specific metric. We note that the threefold strategy employed by HACs is associated with the three ‘baseline’ execution times, where Δt_r , Δt_g and Δt_s correspond to the first strategy of reinitialising a resource, the second strategy of resource group failover and the third strategy of system failover, respectively. Thus, the system failover time (S) for the baseline HAC solution can be calculated as follows

$$S = \Delta t_d + \Delta t_r + \Delta t_g + \Delta t_s. \quad (10.1)$$

Table 10.16 Metrics and the related notation associated with the components of the Bayesian prognostic framework preparation (BFPF) module

Notation	BFPF Component	Description
Δt_{ia}	2	Polling frequency for extracting log entries
t_{ib}	2	Time to query the configuration information
t_{ic}	2	Time to extract
t_{id}	2	Time to parse the log entries into a structured form
t_{ie}	2	Time to enrich data by obtaining more information from the configuration and runtime environment
t_{if}	3	Time to complete the transformation of certain values
t_{ig}	3	Time required for conversion
t_{ih}	4	Time to complete filtering

BFPF components: 2 - log interface, 3 - transformation and conversion, 4 - filter.

We further break down the module-specific metrics for our BP framework into multiple metrics that measure the execution time and computational overhead at a fine-grained level. Table 10.15 lists the metrics associated with the module metrics of BDN-HAC and BN-HAC. Each module-specific metric is broken down into three metrics. For example, the module metric associated with the BN-HAC comprises *BN input time*, *BN inference time* and *BN output time*. Thus, *input time* is the time required to invoke and deliver data to the model, and *inference time* is the model inference time. In contrast, *output time* is required to prepare the model output (e.g., interpreting the utility value of the BDN model and sending it to the next model). Similarly, the BN-HAC model metric comprises three metrics with the same meaning as for the BDN-HAC model metric. The following equations present the connection between the module metrics and their components

$$\Delta t_j = \Delta t_{ja} + \Delta t_{jb} + \Delta t_{jc}, \quad (10.2)$$

$$\Delta t_k = \Delta t_{ka} + \Delta t_{kb} + \Delta t_{kc}. \quad (10.3)$$

M4. Runtime overhead of the BFPF module. We break down the four runtime-related components of BFPF presented in Chapter 8 to define several execution time metrics for the BFPF components used at runtime (Table 10.16). The total BFPF execution time can be calculated as follows

$$\Delta t_i = \Delta t_{ia} + t_{ib} + t_{ic} + t_{id} + t_{ie} + t_{if} + t_{if} + t_{ig} + t_{ih}. \quad (10.4)$$

Evaluation

Δt_{ia} is considered even though the execution time depends on the polling frequency because it helps measuring the total time required to execute the complete BPF.

M5. MTTR evaluation. We use two MTTR representations to evaluate the improvements provided by the BP framework. The first, $MTTR_{base}$, represents the metric associated with the baseline, which measures the time to recovery by the HAC until the application is fully operational. The second, $MTTR_{BPF}$, measures the time to recovery when the BP framework is employed. In general, an MTTR is derived from all $N > 0$ TTR (time to recover) durations within a specific time window and can be formally expressed as follows

$$MTTR = \frac{1}{N} \sum_{i=1}^N TTR_i. \quad (10.5)$$

The first MTTR representation for baseline can be expressed as follows

$$MTTR_{base} = \frac{1}{N} \sum_{i=1}^N TTR_{base,i}, \quad (10.6)$$

where $TTR_{base,i}$ represents the TTR for the i -th failure within the time window used to calculate $MTTR_{base}$. Note that this TTR always includes the terms $\Delta t_d + \Delta t_r$. Additionally, if a resource group failover is performed by the HAC, the term Δt_g will also be included in the calculation. Finally, if a system failover is required, the term Δt_s will be included in the TTR calculation as well.

The second representation that describes the MTTR when the BP framework is employed is expressed as follows

$$MTTR_{BPF} = \frac{1}{N} \sum_{i=1}^N TTR_{BPF,i}, \quad (10.7)$$

where $TTR_{BPF,i}$ represents the TTR for the i -th failure within the time window used to calculate $MTTR_{BPF}$. In the worst-case scenario, this TTR includes all the time periods: Δt_d , Δt_r , Δt_g , Δt_s , Δt_i , Δt_j and Δt_k , but fewer time periods may be needed, e.g., only Δt_d , Δt_r , Δt_i and Δt_j , when the BDN-HAC module correctly predicts that the i -th failure of a resource can be managed locally.

M6. Analysis of availability over time. The objective of this metric is to evaluate the availability of the application using the previously defined MTTR representations, as described in M5. Thus, percent availability is measured using the following equations [167]

$$availability(HAC_{base}) = \left(\frac{MTBF}{MTBF + MTTR_{base}} \right) \times 100\%, \quad (10.8)$$

$$availability(HAC_{BPF}) = \left(\frac{MTBF}{MTBF + MTTR_{BPF}} \right) \times 100\%. \quad (10.9)$$

In these calculations, the values of MTBF are the same for the two measurements because using the BP framework (or not) does not influence the operation of the IT application between failures. Equation (10.8) calculates the availability over time for the baseline, and eq. (10.9) presents the measurement when the BPF is employed.

- M7. **Runtime overhead.** Two metrics are captured: CPU and memory utilisations. These metrics are used when evaluating the performance of the individual components of a BP framework module or the complete module (e.g., the HMTA tool).
- M8. **Execution time.** The aim is to capture the execution time of individual components of a module and to present the total execution time for a module. For example, to measure the execution time of the BDN-HAC model, the execution time of its three components are captured and added to obtain the entire execution time of the model.

10.2.5 Expected Outcome

Expected outcomes are results that are expected from the HAC when a particular type of resource fails. For example, when a specific type of resource fails, the expected result may suggest that a resource group failover is sufficient to resolve the problem. To assess a typical HAC outcome upon a failure, we identified the set of expected outcomes. The actual outcomes of the HAC, BDN and BN were then checked against the expected outcomes to evaluate the quality of the actions taken by the HAC and the quality of the predictions by the BDN and BN.

To obtain the expected outcomes, we reviewed the results and test cases that the vendors and service providers published for the EA under analysis [190, 110, 55, 117, 175, 263, 264]. Furthermore, we validated these outcomes with further experiments on the HAC in the testbed. Hence, the empirical evidence was obtained using the following three approaches:

1. We analysed the HAC log files from different HAC solutions for the same type of EA architecture.
2. We performed multiple iterations of experiments in the testbed to study the results. For example, we performed failovers manually to ensure that the resource groups can failover independently and we verified that the HAC in question could handle such failures and take suitable mitigation actions.
3. We analysed the data produced by our experiments and derived conclusions. For example, the HAC initiated a system failover when a shared resource failed. However, in the subsequent experiments, the HAC reinitialised the resource and achieved no failure.

10.3 Evaluation of Bayesian Prognostic Framework Modules

10.3.1 Evaluation of the Holistic Modelling Technique for High Availability

This section presents the evaluation of the HMTHA technique described in Chapter 5, and we use the technique to create an HHAM model and the accompanying M-table for the testbed application. The T-rules are reused from Chapter 5. In Appendix C, we present the results from investigating the computational overheads associated with creating an HHAM model, notating here that this represents a one-off, offline activity, and that these overheads are comparable to those of other modelling tools.

10.3.1.1 Evaluation of the Model Construction

The HMTHA is the first step in designing and implementing the BP framework, as illustrated in Figure 9.1. The software tool that we developed (described in Section 5.5) is written in Java [257], which we installed and ran on a computer with a 3.4 GHz Intel Core i7 and 64 GB of memory running Windows 10 64-bit. The database was MySQL 8.0.18 (64-bit).

Technique for Obtaining an Instance of the Model for the Testbed Application The five-step modelling process from Section 5.3.3 was used to identify the components of the testbed environment to create the HHAM model, M-table and T-rules as detailed below.

1. Obtain the high availability application requirements

The first step in the process was to identify the availability requirements for the ERP solution, and we assumed an annual SLA of 99% (Category 1). We identified all critical resources by employing a diverse range of techniques, such as identifying an SPOF component, evaluating single resource failure effects, and performing a risk assessment on the resource-level [49]. Furthermore, documentation and guidelines from the application vendor were also employed to identify the relevant resources. Subsequently, we combined the identified resources into resource groups. There was a requirement to support two non-SPOF resources, file systems for interface and software logistics (Category 2), as part of the HA setup. The deployment pattern was a public cloud so that the capabilities available in that environment could be used (Category 3).

2. Apply the holistic modelling technique for high availability

This section describes the application of the HMTHA tool to create an HHAM model and manually construct the accompanying M-table and T-rules for the testbed environment. The top-level application vertex was identified first. Subsequently, five resource group vertices were added. Next, simple vertices representing service vertices were added using the resource group vertices as parents. A vertex representing the file system was added to all resource group vertices.

Moreover, all resource groups except the resource group representing the DLM have a VIP to enable the relocation of the resource groups individually. The DLM group is a shared group and

Table 10.17 Holistic high-availability model grouped by vertex type

Vertex Type	Model
Application	1
Resource group	5
Weak	2
Simple	13
Global	3
Shared	0
Total	24

presents a shared file system (cluster file system) for all other resource groups. Consequently, the DLM group is responsible for retaining locks even during failures to preserve transactional integrity. Three global dependencies were added to represent the CPU, memory and SBD. The first two were created as global and weak vertices to indicate that failure of the corresponding resources does not lead to failure of any other resources.

In contrast, the SBD vertex was created as a global dependency because a failure of the corresponding resource leads to the failure of the entire system. Considering the Category 2 HA requirements, we added two file systems representing the interface and software logistics as weak vertices. We also added the related arc for each vertex type, and this iteration continued until all required vertices were added.

3. Create/update the holistic high-availability model and M-table

The resulting HHAM model is displayed in Figure 10.7, and Table 10.17 lists the identified vertices for the HHAM grouped by vertex type. The model consists of 24 resources in four layers organised into five resource groups. Six of the resources are logical without physical representation. The three global dependency resources are illustrated using thick ellipses, whereas the two non-SPOF resources are depicted using dashed circles. Using the HHAM, we constructed the first part of the M-table, which is presented in Table 10.18. The table is completed with a second part as part of constructing the BN model, which is described in Section 10.3.3.1. The same T-rules presented in Table 5.3 were reused because the rules were explicitly created to map BN-based models.

4. Enable HA Facilitators

The focus of the HAC solution ClusterLabs stack was to protect the critical components of the ERP solution [21]. The implementation and configuration of the HAC are described in Sections 10.1.7 and 10.1.8.

5. Enable other models

Section 10.3.3.1 describes how the HHAM and accompanying M-table and T-rules are used to create the corresponding BN model.

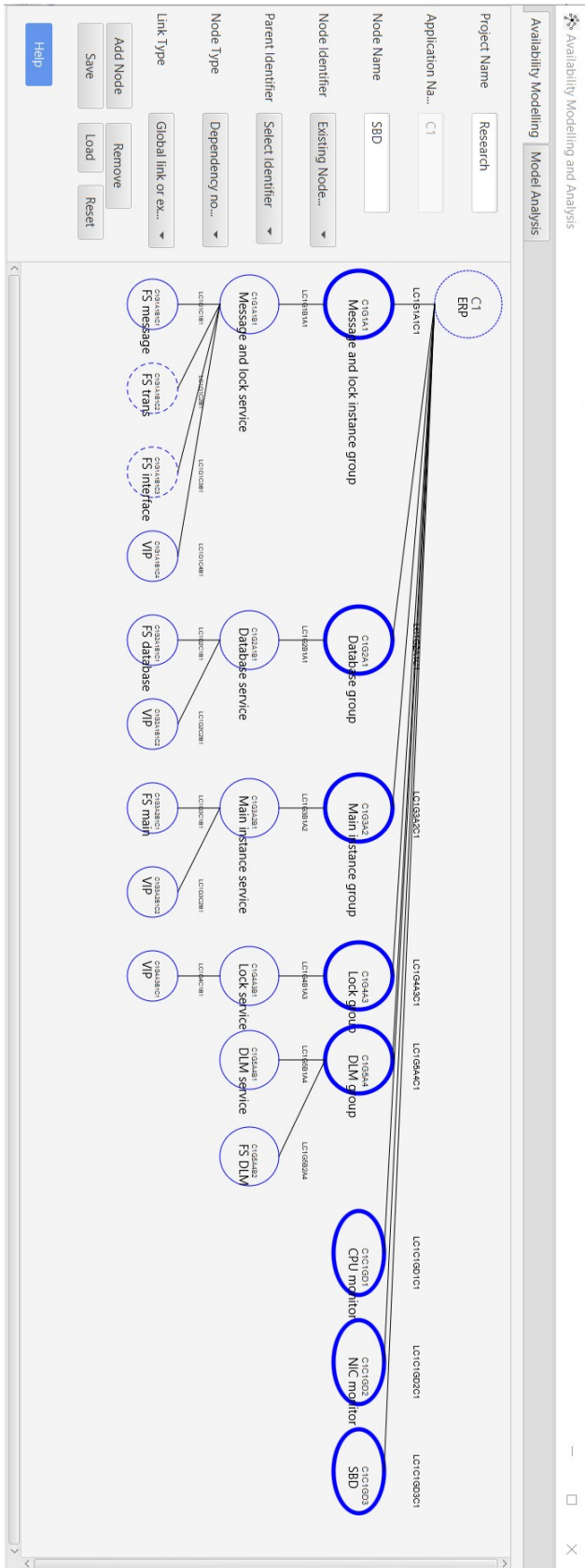


Figure 10.7 Holistic high-availability model (HHAM) for the tested application.

10.3 Evaluation of Bayesian Prognostic Framework Modules

Table 10.18 First part of the mapping table (M-table) for the testbed application

HHAM			
Resource	Vertex Type	Arc Type	Layer
Main application (ERP)	A	A	Top
Message and lock group	RG	RG	a
Message and lock service	S	R	b
VIP	S	R	c
FS message and lock	S	R	c
FS trans	W	R	c
FS interface	W	R	c
Database group	RG	RG	a
Database	S	R	b
VIP	S	R	c
FS database	S	R	c
Main instance group	RG	RG	a
Main instance service	S	R	b
VIP	S	R	c
FS main instance	S	R	c
Backup lock server group	RG	RG	a
Backup lock server service	S	R	b
VIP	S	R	c
DLM group	RG,GD	SG	a
DLM	GD	G	b
FS DLM	GD	G	b
CPU monitor	GW	G	Top
NIC monitor	GW	G	Top
SBD service	GD	G	Top

Vertex Type: A - Application, RG - Resource group, S - Simple, W - Weak, GD - Global dependency, GW - Global dependency, weak.

Arc Type: A - Application, RG - Resource group, R - Resource, SG - Shared resource group, G - Global.

10.3.1.2 Runtime Overhead

Appendix C shows that the memory requirement is cumulative as more vertices are added. The mean utilisation is between 2 and 4 MB per node. Similarly, the CPU utilisation was under 2.5% when adding nodes, which is experienced for a period of 0.6 ± 0.4 s. The conclusion is that scalability is not a concern as modern PCs are powerful and can support constructing a model with many vertices and arcs in a model.

Evaluation

Table 10.19 Details of the two BDN models

Models	Total Nodes	Chance Nodes	Latent Nodes	Utility Nodes	Decision Nodes	ALU Node	States	Parameters
BDN-HAC-1	12	10	2	1	1	N/A	23	89
BDN-HAC-2	12	7	N/A	2	2	1	19	51

N/A - Not applicable.

10.3.2 Evaluation of the Locally Manageable Resource Failure Prediction

The evaluation of the BDN-HAC model is presented in two steps. First, we compared the two models, BDN-HAC-1 and BDN-HAC-2, to identify the most accurate of the two for use within the BDN-HAC module of our BP framework. Next, we evaluated the ability of the selected model to deal with incomplete data, with the difference in prediction quality between data, and with critical nodes versus noncritical nodes between established characteristics (ECs) and new characteristics (NCs).

We present the details of the two implemented BDN models in Table 10.19. Using these models, we used the data sets prepared in Section 10.2.2 to perform the evaluation. All the latent nodes were excluded when the model was inferred to ensure they did not obtain any data. The decision node “*current_state*” was set to offline because the BDN-HAC models were constructed for use after failures of individual HAC resources. Furthermore, we employed policy evaluation as the primary algorithm for BDN inference.

We used the GeNIe modeller v3.0 as the primary tool [18] to construct and test the BDN models. We also employed multiple software solutions such as SPSS Statistics, Power BI and Excel to analyse and visualise the results.

10.3.2.1 Evaluation of the Models

Utility Analysis We used the evaluation metric M2 described in Section 10.2.4 to evaluate the utility values for the two models using Data Set 1 and 2. We selected these two data sets because they cover a broad range of failure types. Utility values are in the prediction outputs when data are supplied to the BDN model as described in Section 4.4. Using the outputs, we plotted the related curves for both models for each data set, which are depicted in Figure 10.8⁴. Figure 10.8 (a) presents the results from Data Set 1, and Figure 10.8 (b) presents the results from Data Set 2. The horizontal lines represent the cut-off values to indicate positive (no failure) and negative (failure) outcomes for the two BDN-HAC models. Any results above the line are positive (failures can be managed locally) and are shown by filled markers, whereas the ones below the line are negative (unmanageable failures) and are shown by empty markers. As the figure displays, we obtained identical predictions from both models, which

⁴The figures only show failures from the original Data Sets 1 and 2 because the duplicated failures provided identical results from the BDN-HAC-1 and BDN-HAC-2 models.

10.3 Evaluation of Bayesian Prognostic Framework Modules

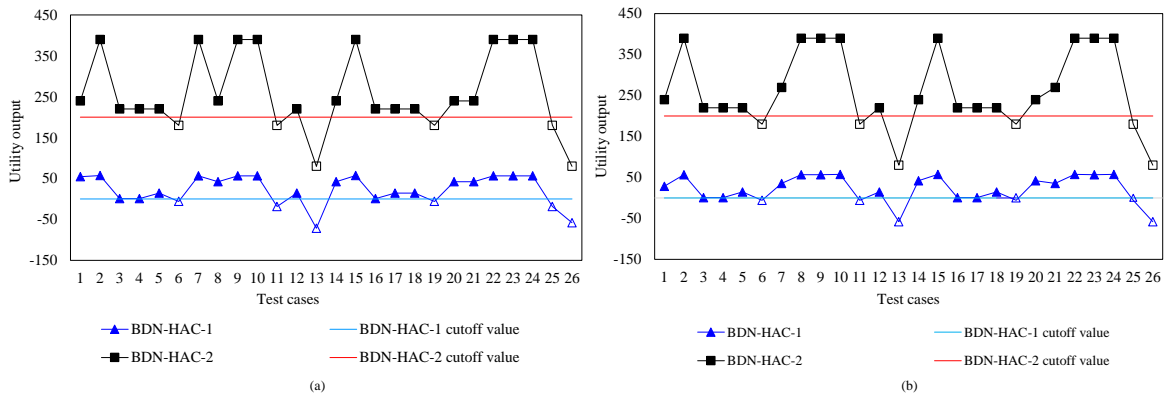


Figure 10.8 Utility analysis results between BDN-HAC-1 and BDN-HAC-2 models for (a) Data Set 1 and (b) Data Set 2. The horizontal cutoff lines at zero utility for BDN-HAC-1 (cf. eq. (6.10)) and at utility 200 for BDN-HAC-2 (cf. eq. (6.17)) separate the positive outcomes (filled markers) and negative outcomes (empty markers).

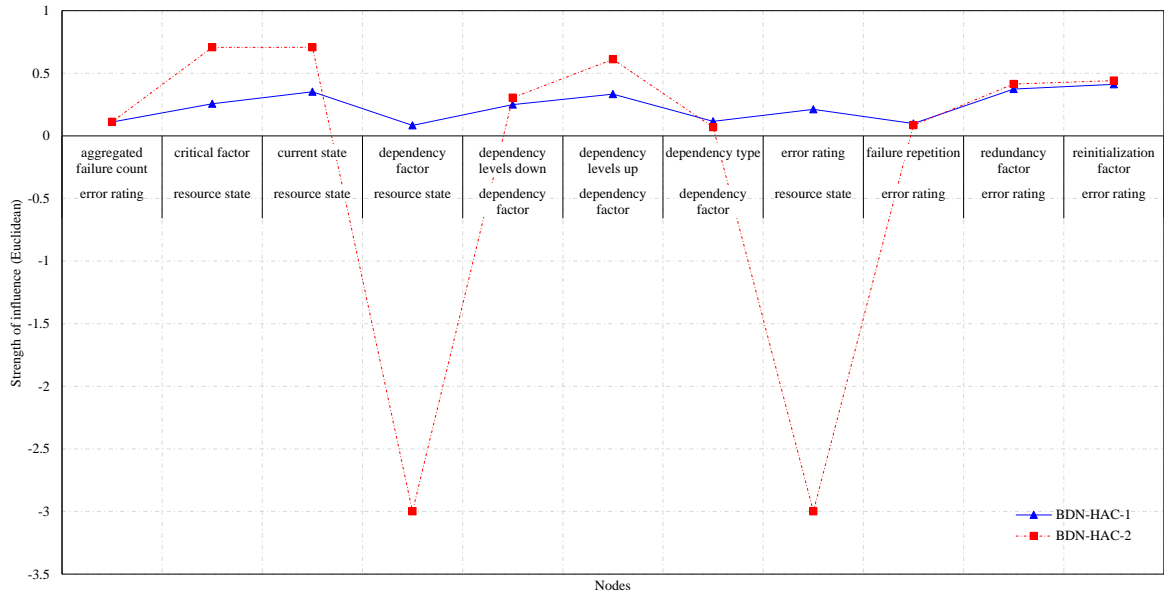


Figure 10.9 Strength of influence for the parent-child node pairs from the two BDN-HAC models.

we validated against the expected outcome (Section 10.2.5), confirming that both models provided accurate predictions for all test cases.

Strength of Influence We used GeNIe Modeler is to compute the strength of influence for the parent-child node pairs from our two BDN-HAC models. Figure 10.9 displays the strength of influence for the two models using the Euclidean measure. The length of the link indicates the influence, where a longer link indicates a stronger influence. Each value is labelled with the names of the corresponding parent node (top label) and child node (bottom label). In a BDN model, the strength of influence should reflect the conditional probabilities in the child nodes. However, this is not the case with the

BDN-HAC-2 model. For example, the parent node ‘dependency factor’ significantly influences the child node ‘resource state’. Similarly, the parent node ‘error rating’ strongly influences the child node ‘resource state’ as the ‘dependency factor’. On the contrary, the influence between the parent node ‘dependency factor’ and the child node ‘resource state’ in the BDN-HAC-1 reflects the intended use correctly. One reason for this behaviour of BDN-HAC-2 could be that BDN-HAC-2 does not have chance nodes for the dependency factor, error rating and resource state; instead, utility nodes significantly reduce the flexibility provided by the use of additional parameters. For example, BDN-HAC-1 has 89 parameters, whereas BDN-HAC-2 works with 51 parameters (listed in Table 10.19). These results suggest that the BDN-HAC-1 can perform the required prediction with better accuracy than BDN-HAC-2.

Selection of a Model The conclusion is that both models performed well in terms of prediction quality. However, BDN-HAC-2 uses three utility nodes, as listed in Table 10.19, and thus it tends to be more deterministic and limits the available options (e.g., the flexibility to define conditional probabilities and to work with more parameters). Moreover, the experimental result from the strength of influence analysis indicates that BDN-HAC-1 reflects conditional probabilities and, thus, more accurately reflects the relationship between parent and child nodes. Therefore, we selected the BDN-HAC-1 model for inclusion into the BP framework. Although we used the HAC in the testbed to select a model, the results are expected to be the same because the BDN-HAC module is a HAC independent module that can work with any HAC solution. As such, only the term ‘BDN-HAC’ is used to refer to this model and a model-specific name (e.g., BDN-HAC-1) is no longer used.

10.3.2.2 Prediction Quality

Using the experimental design for ROC analysis from Section 10.2.4, we plotted the ROC curves for Data Sets 1, 2, 5 and 7 (i.e., for all the data sets that were processed by the BDN-HAC model) as shown in Figures 10.10, 10.11, 10.12 and 10.13. The HAC outcome had 8% FNs in Data Set 1 (Figure 10.10), indicating that the HAC performed a system failover when a service such as a database failed. However, the expected outcome was ‘manageable (no failure)’ because the HAC was expected to reinitialise the resource without triggering a complete system failover.

In contrast, the BDN-HAC model detected the failure and predicted it as a locally manageable failure, and the FN therefore became a TP. Another example is that the HAC performed a resource group failover when the resource ‘main instance service’ was terminated as part of the fault injection in Data Set 2 (Figure 10.11). This event occurred only once, and in the subsequent events, the HAC was able to restart the same service successfully. The BDN-HAC model correctly predicted that the HAC would be able to manage the failure locally (i.e., restart the service). In Data Set 5, both the HAC and BDN model performed identically, reaching a higher AUC (Figure 10.12). In Data Set 7, the HAC initiated unnecessary system failures for a shared resource failure (cluster file system), whereas the BDN-HAC treated the failure as locally manageable (no failure) (Figure 10.13).

10.3 Evaluation of Bayesian Prognostic Framework Modules

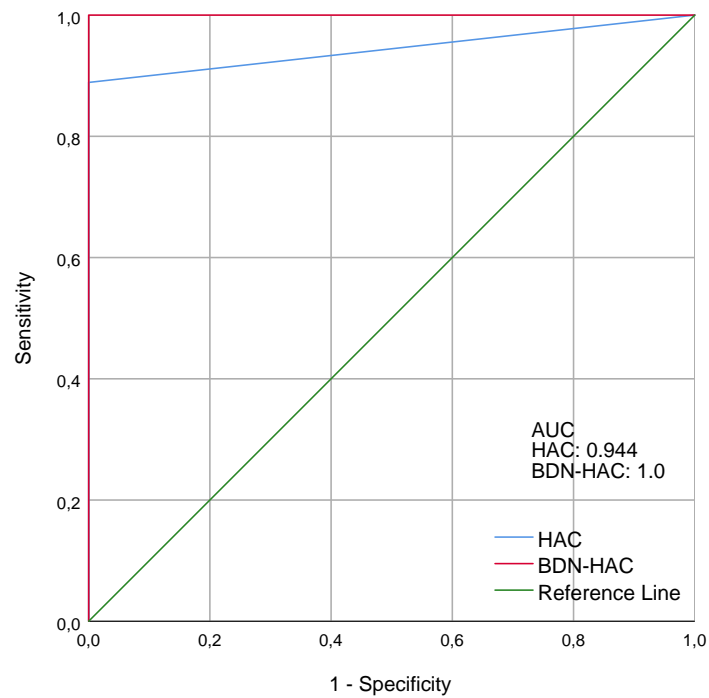


Figure 10.10 Receiver operating characteristic (ROC) curve showing prediction quality for Data Set 1.

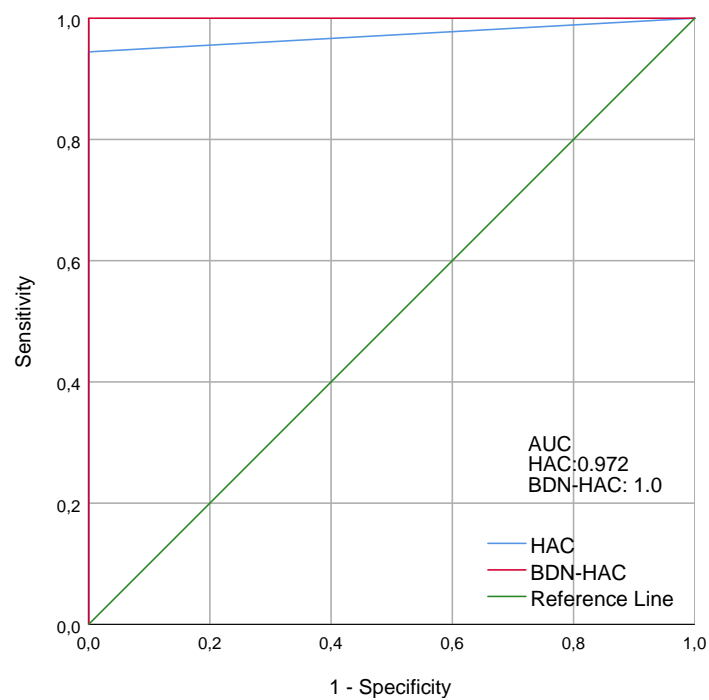


Figure 10.11 Receiver operating characteristic (ROC) curve showing prediction quality for Data Set 2.

We calculated the results from Data Sets 1, 2, 5 and 7, and the BDN-HAC model achieved a mean AUC that was 4.85% better (i.e., higher) than that achieved by the HAC. Furthermore, the BDN-HAC

Evaluation

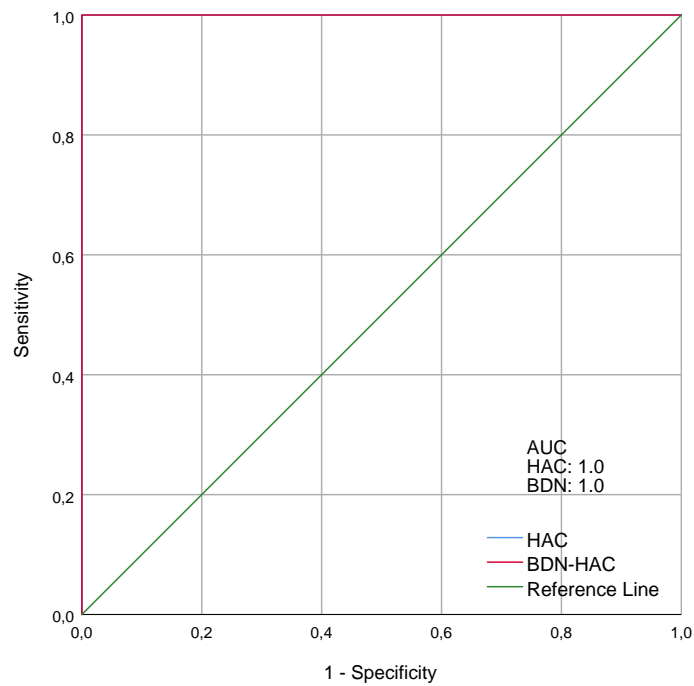


Figure 10.12 Receiver operating characteristic (ROC) curve showing prediction quality for Data Set 5.

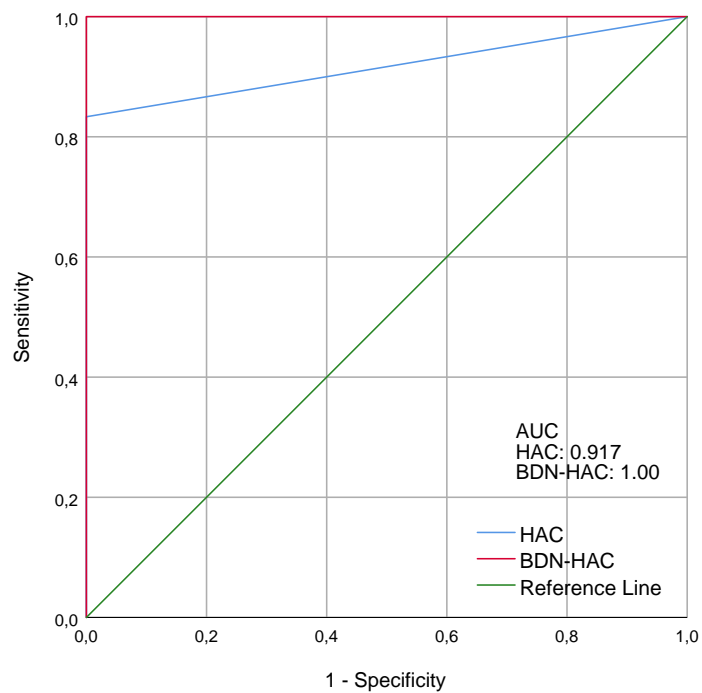


Figure 10.13 Receiver operating characteristic (ROC) curve showing prediction quality for Data Set 7.

model using the data sets without stickiness (Data Sets 1 and 5) performed better than those with

10.3 Evaluation of Bayesian Prognostic Framework Modules

Table 10.20 Summary of the receiver operating characteristic (ROC) analysis grouped by data set

Test Result	AUC	AS	SE	Confidence Interval	
				Lower Bound	Upper Bound
Data Set 1					
HAC result	0.944	0.016	0.000	0.913	0.976
BDN-HAC Data Set 1	1.000	0.000	0.000	1.000	1.000
Data Set 2					
HAC result	0.972	0.011	0.000	0.950	0.995
BDN-HAC Data Set 2	1.000	0.000	0.000	1.000	1.000
Data Set 5					
HAC result	1.000	0.000	0.000	1.000	1.000
BDN-HAC Data Set 5	1.000	0.000	0.000	1.000	1.000
Data Set 7					
HAC result	0.917	0.020	0.000	0.878	0.955
BDN-HAC Data Set 7	1.000	0.000	0.000	1.000	1.000

AUC - Area under curve

AS - Asymptotic significance (null hypothesis: true area = 0.5). It is used to determine the statistical significance of the relationship between the variables. $p < 0.05$ indicates a statistical significance relationship.

SE - Standard error (under the nonparametric assumption)

HAC - High availability cluster (Pacemaker/Corosync HAC)

BDN-HAC - Bayesian decision network for predicting locally manageable resource failures

stickiness (Data Sets 2 and 7). For example, the BDN-HAC model using the Data Set 5 with stickiness achieved the same result as the HAC. The results are summarised in Table 10.20.

Effect of Incomplete Data on Prediction Quality We analysed the effect of incomplete data on the prediction quality by providing data to only a subset of the BDN-HAC nodes (and using the default probability distributions for the other nodes). In a first step, we started with node A_2 . In the second step, we delivered data to both A_2 and A_3 . In the third step, we provided data to three nodes, A_2 , A_3 and A_4 , and we continued until all nodes were supplied with data in the final step.⁵ Figure 10.14 illustrates the model output as each node was provided with data, and it is presented using the accuracy, F1-score and Matthews correlation coefficient (MCC) evaluation metrics. Including MCC overcomes imbalanced classes, as we noted that our data sets are imbalanced since they contain many FPs and FNs. However, the BDN-HAC model aims to eliminate FPs and FNs because the purpose of the

⁵BDN uses the prior probability distribution for the nodes not provided with data, except for the latent and utility nodes.

Evaluation

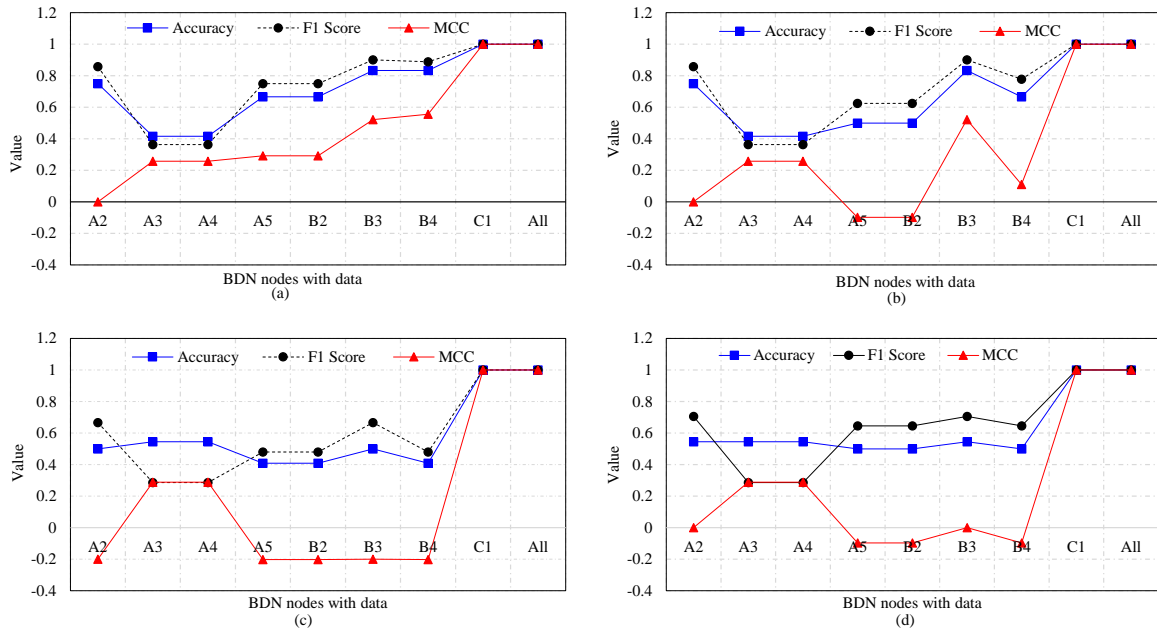


Figure 10.14 Changes in the prediction outcome based on nodes receiving data for (a) Data Set 1, (b) Data Set 2, (c) Data Set 5 and (d) Data Set 7. The labels on the horizontal axis show the last node that was supplied with data, e.g., the values for ‘ B_2 ’ were obtained when the nodes A_2 , A_3 , A_4 , A_5 and B_2 were supplied with data, and all of the other BDN nodes were not.

model is to reduce the downtime for business-critical systems. Therefore, a balanced measure is required when measuring the model quality. Hence, the primary metric for measuring prediction quality with incomplete data is MCC.

A high percentage of FPs was observed when only A_2 obtained data. For instance, the percentage was 27% in Data Set 1 and 45% in Data Sets 5 and 7. This resulted in relatively high accuracy (0.4167, 0.75 and 0.5455, respectively) and F-score (0.3636, 0.8571 and 0.6667, respectively) but a low value for MCC. However, when the second node, A_3 , also received data, a shift from FPs to FNs occurred. We observed 58% FNs in Data Set 1 and 45% FNs in Data Sets 5 and 7. The curve presents this shift by reducing the accuracy to 0.4167 in Data Set 1. However, no change in accuracy or F1-score occurred in Data Sets 5 and 7, but a significant change in MCC occurred. In Data Set 7, 45% of the FPs shifted to 45% FNs in A_3 .

These results show that incomplete data, primarily when very few nodes are supplied with data, provide poor predictions. When only A_1 is instantiated and all the other nodes use prior probabilities, it leads to high FPs. Moreover, the results demonstrate that whenever the next node that receives data is a critical node, it shifts the FPs to FNs in most cases (i.e., A_3 and A_5), which results in negative values for MCC.

Influence of Critical and Noncritical Nodes Figure 10.15 displays a comparison between the prediction quality achieved when only the critical nodes were supplied with data, and that achieved

10.3 Evaluation of Bayesian Prognostic Framework Modules

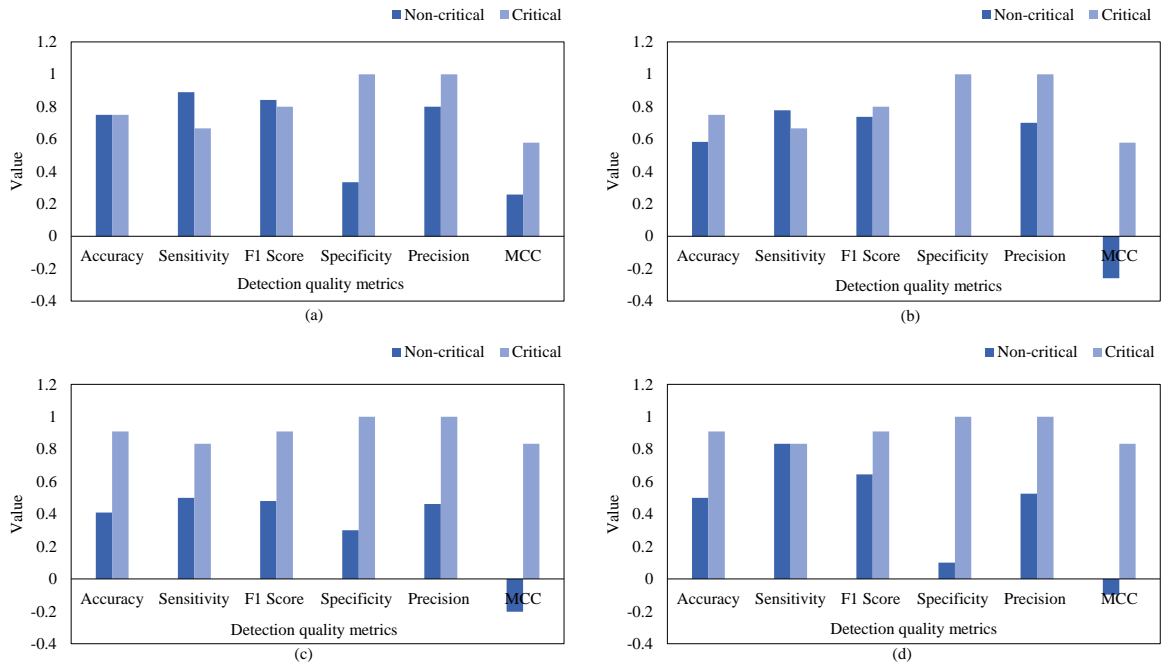


Figure 10.15 Comparison of prediction quality for experiments in which only the critical or only the noncritical nodes were supplied with data for: (a) Data Set 1, (b) Data Set 2, (c) Data Set 5, and (d) Data Set 7.

when only the noncritical nodes were supplied with data. The number of noncritical nodes is five (A_2 , A_4 , B_2 , B_3 and B_4), whereas the number of critical nodes is three (A_3 , A_5 and C_1). However, the three critical nodes have a higher influence than the five noncritical nodes due to weight assignment. In Data Set 1, the data to critical nodes results in 27% FPs, whereas for noncritical nodes, the outcome was 9% FNs and 18% FPs. This results, in general, in higher values for critical nodes across all the evaluation metrics, as depicted in Figure 10.15. A similar pattern was observed in all data sets. For example, we observed 9% FNs for critical nodes in Data Set 7, but the figures were 9% FNs and 40% FPs for noncritical nodes, which results in a negative value for MCC for noncritical nodes. Therefore, when data are supplied only to noncritical nodes, it tends to create more FPs, while when data are supplied only to critical nodes, it creates FNs. The results confirm that critical nodes play a crucial role in providing prediction capabilities for the BDN-HAC model.

Existing vs New Characteristics As described in Chapter 6, the BDN nodes take into account two types of HAC characteristics, namely characteristics also used outside of our project (which we termed ‘existing characteristics’ or ECs) and characteristics proposed by this project (which we term ‘new characteristics’ or NCs). There are three EC nodes (A_2 , A_4 and A_5), and the fourth node is a decision node for which a negative decision to indicate failure is assumed. The NCs consists of five nodes (C_1 , A_3 , B_2 , B_3 and B_4). We investigated the BDN-HAC prediction quality when only EC nodes receive data and compared the result to the scenario when only the NC nodes receive data.

Evaluation

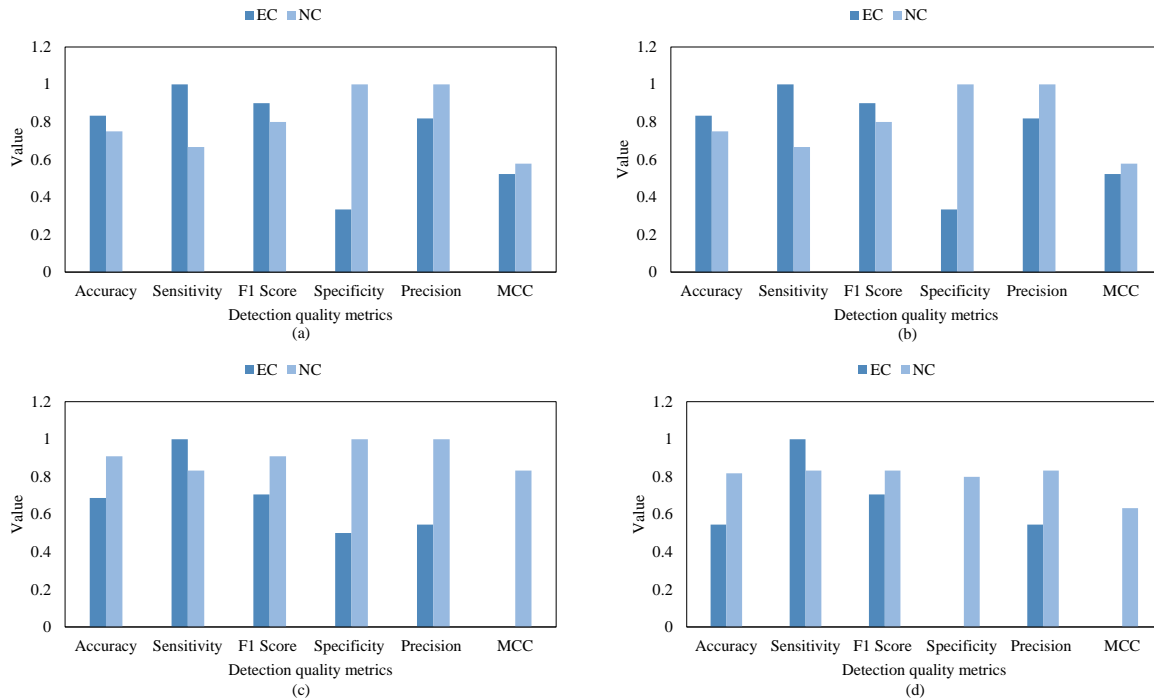


Figure 10.16 Comparison of prediction quality between existing characteristics (ECs) and new characteristics (NCs) for (a) Data Set 1, (b) Data Set 2, (c) Data Set 5 and (d) Data Set 7.

Figure 10.16 depicts the results from the evaluation. There are 17% FPs in ECs compared to 25% FNs in NCs in Data Set 1, and a similar pattern can be observed in Data Set 2. As shown in Figure 10.16 (a) and (b), accuracy is reduced in NCs as a result. However, significant changes exist in Data Set 5, which indicates that ECs have 45% FPs but 9% FNs, resulting in lower values for ECs across all evaluation metrics except for sensitivity.

Similarly, 45% FPs could be observed in ECs, whereas 9% FPs and 9% FNs in NCs could be observed in Data Set 7. However, ECs have only one critical node (A_5), whereas NCs have two (C_1 and A_3), and this could influence the outcome considerably. Overall, the results reveal that NCs improve the prediction quality. For example, if the accuracy is 0.69 with EC nodes, the quality is improved by 32% when NCs are introduced (Data Set 5).

BDN-HAC Model Execution Time Three steps are associated with using the BDN-HAC model to obtain predictions: BDN input, BDN inference and BDN output. The BDN input transfers data to the BDN model to be processed, and the BDN inference performs the actual model execution using data from the previous step. Subsequently, the outcome is interpreted as either a failure or not and is transferred to the BN-HAC model. The mean execution time for each step is depicted for the four data sets in Figure 10.17, which were obtained from running the experiments on a computer with a 3.4 GHz Intel Core i7 and 64 GB of memory running Windows 10. The BDN model requires, on average, 1ms to complete the inference as the model is small (with only 12 nodes) and employs a utility node

10.3 Evaluation of Bayesian Prognostic Framework Modules

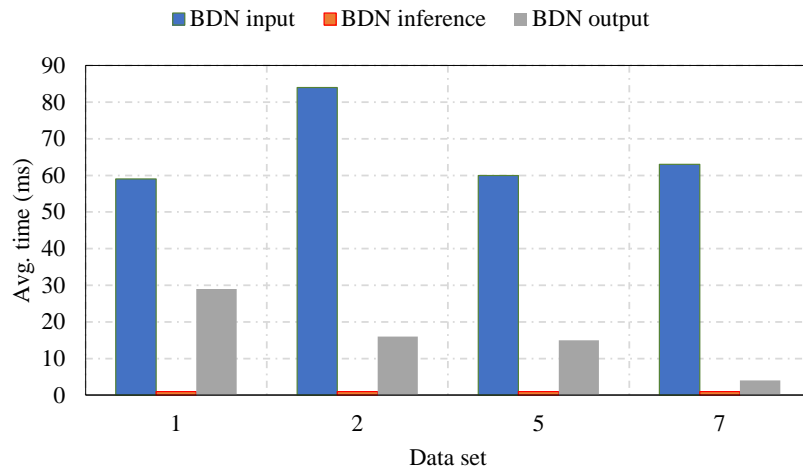


Figure 10.17 Mean execution time for the three steps associated with the invocation of the BDN-HAC model.

with predefined preferences, which reduces the computational complexity. The BDN input and output execution times are higher because they correspond to steps that convert the input/output values of the BDN. Overall, the evaluation result (a mean value of 84ms for all steps and data sets considered, which is 1.3% of the overall BP framework execution time) indicates that the BDN-HAC module's execution time is negligible. Furthermore, a significant part of the execution times associated with 'BDN input' and 'BDN output' would be eliminated if the BDN-HAC was integrated into a future HAC solution, as in that case, the reading and processing of the logs would be replaced by using events received by the HAC.

10.3.2.3 Runtime Overhead

We used GeNIe modeller to perform inference on a computer with a 3.4 GHz Intel Core i7 and 64 GB of memory running Windows 10. The two steps, BDN input and BDN output, were executed using Linux scripts in the testbed. We measured the CPU utilisation and memory by monitoring the utilisation at the process level and the CPU utilisation was experienced for a period of < 85ms as displayed in Figure 10.17. Figure 10.18 depicts box plot distributions of CPU utilisation for the three steps presented for the data sets. The median values for the BDN input are 0.20%, 0.28%, 0.27% and 0.26% for Data Sets 1, 2, 5 and 7, respectively. The maximum values are 0.32%, 0.42%, 0.42% and 0.41%. The BDN inference also has little CPU utilisation, and the maximum values are 0.40%, 0.50%, 0.42% and 0.42%. The reason could be that the BDN model has only 89 parameter, and thus it does not require complex runtime calculations. The BDN output has the lowest utilisation of all three, and the maximum values are 0.20%, 0.36%, 0.35% and 0.38%. The results reveal that most values are in the lower quartile except for the BDN inference in Data Sets 1 and 7, which are in the upper quartile.

Evaluation

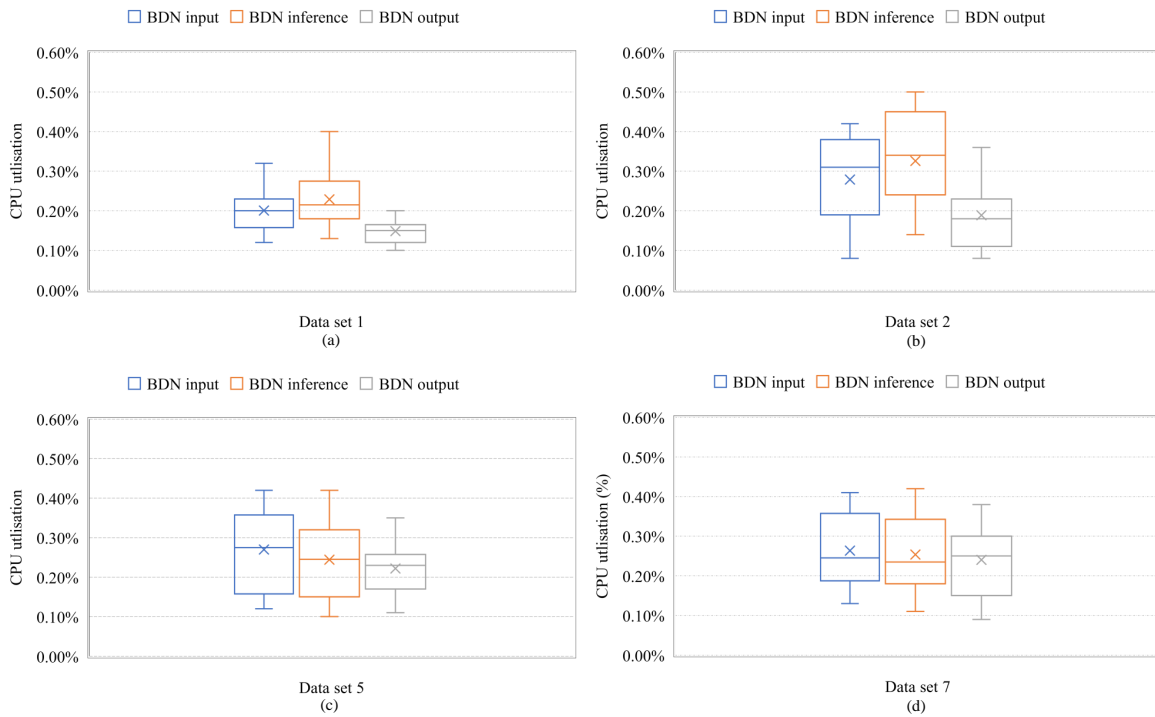


Figure 10.18 Box plots of CPU utilisation of the BDN-HAC steps for (a) Data Set 1, (b) Data Set 2, (c) Data Set 5 and (d) Data Set 7.

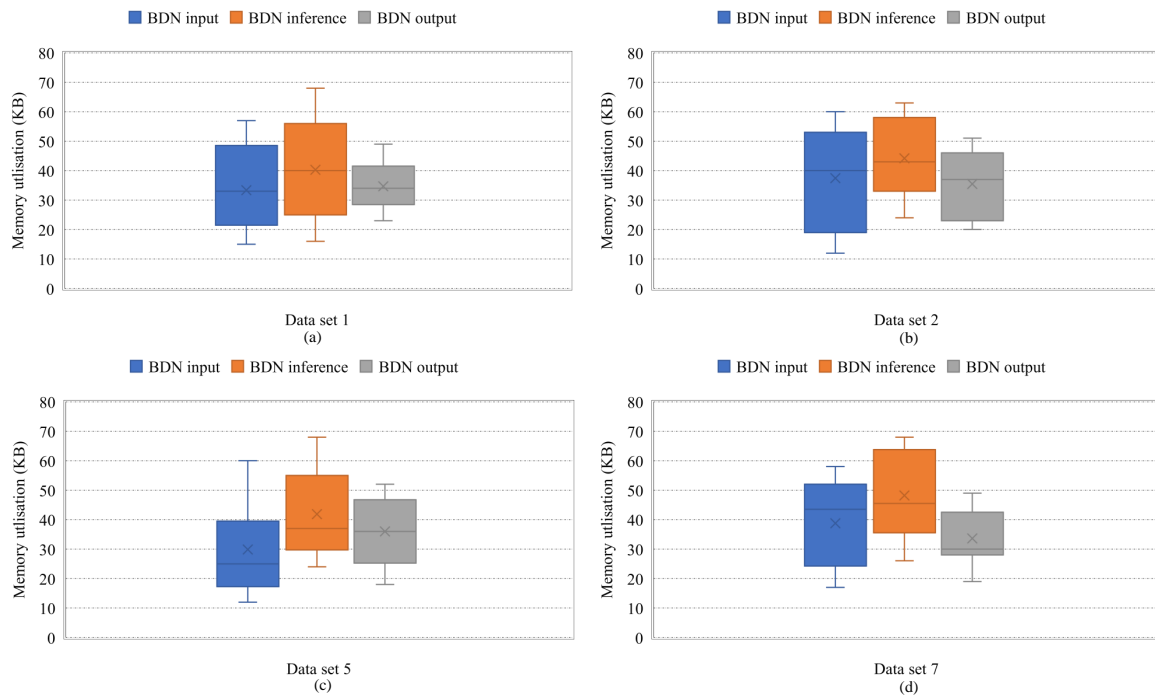


Figure 10.19 Box plot of memory utilisation of the BDN-HAC steps for (a) Data Set 1, (b) Data Set 2, (c) Data Set 5 and (d) Data Set 7.

Figure 10.19 presents box plot distributions of memory utilisation (measured in KB) for the three steps. The BDN inference has the highest utilisation, indicated by the maximum values (68, 63, 68 and 68 KB) in the four data sets. Comparatively, the other two steps used less memory because they dealt with fewer variables. The BDN input processes data for only those variables that the BDN model requires. Moreover, the BDN output has only the utility value from the BDN model, which is translated into a value that the BN model can use. Even if the maximum memory utilisation for all three steps is combined, it is still less than 180 KB. However, considering that each step is executed sequentially, the computational overhead associated with the CPU and memory is only linked to one step at a time. Therefore, the conclusion is that the overhead associated with executing all three steps of the BDN-HAC model is negligible.

10.3.3 Evaluation of the Bayesian Network model for Failure Propagation and Prediction

In this section, we first evaluate the method for constructing a BN model for the testbed HAC, and we then validate the model before evaluating the prediction quality and runtime overhead of the model.

10.3.3.1 Evaluation of the Model Construction

The modelling method introduced in Chapter 7 was used to construct a BN model for the testbed HAC. Unlike the BDN-HAC model, the BN-HAC model must be explicitly modelled for the corresponding HAC solution and subsequently trained and tested. The model structure was obtained from the outcomes of the HMTHA (described in Section 10.3.1) and the specifics of the HAC, and its variables and states were obtained from the HHAM and M-table. Prior probability distributions for the model were assigned using the T-rules. Finally, log data from the testbed was used to train and test the model; thus, parameter learning was used to update the probability distributions. This section presents the different steps associated with constructing and implementing the BN-HAC model.

Transformation from HMTHA to BN-HAC The HHAM model from Section 10.3.1 and the accompanying M-table were used to transform the HHAM into a BN model. We used the six-step mapping approach (Section 5.3.3) to map and complete the M-table. Hence, the first part of the M-table from Section 10.3.1 was completed with a second part, presented in Table 10.21.

There were three layers in the HHAM (a, b, c), mapped to four layers (Layers 1 to 4) of the BN model. All resource groups and the application vertex types in the HHAM were mapped to latent nodes in the BN model. Two simple vertex types were identified as weak. Similarly, two global dependency vertex types were classified as weak. All weak vertices were handled by assigning conditional probabilities to reflect the influence on other resources.

Evaluation

Table 10.21 Completed M-table for the testbed high availability cluster (HAC) showing the structure of the mapped BN-HAC model

HHAM				BN		
Resource	Vertex Type	Arc Type	Layer	BN Node	BN Node Type	Layer
Main application (ERP)	A	A	Top	X_1	Latent	1
Message and lock group	RG	RG	a	A_1	Latent	2
Message and lock service	S	R	b	A_2	Node	3
VIP	S	R	c	A_3	Node	4
FS message and lock	S	R	c	A_4	Node	4
FS trans	W	R	c	A_5	Node	4
FS interface	W	R	c	A_6	Node	4
Database group	RG	RG	a	B_1	Latent	2
database	S	R	b	B_2	Node	3
VIP	S	R	c	B_3	Node	4
FS database	S	R	c	B_4	Node	4
Main instance group	RG	RG	a	C_1	Latent	2
Main instance service	S	R	b	C_2	Node	3
VIP	S	R	c	C_3	Node	4
FS main instance	S	R	c	C_4	Node	4
Backup lock server group	RG	RG	a	D_1	Latent	2
Backup lock server service	S	R	b	D_2	Node	3
VIP	S	R	c	D_3	Node	4
DLM group	RG,GD	SG	a	E_1	Latent	2
DLM	GD	G	b	E_2	Node	3
FS DLM	GD	G	b	E_3	Node	3
CPU monitor	GW	G	Top	G_1	Node	1
NIC monitor	GW	G	Top	G_2	Node	1
SBD service	GD	G	Top	G_3	Node	1

Resource: FS - file system, VIP - Virtual IP, DLM - distributed lock manager, SBD - STONITH block device source group, G - Global

Vertex Type: A - Application, RG - Resource group, S - Simple, W - Weak, GD-Global dependency, GW - Global dependency and weak

Arc Type: A - Application, RG - Resource group, R - Resource, SG - Shared resource group, G - Global

Construction of the BN-HAC Model In this step, we used the HHAM and the completed M-table to construct the BN model. Figure 10.22 displays the resulting BN-HAC model representing the HAC in the testbed environment.

The HHAM layers were mapped in reverse order to construct the BN-HAC model. The BN-HAC model consists of 24 random nodes organised into four layers. Each node has two states: failure to indicate failure and no_failure to indicate nonfailure. Thus, the model has 48 states and 620

10.3 Evaluation of Bayesian Prognostic Framework Modules

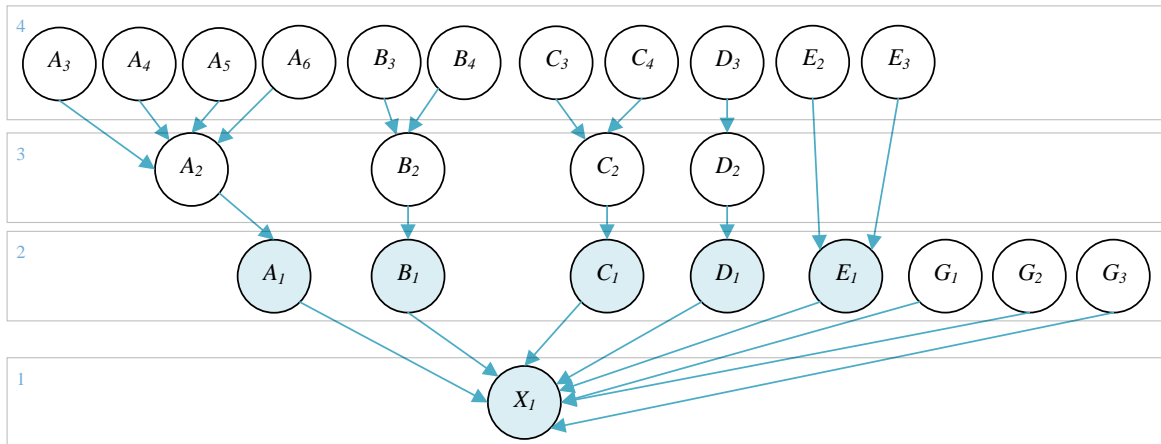


Figure 10.20 Bayesian network model for failure propagation and prediction (BN-HAC), representing the testbed high-availability cluster (HAC). The model details are presented in Table 10.22.

Table 10.22 Details of the Bayesian network for the failure propagation and prediction (BN-HAC) model for the high availability cluster (HAC) in the testbed

	Total Nodes	Latent Nodes	Weak Nodes	States	Parameters
BN-HAC	24	6	4	48	620

parameters. Most HAC resources are in Layer 4, whereas the latent and global dependency nodes are in Layer 2. The application node is in Layer 1. All nonlatent nodes represent physical resources.

In contrast, the latent nodes representing resource groups and the main application are highlighted in blue. Four nodes are identified as weak nodes. Two nodes, A_5 and A_6 , are created to meet business requirements, whereas the other two are global resources, G_2 and G_3 , which only monitor two critical server resources (CPU and memory).

Assigning Prior Probabilities We used Approach 1 from Section 5.3.4 (eqs. (5.2) and (5.3)) to calculate probabilities for all Layer 4 nodes and global dependency nodes in Layer 2 and obtained the following values for the defined SLA of 99.0%:

$$\text{Failure} = .01$$

$$\text{No failure} = .99$$

Thus, we assigned uniform probability distributions to nodes $A_3, A_4, A_5, A_6, B_3, B_4, C_3, D_3, E_2, E_3, G_1, G_2$ and G_3 . Further, we employed Approach 2 to calculate the probabilities for the nodes in Layers 2 and 3 and set the conditional probabilities accordingly. This included nodes A_2, B_2, C_2 and D_2 , and latent nodes A_1, B_1, C_1, D_1 and E_1 . Thus, the failure of weak nodes A_5 and A_6 does not result in the failure of child node A_2 (e.g., failure = .03 for a weak node).

Similarly, conditional probabilities were assigned to the application node X_1 to ensure that the failure of the global dependency weak nodes G_2 and G_3 does not result in a failure of the application.

Evaluation

Table 10.23 Local probabilities and conditional probabilities of the Bayesian network for the failure propagation and prediction (BN-HAC) model

Node	Probability Distribution
A_1	$P(A_1 A_2)$
A_2	$P(A_2 A_3, A_4, A_5, A_6)$
A_3	failure =.01 no_failure =.99
A_3	failure =.01 no_failure =.99
A_4	failure =.01 no_failure =.99
A_5	failure =.01 no_failure =.99
A_6	failure =.01 no_failure =.99
B_1	$P(B_1 B_2)$
B_2	$P(B_2 B_3, B_4)$
B_3	failure =.01 no_failure =.99
B_4	failure =.01 no_failure =.99
C_1	$P(C_1 C_2)$
C_2	$P(C_2 C_3, C_4)$
C_3	failure =.01 no_failure =.99
C_4	failure =.01 no_failure =.99
D_1	$P(D_1 D_2)$
D_2	$P(D_2 D_3)$
D_3	failure =.01 no_failure =.99
E_1	$P(E_1 E_2, E_3)$
E_2	failure =.01 no_failure =.99
E_3	failure =.01 no_failure =.99
G_1	failure =.01 no_failure =.99
G_2	failure =.01 no_failure =.99
G_3	failure =.01 no_failure =.99
X_1	$P(X_1 A_1, B_1, C_1, D_1, E_1, G_1, G_2, G_3)$

However, the failure of critical global dependency G_1 (SBD) affects the entire cluster (the primary node) because it can lead to fencing off the entire node upon failure, affecting all resources running on that node. Therefore, the conditional probability at child level X_1 was set accordingly using the first rule and Approach 2 in the T-rules.

Finally, we used Approach 3 to calculate the failure effect of the resource groups (latent nodes) on the application. We used eq. (5.6) and (5.7) to calculate the failure of one or more resource groups, A_1, B_1, C_1, D_1 and E_1 , and assigned the conditional probabilities accordingly. The application has five resource groups, and using eq. (5.7), we calculated that the HAC could tolerate the failures of up to four resource groups; this is reflected in setting the conditional probabilities in X_1 . Table 10.23 lists all probabilities and conditional probabilities.

Implementation of the Model To simplify the evaluation, we developed and deployed the BN-HAC model using the software package GeNIe Modeler. However, in a live production environment, the model must be implemented using a script (e.g., encoded in R using open-source BN libraries) that can be invoked as and when required after the invocation of the BDN-HAC model.

Substitute Incomplete Data The incomplete data in the training data sets are substituted with the value of no failure to ensure that the probability distributions are updated accurately, significantly improving the prediction capability.

Augment Data Set An optional augmentation step can be performed to ensure that the correct failure distribution is in place if there are insufficient failures. Hence, the training data sets were augmented to reflect operations of an actual HAC environment and to optimise the parameter learning process. For each failure instance, 100 nonfailure instances were added. Moreover, the failure frequency was increased in the second data set to ensure that the repeating failures were also captured. In the third data set, the failure frequency for one resource (node) was increased first. Then, a test case was initiated on that node to determine the prediction quality when historical failures were observed with a particular resource.

Enable Parameter Learning We used the test cases (described in Section 10.2.1.2) to create training data sets to learn the parameter distributions of the model. Subsequently, the training data are first input into the BDN-HAC model. Then, the output was transformed and input into the BN-HAC model. The training employed the EM algorithm to perform maximum likelihood estimation.

Inference Using Production data The BN-HAC model processes only those failures that the BDN-HAC model predicts as unmanageable failures. The subsequent inference of the BN-HAC model relies on the learnt distributions to predict a potential resource group or system failure. Thus, we used test cases T9–T16 to perform the inference, and the details regarding the data sets and evaluation are described in Chapter 10.

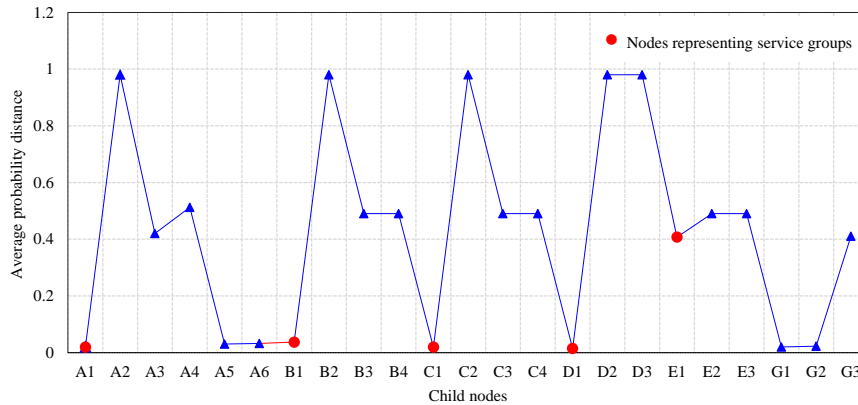


Figure 10.21 Strength of influence analysis for the BN-HAC model.

10.3.3.2 Prediction Quality

The evaluation of the BN-HAC model was conducted in two steps. First, we initiated parameter learning using the EM algorithm and performed inference using the clustering algorithm to observe the results. Data Sets 3 and 4 were used to learn the parameters. Data Sets 9 and 10 were created using Data Set 2 but with different sizes to test the correlation between learning data set sizes and predictions. Data Sets 6 and 8 were used to perform the inference after parameter learning. All latent variables were excluded in the parameter learning process.

Moreover, the parameter initialisation was set to keep the latest distributions as priors so that new distributions could continue from the last learning. This approach enables running parameter learning multiple times to improve the distributions, i.e., the incremental parameter learning mode (as presented in Section 4.3.4). The update of the distributions by parameter learning on existing distributions can be expressed by a confidence parameter (equivalent sample size or ESS) [269], which uses a weighted approach to update new distributions. If it is set to a higher value, the update progresses slowly for new data, whereas a low value indicates that even a small data size can considerably change the network distributions when new data are learnt.

Therefore, we experimented with several data set sizes and confidence parameter values and we identified the equivalent sample size 125 as the optimum value for the learning process. This is because our objective was to consider both historical data and also changes due to new data. Hence, the value means sufficient weight is given to new data to update the parameter distributions without significantly affecting the prior distributions. Similarly, the changes to conditional probability distributions also occur progressively. We employed the clustering algorithm as the primary algorithm for inference. The prediction quality was measured using ROC curves with a 95% confidence interval. The experimental design for ROC analysis is presented in Section 10.2.4. We used GeNIe modeller v3.0 [18] and the bnlearn library [180] to construct the BN models. Parameter learning and inference was also conducted using the GeNIe modeller.

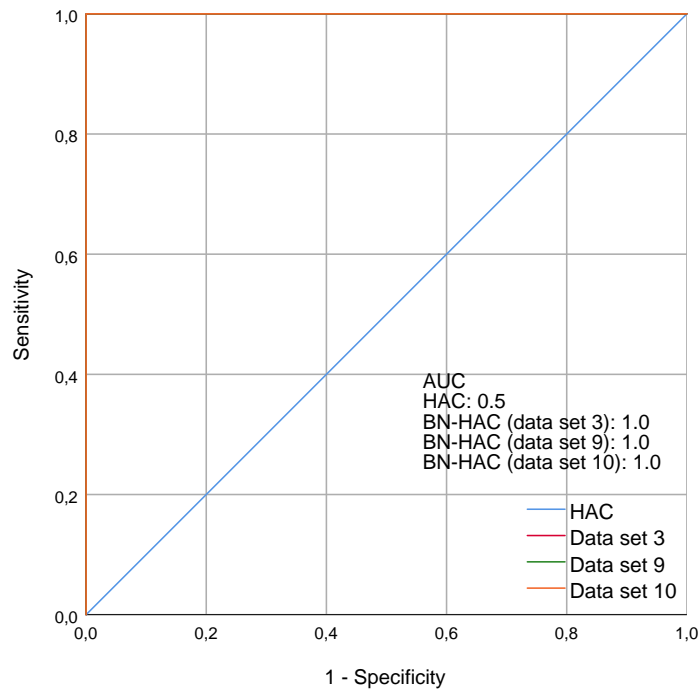


Figure 10.22 Receiver operating characteristic (ROC) curve showing prediction quality for Data Set 6 using Data Sets 3, 9 and 10 for parameter learning.

Model Validation Using GeNIe Modeler we computed the strength of influence for the parent-child node pairs from our BN-HAC model to validate the model construction. Figure 10.21 displays the results using the Euclidean measure, and the length of the link indicates the influence; the longer a link is, the stronger the influence becomes. The figure shows the same strength of influence between the nodes representing resource groups (parents) and child nodes. Moreover, the nodes A_1 , B_1 , C_1 and D_1 are parent nodes to child node X_1 (the application). These parent nodes influence X_1 equally, reducing the likelihood of the failure of one specific resource group causing a system failure, reflecting the intended use correctly. However, E_1 represents a shared resource group; hence, the failure of the resource group results in a system failure, and the strength of influence illustrates this. Overall, the strength of influence reflects the preferred influence on child nodes using conditional probabilities.

Prediction Quality We evaluated the prediction quality of the BN-HAC model by initiating parameter learning using a training data set and by subsequently performing inference using a production data set. The training data sets for parameter learning we used were as follows:

1. We used Data Sets 3, 9 and 10 with sample sizes, and these were created as described in Section 10.2.2.
2. Data Set 4 which was created as described in Section 10.2.2.

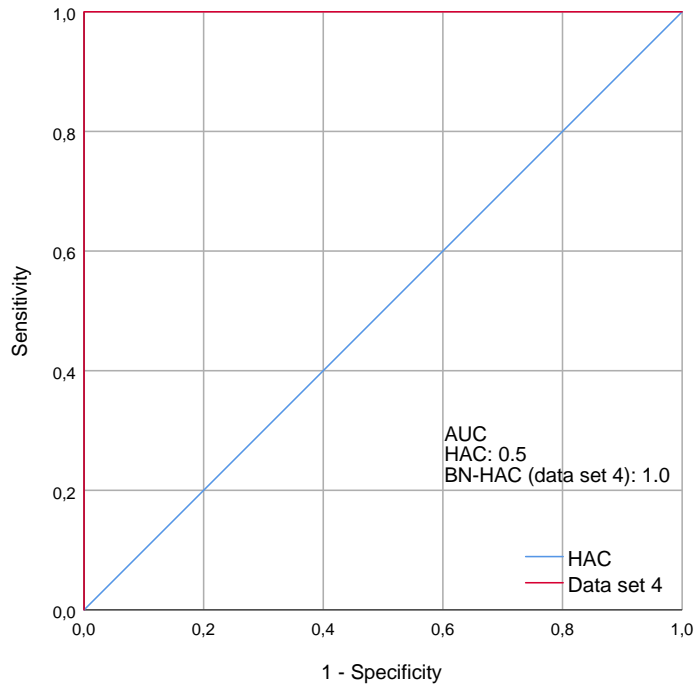


Figure 10.23 Receiver operating characteristic (ROC) curve showing prediction quality for Data Set 8 using Data Set 4 for parameter learning.

Subsequently, we performed inference using the corresponding production data sets, listed as follows:

1. Data Set 6 was created as described in Section 10.2.2 with the stickiness policy disabled.
2. Data Set 8 was created as described in Section 10.2.2 with the stickiness policy enabled.

Using the conditions for ROC analysis presented in Section 10.2.4, we plotted the ROC curves presented in Figures 10.22 and 10.23. Figure 10.22 depicts inference results using Data Set 6 and the results per the parameter learning (training) data sets.

Similarly, the HAC outcome was also validated using the expected outcome, and the result is plotted as a diagonal line with an AUC value of 0.500. The curves plotted by the model outcomes overlap because they all have an AUC value of 1.0, which means the HAC solution that uses the BN-HAC model can improve the baseline HAC by 50%. The primary reason for this was that many of the HAC-determined system failovers were unnecessary, as a resource group failover was sufficient for these, as correctly predicted by the BN-HAC model. Such resource group failovers are less expensive than system failovers. For example, when the file system of the main instance failed, the HAC decision was to perform an unnecessary system failover after going through the threefold strategy of first trying to reinitialise the failed resource and then attempting to failover the concerned resource group. In contrast, the BN-HAC model correctly predicted that a resource group failover was sufficient to resolve the problem. In other cases, when the HAC opted for a system failover, the

10.3 Evaluation of Bayesian Prognostic Framework Modules

Table 10.24 Summary of the prediction results grouped by the inference data set

Test Result	AUC	AS	SE	Confidence Interval	
				Lower Bound	Upper Bound
Data Set 6					
HAC result	0.500	0.089	1.000	0.325	0.675
Data Set 3	1.000	0.000	0.000	1.000	1.000
Data Set 9	1.000	0.000	0.000	1.000	1.000
Data Set 10	1.000	0.000	0.000	1.000	1.000
Data Set 8					
HAC result	0.500	0.050	1.000	0.402	0.598
Data Set 4	1.000	0.000	0.000	1.000	1.000

AUC - Area under curve

AS - Asymptotic significance (null hypothesis: true area = 0.5). It is used to determine the statistical significance of the relationship between the variables. $p < 0.05$ indicates a statistical significance relationship.

SE - Standard error (under the nonparametric assumption)

HAC - High availability cluster (Pacemaker/Corosync HAC)

BN - HAC Bayesian network for failure propagation and prediction

Table 10.25 Performance of the parameter learning for Data Sets 3, 4, 9 and 10

Data Set	#Instances	Log(p)	Elapsed Time (s)
3	30240	-1549	2
4	30240	-1380	2
9	672	-748	1
10	10080	-1320	1

BDN-HAC model predicted no failure. An example of this behaviour is where the BDN-HAC model predicted consistently that no action is required when the file system of the DLM service (node E_3) fails because HAC reinitialises it. However, the HAC in the testbed performed system failovers for such failures in Data Sets 7 and 8, but in Data Sets 5 and 6, it could reinitialise the resource, having identified that no failover was required.

Table 10.24 summarises the prediction results grouped by the inference data sets, and the results are presented per training data set. The HAC result indicates that the observed results are verified against the expected outcomes.

Parameter learning performance Details about the learning performance are presented in Table 10.25, which shows the elapsed time and learning quality in terms of $\log(p)$ for different data sets.

Evaluation

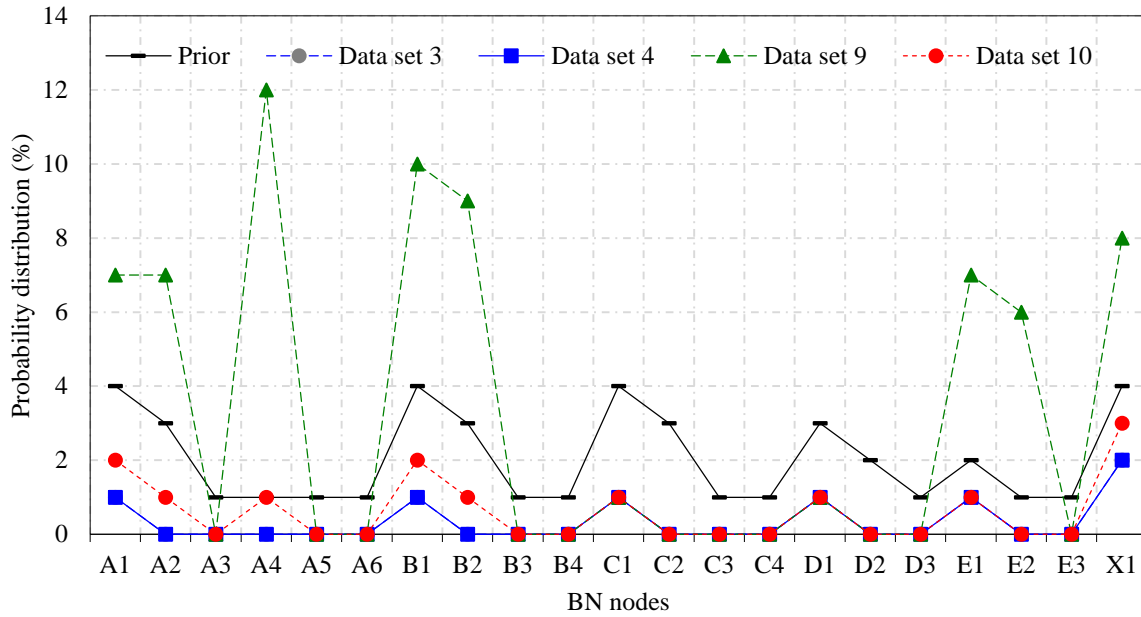


Figure 10.24 Failure probability distributions for each node in the BN-HAC model (in percentage) after parameter learning with different data set sizes, shown for Data Sets 3, 4, 9 and 10.

The metric $\log(p)$ measures how well a model fits the data, and this value ranges between negative infinity and zero, where a higher value indicates a better fit [19].

As presented in Table 10.25, the largest data sets, Data Sets 3 and 4, required 2s to complete the learning, whereas the small data set, Data Set 9 with 672 instances, required 1s to complete the learning. The obtained values for $\log(p)$ indicate that the model fit is better with smaller data set sizes.

Parameter Learning and Data Size To study how parameter learning changes the failure probability distributions for each node in our BN-HAC model when data sets with different sizes are used, we used the four training data sets listed in Table 10.25. Accordingly, we executed parameter learning for each data set and obtained the results. We reset the distributions after each learning to ensure that we can study the relationship between parameter learning in our model and the data set sizes. We then used the results to plot curves representing the failure distributions for each node in Figure 10.24.

The large data sets (3 and 4) exhibit no increase in the likelihood of failures because the number of failures is insignificant relative to nonfailures because the ratio of failures to nonfailures is 13:2160. In contrast, the small data set (9) with 672 instances, including 184 failures (27% of failures or a ratio of failures to nonfailures is 23:84), significantly increases the likelihood of failures in those nodes that have failed frequently. For example, node A_4 exhibits a failure distribution of 12%. The conditional probabilities in the upper-level (latent) nodes are updated automatically to reflect the increase in the likelihood of failures. For example, the latent node representing resource group node (A_1) has a failure distribution of 7%. Similar patterns were observed with other nodes with frequent failures. When the likelihood of failures increases for the individual and related latent nodes, the

10.3 Evaluation of Bayesian Prognostic Framework Modules

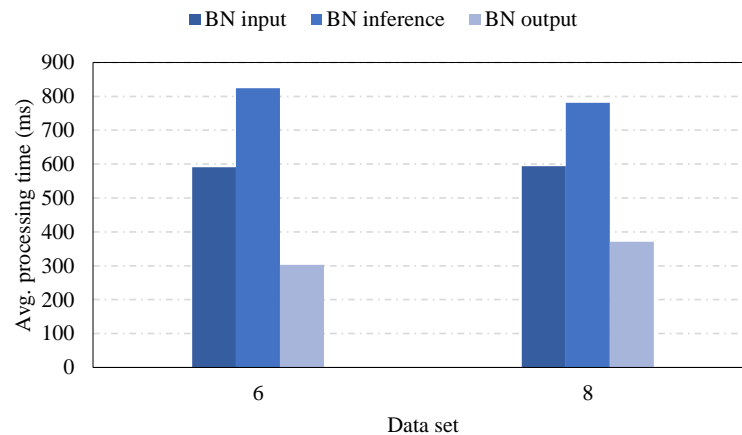


Figure 10.25 Mean execution times for the three steps associated with the invocation of the BN-HAC model.

distribution of the main application (a latent node) is also updated. For example, the application node X_1 is updated with a failure distribution of 8% for Data Set 9, whereas the value is only 2% for Data Sets 3 and 4. Although all data sets that are used to update the parameters resulted in no change in the final prediction quality, the results reveal that smaller data sets with numerous failures tend to increase the failure probability distributions of the BN model. When the failure distribution in the BN model increases, it also increases the likelihood of a resource, resource group or system failure.

10.3.3.3 Runtime Overhead

BN-HAC Model Execution Time The model execution time is evaluated to ensure that it does not add a notable increase to the overall execution time of the HAC. Therefore, we measured the execution times of three steps (BN input, BN inference and BN output) that are part of the invocation of the BN-HAC model. In the first step, the input is prepared for the required format of the BN-HAC model and includes steps to map the failure information from the BDN-HAC to the appropriate BN-HAC node. In the second step, inference using new information occurs, and in the third step, the prediction is provided for the resource group to which the failed resource belongs and for the system. We evaluated BN inference using GeNIe Modeler on a computer with a 3.4 GHz Intel Core i7 and 64 GB of memory running Windows 10. The two others steps were executed using Linux scripts in the testbed. The mean execution times for the three steps, BN input, BN inference and BN output, over the instances in Data Set 6 were 591ms, 824ms and 303ms. Similarly, the mean execution times over the instances in Data Set 8 are 594ms, 781ms, and 371ms, as depicted in Figure 10.25. When taken together, the mean execution time for Data Set 6 is 1.72s, 1.75s for Data Set 8, and the mean value for both data sets is 27.3% of the mean execution time of the BP framework. The BN model requires, on average, 802ms to complete the inference, and the model has 24 nodes and 620 parameters, which means the model complexity could be one reason for the high inference time. For example, the BDN-HAC model with 12 nodes and 89 parameters required 1ms on average to complete

Evaluation

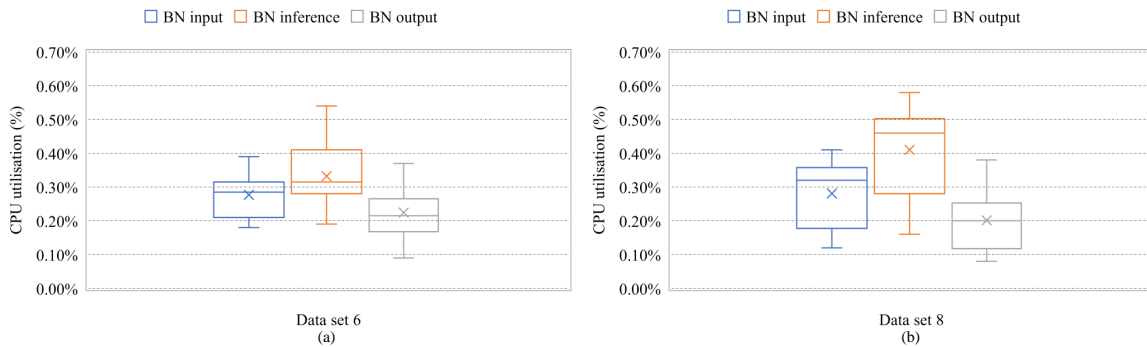


Figure 10.26 CPU utilisation of the BN-HAC model invocation steps for inference (a) Data Set 6 and (b) Data Set 8.

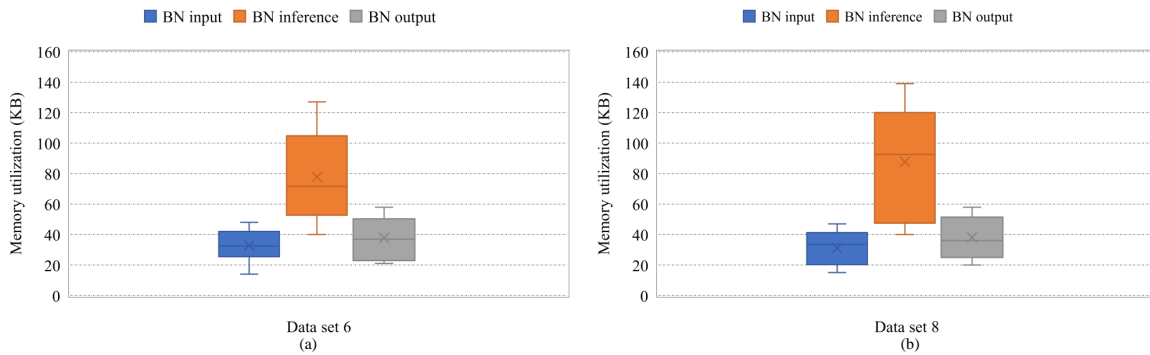


Figure 10.27 Memory utilisation of the BN-HAC model invocation steps for inference (a) Data Set 6 and (b) Data Set 8.

the inference and a mean of 84ms for all steps combined in all data sets. However, integrating the BP framework into future HAC solutions would significantly reduce the overall BN-HAC execution time by removing the need for the BN input or output because an event-based and instantaneous message transfer can replace these.

Furthermore, high-performing servers are typically used in a production HAC solution to protect business-critical EA, significantly reducing execution time. Therefore, while this result shows that the BN-HAC model adds a short delay, this can be mitigated using high-end servers and integrating the BP framework into HACs.

Runtime Overhead The overhead evaluation for all three steps is presented in Figure 10.26 for CPU utilisation and Figure 10.27 for memory utilisation using the inference Data Sets 6 and 8. The CPU utilisation was experienced for a period of 500 ± 300 ms. Figures 10.26 (a) and (b) indicate that the BN inference step used more CPU than the other two steps and recorded a maximum utilisation of 0.58% in Data Set 8. The median is 0.33% in Data Set 6 and 0.41% in Data Set 8. Most inner points are in the upper quartile in Data Set 6, whereas the points are reversed in Data Set 8 (most inner points are in the lower quartile). The BN input has a maximum value of 0.39% in Data Set 6 and 0.41% for

Data Set 8, and the median value is 0.28% in both data sets. The BN output exhibits a similar pattern; the maximum value is 0.37% and 0.38% in Data Sets 6 and 8, and the median values are 0.22% and 0.08%, respectively. On average, the BN inference step uses more memory than the other two steps, though the values are still low. For example, the maximum values are 127 KB and 139 KB in Data Sets 6 and 8, respectively. The median values are 71.5 KB and 92.5 KB, respectively. However, most of the inner points are in the lower quartile. Comparatively, both the BN input and output utilisation values are significantly lower. The maximum values are 48 KB and 47 KB for BN input in Data Sets 6 and 8, whereas the BN output has a maximum value of 58 KB in both data sets. The median values (31 to 38 KB) also confirm the low utilisation. We observed that the BN inference uses more CPU cycles and memory than the other two steps because it manages 24 nodes and 616 parameters. However, overall, the results show that the overhead associated with the BDN model is insignificant.

10.3.4 Evaluation of Bayesian Prognostic Framework Preparation

We introduced the BP framework preparation module in Chapter 8 and validated it successfully by implementing it in the testbed environment (described in Appendix D).

10.3.4.1 Runtime Overhead

We also investigated the runtime overheads associated with the execution of the BPFP, and we present the results of this investigation in Appendix D. We show that polling is the only component with a significant execution time of up to 10s. However, the polling time depends on the scheduling frequency and therefore increasing the frequency will reduce the polling time. Though the polling time was set to 10s, the actual polling time was observed to be less. If the polling takes place 1s after the failure, the HAC records the failure directly, and therefore the polling time becomes 1s. The CPU and memory utilisation evaluation shows that the overhead is negligible, and CPU utilisation was experienced for a period of 5 ± 0.8 s as listed in Table D.1.

10.4 Evaluation of Bayesian Prognostic Framework

In this section, the improvements achieved using the BP framework are presented, which means all the modules that are part of the framework contribute to the results. The consolidated result is compared with the HAC baseline measurement.

We prepared Data Sets 11 and 12 for MTTR and availability analysis. Each was created with 10080 instances to emulate continuous data for a week with one minute intervals to represent availability over time. Each instance consists of a timestamp, availability with TTR_{base} and availability with TTR_{BPF} . The availability figures were either 100 or 0 (default 100 to indicate 100% availability) to indicate whether the EA is available or not.

In the next step, we extracted failures that require downtime (e.g., resource group failover and system failover) from Data Sets 6 and 8. We distributed the failures over a week and inserted the

Evaluation

Table 10.26 Mean failover time in seconds for different types of resource groups

BN Node	Description	Failover Time (s)
A ₁	Message and lock instance group	41
B ₁	Database group	152
C ₁	Main instance group	153
D ₁	Backup lock server group	11

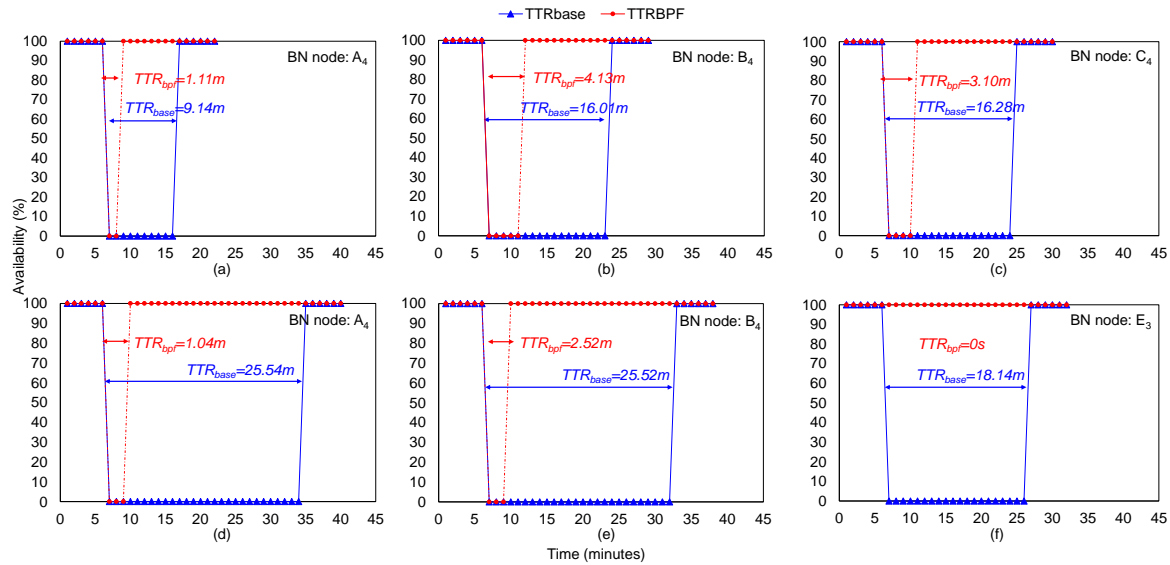


Figure 10.28 TTR analysis with the recovery time when only HAC results are observed in blue and when the BP framework is employed is shown in red. Panels (a), (b) and (c) are from Data Set 11, and (d), (e) and (f) are from Data Set 12.

related downtimes into the newly created data sets. Then, we calculated the TTR for each failure when the BPF is implemented and inserted these values into the data sets. The TTR per failure consists of all execution times presented previously (e.g., the BN inference time and resource group failover time). The resource failover time for the different resource groups was obtained by performing multiple failovers, and Table 10.26 lists the mean values for the different resource groups. Finally, we analysed the two data sets, and the results are presented in the next section. We did not consider the resources subject to reinitialisation in the calculation because such resolutions typically do not affect the overall availability.

We present the results from the evaluations using the TTR evaluation and availability analysis metrics in this section.

MTTR Analysis We observed that the use of the BP framework within a HAC solution can improve the MTTR significantly. To illustrate this, we selected three instances of failures from each of Data Sets 11 and 12. Figure 10.28 depicts the TTR for the three failures from each data set.

10.4 Evaluation of Bayesian Prognostic Framework

Table 10.27 Availability analysis results between the standalone HAC and the HAC supported by the BP framework

Data Set	HAC_{base}	HAC_{base}	HAC_{BPF}	HAC_{BPF}	Downtime	Uptime
	Downtime (min)	Uptime (%)	Downtime (min)	Uptime (%)	Improvement (%)	Improvement (%)
Data Set 11 *	144	98.57	26	99.74	81.94	1.19
Data Set 12 *	238	97.64	30	99.7	87.39	2.11

* Indicates that the failure information for the Data Sets 11 and 12 originates from Data Sets 6 and 8.

Figure 10.28(a) displays the system availability and TTR achieved by the baseline HAC and by using our BP framework (as part of a HAC) in the scenario in which the message and lock service file system (A_4) experienced a failure. The correct mitigation for this failure is a resource group failure, which is suggested by our BP framework. By applying this mitigation, a HAC using the BP framework would have restored the 100% availability with a short TTR_{BPF} of only 1.1m (this time includes the delay introduced by the BP framework). In contrast, the baseline HAC solution performed a time-consuming system failover, achieving a much poorer TTR_{base} of over 9m. Similarly, Figure 10.28 (b) displays the availability and TTR achieved for the failure of the database file system (B_4), where the BP framework correctly recommended a resource group failover, which reduced the recovery time compared to the system failover performed by the baseline HAC solution from the testbed. Figure 10.28 (c) shows the TTR reduction by the BP framework when the main instance file system (C_4) failed.

In the second data set, the failures of the same resources as in (a) and (b) are illustrated in (d) and (e). However, both have increased downtime by the baseline HAC, whereas the BP framework provided values with only minor differences. However, the TTR analysis in Figure 10.28 (f) shows that the resource DLM file system (E_3) failure is not considered a failure by the BP framework because it was predicted as a locally manageable failure by the BDN-HAC module.

We analysed six failures, out of which the BP framework predicted that five failures would only require a resource group failover, and the sixth failure would require no action because the failed resource can be reinitialised. Hence, the results illustrate how the BP framework can reduce the MTTR compared to the unnecessarily conservative baseline HAC.

Availability Analysis Table 10.27 presents the availability results across the whole data sets used in the evaluation. The total downtime with the HAC_{base} is 144 minutes in Data Set 11 and 238 minutes in Data Set 12. The BP framework predicted and advised changing many of the decisions presented in the previous section, reducing the MTTR. Therefore, the HAC_{BPF} improved the availability by 1.19% in Data Set 11 and 2.11% in Data Set 12. The downtime improvement is 81.94% in Data Set 11 and 87.39% in Data Set 12. The results demonstrate that the HAC_{BPF} can achieve significant improvement compared to the HAC_{base} .

10.5 Discussion

We injected a total of 213 failures for both training and production scenarios. The total count included 12 test cases where failures were injected five times consecutively for each test case, resulting in 60 fault injections. However, we treated them as 12 test cases in the evaluation because these failures were often not recorded by the HAC. For example, the application used the self-healing capability to reinitialise the resource. Therefore, we excluded these repeated failure injections (48) from the total number of failure injections and only included test cases in the count, which resulted in 165 failure injections. Further, we used two fault injections in some test cases to evaluate the impact of two resources failing simultaneously. When the first failure resulted in either resource group failover or a system failover, the result of the second failure injection became unobservable. Thus, nine failure injections affected by this were not considered. Out of the remaining failures, the HAC reinitialised 65 resources and carried out one resource group failover and 36 system failovers.

The BDN-HAC model predicted correctly that 70 failures could be managed locally and 32 failures were unmanageable locally. The BDN-HAC produced more true positives (TPs) and fewer true negatives (TNs) compared to the results of the HAC because it considered the additional properties introduced in this thesis as follows:

- The HAC performed four system failovers for a critical resource (database service) that could be reinitialised. At the same time, the BDN-HAC produced TPs for all of them, considering the properties in groups 1 and 2 as described in Section 6.1.
- Two failures associated with noncritical resources led to a resource group failover by the HAC. In contrast, the BDN-HAC produced TPs for all considering the property group three.
- The test cases T1 and T10 were used to consecutively inject failures five times on a resource with self-healing capabilities (redundancy factor in Table 10.7). The application reinitialised the resource rapidly for all these 60 failures. The first three fault injections in the same test case were quickly remedied within under 100ms, but the application required a longer time for the last two. The HAC was not aware of any of the failures or the self-healing capability provided by the application. However, when the delay in reinitialising the resource by the application in the later part of the fault injections coincided with the HAC's monitoring interval for that resource, and therefore the HAC was able to identify the failures and record them in the log file. The HAC then proceeded with reinitialising the resource though the application had already reinitialised it. When the HAC reinitialised the resource, it always reinitiated the four parent nodes (A_3, A_4, A_5 and A_6), which which resulted in a delay of over one second.
- In the same resource group, eight fault injections were associated with two resources, one noncritical (weak) and one with self-healing capability. The HAC reinitialised the weak resource, and the application started the second resource. However, when we injected 12 failures on two resources, one critical and one with self-healing capability, the BN-HAC model

predicted consistently that only a resource group failover was required for the critical resource. However, the HAC performed a system failover which brought down the second resource. In the future, a BP framework-based HAC would have only performed a resource group failover without disrupting other resources while also considering the self-healing capabilities of the application to avoid unnecessary delays and disruptions.

For the 44 unmanageable failures predicted by the BDN-HAC, the HAC performed system failover for all of them, but the BN-HAC model predicted correctly as follows:

- Thirty-six failures were predicted to require only a resource group failover. The failover time differs for the different resources as listed in Table 10.26. The resource group represented by node A_1 required an average of $> 41s$, while the resource group C_1 required $> 151s$.
- The model predicted that eight failures still required a system failover due to global dependency resource and other critical resource failures.

We have grouped the results from both training and inference together to provide the results discussed so far. However, the inference was typically performed using the failure distributions learnt by parameter learning and the production data. Therefore, we present the results observed for inference as follows:

- The BDN-HAC model delivered information about 20 unmanageable failures to the BN-HAC model. The BN-HAC correctly predicted that resource group failovers were sufficient to deal all 20 failures, while the HAC performed system failover for all of them.

Considering that some resource group failovers are fast (as listed in Table 10.26), there were only minor disruptions for the application. In other cases, the upper-level application layers could ensure connectivity using a reconnect functionality [87], thus recording only small windows of downtime. Therefore, the BP framework exploits the capabilities provided by the EA and other properties to achieve a significant improvement in reducing downtime compared to the HAC in the testbed.

10.6 Threats to Validity

We identified several threats that could affect the validity of the evaluation presented in this chapter:

- Unlike the EA used in our testbed, resource groups of EAs may have dependencies on one another. In such a case, the BN-HAC would need to take into account the fact that a failover of one resource group may have a cascading effect resulting in a system failover. We addressed this threat by providing a general-purpose method for constructing a BN model by using the exact composition of the HAC aided by the holistic availability modelling technique. Therefore, the dependencies between resource groups can also be defined as conditional probabilities in latent nodes. Thus, the BN-HACs constructed using our method should be able to deal with these situations, although evaluation with such EAs is needed to confirm this.

- The experiments were conducted using one primary HAC solution. The different HAC solutions available have distinct characteristics that can influence the prediction outcome. To reduce this threat, we used logs from multiple HAC solutions to develop the BP framework. Furthermore, we evaluated our BP framework using a HAC solution with different configurations of HAC, such as stickiness, to ensure that the evaluation captures the behaviour of multiple types of HACs.
- While the empirical evaluation of the BP framework was performed in a production-like environment, with a complete ERP solution, only data from one EA were used in the evaluation. To mitigate this threat, we used data from multiple HAC solutions to develop the framework.
- The HAC supporting an EA could have numerous configuration options, as described in our taxonomy. However, we only used one combination of configuration options, such as an active-passive configuration for the HAC topology. The results using different combinations of HAC configuration options may differ. To reduce this threat, we developed our BP framework so that it can be deployed across all available nodes of a HAC, in order to capture the individual nodes' failure information and assess them accordingly.
- The testbed does not have all the possible components, redundancy setups and related configurations required to fully mirror an environment for hosting a business-critical system. For example, a business-critical setup typically requires a separate network to be set up to allow quorum communication, but we could not provide that in the public cloud. Another example is that a business-critical system may use a quorum with a redundant setup using external devices. However, we could only deploy a quorum on the two available nodes. Therefore, HACs may perform better than in the evaluation results presented. To reduce this threat, we set up the testbed as close as possible to an environment used to support a business-critical system. Moreover, our objective is to demonstrate this new approach using probabilistic reasoning and improved detection and prediction capabilities. Therefore, the overall solutions are expected to perform better when the BP framework is deployed on HACs.

10.7 Summary

This chapter presented the evaluation methodology and evaluated the individual BP framework modules and the complete BP framework. First, we established the testbed to mirror the environment of a business-critical system, which included servers, storage and a network. Subsequently, we implemented a real ERP application and an open-source HAC to protect the application. We included all applications and related infrastructure resources in the HAC. We presented the related evaluation metrics, including the basic metrics and other metrics derived from the basics. Furthermore, we also proposed two metrics to measure MTTR and the availability of the HAC for the baseline scenario and for when the BP framework is implemented. These two metrics aid in quantifying the improvements

achieved by using the BP framework. A fault injection methodology was described, and associated test cases were also presented.

To evaluate the HMTHA technique, we assessed the construction of a model for the testbed HAC, and demonstrated that the technique satisfies its primary objective, which is to capture the complex dependencies of an EA. Furthermore, we evaluated the runtime overhead of the technique when creating models for three IT solutions, including the one for the testbed HAC; Appendix C presents the results of this evaluation in detail.

We evaluated the two BDN-HAC models and selected BDN-HAC-1 for use as the BDN-HAC module of our BP framework. Then, we evaluated the model for prediction quality, including in multiple scenarios in which incomplete data are involved. We also analysed different variations of incomplete data, such as when critical nodes receive data compared to noncritical nodes receive data. Overall, the use of our BDN model improves the AUC by 4.2% compared to the baseline HAC results. The evaluation of execution time demonstrates that it rarely exceeds 100ms, which does not add significant time to the overall execution time of the BP framework. The results from the runtime overhead evaluation demonstrate that the overhead is insignificant.

We evaluated the method for constructing the BN-HAC model in three steps. In Step 1, we transformed the HHAM and M-table to a corresponding BN model representing the underlying HAC. We allowed parameter learning to learn the distributions in Step 2. In Step 3, the model was inferred with runtime data to provide failure propagation and prediction. Then, we assessed the model prediction quality, revealing that the BN-HAC model can improve the AUC by 50% compared to the baseline results delivered by the HAC. We studied the reason for such improvement and found out that the BN-HAC model predicted a resource group failover correctly in scenarios where the baseline HAC performed system failovers unnecessarily. Naturally, a system failover is more expensive than a resource group failover. We also investigated the relationship between the learning data size and subsequent inference, and revealed that smaller data sets could affect the prediction quality.

The BPF module was evaluated by analysing the computational overhead. We demonstrated that the CPU and memory utilisation is negligible for its components, and the overall execution time was also not expected to increase.

Finally, we evaluated the complete BP framework, finding that it can reduce the MTTR and can improve the availability of the testbed application by a mean value of 1.65%, which is a significant improvement.

Chapter 11

Conclusion and Future Work

Multiple challenges stand in the way of ensuring the availability of modern EAs. Some of these come from the significant changes in EA deployment patterns, for example, due to moving applications to public clouds. Other challenges arise from changes in the architecture composition of EAs, for instance, from the transition to microservice-based solutions, and from the integration of new fault-tolerance capabilities (e.g., self-healing) so that some SPOF components can be managed by the application itself. As such, HACs need to continue to evolve to address these challenges while continuing to improve the availability of the protected applications. The Bayesian prognostic framework introduced in this thesis is intended to enable future HACs in this endeavour in two ways. First, the framework supports a more accurate prediction of locally manageable resources. Second, the framework allows the unmanageable failures to be accurately propagated to the resource group and system levels, supporting the prediction of potential failures earlier than currently possible.

To achieve these two improvements, the BP framework integrates and exploits novel capabilities provided by four modules:

1. We devised the HMTHA to discover the complex dependencies between components and represent them in the corresponding BN-HAC model. For example, we introduced the concept of a weak resource, which is presented as a weak vertex in the HHAM. Then, we mapped it to a weak node in the corresponding BN-HAC model using conditional probabilities.
2. We developed the BPF to prepare the environment to deploy the BPF and to aid in preparing HAC failure information.
3. We constructed the BDN-HAC model to incorporate the existing and new characteristics to predict locally manageable resource failures. The existing characteristics include common characteristics that can influence resource availability. The new characteristics identified in this thesis can be used to assess the hierarchy, resource types, resource criticality and application-provided self-healing capabilities. The characteristics are obtained from both the runtime and configuration environments.

4. We provided a method for constructing a BN-HAC model capable of propagating individual resource failures to predict resource group and system failures. The exploitation of the historical failures is realised using characteristics that measure a resource's repeated failure and failure count in the BDN-HAC model. Furthermore, in the BN-HAC model, historical data are used to learn the parameter distributions of past failures so that inference considers them appropriately at runtime.

This chapter summarises the contributions underpinning the BP framework and its four modules, and proposes further research directions.

11.1 Taxonomy of the High Availability Clusters

11.1.1 Research Contributions and Discussion

We developed a novel taxonomy of HACs that organises many classes and subclasses in the HAC domain. We introduced new terms to distinguish the critical areas so that HAC elements can be addressed with more clarity. The new terms are: symmetric application-based, symmetric server-based, cluster-based replication, client-state synchronisation, cluster-state synchronisation and application-state synchronisation. We used the taxonomy as a knowledge repository throughout this thesis, and to construct the BDN and BN models.

The taxonomy can be used to discover, explore, analyse and understand the composition of a HAC environment for an application, which was demonstrated by applying it to the testbed environment (described in Appendix B). We also revealed how the taxonomy could aid in identifying the correct configuration of a HAC, which simplified the implementation of the testbed.

11.1.2 Further Research Directions

We envisage that our taxonomy could support developing a tool-supported method enabling the developers of HHAM models to exploit the domain knowledge encoded by our taxonomy. This tool-supported method could automate key steps associated with devising HHAM models for a specific HAC solution, including discovering the components of a system, identifying the relationship between the components, and using that information to construct the HHAM model automatically.

11.2 Survey of High Availability Clusters

11.2.1 Research Contributions and Discussion

We presented an extensive survey of the HAC solutions using the taxonomy. We developed a systematic approach for selecting the HACs, focusing on EAs that apply filters to narrow the list of HACs for the survey. We designed a comprehensive questionnaire to collect data from the vendors of

HACs. Finally, we analysed the survey results, presented the limitations and challenges associated with the current HACs and proposed further research opportunities.

11.2.2 Further Research Directions

A wide range of further research opportunities was already identified in our survey and this is detailed in Section 3.2.6.

Additionally, several future directions can add further value to the research in the HAC domain. A survey on downtime and the cost associated with all aspects of downtime (i.e., loss of productivity, support costs, operational costs, delays in deliveries and escalation efforts) can produce a wealth of information while also presenting a holistic way of measuring the downtime effects.

A survey on the current challenges related to HAC solutions can help identify and group the challenges into either HAC-related or operations-related challenges to be addressed accordingly. For example, if the challenges are related to HACs, they can be resolved by proposing potential solutions to the HACs. In contrast, if the challenges are related to operations and a common pattern can be observed, they can be addressed by proposing changes in the operational model.

11.3 Holistic Modelling Technique for High Availability

11.3.1 Research Contributions and Discussion

We introduced HMTHA to model HA for IT systems to discover and visualise all components required to set up HA while also depicting the dependency relationships between the components. We defined a graphical notation to represent the different component types of an IT system using vertices and arcs. We proposed a simplified process for gathering requirements and creating a model. The resulting model can analyse different component characteristics, such as the criticality and hierarchy position. The model is accompanied by an M-table and T-rules to simplify mapping the HHAM model to a probabilistic model. Furthermore, we developed a software tool to ease the applicability of the technique. The HMTHA is a crucial technique to create the BN-HAC model for a given HAC. Further, we demonstrated that the HMTHA outcomes could prepare the deployment environment of the BP framework by providing input to the BFPF module in the testbed.

We evaluated the technique by creating three distinct models. The first model represents the testbed environment, while the other two models represent realistic IT systems (Appendix C). All three models have different vertices and arcs with varying depths (layers). We analysed the computational overhead (CPU and memory utilisation) associated with creating the models using the software-based tool. We noted that the CPU cycles were used when adding vertices and that the CPU utilisation was insignificant (under 2.5%). However, the memory utilisation was linear and increased with each added node. We discussed linear memory utilisation and its effect on the overall BP framework. We concluded that applying HMTHA is a one-time activity, and therefore it does not add any overhead to the runtime of the BP framework.

11.3.2 Further Research Directions

The HMTHA can be extended to discover the components of an IT solution automatically, which may require connecting to different information sources across multiple layers (e.g., server) to discover such information. A possible source of such information is a configuration management database (CMDB) that stores information about IT systems and the related components [248]. Open standards-based techniques can also be utilised, such as standards by Distributed Management Task Force (DMTF) [34].

An application programming interface could be developed to provide model information to other solutions automatically. For example, such information can be used to construct the structure of a BN model with significantly reduced effort.

11.4 Bayesian Prognostic Framework Preparation

11.4.1 Research Contributions and Discussion

We developed a BFPF module to satisfy two objectives. The first was to enable the deployment of the BP framework, and the second was to enable the processing and preparation of the failure information to be input into the BDN-HAC model. To support the two objectives, we designed and developed several components: configuration refinement, log interface, transformation and conversion, and the filter. We also introduced a new algorithm for uniquely extracting failure entries using a timestamp and a process identifier in the component log interface.

We evaluated the module and its subcomponents and showed that little computational overhead is associated with the module runtime. The measurement of the execution time indicated that the polling used by the log interface requires a significant portion of the overall execution time of the BP framework. However, the value depends on the polling frequency. Therefore, this frequency can be reduced to improve the overheads.

11.4.2 Further Research Directions

The log interface can be extended to support multiple HACs, in order to enable the BP framework to work with several HACs. The BFPF is already developed to function as an independent “interface” layer between a HAC solution and the other modules of the BP framework. Therefore, the approach should connect the HAC logs to the BFPF intermediate layer. However, this could present a significant challenge because of the lack of standardisation regarding log structure and format as described in Section 3.2.6. A machine learning approach could potentially be employed to identify the structure of a HAC log before proposing how such information can be utilised by the log interface [130].

11.5 Bayesian Decision Network for Predicting Locally Manageable Resource Failures

11.5.1 Research Contributions and Discussion

We introduced a HAC solution-independent BDN model to predict locally manageable resource failures of HACs by incorporating a set of characteristics comprising both existing and new characteristics proposed in this thesis. Each characteristic is mapped to a BDN node and most are mapped to a chance node in the BDN model. A decision node indicates the failure status of a HAC resource, and a utility node presents the prediction outcome. Finally, conditional probabilities are used to encode the node dependencies and add weight to indicate the degree of influence between the parent and child nodes. In contrast, the utility node uses preferences to add weight to the child nodes.

We constructed two models with two distinct paths to enable the comparison and validation of alternative BDN structures. We employed two metrics, utility analysis and strength of influence, to select the better of the two models as the BDN-HAC module of our BP framework.

We evaluated the prediction quality of this model using the testbed environment and showed that the model performed prediction better than the established HAC solution used as a baseline. Moreover, we also investigated the impact of incomplete data and demonstrated that it significantly affected the prediction quality. We also evaluated the scenario when critical nodes (nodes with a high weight factor) received data and compared this to the scenario when noncritical nodes were supplied with data. The experimental results confirmed that the critical nodes play a vital role in the BDN-HAC model. Similarly, we tested the differences between roles of the existing and new characteristics of the BDN-HAC model, and we showed that the new characteristics improved the prediction quality more than the existing characteristics. Furthermore, we measured the computational overhead and execution time associated with the model. We found very little overhead (CPU and memory utilisation), and the execution time is also on the order of milliseconds, which is acceptable for the intended use of this model.

11.5.2 Further Research Directions

In Chapter 2, we demonstrated that a BDN utility output could represent the probability of managing failure locally at two levels, manageable or unmanageable. This shows that the utility output can be correlated to the different levels of failures. Therefore, it is worth exploring and interpreting the utility outcomes at a granular level, which can then be connected to multiple failure states of a resource. This has the potential to improve the prediction quality in the BDN-HAC and BN-HAC models.

If the BDN-HAC model is integrated with HAC solutions, the model can be used in the decision-making process of HACs, resulting in faster failure mitigation times and reduced overhead. The overall purpose of the BP framework is to be integrated into HAC solutions. As a first step, even integrating only the BDN-HAC module into a HAC solution is bound to improve detection and

prediction capabilities because the model can consider a wide range of properties associated with EAs and HACs.

Considering additional HAC characteristics in the BDN-HAC model could improve the detection and prediction qualities. For example, the self-protection ability provided by modern EAs to block cyberattacks, and thereby avoiding potential downtimes caused by such attacks [44, 53], can also be added as a new characteristic.

11.6 Bayesian Network for Failure Propagation and Prediction

11.6.1 Research Contributions and Discussion

We introduced a method for constructing a HAC-specific Bayesian network model. The resulting model is used to propagate unmanageable failures from the BDN-HAC model to predict failures at the resource group or system level. Our method obtains input from the HMTHA outcomes for the structure of the BN model, and assigns appropriate prior probabilities to complete the construction of this model. Furthermore, we presented an approach for mapping the different vertex types in the HHAM model to the corresponding elements of the BN model, for example, mapping weak vertex types to weak BN nodes by assigning conditional probabilities at the related child node level. The model learns the parameter distribution and uses this information to provide predictions. However, the HAC failure data are incomplete because only the failed node passes the information. To resolve this, we added a substitution step to replace all incomplete data with positive values, and we demonstrated that this approach correctly updates the BN parameter distributions.

We evaluated the method for constructing a model using the testbed environment, and showed that the resulting model could accurately predict all cases considered in our experiments. Moreover, we showed that the model could distinguish between different type of failures. For example, the model could predict a resource group failure in many scenarios in which the baseline HAC used in the testbed performed a system failover unnecessarily. Therefore, the model predictions reduced the MTTR and improved the availability of the protected EA. We measured the computational overhead and execution time associated with the model and found both to be insignificant.

11.6.2 Further Research Directions

Online learning can further improve the framework because the accurate and continuous updates of the parameter distributions has the potential to improve the prediction quality of the BN-HAC model when considering new failures. Online parameter learning could be used to update the parameter distributions continuously [225], ensuring that the parameter distributions reflect up-to-date failure occurrences in the HAC, ultimately improving the probability of predicting failures correctly.

Changing the BN-HAC model to a DBN-based model can improve prediction quality using time series data because the HAC log data are based on a time series [133, 134]. Thus, both failures and

nonfailures can be input into the model using online learning. Inference can then use the updated parameter distributions to provide accurate predictions.

Another area worth exploring is the automatic creation of the BN model structure using the underlying structure of the HAC. This thesis laid the foundations for constructing a BN-HAC model from the outcomes of the HMTHA. Automating these steps would simplify the model creation process while reducing the errors associated with the manual creation of the models.

11.7 Bayesian Prognostic Framework for High-Availability Clusters

11.7.1 Research Contributions and Discussion

We devised the BP framework, the first end-to-end solution that employs probabilistic reasoning within the domain of HACs. The BP framework consists of four modules: HMTHA, BPF, BDN-HAC and BN-HAC. The integrated framework functions as a single solution to enable a series of tasks. The framework, for example, enables modelling HA for an IT solution using HMTHA. The outcomes are subsequently used in the BPF module to update information about the HA environment, whereas the BN-HAC module uses them to construct the BN-HAC model. The BPF module extracts distinct failure information from HAC logs and prepares such data for the BDN-HAC module. The BDN-HAC model utilises the prediction capabilities by incorporating a set of characteristics, existing and new, to determine the manageability of the failure. The confirmed failure is then passed to the BN-HAC model to propagate the failure and to predict a potential failure at high-level components (the resource group or system).

The framework supports complete lifecycle management by providing three views: the design, implementation and runtime. The framework is designed specifically for the underlying HAC and EA in the design phase. Therefore, in the design phase, the HHAM model, M-table and T-rules are created. Furthermore, the BN-HAC model is constructed, reflecting the precise structure of the HAC. In the implementation phase, all modules are implemented in the HAC environment, and a training data set is prepared to enable the BN-HAC model to learn the parameter distributions. In the runtime phase, failure information is captured and processed before applying the prediction of locally manageable failures. The information related to unmanageable failures is passed to the BN-HAC model to propagate the failure to upper-level components (the resource group or system) and predict a potential failure.

We established the applicability of the framework by implementing it in a realistic tested environment. We performed an MTTR analysis to evaluate the framework efficiency, and the results revealed a significant reduction in the MTTR. Furthermore, we applied the TTR values from both the baseline HAC and the HAC when the BP framework is deployed (BPF) to two data sets to obtain the availability over time for both. The results show that the BP framework improved the weekly availability by 1.65% (from 98.11% to 99.72%), which is a significant improvement. Furthermore, we evaluated the runtime overhead and execution time for all modules, and neither was significant.

11.7.2 Further Research Directions

The full potential of the framework can be realised once it is fully integrated with the failover management or cluster management module of a HAC. Therefore, the main area of future research is this integration, which has the potential to deliver the following benefits:

1. The BP framework can assess more characteristics, as proposed in the 'Further Research Directions' (Section 11.5.2) of the BDN-HAC, which significantly improves the detection quality and the ability to predict locally manageable failures.
2. The BN-HAC introduces Bayesian probabilistic reasoning using prior and posterior failure probability distributions and can assess the dependencies between a large number of resources simultaneously. For example, the framework can even consider the effect of a resource failure that is not directly related, e.g., in a different resource group.
3. The response time for detection, prediction and decision will also be faster because the communication could be event-based and in real-time, similar to how current HACs manage internal communication between the different modules today.
4. The integrated framework can capture failures directly from the monitoring module of HACs, and this can speed up the entire process.
5. The framework enables implementing more complex solutions such as capturing information regarding multiple states of a node/resource, which could improve the BP framework's detection and prediction qualities due to a granular interpretation of failures can be achieved.

Therefore, the integrated solution can reduce the MTTR significantly, which could improve the overall availability.

The BP framework can be developed further to support assessing a large number of interconnected components where assessing the individual component's state changes and its effect on other components is essential. An analogous BP framework can be developed to detect, predict and mitigate cyberattacks in public clouds. The BDN-HAC can assess the individual resources such as a server or a service using a set of characteristics based on the local capabilities available to resolve the issues. If a threat cannot be resolved locally, it will be passed to the BN-HAC model, assessing the impact on other related sources. The BN-HAC model then can invoke the procedure to mitigate the threat, which could be an alert or another system that blocks the traffic into the network.

Bibliography

- [1] Abdollahi Vayghan, L., Saied, M. A., Toeroe, M., and Khendek, F. (2019). Microservice based architecture: Towards high-availability for stateful applications with kubernetes. In *2019 IEEE 19th International Conference on Software Quality, Reliability and Security (QRS)*, pages 176–185.
- [2] Agrahari, R., Foroushani, A., Docking, T. R., Chang, L., Duns, G., Hudoba, M., Karsan, A., and Zare, H. (2018). Applications of Bayesian network models in predicting types of hematological malignancies. *Scientific reports*, 8(1):1–12.
- [3] Aiken, S., Grunwald, D., Pleszkun, A. R., and Willeke, J. (2003). A performance analysis of the iSCSI protocol. In *20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies, 2003.(MSST 2003). Proceedings.*, pages 123–134. IEEE.
- [4] Alahmad, Y., Agarwal, A., and Daradkeh, T. (2018). High availability management for applications services in the cloud container-based platform. In *2018 IEEE/ACS 15th International Conference on Computer Systems and Applications (AICCSA)*, pages 1–8. IEEE.
- [5] Algirdas Avižienis, Laprie Jean-Claude, Randell Brian, L. C. (2004). Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33.
- [6] Amazon Web Services Inc (2016). SAP on Amazon Web Services High Availability Guide.
- [7] Amazon Web Services Inc. (2018). Shared Responsibility Model.
- [8] Arroyo-Figueroa, G. and Sucar, L. E. (2005). Temporal Bayesian network of events for diagnosis and prediction in dynamic domains. *Applied Intelligence*, 23(2):77–86.
- [9] Avizienis, A., Laprie, J.-C., Randell, B., and Landwehr, C. (2004). Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33.
- [10] Ayuso, P. N., Gasca, R. M., and Lefèvre, L. (2009). Demystifying cluster-based fault-tolerant firewalls. *IEEE Internet Computing*, 13(6):31–38.
- [11] Bailey, D., Frank-Schultz, E., Lindeque, P., and Temple III, J. L. (2008). Three reliability engineering techniques and their application to evaluating the availability of IT systems: An introduction. *IBM Systems Journal*, 47(4):577–589.
- [12] Bajohr, M. and Margaria, T. (2008). High service availability in MaTRICS for the OCS. In *International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*, pages 572–586. Springer.
- [13] Baldoni, R., Montanari, L., and Rizzuto, M. (2015). On-line failure prediction in safety-critical systems. *Future Generation Computer Systems*, 45:123–132.

Bibliography

- [14] Barber, D. (2012). *Bayesian reasoning and machine learning*. Cambridge University Press.
- [15] Barroso, L. A. and Hölzle, U. (2009). The datacenter as a computer: An introduction to the design of warehouse-scale machines. *Synthesis Lectures on Computer Architecture*, 4(1):1–108.
- [16] Bartkowski, S., De Buitlear, C., Kalicki, A., Loster, M., Marczewski, M., Mosaad, A., Nelken, J., Soliman, M., Subtil, K., Vrhovnik, M., et al. (2012). *High availability and disaster recovery options for DB2 for Linux, UNIX, and Windows*. IBM Redbooks.
- [17] BayesFusion, L. (2017). Genie modeler. *BayesFusion, LLC, Pittsburgh, PA, accessed Nov, 30:2017*.
- [18] BayesFusion, L. (2019). Genie modeler. *User Manual. Available online: <https://support.bayesfusion.com/docs/>(accessed on 19 January 2019)*.
- [19] BayesFusion, L. (2020). Genie modeler. *BayesFusion, LLC, Pittsburgh, PA*.
- [20] Beekhof, A. (2017). Pacemaker 1.1 Configuration Explained An A-Z guide to Pacemaker’s Configuration Options.
- [21] Benz, K. and Bohnert, T. M. (2014). Impact of Pacemaker failover configuration on mean time to recovery for small cloud clusters. In *2014 IEEE 7th International Conference on Cloud Computing*, pages 384–391. IEEE.
- [22] Bharat, B., Chua, S., Elkins, C., Scicluna, M., Yang, W.-D., et al. (2010). *High Availability in WebSphere Messaging Solutions*. IBM Redbooks.
- [23] Bielza, C. and Larranaga, P. (2014). Bayesian networks in neuroscience: a survey. *Frontiers in Computational Neuroscience*, 8:131.
- [24] Bielza, C. and Larrañaga, P. (2014). Discrete Bayesian network classifiers: a survey. *ACM Computing Surveys (CSUR)*, 47(1):1–43.
- [25] Birman, K., Van Renesse, R., and Vogels, W. (2004). Adding high availability and autonomic behavior to web services. In *Proceedings of the 26th International Conference on Software Engineering*, pages 17–26. IEEE Computer Society.
- [26] Birman, K. P. (2012). *Guide to Reliable Distributed Systems: Building High-Assurance Applications and Cloud-Hosted Services*. Springer Science & Business Media.
- [27] Bonomi, F., Milito, R., Natarajan, P., and Zhu, J. (2014). Fog computing: A platform for internet of things and analytics. In *Big Data and Internet of Things: A roadmap for smart environments*, pages 169–186. Springer.
- [28] Bottone, S., Lee, D., O’Sullivan, M., and Spivack, M. (2008). Failure prediction and diagnosis for satellite monitoring systems using Bayesian networks. In *MILCOM 2008-2008 IEEE Military Communications Conference*, pages 1–7. IEEE.
- [29] Bouizem, Y., Parlavantzas, N., Dib, D., and Morin, C. (2020). Active-Standby for High-Availability in FaaS. In *Proceedings of the 2020 Sixth International Workshop on Serverless Computing*, pages 31–36.
- [30] Buyya, R. et al. (1999). High performance cluster computing: Architectures and systems (volume 1). *Prentice Hall, Upper SaddleRiver, NJ, USA*, 1:999.

- [31] Cai, Z.-q., Guo, P., Si, S.-b., Geng, Z.-m., Chen, C., and Cong, L.-l. (2017). Analysis of prognostic factors for survival after surgery for gallbladder cancer based on a Bayesian network. *Scientific reports*, 7(1):1–10.
- [32] Calinescu, R. and Di Giandomenico, F. (2021). Special issue on resilient software and software-controlled systems. *Computing*, 103(4):533–534.
- [33] Calinescu, R., Weyns, D., Gerasimou, S., Iftikhar, M. U., Habli, I., and Kelly, T. (2017). Engineering trustworthy self-adaptive software with dynamic assurance cases. *IEEE Transactions on Software Engineering*, 44(11):1039–1069.
- [34] Carlson, M. (2014). Systems and Virtualization Management: Standards and the Cloud. A report on SVM 2013. *Journal of network and systems management*, 22(4):709–715.
- [35] Carriger, J. F. and Parker, R. A. (2021). Conceptual Bayesian networks for contaminated site ecological risk assessment and remediation support. *Journal of Environmental Management*, 278:111478.
- [36] Carvalho, C. M., Christina, D., Saade, M., Conci, A., Seixas, F. L., and Laks, J. (2017). A clinical decision support system for aiding diagnosis of Alzheimer’s disease and related disorders in mobile devices. In *2017 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE.
- [37] Casale, G., Ragusa, C., and Parpas, P. (2013). A feasibility study of host-level contention detection by guest virtual machines. In *2013 IEEE 5th International Conference on Cloud Computing Technology and Science*, volume 2, pages 152–157. IEEE.
- [38] Chaves, I. C., de Paula, M. R. P., Leite, L. G., Queiroz, L. P., Gomes, J. P. P., and Machado, J. C. (2016). BaNHFaP: A Bayesian Network Based Failure Prediction Approach for Hard Disk Drives. In *2016 5th Brazilian Conference on Intelligent Systems (BRACIS)*, pages 427–432. IEEE.
- [39] Check Point Software Technologies Ltd (2018). ClusterXL Administration Guide R80.10.
- [40] Cheng, F. T., Wu, S. L., Tsai, P. Y., Chung, Y. T., and Yang, H. C. (2005). Application cluster service scheme for near-zero-downtime services. *Proceedings - IEEE International Conference on Robotics and Automation*, 2005(April):4062–4067.
- [41] Cheng, J. and Greiner, R. (2001). Learning Bayesian Belief Network Classifiers: Algorithms and System. In *Conference of the Canadian Society for Computational Studies of Intelligence*, pages 141–151. Springer.
- [42] Chow, R., Golle, P., Jakobsson, M., Shi, E., Staddon, J., Masuoka, R., and Molina, J. (2009). Controlling data in the cloud. In *Proceedings of the 2009 ACM workshop on Cloud computing security - CCSW '09*, page 85. ACM.
- [43] Christoforou, A. and Andreou, A. S. (2013). A cloud adoption decision support model using influence diagrams. In *IFIP International Conference on Artificial Intelligence Applications and Innovations*, pages 151–160. Springer.
- [44] Claudel, B., De Palma, N., Lachaize, R., and Hagimont, D. (2006). Self-protection for distributed component-based applications. In *Symposium on self-stabilizing systems*, pages 184–198. Springer.
- [45] Clemente, R., Bartoli, M., Bossi, M. C., D’orazio, G., and Cosmo, G. (2005). Risk management in availability SLA. In *DRCN 2005. Proceedings. 5th International Workshop on Design of Reliable Communication Networks, 2005.*, pages 8–pp. IEEE.

Bibliography

- [46] Codetta-Raiteri, D. and Portinale, L. (2014). Dynamic Bayesian networks for fault detection, identification, and recovery in autonomous spacecraft. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 45(1):13–24.
- [47] Constantinou, A. C., Fenton, N., Marsh, W., and Radlinski, L. (2016). From complex questionnaire and interviewing data to intelligent Bayesian network models for medical decision support. *Artificial intelligence in medicine*, 67:75–93.
- [48] Corsava, S. and Getov, V. (2003). Intelligent architecture for automatic resource allocation in computer clusters. In *Proceedings International Parallel and Distributed Processing Symposium*, pages 8–pp. IEEE.
- [49] Critchley, T. (2014). *High availability IT services*. Auerbach Publications.
- [50] Cully, B., Lefebvre, G., Meyer, D., Feeley, M., Hutchinson, N., and Warfield, A. (2008). Remus: High availability via asynchronous virtual machine replication. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, pages 161–174. San Francisco.
- [51] Dake, S. C., Caulfield, C., and Beekhof, A. (2008). The Corosync cluster engine. In *Linux Symposium*, volume 85. Citeseer.
- [52] Dantas, J., Matos, R., Araujo, J., and Maciel, P. (2012). An availability model for Eucalyptus platform: An analysis of warm-standby replication mechanism. In *Systems, Man, and Cybernetics (SMC), 2012 IEEE International Conference on*, pages 1664–1669. IEEE.
- [53] De Palma, N., Hagimont, D., Boyer, F., and Broto, L. (2011). Self-protection in a clustered distributed system. *IEEE Transactions on Parallel and Distributed Systems*, 23(2):330–336.
- [54] Défago, X., Schiper, A., and Urbán, P. (2004). Total order broadcast and multicast algorithms: Taxonomy and survey. *ACM Computing Surveys (CSUR)*, 36(4):372–421.
- [55] Dell Technologies Inc. (2012). EMC Mission-Critical Business Continuity for SAP.
- [56] DelValle, R., Kaushik, P., Jain, A., Hartog, J., and Govindaraju, M. (2017). Electron: Towards efficient resource management on heterogeneous clusters with apache mesos. In *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, pages 262–269. IEEE.
- [57] Demchenko, Y., De Laat, C., and Membrey, P. (2014). Defining architecture components of the big data ecosystem. In *Collaboration Technologies and Systems (CTS), 2014 International Conference on*, pages 104–112. IEEE.
- [58] DH2i. (2020). DxEnterprise.
- [59] Distefano, S., Longo, F., and Scarpa, M. (2010). Availability assessment of HA standby redundant clusters. In *2010 29th IEEE Symposium on Reliable Distributed Systems*, pages 265–274. IEEE.
- [60] Dolev, D. and Malki, D. (1996). The Transis approach to high availability cluster communication. *Communications of the ACM*, 39(4):64–70.
- [61] Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R., and Safina, L. (2017). Microservices: yesterday, today, and tomorrow. In *Present and Ulterior Software Engineering*, pages 195–216. Springer.
- [62] Du, M., Li, F., Zheng, G., and Srikumar, V. (2017). Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1285–1298.

- [63] Dukaric, R. and Juric, M. B. (2013). Towards a unified taxonomy and architecture of cloud frameworks. *Future Generation Computer Systems*, 29(5):1196–1210.
- [64] Endo, P. T., Rodrigues, M., Gonçalves, G. E., Kelner, J., Sadok, D. H., and Curescu, C. (2016). High availability in clouds: systematic review and research challenges. *Journal of Cloud Computing*, 5(1):16.
- [65] Engelmann, C. (2008). *Symmetric Active/Active High Availability for High-Performance Computing System Services*. PhD thesis, Department of Computer Science, University of Reading, UK.
- [66] Engelmann, C., Scott, S. L., Leangsuksun, C., and He, X. (2008). Symmetric active/active high availability for high-performance computing system services: Accomplishments and limitations. In *2008 Eighth IEEE International Symposium on Cluster Computing and the Grid (CCGRID)*, pages 813–818. IEEE.
- [67] Engelmann, C., Scott, S. L., Leangsuksun, C., and He, X. B. (2006). Symmetric active/active high availability for high-performance computing system services. *JCP*, 1(8):43–54.
- [68] Fawcett, T. (2006). An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874.
- [69] Fernandes, D. A. B., Soares, L. F. B., Gomes, J. V., Freire, M. M., and Inácio, P. R. M. (2014). Security issues in cloud environments: a survey. *International Journal of Information Security*, 13(2):113–170.
- [70] Fernandez, J., Martinez-Selles, M., and Arredondo, M. (2004). Bayesian networks and influence diagrams as valid decision support tools in systolic heart failure management. In *Computers in Cardiology, 2004*, pages 181–184. IEEE.
- [71] Fishburn, P. C. (1988). *Nonlinear preference and utility theory*. Number 5. Johns Hopkins University Press Baltimore.
- [72] Fishburn, P. C. (2013). *The Foundations of Expected Utility*, volume 31. Springer Science & Business Media.
- [73] Fondo-Ferreiro, P., Gil-Castiñeira, F., González-Castaño, F. J., and Candal-Ventureira, D. (2020). A software-defined networking solution for transparent session and service continuity in dynamic multi-access edge computing. *IEEE Transactions on Network and Service Management*, 18(2):1401–1414.
- [74] Forouzan, A. B. (2007). *Data communications & networking (sie)*. Tata McGraw-Hill Education.
- [75] Franke, U. (2011). Optimal it service availability: Shorter outages, or fewer? *IEEE Transactions on Network and Service Management*, 9(1):22–33.
- [76] Friedman, N., Linial, M., Nachman, I., and Pe’er, D. (2000). Using Bayesian networks to analyze expression data. *Journal of computational biology*, 7(3-4):601–620.
- [77] Fujitsu Limited (2017). Primecluster: Installation and Administration Guide 4.5.
- [78] Fuster-Parra, P., Tauler, P., Bennasar-Veny, M., Ligeza, A., Lopez-Gonzalez, A., and Aguilo, A. (2016). Bayesian network modeling: A case study of an epidemiologic system analysis of cardiovascular risk. *Computer Methods and Programs in Biomedicine*, 126:128–142.

Bibliography

- [79] Gainaru, A., Cappello, F., Fullop, J., Trausan-Matu, S., and Kramer, W. (2011). Adaptive event prediction strategy with dynamic time window for large-scale hpc systems. In *Managing Large-scale Systems via the Analysis of System Logs and the Application of Machine Learning Techniques*, pages 1–8.
- [80] Gartner Inc. (2016). Market Share Analysis: Customer Relationship Management Software, Worldwide, 2016. Technical report, Gartner, Inc., Stamford.
- [81] Gartner Inc. (2017a). Magic Quadrant for Operational Database Management Systems. Technical report, Gartner, Inc., Stamford.
- [82] Gartner Inc. (2017b). Market share analysis: ERP software, worldwide, 2017. Technical report, Gartner, Inc., Stamford.
- [83] Ghahramani, Z. (1997). Learning dynamic Bayesian networks. In *International School on Neural Networks, Initiated by IIASS and EMFCSC*, pages 168–197. Springer.
- [84] Ghosh, D., Sharman, R., Rao, H. R., and Upadhyaya, S. (2007). Self-healing systems—survey and synthesis. *Decision Support Systems*, 42(4):2164–2185.
- [85] GILABERT, A. G. E. (2011). Mapping fmea into bayesian networks. *International Journal of Performability Engineering*, 7(6):525.
- [86] Gokhale, A., Schmidt, D. C., Natarajan, B., and Wang, N. (2002). Applying model-integrated computing to component middleware and enterprise applications. *Communications of the ACM*, 45(10):65–70.
- [87] Goldstein, J., Abdelhamid, A., Barnett, M., Burckhardt, S., Chandramouli, B., Gehring, D., Lebeck, N., Meiklejohn, C., Minhas, U. F., Newton, R., et al. (2020). A.M.B.R.O.S.I.A: Providing performant virtual resiliency for distributed applications. *Proceedings of the VLDB Endowment*, 13(5):588–601.
- [88] Gomes, C., Tavares, E., Junior, M. N. d. O., and Nogueira, B. (2021). Cloud storage availability and performance assessment: a study based on NoSQL DBMS. *The Journal of Supercomputing*, pages 1–21.
- [89] Gómez, A., Carril, L., Valin, R., Mouriño, J. C., and Cotelo, C. (2014). Fault-tolerant virtual cluster experiments on federated sites using bonfire. *Future Generation Computer Systems*, 34:17–25.
- [90] Gonçalves, G. E., Endo, P. T., Rodrigues, M., Sadok, D. H., Kelner, J., and Curescu, C. (2020). Resource allocation based on redundancy models for high availability cloud. *Computing*, 102(1):43–63.
- [91] Haddad, I., Leangsuksun, C., and Scott, S. L. (2003). HA-OSCAR: the birth of highly available OSCAR. *Linux Journal*, 2003(115):1.
- [92] Hamill, M. and Goseva-Popstojanova, K. (2015). Exploring fault types, detection activities, and failure severity in an evolving safety-critical software system. *Software Quality Journal*, 23(2):229–265.
- [93] Hanmer, R. S. (2013). *Patterns for Fault Tolerant Software*. John Wiley & Sons.
- [94] Hasenstein, M. (2001). The Logical Volume Manager (LVM). *White paper*.

-
- [95] He, J., Dai, T., Gu, X., and Jin, G. (2020). Hangfix: automatically fixing software hang bugs for production cloud systems. In *Proceedings of the 11th ACM Symposium on Cloud Computing*, pages 344–357.
- [96] He, P., Zhu, J., Xu, P., Zheng, Z., and Lyu, M. R. (2018). A directed acyclic graph approach to online log parsing. *arXiv preprint arXiv:1806.04356*.
- [97] He, X., Ou, L., Engelmann, C., Chen, X., and Scott, S. L. (2009). Symmetric active/active metadata service for high availability parallel file systems. *Journal of Parallel and Distributed Computing*, 69(12):961–973.
- [98] Heimovski, G. B., Turchetti, R. C., Wickboldt, J. A., Granville, L. Z., and Duarte Jr, E. P. (2020). FT-Aurora: A highly available IaaS cloud manager based on replication. *Computer Networks*, 168:107041.
- [99] Hewlett Packard Enterprise Development L. P. (2011). Designing Disaster Recovery HA Clusters using Metrocluster and Continentalclusters.
- [100] Hewlett Packard Enterprise Development L. P. (2012). Understanding and Designing Service-guard Disaster Recovery Architectures.
- [101] Hewlett Packard Enterprise Development L. P. (2016). Managing HPE Serviceguard Extension for SAP for Linux Version.
- [102] Hiep, M. Q., Yang, H., and Kim, Y. (2020). Dynamic policy management system for high availability in a multi-site cloud. In *2020 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 359–362. IEEE.
- [103] Hou, Z., Huang, Y., Zheng, S., Dong, X., and Wang, B. (2003). Design and implementation of heartbeat in multi-machine environment. In *17th International Conference on Advanced Information Networking and Applications, 2003. AINA 2003.*, pages 583–586. IEEE.
- [104] Howard, R. A. (1988). Decision analysis: practice and promise. *Management science*, 34(6):679–695.
- [105] Hsueh, M.-C., Tsai, T. K., and Iyer, R. K. (1997). Fault injection techniques and tools. *Computer*, 30(4):75–82.
- [106] Hu, H., Wen, Y., Chua, T.-S., and Li, X. (2014). Toward Scalable Systems for Big Data Analytics: A Technology Tutorial. *IEEE Access*, 2:652–687.
- [107] Huang, Y., Kintala, C., Kolettis, N., and Fulton, N. D. (1995). Software rejuvenation: Analysis, module and applications. In *wenty-Fifth International Symposium on Fault-Tolerant Computing*, pages 381–390. IEEE.
- [108] Hunter, S. W. and Smith, W. E. (1999). Availability modeling and analysis of a two node cluster. In *Proceedings of the 5th International Conference on Information Systems, Analysis and Synthesis*.
- [109] Hwang, K.-S. and Cho, S.-B. (2009). Landmark detection from mobile life log using a modular Bayesian network model. *Expert Systems with Applications*, 36(10):12065–12076.
- [110] IBM Corp. (2017). Tivoli system automation for multiplatforms v4.1: Failover scenarios.
- [111] IBM Corporation (2016). PowerHA SystemMirror concepts.

Bibliography

- [112] IBM Corporation (2017a). IBM PowerHA SystemMirror for AIX: Geographic Logical Volume Manager.
- [113] IBM Corporation (2017b). IBM Spectrum Scale Concepts, Planning, and Installation Guide.
- [114] IBM Corporation (2017c). Smart Assists for PowerHA SystemMirror.
- [115] IBM Corporation (2018a). Administering PowerHA SystemMirror.
- [116] IBM Corporation (2018b). Supported cluster management software.
- [117] IBM Redbooks. (2008). Building High Availability with SteelEye LifeKeeper for SAP NetWeaver on SUSE Linux Enterprise Server.
- [118] IDC Corporation (2016). Clustering Solutions for Achieving High Availability for Diversifying Platforms: The Future in Advanced Best Practices. Technical report, IDC Corporation, Massachusetts.
- [119] ISO/IEC 25010:2011 (2011). *Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuARE) — System and software quality models*.
- [120] Jammal, M., Kanso, A., Heidari, P., and Shami, A. (2016). Availability analysis of cloud deployed applications. In *2016 IEEE International Conference on Cloud Engineering (IC2E)*, pages 234–235. IEEE.
- [121] Johnson, P., Lagerström, R., Närman, P., and Simonsson, M. (2007). Enterprise architecture analysis with extended influence diagrams. *Information Systems Frontiers*, 9(2):163–180.
- [122] Kaitovic, I. and Malek, M. (2018). Impact of failure prediction on availability: Modeling and comparative analysis of predictive and reactive methods. *IEEE Transactions on Dependable and Secure Computing*, 17(3):493–505.
- [123] Kanagavelu, R., Lee, B. S., Miguel, R. F., Mingjie, L. N., et al. (2013). Software defined network based adaptive routing for data replication in data centers. In *2013 19th IEEE International Conference on Networks (ICON)*, pages 1–6. IEEE.
- [124] Kanso, A. and Lemieux, Y. (2013). Achieving high availability at the application level in the cloud. In *2013 IEEE 6th International Conference on Cloud Computing (CLOUD)*, pages 778–785. IEEE.
- [125] Kanso, A., Toeroe, M., and Khendek, F. (2014). Comparing redundancy models for high availability middleware. *Computing*, 96(10):975–993.
- [126] Kao, H.-Y. (2008). Diagnostic reasoning and medical decision-making with fuzzy influence diagrams. *Computer Methods and Programs in Biomedicine*, 90(1):9–16.
- [127] Khakzad, N. (2021). Optimal firefighting to prevent domino effects: Methodologies based on dynamic influence diagram and mathematical programming. *Reliability Engineering & System Safety*, 212:107577.
- [128] Khan, M., Toeroe, M., and Khendek, F. (2017). Comparing Pacemaker with OpenSAF for availability management in the cloud. In *Edge Computing (EDGE), 2017 IEEE International Conference on*, pages 106–111. IEEE.
- [129] Kim, D. S., Machida, F., and Trivedi, K. S. (2009). Availability modeling and analysis of a virtualized system. In *2009 15th IEEE Pacific Rim International Symposium on Dependable Computing*, pages 365–371. IEEE.

- [130] Kimura, T., Watanabe, A., Toyono, T., and Ishibashi, K. (2019). Proactive failure detection learning generation patterns of large-scale network logs. *IEICE Transactions on Communications*, 102(2):306–316.
- [131] Kjaerulff, U. B. and Madsen, A. L. (2008). Bayesian networks and influence diagrams. *Springer Science+ Business Media*, 200:114.
- [132] Koiter, J. R. (2006). Visualizing inference in Bayesian networks. Master’s thesis, Delft University of Technology.
- [133] Koller, D. and Friedman, N. (2009). *Probabilistic graphical models: principles and techniques*. MIT press.
- [134] Korb, K. B. and Nicholson, A. E. (2010). *Bayesian artificial intelligence*. CRC press.
- [135] Koren, I. and Krishna, C. M. (2007). *Fault-Tolerant Systems*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition.
- [136] Koski, T. and Noble, J. (2011). *Bayesian Networks: An Introduction*, volume 924. John Wiley & Sons.
- [137] Le, L. H., Bezerra, C. E., and Pedone, F. (2016). Dynamic scalable state machine replication. In *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 13–24. IEEE.
- [138] Leander, J., Honfi, D., Ivanov, O. L., and Björnsson, Í. (2018). A decision support framework for fatigue assessment of steel bridges. *Engineering Failure Analysis*, 91:306–314.
- [139] Leangsuksun, C., Liu, T., Rao, T., Scott, S., and Libby, R. (2004). A Failure Predictive and Policy-Based High Availability Strategy for Linux High Performance Computing Cluster. In *The 5th LCI International Conference on Linux Clusters: The HPC Revolution*, pages 18–20. Citeseer.
- [140] Leangsuksun, C., Munganuru, V., Liu, T., Scott, S., and Engelmann, C. (2005). Asymmetric active-active high availability for high-end computing. In *Proceedings of 2nd International Workshop on Operating Systems, Programming Environments and Management Tools for High-Performance Computing on Clusters (COSET-2)*.
- [141] Lee, J., Moon, S., Kim, K. H., Kim, D. H., Cha, S. K., and Han, W.-S. (2017). Parallel replication across formats in SAP HANA for scaling out mixed OLTP/OLAP workloads. *Proceedings of the VLDB Endowment*, 10(12):1598–1609.
- [142] Lee, Y.-J., Kim, H.-Y., and Lee, C.-H. (2008). A Stochastic Availability Prediction Model for Head Nodes in the HA Cluster. In *22nd International Conference on Advanced Information Networking and Applications-Workshops (aina workshops 2008)*, pages 157–161. IEEE.
- [143] Lee, Y.-L., Arizky, S. N., Chen, Y.-R., Liang, D., and Wang, W.-J. (2021). High-availability computing platform with sensor fault resilience. *Sensors*, 21(2):542.
- [144] Levitin, G., Xing, L., and Dai, Y. (2014). Cold vs. hot standby mission operation cost minimization for 1-out-of-n systems. *European Journal of Operational Research*, 234(1):155–162.
- [145] Li, W. and Kanso, A. (2015). Comparing containers versus virtual machines for achieving high availability. In *2015 IEEE International Conference on Cloud Engineering*, pages 353–358. IEEE.
- [146] Li, Y. and Lan, Z. (2006). Exploit failure prediction for adaptive fault-tolerance in cluster computing. In *Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID’06)*, volume 1, pages 8–pp. IEEE.

Bibliography

- [147] Liu, B., Chang, X., Han, Z., Trivedi, K., and Rodríguez, R. J. (2018). Model-based sensitivity analysis of IaaS cloud availability. *Future Generation Computer Systems*, 83:1–13.
- [148] Liu, F., Tong, J., Mao, J., Bohn, R., Messina, J., Badger, L., and Leaf, D. (2011). NIST cloud computing reference architecture. *NIST special publication*, 500(2011):1–28.
- [149] Liu, H.-C., Liu, L., and Liu, N. (2013). Risk evaluation approaches in failure mode and effects analysis: A literature review. *Expert systems with applications*, 40(2):828–838.
- [150] Liu, T., Song, H., et al. (2003). Availability prediction and modeling of high mobility OSCAR cluster. In *2003 Proceedings IEEE International Conference on Cluster Computing*, pages 380–386. IEEE.
- [151] Liu, Y., Shen, Y., Chen, Y., and Gao, F. (2004). The integrated process of project risk management based on influence diagrams. In *2004 IEEE International Engineering Management Conference (IEEE Cat. No. 04CH37574)*, volume 2, pages 746–750. IEEE.
- [152] Long, F., Zeiler, P., and Bertsche, B. (2016). Modelling the production systems in industry 4.0 and their availability with high-level Petri nets. *IFAC-PapersOnLine*, 49(12):145–150.
- [153] Longo, F., Ghosh, R., Naik, V. K., and Trivedi, K. S. (2011). A scalable availability model for infrastructure-as-a-service cloud. In *2011 IEEE/IFIP 41st International Conference on Dependable Systems & Networks (DSN)*, pages 335–346. IEEE.
- [154] Loveland, S., Dow, E. M., LeFevre, F., Beyer, D., and Chan, P. F. (2008). Leveraging virtualization to optimize high-availability system configurations. *IBM Systems Journal*, 47(4):591–604.
- [155] Lu, Y., Yu, X., Cao, L., and Madden, S. (2021). Epoch-based commit and replication in distributed OLTP databases. *Proceedings of the VLDB Endowment*, 14(5):743–756.
- [156] Lumppp, T., Schneider, J., Holtz, J., Mueller, M., Lenz, N., Biazetti, A., and Petersen, D. (2008). From high availability and disaster recovery to business continuity solutions. *IBM Systems Journal*, 47(4):605–619.
- [157] Luoma, E., Nevalainen, L., Altarriba, E., Helle, I., and Lehikoinen, A. (2021). Developing a conceptual influence diagram for socio-eco-technical systems analysis of biofouling management in shipping—A Baltic Sea case study. *Marine Pollution Bulletin*, 170:112614.
- [158] Lyu, H., Li, P., Yan, R., Qian, H., and Sheng, B. (2016). High-availability deployment for large enterprises. In *2016 International Conference on Progress in Informatics and Computing (PIC)*, pages 503–507. IEEE.
- [159] Machida, F., Andrade, E., Kim, D. S., and Trivedi, K. S. (2011). Candy: Component-based availability modeling framework for cloud service management using sysml. In *2011 IEEE 30th International Symposium on Reliable Distributed Systems*, pages 209–218. IEEE.
- [160] Magalhaes, A., Monteiro, J. M., and Brayner, A. (2021). Main memory database recovery: A survey. 54(2).
- [161] Magnanini, F., Ferretti, L., and Colajanni, M. (2021). Scalable, confidential and survivable software updates. *IEEE Transactions on Parallel and Distributed Systems*.
- [162] Malhotra, M. and Trivedi, K. S. (1995). Dependability modeling using Petri-nets. *IEEE Transactions on Reliability*, 44(3):428–440.

- [163] Malkhi, D., Reiter, M. K., Wool, A., and Wright, R. N. (2001). Probabilistic quorum systems. *Information and Computation*, 170(2):184–206.
- [164] Maloy, J. (2004). TIPC: Providing Communication for Linux Clusters. In *Linux Symposium*, volume 2, pages 347–356.
- [165] Mansouri, Y., Toosi, A. N., and Buyya, R. (2018). Data storage management in cloud environments: Taxonomy, survey, and future directions. *ACM Computing Surveys (CSUR)*, 50(6):91.
- [166] Marcot, B. G. (2012). Metrics for evaluating performance and uncertainty of Bayesian network models. *Ecological modelling*, 230:50–62.
- [167] Marcus, E. and Stern, H. (2003). *Blueprints for high availability*. John Wiley & Sons, Indianapolis, Indiana.
- [168] Mayerl, C., Huner, K. M., Gaspar, J.-U., Momm, C., and Abeck, S. (2007). Definition of metric dependencies for monitoring the impact of quality of services on quality of processes. In *2007 2nd IEEE/IFIP International Workshop on Business-Driven IT Management*, pages 1–10. IEEE.
- [169] Medjaher, K., Moya, J.-Y., and Zerhouni, N. (2009). Failure prognostic by using dynamic Bayesian networks. *IFAC Proceedings Volumes*, 42(5):257–262.
- [170] Megargel, A., Shankararaman, V., and Walker, D. K. (2020). Migrating from monoliths to cloud-based microservices: A banking industry example. In *Software Engineering in the Era of Cloud Computing*, pages 85–108. Springer.
- [171] Microsoft Corporation (2011). Failover Cluster Step-by-Step Guide: Configuring the Quorum in a Failover Cluster.
- [172] Microsoft Corporation (2016). Behavior of Dynamic Witness on Windows Server 2012 R2 Failover Clustering.
- [173] Microsoft Corporation (2017). Shared Responsibilities for Cloud Computing.
- [174] Microsoft Corporation (2018). Failover Clustering in Windows Server.
- [175] Microsoft Corporation (2020). High availability for SAP NetWeaver on Azure VMs on SUSE Linux Enterprise Server for SAP applications.
- [176] Minhas, U. F., Rajagopalan, S., Cully, B., Abounaga, A., Salem, K., and Warfield, A. (2013). Remusdb: Transparent high availability for database systems. *The VLDB Journal—The International Journal on Very Large Data Bases*, 22(1):29–45.
- [177] Mondal, S. K., Pan, R., Kabir, H., Tian, T., and Dai, H.-N. (2022). Kubernetes in it administration and serverless computing: An empirical study and research challenges. *The Journal of Supercomputing*, 78(2):2937–2987.
- [178] Mortazavi, S. H., Salehe, M., Balasubramanian, B., de Lara, E., and PuzhavakathNarayanan, S. (2020). SessionStore: A Session-Aware Datastore for the Edge. In *2020 IEEE 4th International Conference on Fog and Edge Computing (ICFEC)*, pages 59–68. IEEE.
- [179] Nabi, M., Toeroe, M., and Khendek, F. (2016). Availability in the cloud: State of the art. *Journal of Network and Computer Applications*, 60:54–67.
- [180] Nagarajan, R., Scutari, M., and Lèbre, S. (2013). *Bayesian networks in R*. Springer.

Bibliography

- [181] Naor, M. and Wool, A. (1998). The Load, Capacity, and Availability of Quorum Systems. *SIAM Journal on Computing*, 27(2):423–447.
- [182] Neapolitan, R., Jiang, X., Ladner, D. P., and Kaplan, B. (2016). A primer on Bayesian decision analysis with an application to a personalized kidney transplant decision. *Transplantation*, 100(3):489.
- [183] NEC Corporation (2017a). EXPRESSCLUSTER X 3.3 for Linux Reference Guide.
- [184] NEC Corporation (2017b). EXPRESSCLUSTER X 3.3 for Windows Reference Guide.
- [185] Nelson, R. and Staggers, N. (2016). *Health informatics-e-book: an interprofessional approach*. Elsevier Health Sciences.
- [186] Nguyen, T. A., Kim, D. S., and Park, J. S. (2016). Availability modeling and analysis of a data center for disaster tolerance. *Future Generation Computer Systems*, 56:27–50.
- [187] Nielsen, J. J. and Sørensen, J. D. (2010). Bayesian networks as a decision tool for O&M of offshore wind turbines. In *ASRANet: Integrating Structural Analysis, Risk & Reliability: 5th International ASRANet Conference, Edinburgh, UK, 14-16 June 2010*. ASRANet Ltd.
- [188] Nielsen, T. D. and Jensen, F. V. (2009). *Bayesian networks and decision graphs*. Springer Science & Business Media.
- [189] Noble, J., Maxwell, D., Hourihan, K. X., and Stephens, R. (2003). *Check Point NG VPN-1/FireWall-1: Advanced configuration and troubleshooting*. Syngress (Canada).
- [190] Novell, Inc. (2014). SAP Applications Made High Available on SUSE Linux Enterprise Server 10.
- [191] O’Connor, P. P. and Kleyner, A. (2012). *Practical Reliability Engineering*. John Wiley & Sons, Ltd, New York, 5th Edition edition.
- [192] Oniško, A. and Druzdzal, M. J. (2014). Impact of Bayesian Network Model Structure on the Accuracy of Medical Diagnostic Systems. In *International Conference on Artificial Intelligence and Soft Computing*, pages 167–178. Springer.
- [193] Oonk, S. and Maldonado, F. J. (2016). Automated maintenance path generation with Bayesian networks, influence diagrams, and timed failure propagation graphs. In *2016 IEEE AUTOTEST-CON*, pages 1–9. IEEE.
- [194] Oppenheimer, D., Ganapathi, A., and Patterson, D. A. (2003). Why do Internet services fail, and what can be done about it? In *USENIX symposium on internet technologies and systems*, volume 67. Seattle, WA.
- [195] Oracle Corporation (2010). Oracle Solaris and Oracle Solaris Cluster: Extending Oracle Solaris for Business Continuity.
- [196] Oracle Corporation (2013). Operating SAP Landscapes on Oracle Engineered Systems Using ITIL Best Practices.
- [197] Oracle Corporation (2014). Solaris Cluster Concepts Guide.
- [198] Oracle Corporation (2015). Oracle Solaris Cluster System Administration Guide.
- [199] Oracle Corporation (2016a). Oracle Solaris Cluster 4.3 Geographic Edition Overview.

- [200] Oracle Corporation (2016b). Solaris Cluster Data Service for Oracle E-Business Suite as of Release 12.2 Guide Oracle Solaris Cluster Data Service for Oracle E-Business Suite as of Release 12.2 Guide.
- [201] Oracle Corporation (2016c). Solaris Cluster Data Service for Oracle Real Application Clusters Guide.
- [202] Oracle Corporation (2017a). Oracle Clusterware Administration and Deployment Guide.
- [203] Oracle Corporation (2017b). Oracle Database High Availability Overview.
- [204] Oracle Corporation (2017c). Oracle Fusion Middleware: High Availability Guide.
- [205] Oracle Corporation (2021). Oracle Database 21c (21.3) comes with full production support for Oracle Sharding on Kubernetes and Docker!
- [206] Oracle Corporation (2022). Oracle WebLogic Server: Oracle WebLogic Kubernetes Toolkit.
- [207] Palo Alto Networks Inc. (2018). PAN-OS 8.0 Admin Guide.
- [208] Park, E., Chang, H.-j., and Nam, H. S. (2018). A Bayesian network model for predicting post-stroke outcomes with available risk factors. *Frontiers in neurology*, 9:699.
- [209] Pearl, J. (2011). Bayesian networks.
- [210] Pedone, F., Guerraoui, R., and Schiper, A. (2003). The database state machine approach. *Distributed and Parallel Databases*, 14(1):71–98.
- [211] Pereira Ferreira, A. and Sinnott, R. (2019). A performance evaluation of containers running on managed kubernetes services. In *2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 199–208.
- [212] Pérez-Miguel, C., Mendiburu, A., and Miguel-Alonso, J. (2015). Modeling the availability of Cassandra. *Journal of Parallel and Distributed Computing*, 86:29–44.
- [213] Pillay, A. and Wang, J. (2003). Modified failure mode and effects analysis using approximate reasoning. *Reliability Engineering & System Safety*, 79(1):69–85.
- [214] Pitakrat, T. (2013). Hora: Online Failure Prediction Framework for Component-based Software Systems Based on Kieker and Palladio. In *KPDAYS*, pages 39–48.
- [215] Pitakrat, T., Okanović, D., van Hoorn, A., and Grunske, L. (2018). Hora: Architecture-aware online failure prediction. *Journal of Systems and Software*, 137:669–685.
- [216] Pohanka, T. and Pechanec, V. (2020). Evaluation of replication mechanisms on selected database systems. *ISPRS International Journal of Geo-Information*, 9(4):249.
- [217] Pourret, O., Naïm, P., and Marcot, B. (2008). *Bayesian networks: a practical guide to applications*. John Wiley & Sons.
- [218] Preslan, K. W., Barry, A. P., Brassow, J., Declerck, M., Lewis, A., Manthei, A., Marzinski, B., Nygaard, E., Van Oort, S., Teigland, D., et al. (2000). Scalability and failure recovery in a linux cluster file system. In *Annual Linux Showcase & Conference*.
- [219] Prior, D., MacNeela, A., Brown, I., Krischer, J., Scott, D., and Green-Armytage, J. (2001). Enterprise guide to gartner’s high-availability system model for SAP. *Gartner, December*.

Bibliography

- [220] Quintero, D., Balappa, V., Bodily, S., De, D., Casali, S., Dengel, J.-G., Dhandapani, M., Hoshino, M., Kiran, J., Langer, K., Sergio, P., Queiroz, L., Radford, M., Socoliuc, A., and Tu, N. K. (2013). *IBM PowerHA SystemMirror 7.1.2 Enterprise Edition for AIX*. IBM Redbooks.
- [221] Rabbat, R., McNeal, T., and Burke, T. (2001). A high-availability clustering architecture with data integrity guarantees. In *Third IEEE International Conference on Cluster Computing (CLUSTER'01)*, pages 178–178. IEEE Computer Society.
- [222] Ramos, F., Viegas, E., Santin, A., Horchulhack, P., dos Santos, R. R., and Espindola, A. (2021). A machine learning model for detection of docker-based app overbooking on kubernetes. In *ICC 2021-IEEE International Conference on Communications*, pages 1–6. IEEE.
- [223] Ranade, D. M. (2003). *Shared Data Clusters: Scaleable, Manageable, and Highly Available Systems (Veritas Series)*, volume 9. John Wiley & Sons.
- [224] Rashid, H., Place, C., Mba, D., Lim, R., Healey, A., Kleine-Beek, W., and Romano, M. (2015). Helicopter MGB oil system failure analysis using influence diagrams and random failure probabilities. *Engineering Failure Analysis*, 50:7–19.
- [225] Ratnapinda, P. and Druzdzal, M. J. (2013). An empirical comparison of Bayesian network parameter learning algorithms for continuous data streams. In *The Twenty-Sixth International FLAIRS Conference*.
- [226] Renooij, S. (2001). Probability elicitation for belief networks: issues to consider. *The Knowledge Engineering Review*, 16(3):255.
- [227] Renooij, S. and Witteman, C. (1999). Talking probabilities: communicating probabilistic information with words and numbers. *International Journal of Approximate Reasoning*, 22(3):169–194.
- [228] Riley, J., Noss, J., Dillingham, W., Cuff, J., and Llorente, I. M. (2017). A high-availability cloud for research computing. *Computer*, 50(6):92–95.
- [229] Rios Insua, D., Couce-Vieira, A., Rubio, J. A., Pieters, W., Labunets, K., and G. Rasines, D. (2021). An adversarial risk analysis framework for cybersecurity. *Risk Analysis*, 41(1):16–36.
- [230] Rosendo, D., Gomes, D., Leoni Santos, G., Silva, L., Moreira, A., Kelner, J., Sadok, D., Gonçalves, G., Mehta, A., Wildeman, M., et al. (2020). Availability analysis of design configurations to compose virtual performance-optimized data center systems in next-generation cloud data centers. *Software: Practice and Experience*, 50(6):805–826.
- [231] Rossi, D. and Turrini, E. (2005). Analyzing the impact of components replication in high available J2EE clusters. In *Joint International Conference on Autonomic and Autonomous Systems and International Conference on Networking and Services-(icas-isns' 05)*, pages 56–56. IEEE.
- [232] Roux, J., Berouille, V., Morin-Allory, K., Leveugle, R., Bossuet, L., Cézilly, F., Berthoz, F., Génévrier, G., and Cerisier, F. (2021). High-level fault injection to assess FMEA on critical systems. *Microelectronics Reliability*, 122:114135.
- [233] Roychoudhury, I. and Daigle, M. (2011). An integrated model-based diagnostic and prognostic framework. In *Proceedings of the 22nd International Workshop on Principle of Diagnosis (DX'11)*. Murnau, Germany. Citeseer.
- [234] Salfner, F., Lenk, M., and Malek, M. (2010). A survey of online failure prediction methods. *ACM Computing Surveys (CSUR)*, 42(3):1–42.

- [235] Santos, G. L., Endo, P. T., Goncalves, G., Rosendo, D., Gomes, D., Kelner, J., Sadok, D., and Mahloo, M. (2017). Analyzing the IT subsystem failure impact on availability of cloud services. In *Proceedings - IEEE Symposium on Computers and Communications*, pages 717–723.
- [236] SAP SE (2018). Certified HA-Interface Partners - SAP Application Server High Availability Interface Certification.
- [237] SAS Institute Inc. (2017). SAS 9.4 Intelligence Platform: Overview, Second Edition.
- [238] Saxena, H. and Pound, J. (2020). A cloud-native architecture for replicated data services. In *12th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 20)*.
- [239] Schmidt, K. (2006). *High availability and disaster recovery: concepts, design, implementation*, volume 22. Springer Science & Business Media.
- [240] Schroeder, B. and Gibson, G. A. (2007). Understanding disk failure rates: What does an MTTF of 1,000,000 hours mean to you? *ACM Transactions on Storage (TOS)*, 3(3):8–es.
- [241] Schroeder, B. and Gibson, G. A. (2009). A large-scale study of failures in high-performance computing systems. *IEEE Transactions on Dependable and Secure Computing*, 7(4):337–350.
- [242] Schurink, C., Lucas, P., Hoepelman, I., and Bonten, M. (2005). Computer-assisted decision support for the diagnosis and treatment of infectious diseases in intensive care units. *The Lancet infectious diseases*, 5(5):305–312.
- [243] Seixas, F. L., Zadrozny, B., Laks, J., Conci, A., and Saade, D. C. M. (2014). A Bayesian network decision model for supporting the diagnosis of dementia, Alzheimer’s disease and mild cognitive impairment. *Computers in Biology and Medicine*, 51:140–158.
- [244] Sen, S. D. and Adams, J. A. (2015). An influence diagram based multi-criteria decision making framework for multirobot coalition formation. *Autonomous Agents and Multi-Agent Systems*, 29(6):1061–1090.
- [245] Serrano, D., Bouchenak, S., Kouki, Y., de Oliveira Jr., F. A., Ledoux, T., Lejeune, J., Sopena, J., Arantes, L., and Sens, P. (2016). SLA guarantees for cloud services. *Future Generation Computer Systems*, 54:233–246.
- [246] Service Availability Forum (2011). Service Availability Forum Service Availability Interface.
- [247] Sethi, T., Mittal, A., Maheshwari, S., and Chugh, S. (2019). Learning to Address Health Inequality in the United States with a Bayesian Decision Network. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 710–717.
- [248] Sfondrini, N., Motta, G., and Longo, A. (2018). Public cloud adoption in multinational companies: A survey. In *2018 IEEE International Conference on Services Computing (SCC)*, pages 177–184. IEEE.
- [249] Shankar, I. and Mishra, A. (2013). Enhanced cluster failover management. US Patent 8,484,510.
- [250] Sheghdara, M. and Hassine, J. (2020). Automatic retrieval and analysis of high availability scenarios from system execution traces: A case study on hot standby router protocol. *Journal of Systems and Software*, 161:110490.
- [251] Shi, Y., Zuo, J., Guo, Y., and Lu, Y. (2020). Distributed file system multilevel fault-tolerant high availability mechanism. In *Proceedings of the 2020 International Conference on Cyberspace Innovation of Advanced Technologies*, pages 431–438.

Bibliography

- [252] SIOS Technology Corp. (2017a). SANless Cluster Software.
- [253] SIOS Technology Corp. (2017b). SIOS Protection Suite for Linux Documentation.
- [254] SIOS Technology Corp. (2018). SIOS Protection Suite for Windows.
- [255] Somasekaram, P. (2017). A Component-based Business Continuity and Disaster Recovery Framework. Master's thesis, Uppsala university.
- [256] Somasekaram, P. (2021a). Bayesian Prognostic Framework Preparation Software. <https://github.com/ps234/logInterface/>.
- [257] Somasekaram, P. (2021b). Holistic Modelling Technique for High Availability Software. <https://github.com/ps234/HMTHA/>.
- [258] Stewart, C. and Shen, K. (2005). Performance modeling and system management for multi-component online services. In *Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation-Volume 2*, pages 71–84.
- [259] Stratus Technologies. (2020). everRun.
- [260] Sucar, L. E. (2015a). Dynamic and Temporal Bayesian Networks. In *Probabilistic Graphical Models*, pages 161–177. Springer.
- [261] Sucar, L. E. (2015b). Probabilistic graphical models. *Advances in Computer Vision and Pattern Recognition. London: Springer London. doi*, 10:978–1.
- [262] Sun, Z., Jin, H., Yong, J., Al-Ismaili, S., Li, C., and Shen, J. (2016). A High Availability Application Service Platform for nuclear power enterprises. *2016 IEEE 20th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, pages 613–618.
- [263] SUSE LLC (2012). SAP on SUSE linux enterprise.
- [264] SUSE LLC (2016). Enqueue Replication - SAP NetWeaver High Availability on SUSE Linux Enterprise (12).
- [265] SUSE LLC (2017). Administration Guide SUSE Linux Enterprise High Availability Extension 12 SP3.
- [266] The Kubernetes Authors (2020). Kubernetes Documentation.
- [267] Tobon-Mejia, D. A., Medjaher, K., and Zerhouni, N. (2012). CNC machine tool's wear diagnostic and prognostic by using dynamic Bayesian networks. *Mechanical Systems and Signal Processing*, 28:167–182.
- [268] Toeroe, M. and Tam, F. (2012). *Service Availability: Principles and Practice*. John Wiley & Sons.
- [269] Tomi Silander and Petri Kontkanen and Petri Myllymäki (2007). On sensitivity of the MAP Bayesian network structure to the equivalent sample size parameter. pages 360–367, International. AUAI Press. AUAI Press cop.; 0-9749039-2-2; ; Conference on Uncertainty in Artificial Intelligence ; Conference date: 19-07-2007 Through 22-07-2007.
- [270] Trivedi, K. S. and Bobbio, A. (2017). *Reliability and Availability Engineering: Modeling, Analysis, and Applications*. Cambridge University Press.

- [271] Trivedi, K. S., Vasireddy, R., Trindale, D., Nathan, S., and Castro, R. (2006). Modeling high availability. In *Proceedings of the 12th Pacific Rim International Symposium on Dependable Computing*, PRDC '06, pages 154–164, Washington, DC, USA. IEEE Computer Society.
- [272] Uhlemann, K., Engelmann, C., and Scott, S. L. (2006). JOSHUA: Symmetric Active/Active Replication for Highly Available HPC Job and Resource Management. In *2006 IEEE International Conference on Cluster Computing*, pages 1–10. IEEE.
- [273] Vacca, J. R. (2016). *Cloud Computing Security: Foundations and Challenges*. CRC Press, 1 edition edition.
- [274] Vaidyanathan, K., Harper, R. E., Hunter, S. W., and Trivedi, K. S. (2001). Analysis and implementation of software rejuvenation in cluster systems. In *Proceedings of the 2001 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 62–71.
- [275] van der Gaag, L. C., Renooij, S., Witteman, C. L. M., Aleman, B. M. P., and Taal, B. G. (1999). How to elicit many probabilities. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, UAI'99, page 647–654, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.
- [276] van der Linde, A., Leitão, J., and Preguiça, N. (2020). Practical client-side replication: Weak consistency semantics for insecure settings. *Proceedings of the VLDB Endowment*, 13(12):2590–2605.
- [277] Van Vugt, S. (2014). *Pro Linux high availability clustering*. Apress.
- [278] Vayghan, L. A., Saied, M. A., Toeroe, M., and Khendek, F. (2018). Deploying microservice based applications with kubernetes: Experiments and lessons learned. In *2018 IEEE 11th international conference on cloud computing (CLOUD)*, pages 970–973. IEEE.
- [279] Vayghan, L. A., Saied, M. A., Toeroe, M., and Khendek, F. (2019). Microservice Based Architecture: Towards High-Availability for Stateful Applications with Kubernetes. In *2019 IEEE 19th International Conference on Software Quality, Reliability and Security (QRS)*, pages 176–185. IEEE.
- [280] Vayghan, L. A., Saied, M. A., Toeroe, M., and Khendek, F. (2021). A Kubernetes controller for managing the availability of elastic microservice based stateful applications. *Journal of Systems and Software*, 175:110924.
- [281] Vercellis, C. (2011). *Business intelligence: data mining and optimization for decision making*. John Wiley & Sons.
- [282] Veritas Technologies LLC (2006). A guide to understanding volume replicator. *Engineering White Paper, Symantec*.
- [283] Veritas Technologies LLC (2013). Symantec High Availability Agent for SAP NetWeaver Installation and Configuration Guide.
- [284] Veritas Technologies LLC (2017a). Cluster Server 7.3 Administrator's Guide - Linux.
- [285] Veritas Technologies LLC (2017b). Storage Foundation for Oracle RAC 7.3 Administrator's Guide - Linux.
- [286] Veritas Technologies LLC (2020). Veritas InfoScale 7.4.3 Support for Kubernetes - Linux.

Bibliography

- [287] VMware Inc. (2015). SAP Solutions on VMware vSphere Guidelines Summary and Best Practices.
- [288] Vogels, W., Dumitriu, D., Birman, K., Gamache, R., Massa, M., Short, R., Vert, J., Barrera, J., and Gray, J. (1998). The design and architecture of the Microsoft Cluster Service—a practical approach to high-availability and scalability. In *Fault-Tolerant Computing, 1998. Digest of Papers. Twenty-Eighth Annual International Symposium on*, pages 422–431. IEEE.
- [289] Wang, G., Xu, T., Tang, T., Yuan, T., and Wang, H. (2017). A Bayesian network model for prediction of weather-related failures in railway turnout systems. *Expert systems with applications*, 69:247–256.
- [290] Wang, H., Wang, H., and Shen, J. (2004). Architectural design and implementation of highly available and scalable medical system with ibm websphere middleware. In *Proceedings. 17th IEEE Symposium on Computer-Based Medical Systems*, pages 174–179. IEEE.
- [291] Wang, S. S. and Franke, U. (2020). Enterprise IT service downtime cost and risk transfer in a supply chain. *Operations Management Research*, pages 1–15.
- [292] Wang, X., Sun, H., Deng, T., and Huai, J. (2015). On the tradeoff of availability and consistency for quorum systems in data center networks. *Computer Networks*, 76:191–206.
- [293] Ward, J. S. and Barker, A. (2014). Observing the clouds: a survey and taxonomy of cloud monitoring. *Journal of Cloud Computing*, 3(1):24.
- [294] Watanabe, Y., Otsuka, H., Sonoda, M., Kikuchi, S., and Matsumoto, Y. (2012). Online failure prediction in cloud datacenters by real-time message pattern learning. In *4th IEEE International Conference on Cloud Computing Technology and Science Proceedings*, pages 504–511. IEEE.
- [295] Weber, P. and Suhner, M.-C. (2002). An application of Bayesian Networks to the Performance Analysis of a Process. In *In European Conference on System Dependability and Safety (ESRA 2002/lambda-Mu13). Lyon, France*, pages 266–273. ESRA-ISdF.
- [296] Wen, Z., Liang, Y., and Li, G. (2020). Design and implementation of high-availability PaaS platform based on virtualization platform. In *2020 IEEE 5th Information Technology and Mechatronics Engineering Conference (ITOEC)*, pages 1571–1575. IEEE.
- [297] Xie, W., Hong, Y., and Trivedi, K. (2005). Analysis of a two-level software rejuvenation policy. *Reliability Engineering & System Safety*, 87(1):13–22.
- [298] Xiong, H., Fowley, F., and Pahl, C. (2016). A database-specific pattern for multi-cloud high availability and disaster recovery. In *Communications in Computer and Information Science*, volume 567, pages 374–388.
- [299] Xu, W., Huang, L., Fox, A., Patterson, D. A., and Jordan, M. I. (2008). Mining console logs for large-scale system problem detection. *SysML*, 8:4–4.
- [300] Yang, H. and Kim, Y. (2020). Design and Implementation of Fast Fault Detection in Cloud Infrastructure for Containerized IoT Services. *Sensors*, 20(16):4592.
- [301] Zhang, C., Kumbhare, A. G., Manousakis, I., Zhang, D., Misra, P. A., Assis, R., Woolcock, K., Mahalingam, N., Warriar, B., Gauthier, D., et al. (2021). Flex: High-availability datacenters with zero reserved power. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pages 319–332. IEEE.

- [302] Zheng, Z., Lan, Z., Park, B. H., and Geist, A. (2009). System log pre-processing to improve failure prediction. In *2009 IEEE/IFIP International Conference on Dependable Systems & Networks*, pages 572–577. IEEE.
- [303] Zhu, L. and Lin, J. (2020). A SCSI3 Persistent Reservation Synchronization Solution for iSCSI Targets Cluster Hosting Ceph RBD with Active/Active Connections. In *2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, volume 1, pages 1785–1793. IEEE.
- [304] Zhu, L. G., Han, D. Z., Zhou, S. Z., and Xie, C. S. (2006). High availability cluster with combining nas and ISCSI. *Proceedings of the 2006 International Conference on Machine Learning and Cybernetics*, 2006(August):4455–4460.

Appendix A

High Availability Resource Characteristics

1. VM indicator

This property helps identify whether a resource is deployed in a VM host or VM guest instance. If it is a guest instance, the host's critical resources can be monitored to capture the state so that the failure probability can be assessed considering such failures [37].

Motivation for exclusion. It is impossible to obtain access at the host level in most operating models (e.g., a public cloud). Hence, it is not possible to deploy a monitoring tool. We excluded this property due to the challenges of gaining access at the host level to deploy an appropriate monitoring tool in the public cloud environment where the testbed was established.

2. Error severity

This characteristic can identify different severity levels of an error to provide detailed information, indicating that different mitigation actions can be devised based on the severity level [92].

Table A.1 Excluded resource properties of high-availability (HAC) clusters

HAC Resource Property	Description	Category	Values
VM indicator	An indicator for either the guest or host of a VM	EC	{0,1}
Error severity	Multiple severity levels of an error	NC	{0,...,n}
Error type	Multiple types of errors (e.g., intermittent, transient)	NC	{0,...,n}
Resource type	Type of failed resource	EC	{0,...,n}
Application state	Current state of an application or a business process	NC	{0,1}
Resource CPU utilisation	CPU utilisation by resource	NC	{0,...,n}
Resource memory utilisation	Memory utilisation by resource	NC	{0,...,n}

VM - Virtual machine

EC - Established characteristics

NC - New characteristics

Motivation for exclusion. The primary challenge is to obtain such values from the HAC logs. Second, it is difficult to interpret such information and map it into detection capabilities because all such severities must be translated into values that the model can understand. Moreover, no standards exist for providing severity because different HACs and protected applications may have specific severity levels and interpretations.

3. Error type

This property can capture different types of errors, such as transient errors [5].

Motivation for exclusion. The HAC logs typically do not provide this information. Moreover, application-specific error types may require such information provided by the application, but usually, HACs do not read or interpret application-provided logs. However, some limited scope is included in the selected characteristics by evaluating the *failure repetition* and *aggregated failure count*.

4. Resource type

A resource type can identify different types of resources and how different resolution actions can be performed upon failure [49].

Motivation for exclusion. The resource type is captured indirectly by associating the resource type with the characteristic reinitialisation factor. Therefore, this property was not included.

5. Application state

An application state indicates the state of an application resource, which includes a business process hosted on an application. Suppose an application is in the hang state [95]. In that case, it can still be considered running by the HAC [239], and only by specifically monitoring the application can one decide whether a failure has occurred. If such an application resource is not responding, understanding the correct state can identify a potential failure to initiate the correct mitigation activities.

Motivation for exclusion. This property requires monitoring an application resource to ensure that the application resource is operational. Therefore, an appropriate monitoring mechanism must be in place to check and understand the behaviour of the application resource, which requires considerable effort. Thus, this property was excluded.

6. Resource CPU utilisation

The CPU utilisation by a resource may indicate a potential failure shortly if the utilisation is high [239]. Moreover, such high utilisation can also affect other resources. Hence, this can lead to the failure of multiple interconnected resources or even high-level components.

Motivation for exclusion. The CPU utilisation for an individual resource is difficult to obtain for different resource types [258]. This property would require additional tools that support extracting such information. Therefore, we chose not to include this property.

7. Resource memory utilisation

A high value for this property can be used to identify or predict a potential resource failure or even the potential failure of other interconnected resources [49].

Motivation for exclusion. Memory utilisation for an individual resource is difficult to obtain for different resource types. Moreover, monitoring does not capture such details but only on memory utilisation at the server level. We chose not to include this characteristic because it requires additional development to capture memory utilisation for all HAC resources continuously.

Appendix B

Use of the Taxonomy

We applied the taxonomy (described in Section 3.1) to identify and organise the information required to set up the environment, implement the HAC, and incorporate the ERP solution into the HAC. Table B.1 lists a selected number of taxonomy classes and the corresponding values for the testbed environment.

Table B.1 Application of the taxonomy to the testbed application

Taxonomy Classes	Value
A: Deployment patterns	
A: Deployment patterns	Environment:public cloud Host:virtual
B: Application Areas	
B: Application areas	Enterprise system
C: Type of Cluster	
C: Type of cluster	C.1: Local
D.2: Asymmetric	D.2.1: Active-passive
E: Cluster Management	
E.1.1: Configuration	E.1.1.1: Disk
E.1.2: Runtime	E.1.2.3: Memory
E.2.1.1: Heartbeat	E.2.1.1.1: LAN-based
E.2.2.1.2: Virtual synchrony	TOTEM
E.3.1.2.1.1: Application	ERP
E.3.1.2.1.2: Database	MaxDB 7.9.10
F: Failure Detection and Recovery	
F.1.2: Type	F.1.2.1: State-based
F.1.3: Method	F.1.3.1: Poll
F.2: Failover	F.2.1: Reactive
G.4.1.1: Resource	SBD
H: Data Synchronisation	
H.1: Shared storage	iSCSI

Use of the Taxonomy

The deployment pattern indicates that the environment is established in a public cloud, and virtual servers were used. The application area is an enterprise system (ERP solution), and the cluster is a local cluster that is set up using the active-passive topology (asymmetric). Moreover, the pattern also provides details at the granular level. For example, the TOTEM protocol supports intercluster communication, and heartbeat communication is based on a local area network. The application class is used to identify the application components, such as specific enterprise application and database so that the appropriate resource agents (e.g., agent for the ERP and database) can be installed. The failure detection and recovery class indicates that the monitoring type is state-based, and the method for collecting monitoring data is through polling. Failover is based on policy, whereas fencing is performed at the node level. Data synchronisation is based on shared storage, and the technology is based on iSCSI.

Appendix C

Evaluation of Runtime Overhead of the Holistic Modelling Technique for High Availability

This section presents the evaluation of the HMTHA technique described in Chapter 5. We validated the technique by using it to create a HHAM model for the HAC from our testbed environment in Section 10.3.1. Subsequently, we used this HHAM model to successfully construct the BN-HAC model. In this section, we present the results from investigating the computational overhead associated with creating a HHAM model. Section C.1 presents the experimental setup, and Section C.2 presents and discusses the evaluation results. Section C.3 reviews the threats to the validity of the experiments and the results. Section C.4 summarises the section.

Table C.1 Evaluated HHAM models—vertex types

Vertex Type	Number of vertices		
	Model 1 (testbed)	Model 2	Model 3
Application	1	1	1
Resource group	5	4	5
Weak	2	2	2
Simple	13	16	28
Global	3	2	n/a
Shared	n/a	1	n/a
Total	24	26	36

n/a – no vertex of this type

Table C.2 Evaluated HHAM models—vertices in each layer

Layer	Number of vertices		
	Model 1 (testbed)	Model 2	Model 3
App(1)	1	1	1
a (2)	8	4	5
b (3)	6	6	6
c (4)	9	4	16
d (5)	0	11	5
e (6)	0	0	1
f (7)	0	0	2
Total	24	26	36

C.1 Experimental Setup

For the experimental evaluation, we ran the HMTHA tool on a computer with 3.4 GHz Intel Core i7 and 64 GB memory running Windows 10 64-bit. The database was MySQL 8.0.18 (64-bit). We created three HHAM models. The first model was for the HA solution implemented in the testbed. The other two models represented two realistic IT systems with a variety of components and depths. Table C.1 presents the three models according to the vertex types, whereas Table C.2 lists the three models grouped by layers.

We measured the computational overhead by measuring the CPU and memory when adding a vertex and assessing the type and layer of the vertex. The results are presented in the next section.

C.2 Results and Discussion

When the HMTHA tool was initiated, it used between 152 and 162 MB of memory, and more memory was used gradually as different types of vertices were added, as depicted in Figure C.1. *Memory per vertex* indicates how much memory was added per vertex, and the cumulative curve presents the *cumulative* values. However, the *absolute* value presents the absolute memory usage, which includes all memory used by the model.

We observed that the changes to memory utilisation when adding a vertex was constant, apart from some minor deviations. Figure C.2 (b) displays the deviations more clearly. The mean utilisation is between 2 and 4 MB, but it exceeds the mean value in some cases. For example, when adding nodes 18 or 20, all their models exhibit the same behaviour. Figure C.3 (b) presents the mean memory utilisation per vertex type, and dependency vertices appear to be using more memory. However, only a few vertices of each type are present in each model (e.g., only three global vertices in Model 1). Therefore, the result cannot be treated as statistically significant.

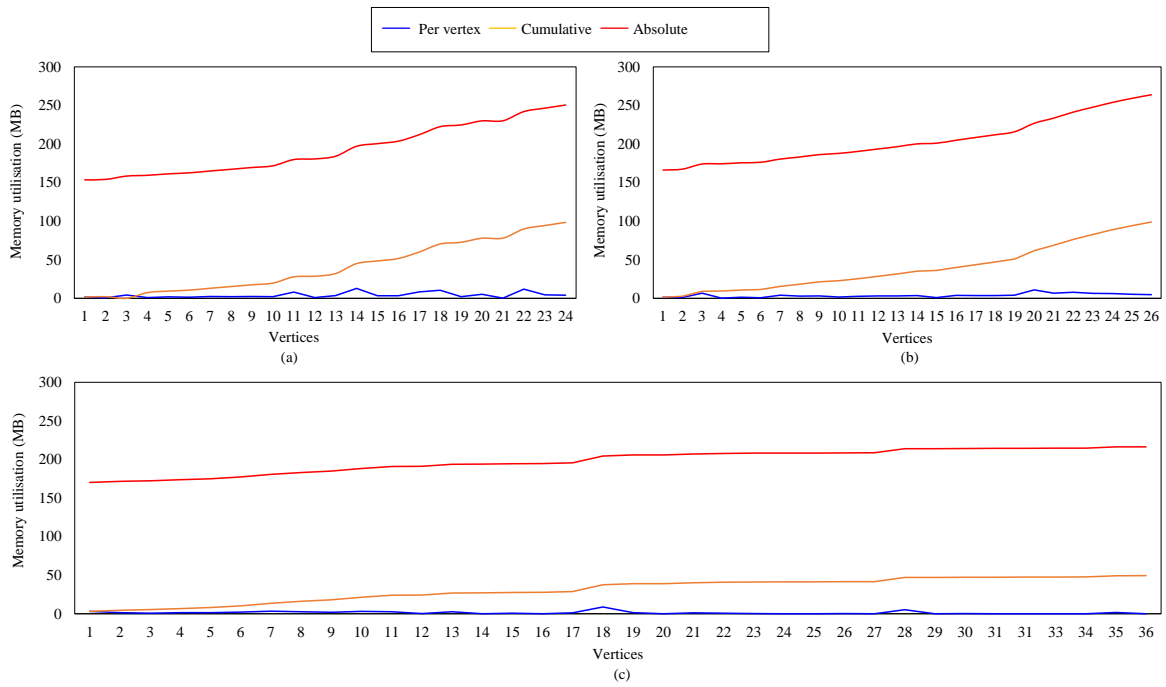


Figure C.1 Utilisation of memory per vertex, cumulative memory, and absolute memory for (a) model 1, (b) model 2, and (c) model 3.

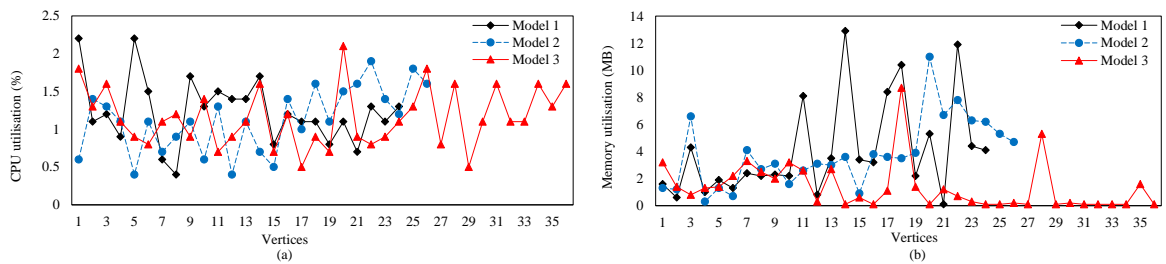


Figure C.2 CPU (a) and memory (b) utilisation when adding vertices.

Figure C.4 presents the CPU and memory utilisation per model layer. Figure C.4(b) shows that the fourth layer (which has more vertices than the other layers) requires the highest amount of memory for Models 1 and 2, while Layer 1 has the highest value for Model 3. Initially, the pattern that emerges from Figure C.4(b) is that when the number of vertices in the HHAM model layers increases, the memory requirements also increase. The only exception is Layer 1 in Model 3, which has a high value. However, the two models (1 and 2) do not have more than six layers; hence, it is difficult to reach a conclusion regarding the correlation between layers and memory.

In contrast to memory utilisation, which grows with each node, CPU utilisation occurs only when a new node is added, and the percentages shown for CPU utilisation were experienced for a period of 0.6 ± 0.4 s. Figure C.2 (a) presents CPU utilisation when adding a node, which is observed to be under

Evaluation of Runtime Overhead of the Holistic Modelling Technique for High Availability

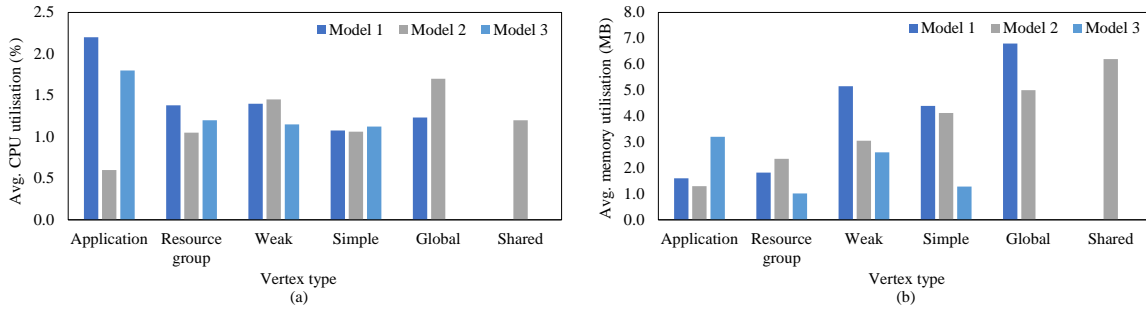


Figure C.3 Mean CPU (a) and memory (b) utilisation per vertex type.

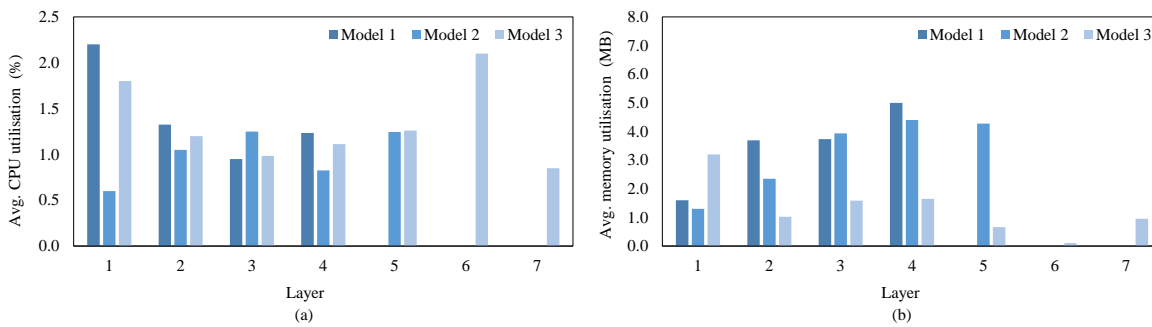


Figure C.4 Mean CPU (a) and memory (b) utilisation per layer.

2.5%. Similarly, Figure C.3 (a) displays the mean CPU utilisation per vertex type, and Figure C.4 (a) displays CPU utilisation per layer. No significant difference can be observed.

The results indicate that memory might be a concern for modelling very large and complex HACs because the technique is typically used on PCs. However, although HMTHA is part of the BP framework, it should be considered an independent module because it does not participate in the runtime operations of the framework. Therefore, resource utilisation does not add additional overhead to the runtime of the BP framework.

C.3 Threats to Validity

There are several threats to the validity of the HMTHA tool evaluation presented in this section:

- We used a high performing PC to evaluate; therefore, the results may be different when a less powerful computer is used. To mitigate this threat, we used a standard PC set up with commonly deployed processors. Furthermore, the amount of memory on the computer does not influence how much memory is needed to store the data associated with a node.
- The HMTHA can be implemented on different operating systems, but we only used one operating system, and the results could be different when other operating systems are used. To mitigate this, we developed the tool in Java to be as operating system independent as possible

using Java VM. Therefore, even if there are changes in the operating system, the software's functionality should not be affected. the software uses Java VM to run

C.4 Summary

In Chapter 10.3.1 we validated the HMTHA tool to create an HA model for the testbed application. We demonstrated that the model could be used to identify all key components while presenting dependency relationships. The outcomes of the HMTHA were used to construct the BN-HAC model for the HAC in the testbed. Moreover, the model aided in creating the configuration environment of the BPPF module.

In this section, we evaluated the runtime overhead of the HMTHA, which was implemented as a software tool. We created three HA models and evaluated them using the metrics of CPU and memory utilisation. We also investigated the correlation between resource utilisation and vertex type and the depth in the hierarchy and resource utilisation. The conclusion is that that the memory requirement is cumulative as more vertices are added. However, CPU utilisation is not persistent but only used when new nodes are added. We also evaluated the scalability of the technique and concluded that it could support numerous components. However, the visualisation and analysis of all such components can be a challenge. In summary, the modelling approach satisfies the primary objective, which is to capture the complex dependencies of an EA.

Appendix D

Implementation and Evaluation of Bayesian Prognostic Framework Preparation

The implementation of the BPPF module is presented in this section. Section [D.1.1](#) presents the steps to prepare the environment, and Section [D.1.2](#) details creating the required database objects. The configuration refinement step is described in Section [D.1.3](#), and the implementation of the log interface is detailed in Section [D.1.4](#). Section [D.1.5](#) describes the transformation and conversion process, and Section [D.1.6](#) presents the implementation of the database object to store the preprocessed data. Finally, Section [D.1.7](#) presents the application of the filter.

D.1 Implementation

D.1.1 Prepare the Environment

The BPPF is a platform- and database-independent solution; thus, only minor changes are required when implementing it on a different platform and database combination. However, the BPPF requires that a set of runtime tools are implemented to support the component execution (e.g., Python, R runtime libraries). Therefore, the first step was to install an appropriate development environment. Furthermore, a separate database (MySQL) was installed to store the configuration and runtime data of the BP framework.

D.1.2 Implement the Database for Storing Log and Configuration Data

Once the required database objects (described in Chapter 8) were created, we used the accompanying scripts to create the required stored procedures and triggers to facilitate transformation, conversion, and filtering.

Implementation and Evaluation of Bayesian Prognostic Framework Preparation

Resource_id:	C1G1A1B1
Resource_name:	message and lock service
HAC_resource_name:	rsc_DEV_ASCS00
Group_group_id:	C1G1A1
Cluster_cluster_id:	cluster1
Node_node_id:	1
Resource_type_resource_type:	service
Reinitialization_factor:	1
Redundancy_factor:	1
Dependency_type:	1
Critical_factor:	1
Dependency_level:	2
Dependency_depth:	3
Dependency_levels_up:	1
Dependency_levels_down:	1

Figure D.1 Configuration refinement for one record using the details from the holistic high-availability model (HHAM), mapping table (M-table), and high availability cluster (HAC) configuration details.

```

Jun 22 07:54:30 [1619] vmi243500 crmd: notice: process_irm_event: Result of probe operation for fs_3_DEV_ASCS on vmi243500: 7 (not running) | call=37 key=fs_3_DEV_ASCS_monitor_0 confirmed=true cib-update=49
Jun 22 07:54:30 [1614] vmi243500 cib: info: cib_perform_op: ++ /cib/status/node_state[@id="771304931"]/irm[@id="771304931"]/irm_resources: <irm_resource id="global_rsc_DEV_CPU" type="HealthCPU" class="ocf" provider="pacemaker"/>
Jun 22 07:54:30 [1614] vmi243500 cib: info: cib_perform_op: ++ <irm_rsc_op id="global_rsc_DEV_CPU_last_0" operation_key="global_rsc_DEV_CPU_monitor_0" operation="monitor" crm-debug-origin="do_update_resource" crm_feature_set="3.1.0" transition-key="16:0:7:b58f891c-6e09-4d3a-be21-ca68613900d7" transition-magic="-1:193;16:0:7:b58f891c-6e09-4d3a-be21-ca68613900d7" exit-reason="" on_node="vmi243500" call-id="-1" rc-code="" </irm_resource>
Jun 22 07:54:30 [1614] vmi243500 cib: info: cib_perform_op: ++ </irm_resource>
Jun 22 07:54:30 [1619] vmi243500 crmd: notice: process_irm_event: Result of probe operation for fs_DEV_Cl on vmi243500: 7 (not running) | call=41 key=fs_DEV_Cl_monitor_0 confirmed=true cib-update=50
Jun 22 07:54:30 [1614] vmi243500 cib: info: cib_process_request: Completed cib_modify operation for section status: OK (rc=0, origin=vmi243500/crmd/47, version=1.589.25)
Jun 22 07:54:30 [1614] vmi243500 cib: info: cib_process_request: Forwarding cib_modify operation for section status to all (origin=local/crmd/48)
Jun 22 07:54:30 [1614] vmi243500 cib: info: cib_process_request: Forwarding cib_modify operation for section status to all (origin=local/crmd/49)
Jun 22 07:54:30 [1614] vmi243500 cib: info: cib_process_request: Forwarding cib_modify operation for section status to all (origin=local/crmd/50)
Jun 22 07:54:30 [1619] vmi243500 crmd: notice: process_irm_event: Result of probe operation for vip_DEV_Cl on vmi243500: 7 (not running) | call=45 key=vip_DEV_Cl_monitor_0 confirmed=true cib-update=51
Jun 22 07:54:30 [1614] vmi243500 cib: info: cib_process_request: Forwarding cib_modify operation for section status to all (origin=local/crmd/51)
Jun 22 07:54:30 [1615] vmi243500 stonith-ng: info: update_cib_stonith_devices_v2: Updating device list from the cib: create irm_resources
Jun 22 07:54:30 [1615] vmi243500 stonith-ng: info: cib_devices_update: Updating devices to version 1.589.25
Jun 22 07:54:30 [1615] vmi243500 stonith-ng: notice: unpack_config: Watchdog will be used via SBD if fencing is required

```

Figure D.2 Extract from the log file from the testbed high availability cluster (HAC).

D.1.3 Apply Configuration Refinement

One critical activity is to insert the information regarding the HAC environment into a set of tables so that the runtime environment can use the information. The information was obtained from the HHAM model, M-table, and HAC configuration and was entered into the appropriate tables using the ‘configuration refinement’ activity. An example of entering a record is displayed in Figure D.1. This step was repeated until entries related to all resources were inserted. Similarly, the configuration table nodes, clusters, resource types, and groups were updated as described in Section 8.3.

D.1.4 Implement the Log Interface

We developed the log interface in Python, and it was provided as an executable Python script. Three changes are required in the script to function correctly in a new environment. The first change is to identify the pattern of the error message to be captured. The second change is to identify the relative position of the error messages. In the third change, the database connection is identified. All changes

Entry_id:	5
Resource_id:	C1G2A1B1
Resource_name:	database
HAC_resource_name:	rsc_DEV_database
Group_id:	C1G2A1
Cluster_id:	cluster1
Node_id:	1
Event_date:	2020-06-22 07:54:30
Current_state:	offline
Critical_factor:	true
Failure_repetition:	low
Redundancy_factor:	false
Aggregated_failure_count:	low
Reinitialization_factor:	true
Dependency_type:	local
Dependency_levels_up:	low
Dependency_levels_down:	low

Figure D.3 Extract from table *model_data* showing the result of the transformation, conversion, and filter application.

can be updated directly in the script. An extract from the log file from the testbed HAC is presented in Figure D.2, where the error messages are depicted in red. The script can be executed either directly or in the background. We scheduled the script to run every 10 s, which also becomes the polling time.

D.1.5 Enable Transformation and Conversion

The three steps, transformation, conversion, and filtering, were combined and implemented as a database trigger (created in Section D.1.2). The trigger was initiated as soon as the log interface enters new data in the *Hac_main* table. The processed data were inserted into the table *model_data*. An example is provided in Figure D.3.

D.1.6 Implement the Database Table for Storing Model Data

In this step, we created the table to store the prepared and preprocessed information after applying the filter step so that the BDN-HAC model can process such information. An SQL script was used to implement the table for storing the model data, as described in Section 8.6.

D.1.7 Apply Filter

As described in Section D.1.5, the filter was implemented as a database trigger. Figure D.3 presents an entry from the table *model_data*.

Table D.1 Mean execution time in seconds for different components of the BPF

Data Sets	Poll	Query	Extract	Parse	Enrich	Transform	Convert	Filter
1	4.54	0.05	0.05	0.07	0.04	0.06	0.03	0.05
2	3.32	0.05	0.05	0.05	0.05	0.06	0.03	0.04
5	3.70	0.06	0.05	0.05	0.05	0.06	0.04	0.04
7	5.27	0.06	0.05	0.05	0.05	0.05	0.04	0.04

D.2 Evaluation of Bayesian Prognostic Framework Preparation

In this section, the evaluation of the BPF module is presented. The model consists of numerous components. Therefore, to evaluate the model, we first evaluated each component individually, and then combined the results from these evaluations to obtain the results for the end-to-end BPF module. Section D.2.1 describes the experimental setup, and Section D.2.2 presents the results from the evaluation. Section D.3 reviews the threats to the validity of the experiments. Finally, Section D.4 summarises the section.

D.2.1 Experimental Setup

The individual components of the BPF module operate differently; therefore, many approaches are required to monitor and capture the outcomes. For instance, polling time was obtained by subtracting the scheduled polling time from the failure time recorded by the HAC. In contrast, a tool in the operating system is required to monitor the CPU and memory utilisation for polling. Therefore, we used several tools, such as MySQL workbench, Windows performance analyzer (WPA), and Linux performance monitor utilities (e.g., top and sar). The polling interval plays a crucial role in changing the execution time and overhead for the entire BPF. However, the polling frequency is adjustable, and we set the polling interval to 10s for the experiments.

D.2.2 Results and Discussion

Execution Time Table D.1 presents the mean execution time for different BPF components. The execution time for polling stands out and consumes a sizeable portion of the total runtime of the BPF module with an interval between 2s and 10s. In contrast, the maximum execution time for each of the other components is less than 0.07s. Although the polling interval is set to 10s, extracting the failure information can occur earlier because a failure can occur close to the polling time. Taken together, the mean execution time of BPF when considering all data set is 4.5s. Moreover, considering all the execution steps in all the related data sets, we obtain 27.3% for the BN-HAC model, 1.3% for the BDN-HAC model and 71.4% for the BPF. The execution time of the BPF contributes to a significant portion of the overall execution time for the BP framework. The total time required by the BPF module is a concern, particularly the polling time. However, the polling time can be increased,

D.2 Evaluation of Bayesian Prognostic Framework Preparation

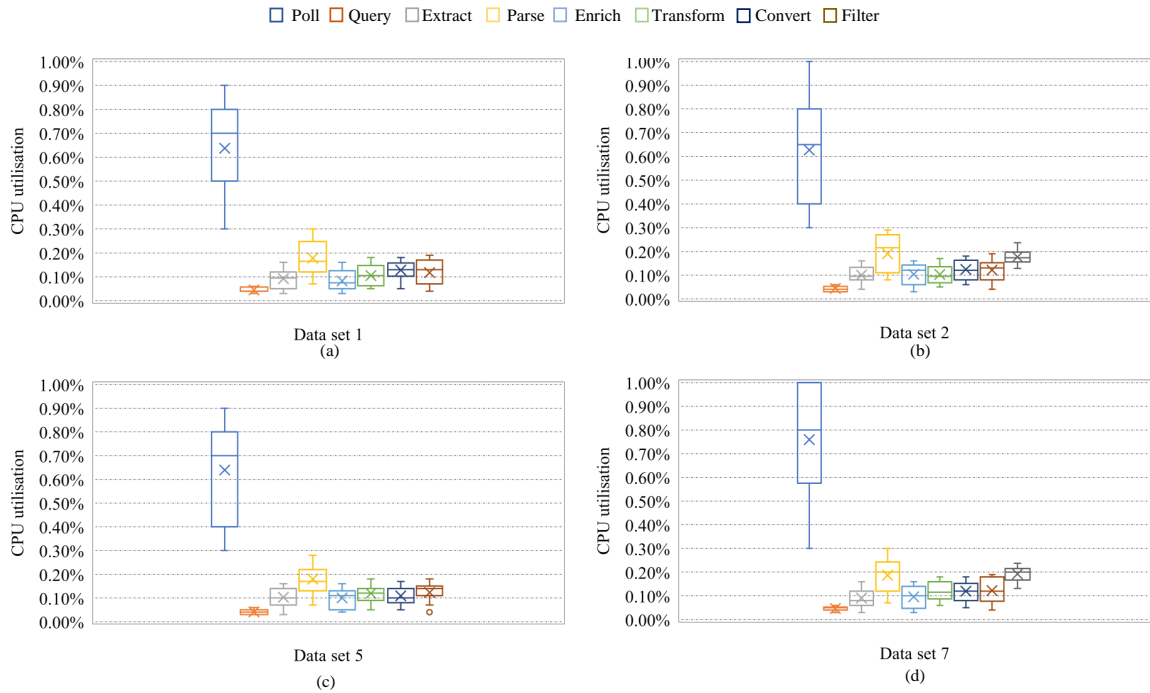


Figure D.4 Box plot of the CPU utilisation of the BPF components presented for (a) Data Set 1, (b) Data Set 2, (c) Data Set 5 and (d) Data Set 7.

which should reduce the overall time required by the module. Moreover, if the BP framework is integrated with HACs in the future, it will eliminate the polling time because the HAC will pass the failure information directly to the BPF in that case.

Runtime Overhead We evaluated the runtime overhead of the BPF module by measuring the individual model components. Figure D.4 presents the CPU utilisation by different modules using Data Sets 1, 2, 5 and 7. The CPU utilisation was experienced for a period of $< 5.27s$ for the duration of the BPF execution times from Table D.1. The polling component stands out from other components because it uses consistently more CPU resources. The maximum utilisation is 0.9% across all data sets, and the median is 0.7%. However, a significant part is in the lower quartile, indicating that the CPU utilisation is below 0.7% in most cases. The component query has little utilisation and uses less than 0.05% in all data sets. The component extract has a slightly higher utilisation between 0.03% and 0.16%. The maximum utilisation for the parse component is 0.3%, and the median is 0.07%. The enrich component also has a similar utilisation pattern, and the maximum value is 0.16%, whereas the median is 0.10% on average, putting most instances in the lower quartile. Transform and convert have a maximum utilisation of 0.18% and 0.19%, respectively. The component filter also has low utilisation, with a maximum of 0.24% and a median between 0.12% and 0.19%. However, the only deviation is that an outlier is identified in Data Set 5 with a value of 0.04% (Figure D.4 (c)). Overall, the results indicate negligible CPU utilisation for all components except for polling. However, because

Implementation and Evaluation of Bayesian Prognostic Framework Preparation

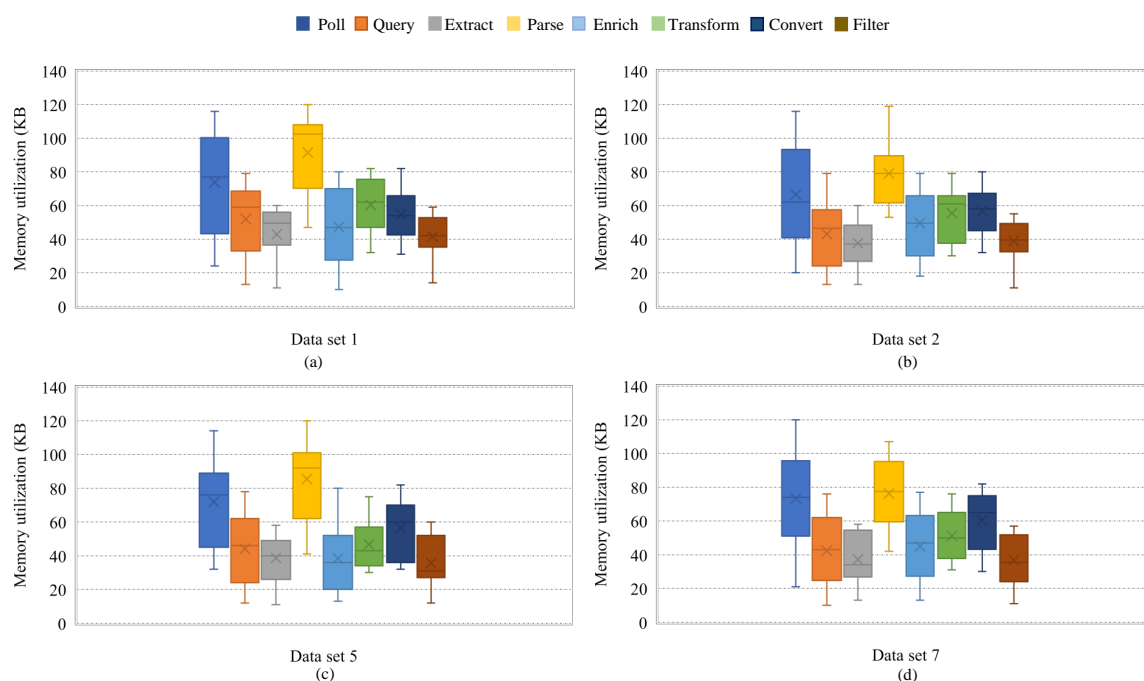


Figure D.5 Box plot of the memory utilisation of the BPF components presented for (a) Data Set 2, (b) Data Set 3, (c) Data Set 5 and (d) Data Set 7.

polling was scheduled to run every 10 s, sometimes a polling process may create a new process while the previous one is still running, explaining the standout behaviour.

Memory utilisation is depicted in Figure D.5 for Data Sets 1, 2, 5 and 7, and the maximum utilisation is shared by both poll and parse with a value of 120 KB. The median for poll is between 66 and 77 KB, whereas the median for parse is between 76 and 102 KB, and both have more than 50% of instances in the lower quartile. The component query has a median value between 43 and 59 KB. The component enrich has a pattern similar to that of the query. Enrich has a median value between 43 and 61 KB, whereas transform has a median value between 46 and 61 KB. Convert has a maximum value of 82 KB (in Data Sets 1, 5 and 7), and filter has a maximum value of between 55 and 60 KB. The results reveal that the memory utilisation by the BPF is very low.

The combined view of resource utilisation demonstrates that the BPF adds little overhead. The components of the BPF are executed sequentially; therefore, CPU utilisation is not cumulative. The memory utilisation tends to be sequential, as the next component is initiated only when the previous component is finished. Therefore, even the combined memory utilisation is considered low.

D.3 Threats to Validity

The identified threats to validity are the following:

- In the empirical evaluation, we used data from one HAC solution. Because different HAC solutions produce and manage log data differently, the log interface could be considered specific to the HAC from our testbed. To mitigate this threat, we employ a modular architecture in which only the log interface needs to be adjusted for a different HAC solution.
- The measurements were performed only in one environment, and the results may likely differ if the experiments are executed in a different environment, for example, if the experiments were executed in an environment with lower-spec hardware. To mitigate this, the hardware used in the testbed is not high spec compared to what an organisation may use to run their EAs.

D.4 Summary

We introduced the BP framework preparation module in Chapter 8 and validated it by implementing it in the testbed environment described in Section 10.1). In this section, we evaluated the execution time and overhead of the individual components of the BFPF. The results demonstrate that polling is the only component with significant execution time of up to 10s, whereas all the other components require much less time (typically tens of milliseconds). However, the polling time depends on the scheduling frequency and therefore increasing the frequency will reduce the polling time. Moreover, only the maximum time is 10s because if the polling is 1s after the failure, the polling time becomes 1s. Additionally, future integration of the BP framework into HACs can also eliminate the polling time because failure events will be captured and passed in realtime to the BP framework in such cases, We evaluated the overhead by measuring the CPU and memory utilisation, and the results indicate that the overhead is negligible, and CPU utilisation was experienced for a period of $5 \pm 0.8s$.

Appendix E

Implementation of the Bayesian Decision Network for Predicting Locally Manageable Resource Failures

This section describes how the BDN-HAC model was implemented. The BDN-HAC is a HAC solution-independent model that can be implemented without changes to the model. We implemented the two BDN models as described in Chapter 6 to evaluate the detection quality and select the best performing model. Hence, the evaluation is described in Section 10.3.2. However, we only describe the implementation of one model in this section, considering that only one model must be set up during implementation.

E.1 Implement the Model

The model can be implemented as a script, and a prerequisite is used to prepare the environment. For example, if the script is based on R, an R environment must be prepared using R software and the required libraries specifically for the operating environment in scope. However, we used a BN modelling tool to create and evaluate the model.

E.2 Change the Target Environment Details

The R script can be triggered upon completion of preparing and preprocessing the failure information. Similarly, upon completing the execution of the BDN-HAC model, the script must call the next model, BN-HAC. Thus, the location of the next model must be specified in the script.

E.3 Test the Model

We used the test cases T1-T9, described in Section [10.2.1.2](#), to create data sets for training, and the model uses the data sets to apply the improved detection capabilities to predict whether a resource failure can be managed locally. Only those failures considered unmanaged are passed to the BDN-HAC model.

E.4 Inference Using Production data

The model infers using production data in the runtime phase, and we used test cases T10-T19, described in Section [10.2.1.2](#), to create the data sets and evaluate the model. The unmanaged failure information was passed to the BN-HAC model. All such steps can be fully automated in the runtime phase.