A Management System for Service Level Agreements in Grid based Systems

James Joseph Padgett

Submitted in accordance with the requirements for the degree of Doctor of Philosophy

University of Leeds
School of Computing

September 2006

Abstract

Grid based systems have increased the opportunity for users to deploy and execute their applications using Grid resources. These resources have varying reliability and performability, particularly when demand is high. If a Grid application is executed at such times, performance may suffer and results may be delayed. In order to overcome this problem, application management is needed to support Quality of Service (QoS) requirements. The Distributed Aircraft Maintenance Environment (DAME) is an example of a Grid based system in which users wish to attach application QoS requirements.

In light of this, an adaptive SLA (Service Level Agreement) management system is presented which has the ability to interpret application requirements and deliver management using application adaptation. An SLA specification is presented which improves contract non-repudiation by way of elements which allow requirements, guarantees to be specified and provenance to be recorded. To predict the execution time of an application, a technique using historical observations is proposed. An approach which is highly appropriate for Grid based systems which perform countless runs of the same application. This prediction is used in combination with application monitoring to determine the progress made by the application during run-time. Progress is determined by comparing an estimate of the applications remaining execution time and an execution schedule. If application progress is insufficient, a rule-based control algorithm monitors progress and infers control actions which adapt the behaviour of the application.

Experimental analysis is conducted on a local Grid test-bed and a large scale Grid infrastructure, the White Rose Grid. This shows the solution supports application executions with attached time or performance constraints; where use of the system prevents application failure or delay. Migration is useful in reducing the execution time of applications when performance degradation occurs. Mechanisms for automated monitoring and provenance capture are presented, both of which support the operation of the SLA management system.

Adaptive SLA management benefits the users of Grid based systems such as DAME, by providing Grid application management. This is in contrast to current best-effort provision which offers no such guarantee. The ability to provide these guarantees and an SLA specification makes commercial exploitation of these Grid based systems more realistic.

Acknowledgements

I would like to acknowledge the support and advice given to me by my work colleagues at the University of Leeds, in particular my supervisors Karim Djemame and Peter Dew. Their wealth of experience has provided me with encouragement and critical feedback in equal measures and for that I thank you.

The work in this thesis was partly supported by the DAME project under UK Engineering and Physical Sciences Research Council Grant GR/R67668/01. I am also grateful for the support of the DAME partners, including the help of staff at Rolls-Royce, Data Systems & Solutions, Cybula, and the Universities of York, Sheffield and Oxford.

Finally, I would like to acknowledge the support of my family and friends, particularly my parents, who have always reminded me to give my best rather than to change the world.

Declarations

Some parts of the work presented in this thesis have been published in the following articles:

Towards automated service level agreement management for Grid Services. J. Padgett. in Proceedings of the Postgraduate Research Conference in Electronics, Photonics, Communications, Networks and Computing Science (PREP2004), Hatfield, UK, April 2004.

Grid-based SLA Management. J. Padgett, K. Djemame and P. Dew. in Proceedings of the European Grid Conference (EGC'2005), Amsterdam, The Netherlands, February 2005, Lecture Notes in Computer Science 3470, pp.1076-1085.

SLA Management for Dynamic Virtual Organisations. J. Padgett, I. Gourlay and K. Djemame. in Proceedings of the Postgraduate Research Conference in Electronics, Photonics, Communications, Networks and Computing Science (PREP2005), Lancaster, UK, April 2005.

SLA Management in a Service Oriented Architecture. J. Padgett, M. Haji and K. Djemame. in Proceedings of the International Conference of Computer Science and its Applications (ICCSA2005), Singapore, May 2005, Lecture Notes in Computer Science 3483, pp.1282-1291.

Predictive Run-time Adaptation for Service Level Agreements on the Grid. J. Padgett, K. Djemame, and P. Dew. in Proceedings of the 21st UK Performance Engineering Workshop (UKPEW2005), Newcastle, UK, July 2005.

Grid Service Level Agreements Combining Resource Reservation and Predictive Run-time Adaptation. J. Padgett, K. Djemame and P. Dew. In Proceedings of the UK e-Science All Hands Meeting (AHM2005), Nottingham, UK, September 2005.

Predictive Adaptation for Service Level Agreements on the Grid. J. Padgett, K. Djemame and P. Dew. International Journal of Simulation. 7(2), pp29-42.

Contents

Abstract ..... **ii**

Acknowledgements ..... **iv**

Declarations ..... **v**

Contents ..... **vi**

List of Figures ..... **xi**

List of Tables ..... **xiv**

Abbreviations ..... **xv**

Chapter 1 Introduction **1**

1.1 Research Motivation 1

1.2 Thesis Objectives 3

1.3 Scope of Thesis 3

1.4 Methodologies 4

1.5 Major Contributions 5

1.6 Thesis Overview 6

1.7 Summary 8

Chapter 2 Background and Related Work **9**

2.1 Grid Computing 9

2.1.1 Architecture 10

2.1.1.1 Open Grid Services Architecture (OGSA) 11

2.1.1.2 Web Services Resource Framework (WSRF) 12

2.1.2 Middleware 13

2.1.2.1 Globus Toolkit 14

2.1.2.2 Unicore 15

2.1.2.3 Crown 15

2.1.2.4 OMII 15

2.1.3 Grid Infrastructures 16

2.1.4 Projects and Applications 16

2.2 Resource Management and Scheduling 19

2.2.1 Resource Scheduling 21

2.2.2 Resource Brokering 22
2.2.3 Grid Monitoring 23
2.3 SLA Specifications 25
2.3.1 Job Submission Description Language (JSDL) 25
2.3.2 Web Service Level Agreement (WSLA) Specification 26
2.3.3 WS-Agreement 27
2.3.4 Usage Record (UR) 27
2.3.5 Grid Economic Services Architecture (GESA) 28
2.3.6 Service Negotiation and Acquisition Protocol (SNAP) 28
2.3.7 e-Contracts in e-Commerce 29
2.4 SLA Management Systems 29
2.4.1 Utility Grid Systems 30
2.4.2 Commercial Grid Systems 32
2.5 SLAs Within UK e-Science Projects 33
2.6 Predicting application resource usage 35
2.6.1 Learning based approach 36
2.6.2 Code based approach 37
2.7 Checkpointing 37
2.8 Application Adaptation in Grid systems 38
2.9 Summary 42
Chapter 3 SLA Management Architecture and Specification 43
3.1 SLA Management Usage Scenario 43
3.2 Application Model 46
3.3 SLA Management Architecture 47
3.3.1 SLA Instantiation and Modification 49
3.3.2 Initial prediction of Application execution time 49
3.3.3 Resource Selection and Reservation 50
3.3.4 Resource and Application Monitoring 51
3.3.5 Estimating the Applications Remaining Execution Time 51
3.3.6 Application Adaptation 52
3.4 System Use Cases 54
3.4.1 End-User Use Cases 55
3.4.2 SLA Manager Use Cases 56

3.4.3 Broker Use Cases 58
3.5 System Components 58
3.6 Intended Usage Scenarios 60
3.6.1 Request Initial Prediction 61
3.6.2 Request SLA for task 61
3.6.3 Accept SLA 62
3.6.4 Monitor application progress 63
3.7 Implementation 65
3.7.1 Updating the Globus Toolkit 65
$GLOBUS_LOCATION/share/globus_gram_job_manager/sge.rvf 65
$GLOBUS_LOCATION/lib/perl/Globus/GRAM/JobManager/sge.pm 66
3.7.2 Extending support in SGE. 66
3.7.3 Application Migration 67
3.8 SLA Specification 67
3.8.1 Overview 69
3.8.1.1 slaType 70
3.8.1.2 partiesType 70
3.8.1.3 consumerType 71
3.8.1.4 providerType 71
3.8.1.5 cpuType 72
3.8.1.6 ramType 72
3.8.1.7 hddType 73
3.8.1.8 osType 73
3.8.1.9 completionType 74
3.8.1.10 violationType 74
3.8.1.11 warningType 75
3.8.1.12 migrationType 75
3.9 Summary 75
Chapter 4 Prototype testing on a local Grid Test-bed 77
4.1 Overview 77
4.2 Experimental Design 78
4.2.1 Grid Application 80
4.2.2 Resource Monitoring 81

4.2.2.1 Load Calculations 81
4.2.2.2 Motivation for using MDS 81
4.2.3 Specifying the remaining execution time control variable 82
4.3 Scenario 1 – Performance Guarantees 85
4.4 Scenario 2 – Best Effort 88
4.5 Scenario 3 – SLA Management 93
4.6 Summary 97
Chapter 5 Testing adaptive SLA management on the White Rose Grid 98
5.1 Overview 99
5.2 Experimental Design 100
5.2.1 Grid Application 103
5.2.2 Resource Monitoring 104
5.2.3 Initial Prediction 105
5.2.4 Specifying the remaining execution time control variable 106
5.2.5 Rule Base and SLA 108
5.3 Scenario 1 – Single Provider 110
5.3.1 Run 1 111
5.3.2 Run 2 114
5.4 Scenario 2 – Multiple Providers 118
5.5 Scenario 3 – DAME XTO Application 120
5.6 Summary 124
Chapter 6 Evaluation 125
6.1 Overview 125
6.2 Methodology 126
6.3 Experiments 126
6.4 Related Work 131
6.5 Non Functional Requirements 134
6.6 Summary 137
Chapter 7 Conclusion 138
7.1 Summary of Work 138
7.2 Results 140
7.2.1 Methodology 141
7.3 System Drawbacks 142

7.4 Future Work 143
7.5 Overall Conclusions 146
Appendix A 148
Appendix B 152
References 153

List of Figures

Figure 1 Layered Grid Protocol Architecture 10

Figure 2 GRAM and the sequence in job submission from client to server [48] 21

Figure 3 JSDL Pseudo Schema 26

Figure 4 WSLA Pseudo Schema 26

Figure 5 WS-Agreement Pseudo Schema 27

Figure 6 UR Pseudo Schema 28

Figure 7 Overall System Usage Scenario 44

Figure 8 SLA Management Architecture application model 46

Figure 9 SLA Management Architecture 49

Figure 10 Resource Reservation Process 51

Figure 11 Adaptive Decisions: An Example 54

Figure 12 System use case diagram 55

Figure 13 System Implementation Diagram 59

Figure 14 Request Initial Prediction Intended Usage Scenario 61

Figure 15 Request SLA for Task 62

Figure 16 Accept SLA Intended Usage Scenario 63

Figure 17 Monitor Application Progress – Single Provider 64

Figure 18 Changes to Globus RSL Schema 66

Figure 19 Changes to Globus SGE script builder 66

Figure 20 SGE Checkpointing Interface Construct 67

Figure 21 SLA Element 70

Figure 22 SLA Type 70

Figure 23 Parties Type 71

Figure 24 Consumer Type 71

Figure 25 Provider Type 71

Figure 26 CPU Type 72

Figure 27 RAM Type 73

Figure 28 HDD Type 73

Figure 29 OS Type 73

Figure 30 Completion Type 74

Figure 31 Violation Type 74

Figure 32 Warning Type 75

Figure 33 Migration Type 75

Figure 34 Snippet of application code responsible for the approximation of π 80

Figure 35 RSL code for submission of the SLA-bound application 80

Figure 36 SLA used in scenario 1 86

Figure 37 Normalised measure of system load vs. Elapsed Time: Performance Guarantee 87

Figure 38 Changes to the SLA for experiment in Figure 37 87

Figure 39 Rule Base: Scenario 2 88

Figure 40 SLA used in scenario 2 and 3 89

Figure 41 Normalised measure of system load vs. Elapsed Time: Scenario 2 - Disturbance added early 90

Figure 42 Normalised measure of system load vs. Elapsed Time: Scenario 2 - Disturbance added late 90

Figure 43 Time Remaining vs. Time elapsed: Scenario 2 - Disturbance added early 91

Figure 44 Time Remaining vs. Elapsed Time: Scenario 2 - Disturbance added late 92

Figure 45 Changes to the SLA for experiment in Figure 43 92

Figure 46 Changes to the SLA for experiment in Figure 44 92

Figure 47 Rule Base: Scenario 3 93

Figure 48 Normalised measure of system load vs. Elapsed Time: Scenario 3 - Disturbance added early 94

Figure 49 CPU Load vs. Elapsed Time: Scenario 3 - Disturbance added late .. 94

Figure 50 Time remaining vs. Elapsed Time: Scenario 3 - Disturbance added early 95

Figure 51 Time remaining vs. Elapsed Time: Scenario 3 - Disturbance added late 96

Figure 52 Changes to the SLA for experiment in Figure 50 96

Figure 53 Changes to the SLA for experiment in Figure 51 97

Figure 54 The White Rose Grid (WRG) Architecture 99

Figure 55 RSL code for submission of the SLA-bound application 103

Figure 56 Grid application submission script 104

Figure 57 Information maintained by the OS for each process in the /proc file system..... 105

Figure 58 Rule Base experiments in Chapter 5 109

Figure 59 SLA for scenarios 1 - 3 110

Figure 60 CPU Time (jiffies/s) vs. Elapsed Time (minutes) Single Provider with adaptation 111

Figure 61 CPU Time (jiffies/s) vs. Elapsed Time (minutes) Single Provider without adaptation 112

Figure 62 Time Remaining (minutes) vs. Elapsed Time (minutes) Single Provider with adaptation 113

Figure 63 Time Remaining (minutes) vs. Elapsed Time (minutes) Single Provider without adaptation 113

Figure 64 Changes to the SLA for experiment in Figure 62 114

Figure 65 Changes to the SLA for experiment in Figure 63 114

Figure 66 CPU Time (jiffies/s) vs. Elapsed Time (minutes) Single Provider with adaptation 115

Figure 67 CPU Time (jiffies/s) vs. Elapsed Time (minutes) Single Provider without adaptation 115

Figure 68 Time Remaining (minutes) vs. Elapsed Time (minutes) Single Provider with adaptation 116

Figure 69 Time Remaining (minutes) vs. Elapsed Time (minutes) Single Provider with no adaptation..... 117

Figure 70 Changes to the SLA for experiment in Figure 68 117

Figure 71 Changes to the SLA for experiment in Figure 69 118

Figure 72 CPU Time (jiffies/s) vs. Elapsed Time (minutes) Multiple Providers119

Figure 73 Time Remaining (minutes) vs. Elapsed Time (minutes) Multiple Providers..... 120

Figure 74 Changes to the SLA for experiment in Figure 73 120

Figure 75 CPU Time (jiffies/s) vs. Elapsed Time (minutes) XTO..... 122

Figure 76 Time Remaining (minutes) vs. Elapsed Time (minutes) XTO..... 123

Figure 77 Changes to the SLA for scenario 3 123

Figure 78 Inferential Controller 127

Figure 79 Inferential Observer 128

Figure 80 SLA Manager GUI..... 135

Figure 81 Grid application source code 151

Figure 82 Stat field key..... 152

List of Tables

Table 1 SLA Specification for a compute service..... 69

Abbreviations

AppLeS	Application Level Scheduler
CPU	Central Processing Unit
B2B	Business to business
BSLA	Binding Service Level Agreement
DAME	Distributed Aircraft Maintenance Environment
GESA	Grid Economic Services Architecture
GIIS	Grid Index Information Service
GrADS	Grid Analysis and Display System
GRAM	Grid Resource Allocation Manager
GridFTP	Grid File Transfer Protocol
GRIS	Grid Resource Information Service
HPC	High Performance Computing
JSDL	Job Submission Description Language
JAXB	JAVA Architecture for XML Binding
LAN	Local Area Network
LDAP	Lightweight Directory Access Protocol
LSF	Load Sharing Facility
MDS	Monitoring and Discovery System
MPI	Message Passing Interface
NBQS	Network Batch Queuing System
NWS	Network Weather Service
OGSA	Open Grid Services Architecture
OS	Operating System
PBS	Portable Batch System
POSIX	Portable Operating System Interface for UNIX

PUNCH	Purdue University Network-Computing Hubs
QoS	Quality of Service
RAM	Random Access Memory
RSL	Resource Specification Language
RSLA	Resource Service Level Agreement
SGE	Sun Grid Engine
SLA	Service Level Agreement
SLI	Service Level Indicator
SLO	Service Level Objective
SNAP	Service Negotiation and Acquisition Protocol
TSLA	Task Service Level Agreement
TTL	Time to Live
UML	Unified Modelling Language
UR	Usage Record
VO	Virtual Organisation
WASS	Wide Area Scheduling System
WAN	Wide Area Network
WRG	White Rose Grid
WSLA	Web Service Level Agreement
WSRF	Web Service Resource Framework
XML	Extensible Mark-up Language
XSD	XML Schema Definition
XTO	Extract Tracked Orders
YHMAN	Yorkshire and Humberside Metropolitan Area Network

Chapter 1

Introduction

The purpose of the work presented in this thesis is to develop and evaluate an adaptive Service Level Agreement (SLA) Management system to enhance the Quality of Service (QoS) capability of applications executing within Grid based systems. This chapter discusses the motivation, scope and objectives behind the research as well as the methodologies and major contributions.

1.1 Research Motivation

Since its conception, Grid computing [1] has increased the opportunity for research collaboration between educational and research institutions. These so-called virtual organisations (VO) broaden access to expertise and services by sharing resources. Commercial uptake of Grid computing has been less prolific, yet there are a number of potential advantages for users looking to outsource part or entire processes. Opportunities exist to deploy or share resources and services, avoid large investment overheads associated with in-house computing infrastructures to name a few. A bar to commercial take-up is the prospect that outsourced processes may fail because current Grid middleware solutions do not offer to match process requirements with any form of management. In addition, a requirement of many commercial organisations is that usage is based on non repudiation; a concept that ensures an agreement cannot later be denied by one of the parties involved.

However users intend to take advantage of Grid computing, assurances that their processes will complete according to their requirements is a priority. Current Grid middleware solutions leave users with nothing if these processes fail. Adaptive SLA Management is a solution which can meet the requirements of process management and

solve the problem of reliable non repudiation. The work in this thesis showcases a scenario in which a commercial organisation lacks the computing infrastructure to process a large number of computationally intensive batch Grid application executions. An application requirement is for the timely completion and return of the results. Adaptive SLA Management can provide the monitoring and control to manage this process and prevent missed deadlines.

The type of applications which are executed on Grid systems can vary from long running computationally intensive simulations to high demand and high priority time critical transaction based executions, to real-time interactive visualisations. Of these examples, each has a different QoS requirement, yet under current Grid middleware deployments each is expected to execute and complete on resources configured identically, with no consideration given to their differing requirements. This is in addition to an expectation that neither one will effect the performance of the other. Grid resources are owned and administered by different organisations, making cross resource management difficult. Challenges include availability, reliability and performability, qualities which cannot be relied upon, particularly when demand is high. In this situation competition for resources can easily swamp a resource, adversely affecting the performance of any application which is running. If this coincides with the execution of a time critical application, results can be delayed or lost entirely. This has consequences in the real world and can lead to broken commitments and penalty fees. In situations where the results influence the actions of others, there can be knock on effects in other walks of life.

If Grid applications could be executed with Quality of Service (QoS) commitments guaranteeing requirements such as timely execution, without the need to re-engineer the software, not only would it help mitigate the issue of reliability and performability, it would increase the chances of commercial take-up. An example of a commercial process where this is a requirement is the Distributed Aircraft Maintenance Environment (DAME) [2]. This Grid based decision support / expert system is used by members of the commercial aviation industry who require the results of aircraft engine health monitoring applications to be presented to them within strict time deadlines. If this requirement is not met, commercial aircraft departure delays are highly likely. Therefore in this showcase scenario, upholding commitments and ensuring application reliability and performability in order to meet strict timing constraints is key. The work

presented in this thesis proposes an adaptive Service Level Agreement (SLA) management system which will meet the requirements of this scenario.

1.2 Thesis Objectives

[O1]. Demonstrate, with the aid of a showcase scenario, how adaptive SLA Management in combination with Grid computing can be introduced into a process to enhance application management and contract non repudiation mechanisms. Its purpose is to demonstrate to commercial organisations that adaptive SLA management and Grid computing can meet Grid application reliability and performability requirements which necessitate timely execution.

[O2]. Demonstrate the performance benefits of application management using control loop adaptation mechanisms for compute intensive Grid applications during run-time. This functionality draws heavily on supporting technologies: predicting application execution times, system and application monitoring and checkpointing.

[O3]. Specify a Service Level Agreement (SLA) specification which enhances non repudiation mechanisms by improving agreement traceability and validation. The specification is machine readable and can be updated to reflect actions taken during application management.

[O4]. Link the research areas of SLA management and resource brokering by demonstrating resource selection and reservation using an external broker.

1.3 Scope of Thesis

This project and the resultant technology propose a QoS enhancement to current Grid middleware. The DAME business process is used as a showcase scenario to demonstrate

the benefits of adaptive SLA Management. DAME requirements are motivated by business obligations which rely on timely execution of computationally intensive applications. In addition they include the need for resource selection and application management. The SLA specification and management system presented in this thesis provide enhanced QoS for the DAME system. It allows the user of such a system to specify time or performance constraints for their Grid application execution.

The SLA specification and management system may be used to manage applications independently but can also form part of larger Grid based software systems. The work in this thesis examines the mapping of SLAs to single Grid application invocations on a per instance basis; applying SLAs at the workflow level is out of the scope of the thesis. Although not the subject of research in this thesis, it is envisaged that adaptive SLA management could work in combination with workflow management systems which have responsibility for invoking sets of applications as part of larger workflow orchestrations.

1.4 Methodologies

The research methodology is described below:

[M1]. *Capturing Requirements*. The requirements were initially drawn from the DAME project. These are heavily influenced by business requirements which rely on timely execution of Grid applications and contract traceability and validation.

[M2]. *UML Modelling and Experimental design*. Drawing from the requirements, the Unified Modelling Language (UML) [3-6] is used to model the adaptive SLA Management system. This allows adaptive SLA management to be integrated within the DAME business process and in addition, the specification of software components and their responsibilities within the system. The experimental design recreates usage patterns which mirror those of a DAME user.

[M3]. *Gathering historical data*. This supports the prediction of application execution times. Before the prediction system can be used it is necessary to gather historical data from previous runs of the application which will be used in the experiments.

[M4]. *Performance comparison* which examines if adaptive SLA Management can prevent a Grid application from failing to complete within a specified time period given a reduction in performance caused by competing applications. This is compared against a Grid application which is executing without adaptive SLA Management on a standard Grid deployment. The comparison considers the performance on a local Grid test-bed and a large distributed Grid infrastructure.

[M5]. *Performance evaluation* of adaptive SLA Management as a solution to the research questions raised by the showcase scenario and during development.

1.5 Major Contributions

The major contributions of this work include:

[C1]. An SLA Management Architecture defining the general principles needed to support management of QoS implemented through SLA within a Grid based system.

[C2]. An SLA specification is provided to support the functionality of the management system [C1] and enhance contract non repudiation mechanisms. Notable features include job specification elements and the ability to dynamically update the SLA with significant actions performed during application management.

[C3]. A system implementation of [C1] incorporating resource reservation and predictive adaptation. The system demonstrates SLA

management within a Grid based system using the principles defined in the architecture. A method of reserving Grid resource using a broker service is described; the example used is the SNAP-based resource broker [7], which provides reservations for compute resources.

[C4]. An initial prediction technique which can estimate the execution time of the Grid application used in the experiments, from information from previous runs. This historical information forms the basis of the prediction, which uses locally weighted linear regression to estimate future application run-times based on a predictor variable.

[C5]. Techniques to monitor at the system level using system load average and at the application level using the amount of CPU time received. Both methods enable the SLA Manager to detect changes in application performance.

[C6]. An adaptive rule-based controller which uses monitoring and a control loop to manage the Grid application execution and determine when control actions are needed. The behaviour is configured using a rule based control scheme.

[C7]. Performance comparison of Grid application execution given the enhancements [C1] - [C6] against an execution running in the absence these.

1.6 Thesis Overview

The remainder of this thesis is organised as follows.

- Chapter 2 reviews background material which helps to scope the area of research, followed by a critical evaluation of relevant research within Grid based SLA Management systems. A review of current SLA specifications is provided, with emphasis on those designed for Grid based systems. Prediction

as a method of estimating application execution times is provided, sampling a range of current technologies.

- Chapter 3 presents the SLA Management Architecture including functional responsibilities. An SLA specification is presented and described in detail. The DAME showcase scenario is used to demonstrate how adaptive SLA Management can be introduced into a process to enhance application management and contract non repudiation mechanisms. The main actors are identified and requirements assigned use cases. From this the system implementation is described, including intended usage scenarios showing the expected use of the system. Finally, an implementation section showcases the SLA Manager and discusses some modifications which were needed to the underlying Grid infrastructure.
- Chapter 4 examines the performance of the SLA Manager implementation using a monitoring technique which measures system load average to detect changes in Grid application performance. The experimental scenarios make use of a local Grid test-bed and system level monitoring to examine the benefit of executing a Grid application with adaptive SLA Management compared with best-effort execution using a standard Grid middleware installation.
- Chapter 5 examines the performance of an SLA Manager implementation using an application level monitoring technique rather than the system level, to detect changes in Grid application performance. The experimental scenario makes use of a large distributed Grid infrastructure – the White Rose Grid (WRG) and application level monitoring to examine added benefit of executing a Grid application with adaptive SLA Management compared with best-effort execution using a standard Grid middleware installation.
- Chapter 6 provides an evaluation of the results from Chapters 4 and 5 and discusses the significance of adaptive SLA Management in a Grid context. The implications of the results, the difficulties and successes encountered and the realistic aspirations of increasing Grid commercialisation.
- Chapter 7 summarises the work chapter by chapter. Results and future directions for the research are followed by an overall conclusion.

1.7 Summary

This chapter has introduced the area of research, including a discussion about the motivations and an outline of the scope. The objectives have been clearly defined and the contributions discussed. An overview of the thesis structure has been provided for ease of reference. The next chapter introduces a background discussion of Grid computing and a discussion of the related work more closely linked to SLA Management and other research issues needed to support the objectives of the research.

Chapter 2

Background and Related Work

This chapter examines the definition and background of Grid computing including a survey of deployed projects and a description of the middleware which enables coordinated resource sharing. Grid resource management and scheduling techniques are discussed including the challenges which exist within such infrastructures. A description of SLA specifications is provided with reference to technologies which exist within the Grid community. A survey of Grid based SLA Management implementations are provided, focusing on functionalities and limitations. At the application level, performance prediction techniques and checkpointing methods are discussed focussing on solutions and their benefits and limitations. A framework for adaptive application execution is discussed, as well as a migration technique for applications deployed as Grid systems. Finally a summary is provided of the key points.

2.1 Grid Computing

Grid computing [8] allows coordinated sharing of resources across organisational boundaries without the need for centralised control. This is in contrast to another form of distributed computing, wide area scheduling systems such as Nimrod/G [9, 10] which provide similar functionality but rely on centralised control in order to maintain a global view of scheduling. The lack of centralised control makes it difficult to rely on management protocols across organisations. However, this looseness of coupling allow Grid systems to scale well in contrast to wide area scheduling systems which scale poorly due to the associated overhead of maintaining global system state. In order to overcome the lack of standardised management protocols, a number of general purpose protocols have been defined. These provide standard APIs for resource discovery,

communication and security between Grid sites. These protocols and interfaces are open standards, unlike other distributed system technologies such as Corba [11] or DCOM [12] which provide similar protocols for distributed resource usage but remain tightly coupled.

2.1.1 Architecture

A layered Grid Protocol Architecture (Figure 1) detailing functional requirements is proposed by Foster et. al [1]. The architecture is modelled on the Internet Protocol (IP) [13] Architecture, with the Fabric layer synonymous with the IP Link layer. The Connectivity layer is broadly equivalent to the IP Transport and Internet layers. The Resource and Collective layers are equivalent to the IP Application layer. Each layer defines a component with functional responsibilities rather than implementation specific details.

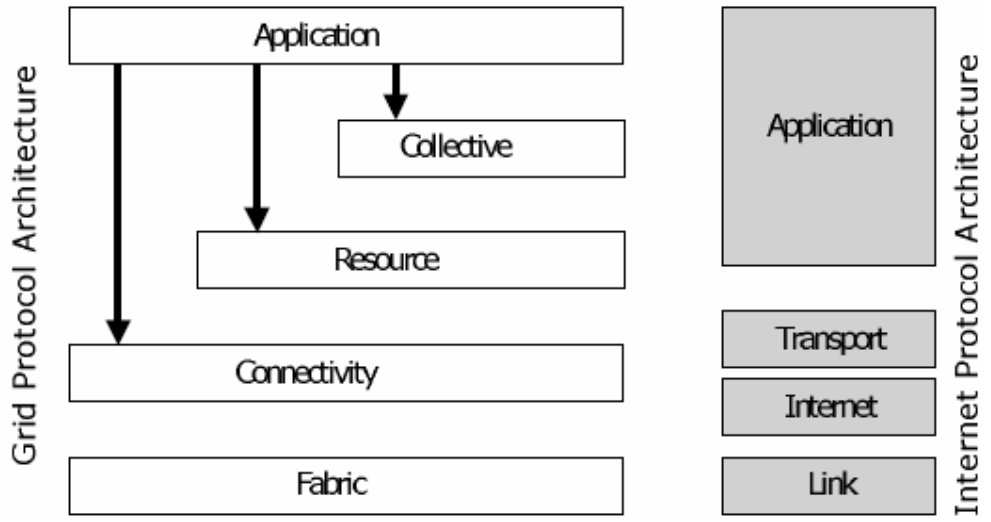

Figure 1 Layered Grid Protocol Architecture

Fabric layer protocols provide resource specific operations for resource sharing. Within this layer are mechanisms for starting, stopping and managing applications; as well as mechanisms for gathering and querying resource information. In addition there are mechanisms for putting and getting of files, including third party file transfer.

Connectivity layer protocols deal with communication and authentication. Authentication provides secure mechanisms for verifying user identity, single sign on and delegation of rights, as well as user based trust relationships. Within this layer are

mechanisms for ensuring secure exchange of data between Fabric layer entities (Grid resources).

Resource layer protocols provide management and information services for individual resources. Information regarding the structure, state and configuration of Grid resources and services are provided through information services. Resource layer management protocols provide negotiated access to resources.

Collective layer protocols are similar to those in the Resource layer but take a coordinated global view. Directory services provide resource discovery over organisational boundaries and help support co allocation and scheduling of tasks. Other services include monitoring, data replication, work load management and software discovery.

The Grid Protocol Architecture provides the functional requirements for sharing resources within a VO. It highlights component services and their functional responsibilities but does not provide specific details on protocols or interfaces, this is provided by the Open Grid Service Architecture (OGSA) [14]. OGSA places the requirements defined in the Grid Protocol Architecture in the context of a Service Oriented Architecture (SoA); a principle which views the components within a system as services. *Grid Services* offer greater interoperability and clear separation between interface and implementation; ideal in a system where resources are heterogeneous and distributed. The separation of implementation from interface allow services to be virtualised for consistent access across different platforms. The Web Service Resource Framework (WSRF) [15] specifies key interface requirements which all Grid services must implement. WSRF is an attempt to unify the diverging paths taken by the Grid Services and Web Services research communities, in an attempt to make them interoperable.

2.1.1.1 Open Grid Services Architecture (OGSA)

OGSA defines a number of architectural concepts and functional requirements which support Grid Service interoperability, integration, virtualisation and management.

Reliable Service Invocation supports the exchange of messages between Grid services in order to communicate. The distributed nature of Grid systems means these messages

are prone to failure. In order to prevent this, Grid services implement an internal state which records if messages have been received.

Authentication identifies users using access control mechanisms which govern service authorisation and usage. Mutual authentication of client and service and delegation of credentials enable mechanisms such as orchestration and enactment of Grid service workflows.

Discovery identifies the location and specifics of the service interface. Applications can dynamically discover, locate and configure Grid Service invocation requests based on the interface specification.

Dynamic Service Creation allows for service instances to be created. A standard creational method exists for all Grid services.

Lifetime Management includes methods which manage and destroy service instances during their lifetime.

Notification enables Grid services to announce changes of internal state to other service instances or applications. The source of the notification is the message sending entity; the sink is the message receiving entity.

2.1.1.2 Web Services Resource Framework (WSRF)

The WSRF specification represents the refactoring of functionality within the Open Grid Services Infrastructure (OGSI) [16] specification. OGSI was a first attempt at specifying the functional concepts within OGSA. One of the criticisms of OGSI was that it had become too large, offering no separation between functions. WSRF has refactored each function into a family of standards which allow a more flexible approach to Grid service implementation. In addition, the refactoring has allowed a smoother acceptance of the possibility that one day Web and Grid services may be interoperable.

OGSI based Grid Services were further criticised as being too object oriented; relating specifically to their stateful nature. Web Services do not support state or instancing which has lead to problems of incompatibility between the two technologies. WSRF has decoupled state from service with the definition of the WS-Resource [17] entity. In addition, the WSRF interface definition has been simplified by the Web Service

Definition Language (WSDL) [18] rather than the Grid WSDL (GWSDL) [19] specification used in OGSI. Defined here are a number of WS-* specifications for WSRF service composition.

WS-Resource is a stateful resource which is associated with a web service. This model allows for the OGSA model to be mapped onto the stateless Web Service model and still retain compatibility.

WS-ResourceProperties defines WS-Resource properties and a specification for querying, updating and deleting the properties associated with a WS-Resource.

WS-ResourceLifetime enables the destruction of WS-Resource entities immediately or by schedule.

WS-Addressing defines the location of a web service or WS-Resource entity.

WS-RenewableReferences is an extension of the WS-Addressing [20] endpoint reference with the ability to resolve a new endpoint should the current one become stale.

WS-ServiceGroup is a means of representing and managing collections of web services; for example to build directory services.

WS-BaseFaults define standard return type faults for web service message exchange.

WS-Notification is a publish / subscribe notification specification allowing clients to subscribe to changes in resource property values described within the WS-ResourceProperties specification.

2.1.2 Middleware

Grid middleware is a layer of software which implements the WSRF specification in order to enable resource sharing over a Grid infrastructure. Once installed, resources can be grouped together to form Grid systems. Development has established four main functional requirements to Grid middleware: data management, information services, execution management and security.

- **Data management** responsibilities include data replication, which enables file redundancy and indexing within a registry. Reliable file transfers enable files

to be copied from one site to another and 3rd party transfer between sites. Distributed database storage and retrieval can be executed over multiple sites, allowing querying tools to invoke search, update and other commands across many sites.

- **Information services** enable monitoring and discovery of resources and services. Availability of Grid services can change rapidly due to the multi-organisational makeup of Grid systems. Monitoring and discovery of services is vital for effective sharing and usage of resources
- **Execution management** enables quick and efficient allocation and management of Grid resources and invocation of Grid services. Methods are needed for locating, submitting, monitoring and terminating Grid services. In addition, interfaces are needed to a number of local resource managers and network batch queuing systems to enable local policy controlled resource usage.
- **Security** enables delegation of credentials and authorisation / authentication mechanisms. Delegating credentials allows multiple Grid service invocations across different sites without the need to authenticate users at each site. Authorisation and authentication allows access policies for Grid services to restrict usage by role or identity. Secure messages between services and data movements at the transport layer are an additional consideration.

A number of Grid middleware implementations have been developed around the architectural ideas discussed in section 2.1.1. An overview of some of these is provided below.

2.1.2.1 Globus Toolkit

The Globus Toolkit (GT) is the most popular example of Grid middleware and pre-dates the OGSA and WSRF initiatives; however, the current version of GT is compliant with both. Its function is to assist application developers to construct Grid systems which can collaboratively share Grid services and utilise heterogeneous resources which are brought together to form VOs. GT offers standard services for security [21], resource management [22], information services [23] and data handling [24]. Resource allocation uses the Grid Resource Allocation Agreement Protocol (GRAAP) [25]. Registration &

discovery uses the Globus Information Services Protocol (GIS) for information services. For security the Grid Security Infrastructure Protocol (GSI) is used.

2.1.2.2 Unicore

Unicore [26, 27] is a Java [28] based OGSA compliant middleware implementing a tiered architecture consisting of user, server and target system. The user tier consists of a GUI which presents a global view of services to the user. A feature of the GUI is a workflow composition tool which allows a sequence of services to be constructed and executed at different sites running the UNICORE middleware. The server tier deals with security and authorisation as well as job submission and management. The target system tier provides an interface to the resource management system. The Portable Batch System (PBS) [29] and Cluster Computing Software (CCS) [30] are examples of network batch queuing systems used with UNICORE.

2.1.2.3 Crown

The China Research and development environment Over Wide-area Networks (CROWN) [31] is a Grid middleware enabling collaboration, especially in the academic research community. A unique feature of the CROWN software is the ability to hot deploy Grid services at run time. This enables end-users of specialist applications to more easily discover resources for their applications and deploy them without administrator intervention. A provenance record allows information to be recorded for individual services, describing usage, errors and messaging. CROWN includes an integrated development environment (IDE) which supports service development and deployment. It provides tools which make service composition less verbose, a criticism which has been levelled at service development using the Globus Toolkit. Automated trust negotiation allows end-users not currently registered with the CROWN system to build a trust relationship. This is achieved through disclosure of their credentials and creation of access control policies.

2.1.2.4 OMII

The Open Middleware Infrastructure Institute (OMII) [32] is a collaboration between partners in the higher education sector and e-Science community. The OMII is a

repository of Grid middleware components, services and tools. The approach is end-user focussed with emphasis on documentation and support – something which has often been overlooked by other Grid middleware providers. The security and authorisation models within the OMII distribution allow service developers to write standard Web services rather than focus efforts on security.

2.1.3 Grid Infrastructures

Grid infrastructures are physical deployments which provide the resources on which Grid applications and services are deployed and executed. They can often be categorised by type according to the suitability of the hardware or type of services deployed. An overview of some Grid infrastructure deployments is provided below.

- The **White Rose Grid** is a community based collaborative project between three UK Universities; Leeds, Sheffield and York. It provides community / shareholder access to cluster resources for users from each of the partner universities. Differentiated classes of service are not offered. However, the work presented in this thesis is an attempt to demonstrate, as a proof of concept, that adaptive SLA management can support the outsourcing of time critical parts of the DAME business process onto a Grid infrastructure.
- The **SunGrid** is a commercial Grid infrastructure deployed by Sun Inc, offering pay as you go access to HPC cluster resources.
- The **Enterprise Grid Orchestrator** (EGO) is deployed by Platform Computing and offers pay per transaction for virtualisation, automation and sharing of IT services for enterprise applications.
- The **European Union Datagrid** (EU-DataGrid) aims to enable next generation scientific exploration which requires intensive computation and analysis of shared large-scale databases, millions of Gigabytes, across widely distributed scientific communities.

2.1.4 Projects and Applications

Many e-Science projects are choosing to implement their solutions as Grid systems. This section lists examples of projects which (i) demonstrates the advantage of

outsourcing a process onto a Grid infrastructure or (ii) enhances Grid middleware in order to boost commercial confidence in Grid technology.

- The **Distributed Aircraft Maintenance Environment** (DAME) [2] is a Grid based decision support / expert system providing analysis of aircraft engine airworthiness. It demonstrates the commercial use of Grid technology by outsourcing a process onto the Grid. DAME analyses in-flight aircraft engine data using a suite of Grid applications to detect component failure. These include in-flight events such as ingress of foreign objects or behaviour which exceeds manufacturer tolerance. This analysis forms part of a routine maintenance schedule which is subject to strict timing constraints. Airline companies operate on the principle of swift aircraft turnaround times within which this analysis has to be completed. The likelihood of variable demand and a fixed cost business requirement make a Grid based solution ideal in this scenario. There are two primary Grid challenges within DAME: the management of large, distributed and heterogeneous data repositories: and rapid data mining and analysis of fault data. DAME Quality of Service requirements include; time constraints, e.g. flight turn around time and a risk that the Grid application will fail to deliver the analysis results within this time.
- **Grid Enabled Remote Instrumentation with Distributed Control and Computation (**GridCC) [33] is a project which aims to improve access to and control of Grid based monitoring and instrumentation. One aim is to develop Grid Services on top of existing Grid middleware to enable control and monitoring of instrumentation and to incorporate this into pilot applications. This project provides an enhancement to Grid monitoring, which improves contract verification techniques; an issue which is a potential barrier to Grid outsourcing of commercial processes.
- **Enabling Grids for e-Science in Europe** (EGEE) [34] is a project which encourages collaboration between industry and the research community. Its goals are to build a Grid network which provides users with computing resources onto which they can deploy their applications. The development and maintenance of Grid services and middleware software is important to the project. EGEE aims to increase the number of providers and users and to

provide them with training and support. EGEE focuses on the application areas of high energy physics and biomedical sciences, as well as others such as search and categorisation.

- The **ICENI** [35] project is a middleware system which gives users the ability to compose and manage custom computational Grids, onto which users can then deploy their applications. The ICENI service architecture makes use of SLAs to grant usage rights to Grid resources. SLAs are used to specify permissions and costs associated with resource usage for specific time periods and durations. SLA usage within the project has advanced contract non repudiation mechanisms which help to improve commercial confidence in Grid usage.

Many of the projects listed above give access to a range of Grid applications. A number of application types are ideally suited for execution on Grid resources, examples include: data mining, computationally intensive, interactive visualisations. Some examples are given below.

- **GRACE** [36] is a Grid based distributed data mining and categorisation engine which searches digital libraries and categorises the results based on a user specified classification scheme.
- **Clinical Decision Support Systems** (CDSS) are Grid based data mining applications which extract medically significant data from a large sets of patient information.
- **Grid protein sequence analysis** (GPS@) [37] is a portal based computationally intensive biomedical application which provides users with access to a suite of protein analysis tools which run as Grid services and present the results back to the user.
- The **Meteorological Pilot Application** [38] uses the Skiron/Eta weather forecasting system to detect hazardous weather events in Greece and the Eastern Mediterranean. It uses Grid based compute, storage and instrumentation resources in order to generate short-term forecasts.
- The **Power Grid Pilot** [39] application simulates the behaviour of electrical power networks, enabling analysis and control algorithms to be tested. It uses Grid based compute resources to forecast the behaviour of power networks

given particular input conditions, so that generators can be more effectively scheduled.

- **gPTM3D** [40] is a Grid based interactive radiological image visualisation application which performs supervised medical data analysis and exploration in real time.

2.2 Resource Management and Scheduling

One of the key functional requirements of Grid usage is the delivery and support for QoS. Many applications have requirements which necessitate a quick response time, reliability, performability [41] and throughput. The provision of these requirements often conflicts with the non-centralised control ethos of Grid computing. Virtualising Grid services which perform identically on resources which are heterogeneous is difficult. The resource architecture on which these services execute may not support the QoS requirements in the same way, if at all. Grid Service scheduling is one example of how QoS delivery can differ significantly. There are a number of local batch queuing systems (SGE, PBS, LSF), which provide controlled access to resources for Grid applications. Each offer similar functionality, but can be configured to control access in different ways. Some have support for advanced reservation whilst others operate access control policies. For this reason, delivery of QoS can be unreliable. These issues and others associated with Grid resource management and scheduling can be categorised into 4 problem areas: site autonomy [42], heterogeneity, co-allocation [43] and policy extensibility.

The resources which make-up Grid systems belong to different organisations. The White Rose Grid (WRG) [44] is made-up of resources from the Universities of Leeds, Sheffield and York. Each **site** has **autonomy** over their own resources, effecting levels of resource availability and application reliability, performability and security.

Heterogeneity relates to the differing hardware and software configurations of resources. Grid resources are drawn from different organisations, the likelihood of heterogeneous configurations is high. The consequences include differing resource functionality and compatibility.

A consequence of both heterogeneity and site autonomy is **policy extensibility**. The rights and privileges over specific resources are likely to differ given their differing configurations and autonomy. This has repercussions on the availability of resources, affecting access rights, times and durations.

Co-allocation of applications deployed through Grid systems may involve multiple resource allocation or workflow orchestration in order to complete successfully. Applications may also execute in parallel using technologies such as MPICH-G2 [45]. If the resources for such scenarios are unavailable from a single Grid resource provider, multiple providers are needed. Grid systems which hope to manage resources should have the ability to co-allocate applications in this way.

Solutions to these problems tend to focus solely on one research area: resource brokering, scheduling or monitoring. The Globus resource management architecture (GRAM) [22] implemented within the Globus Toolkit, attempts to provide a basic solution through dedicated access to heterogeneous distributed resources. GRAM resides above local resource manager systems such as those in section 2.2.1 and below Grid applications and brokers, providing communication between the two. Interfaces have been implemented which allow communication between a number of local resource managers, SGE, PBS and LSF. The main components of the architecture are a Gatekeeper and a job manager. The role of GRAM in the submission of a Grid application is depicted in Figure 2. Inetd is configured to start the Gatekeeper when GRAM requests are submitted, e.g. from a client executing the *globusrun* executable, which is included with the Globus Toolkit or Globus Cog Kit [46].

One of the Gatekeeper's responsibilities is to authenticate and authorise the client's request. When the job is submitted, a Grid credential is included which identifies the end-user. This is compared with an access control list which resides on the server. This maps the Grid user credential to a local user credential. If the request is authorised the Gatekeeper launches a Job Manager to handle the submission of the job. The job is described using the Resource Specification Language (RSL) [47] which the Job Manager uses to prepare the execution and configure such variables as standard out and error. The Job Manager takes care of any subsequent data transfers and associated authentication, e.g. staging in and staging out before and after the execution.

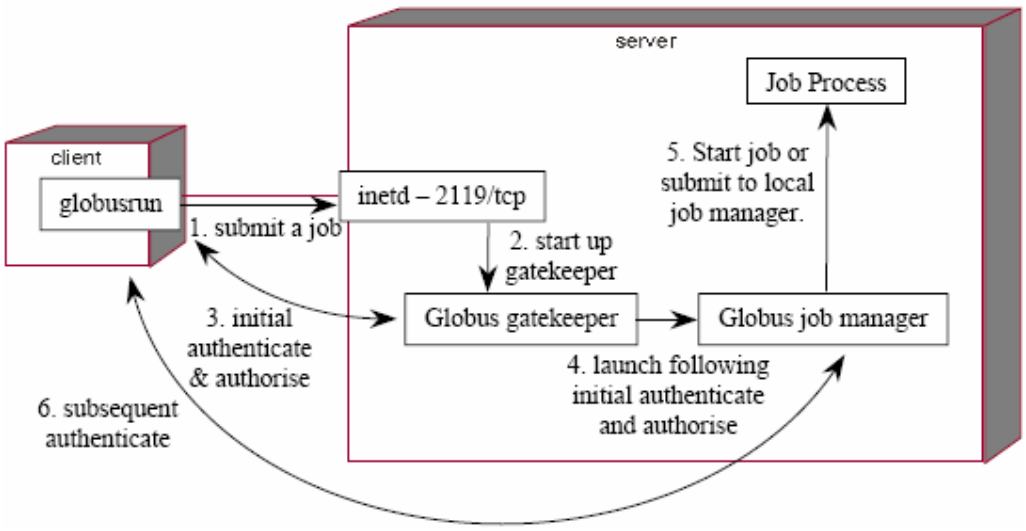

Figure 2 GRAM and the sequence in job submission from client to server [48]

Often a job is run via a local resource manager such as SGE or PBS. In this case, information specified in RSL might include configuration information for the local resource manager. The interfaces which have been provided to a number of local resource managers is one method which has overcome the issue of site autonomy between Grid resource providers. Once the job has been submitted, the Job Manager receives status updates which describe the state of the job: pending, active or done.

2.2.1 Resource Scheduling

Grid resource scheduling is the allocation of tasks or application requests to resources according to a scheduling policy. A scheduling policy [49] dictates how tasks are dealt with as they arrive at the scheduler. Popular algorithms include FCFS (First Come First Served), which schedule tasks in the order they arrive. SJF (Shortest Job First) and LJF (Longest Job First) as their names suggest, give priority to tasks according to their expected duration. Research within this area has produced systems which are classified as network batch queuing systems or wide area scheduling systems.

Network Batch Queuing Systems manage resources at the cluster level rather than over multiple sites. Used in conjunction with Grid middleware they have traditionally offered finer-grained scheduling support than the meta-scheduling offered by middleware. Network batch queuing systems can be further classified into queuing based and planning based. Within queuing based systems, jobs are queued according to the scheduling policy and executed when the job gets to the head of the queue and a

node is free. Planning based systems offer scheduling algorithms which are more suitable for Grid based systems where timely delivery of applications is a key requirement. They are not limited to scheduling algorithms such as FCFS (First Come First Served), SJF (Shortest Job First) or LJF (Longest Job First). In planning based systems, users can attach application start times to enable advanced reservation. In addition these systems make extensive use of backfilling when generating their schedules.

Examples of network batch queuing systems which are queuing based include the Load Sharing Facility (LSF) [50] and PBS. Examples of network batch queuing systems which are planning based include CCS [30] and Maui [51].

Wide-Area Scheduling Systems manage resources over a number of sites. Their functionality enables them to map application tasks to resources spread over a number of sites. Examples of wide-area scheduling systems are Condor/G [52, 53] and Legion [54].

2.2.2 Resource Brokering

Grid brokering is a method of providing brokerage services for negotiating and acquiring resources on behalf of an end-user or customer. Typically, brokers take task requirements from a customer or end-user and attempt to match them with resources or resource access. They differ from wide area scheduling systems as they do not schedule the tasks themselves. Their functionality is enhanced in combination with a wide area scheduling system or a network batch queuing system when they can offer resource reservation. Examples of Grid based resource brokers include the following.

Nimrod/G [9] provides scheduling services for applications that run on the Grid. It aims to solve application requirements based on a set of input parameters expressed in a declarative modelling language. It can also integrate with middleware tools such as the Globus Toolkit [55]. Nimrod/G provides no automated resource discovery; instead the user has to manually choose between a series of Grid machines.

EZ-Grid Broker [56] is a resource brokering system utilising a user interface and information objects for robust multi-site Grid computing. It uses middleware tools such

as the Globus Toolkit [55]. The EZ-Broker provides no method of QoS requirement formalisation.

The **Grid Resource Broker** [57] is accessed though a portal allowing users to create small Grid clusters for their jobs. The broker uses a security model which only allows access to trusted users. There is no formalisation of QoS requirements or SLAs.

The **AppLes** [58] broker uses an agent based architecture to discover and schedule application executions. During runtime, AppLes is able to monitor resource state using the Network Weather Service (NWS) [59]. The forecasting methods within NWS are used to predict changes in resource state. If this occurs, AppLes can reschedule the application onto another resource.

2.2.3 Grid Monitoring

The use of Grid monitoring tools [60] such as the Network Weather Service (NWS) [59] and Netlogger [61] can increase the accuracy of monitoring information within Grid systems. The accuracy of resource information effects the ability of scheduling and brokering systems to allocate tasks reliably. If information is stale or out-dated, the resource status will be inaccurate and tasks submission may fail. Equally, if the sampling period of the monitoring software is too short there may be an abundance of resource information which may choke bandwidth. If this occurs then issues of scalability will prevent the system from functioning correctly. Tierney et al [60] define five non-functional requirements against which all monitoring systems should be validated: low latency data transmission, high data rate, minimal measurement overhead, secure access to data and scalability. Examples of Grid monitoring tools include the following.

MDS 4 [62] is the Monitoring and Discovery System packaged with the Globus Toolkit. It provides a standard protocol for access to and delivery of resource information. In addition it provides a schema for representing this data and interface connections for a number of local information providers. Local information providers are applications which execute on local resource and dynamically monitor resource and system state. Examples include Hawkeye [63] and Ganglia [64, 65]. MDS makes use of query, notification and subscription interfaces defined within the WSRF implementation. In addition to these it implements an index interface which publishes aggregated

information from multiple information sources. A trigger interface is also provided which can perform actions when an information metric condition is reached. MDS4 represents a significant improvement over previous implementations such as MDS2 which was implemented using the LDAP protocol and suffered scalability issues which prevented its use in large Grid systems.

The **Grid Monitoring Architecture** [60] offers an alternative approach to the Index interface within MDS4. It uses a directory service implemented as a relational database to store resource information published by a producer. This can be queried by a consumer who can also subscribe to resource information streams.

The **Network Weather Service** [59] is an example of a local information provider. It provides resource information through its monitoring interface and provides a short-term prediction based on past information. There are 4 main processes which manipulate resource monitoring: reading and writing resource information to secondary storage, sampling, forecasting and a registry of sampling processes. Time-series data is used to produce short term forecasts using one of three techniques based on a mean or median estimate or autogression. The method with the least cumulative error is used as the forecast. Despite claims to the contrary, experience of this software has shown that the sensor process has a high measurement overhead and impacts greatly on resource performance.

Netlogger [61, 66] captures resource and application information using a standardised logging and messaging format. An API is provided for Java which allows applications to call the Netlogger system to produce information logs. The system includes routines for the collection, sorting and visualisation of information. Logging messages can only be generated if Netlogger calls are inserted into the application source code. Limited monitoring can be performed if this is not possible, by creating wrappers for various system components. For a 1% CPU utilisation, Netlogger is limited to a few hundred logging messages a second. This is considered to be inappropriate for efficient monitoring of hardware resources. A binary logging and messaging format is proposed to enhance this ability.

2.3 SLA Specifications

Service Level Agreements provide a mechanism for formalising QoS requirements. They are formal statements for expressing expectations and obligations between parties. Within the Grid research community there have been many SLA specifications, each addressing a particular requirement such as task submission or resource management. Examples of such specifications include the Job Submission Description Language [67], the Web Service Level Agreement Specification [68] and the WSAgreement Specification [69]. There have also been other initiatives which never made it to formal specification, but define methods for expressing aspects of Grid usage. These include the Usage Record [70], the Grid Economic Services Architecture [71], Service Negotiation and Acquisition Protocol [72] and the e-Contracts [73] initiative.

2.3.1 Job Submission Description Language (JSDL)

The Job Submission Description Language (JSDL) [67] defines the application requirements needed in order to submit a Grid job. The elements which make up the schema are shown in Figure 3. It is defined with a Job Definition root element which comprises of a single mandatory element, the Job Description. This description holds information related to the Job such as its ID, the application and resource requirements, plus any data staging requirements. The vocabulary used within JDSL is influenced by a number of network batch schedulers and the DRMAA [74] standard. This allows it to provide job definitions for most local resource managers. JSDL does not provide job definitions to application services which have a web service interface. For this the authors admit that WS-Agreement is more appropriate. JSDL does not provide elements which can be used to record the provenance of the Job. Once the Job has been admitted to the execution host JSDL provides no elements which can be set dynamically. In a business environment this may itself be a requirement of the job. Examples include the verification of the current state of the job or the validation that the SLA completed with no violations. Other omissions include identification of the parties involved.

```
<JobDefinition>
     <JobDescription>
           <JobIdentification/>
           <Application/>
           <Resources/>
           <DataStaging/>
     </JobDescription>
     <xsd:any##other/>
</JobDefinition>
```

Figure 3 JSDL Pseudo Schema

2.3.2 Web Service Level Agreement (WSLA) Specification

The Web Service Level Agreement (WSLA) [68] specification defines a service level agreement language to support dynamic electronic services implementing QoS guarantees within a Web Services Framework [75]. The elements which make up the schema are shown in Figure 4. The language is XML based and able to describe heterogeneous objects relevant to an SLA implementation. An SLA agreement consists of a description of the parties involved; a service definition and obligation description. This describes qualitatively and quantitively, what is covered by the agreement. Action guarantees are included to express what actions should be performed should a pre-condition be met. This form of adaptive SLA management applies mainly to the SLA, rather than application management.

```
<SLA>
<Parties>
    <ServiceProvider/>
    <ServiceConsumer/>
    <SupportingParty/>
</Parties>
<ServiceDefinition/>
     <Schedule/>
     <Operation>
           <SLAParameter/>
           <Metric/>
     </Operation>
</ServiceDefinition>
<Obligations>
     <ServiceLevelObjective/>
     <ActionGuarantee/>
</Obligations>
</SLA>
```

Figure 4 WSLA Pseudo Schema

2.3.3 WS-Agreement

WS-Agreement [69] is a specification intended for the management of WSRF service environments using agreement negotiation. The negotiation is considered stateful and involves a series of automated message exchanges between agreement parties. An agreement Web service factory provides the negotiation interface and a WSRF service instance represents a signed service level agreement. The agreement has state and a lifecycle from creation to termination. The methods used in its implementation are open – allowing for domain specific elements to be added [76]. The elements which make up the schema are shown in Figure 5.

```
<Agreement>
     <Name/>
     <AgreementContext>
           <AgreementInitiator/>
           <AgreementResponder/>
           <ServiceProvider/>
           <ExpirationTime/>
           <TemplateId/>
           <TemplateName/>
     <AgreementContext/>
     <Terms>
           <All/>|
           <OneOrMore/>|
           <ExactlyOne/>|
           {
                 <ServiceDescriptionTerm/>|
                 <ServiceReference/>|
                 <ServiceProperties/>|
                 <GuaranteeTerm/>
           }*
     <Terms/>
     <CreationConstraints>
           <Item/>
           <Constraint/>
     <CreationConstraints/>
< Agreement/>
```

Figure 5 WS-Agreement Pseudo Schema

2.3.4 Usage Record (UR)

The Usage Record (UR) [77] is responsible for defining a common format for exchanging basic accounting and usage data within Grid systems. The elements which make up the schema are shown in Figure 6. The language is XML based providing usage information on elements such as CPU duration, wall duration, processor and node count.

```
<UsageRecord>
     <RecordIdentity/>
     <JobIdentity/>
     <UserIdentity/>
     <Status/>
     <Memory/>
     <Processors/>
     <EndTime/>
     <ProjectName/>
     <Host/>
     <Queue/>
     <WallDuration/>
     <CpuDuration/>
     <Resource/>
</UsageRecord>
```

Figure 6 UR Pseudo Schema

2.3.5 Grid Economic Services Architecture (GESA)

The Grid Economic Services Architecture (GESA) [78] is responsible for defining the GESA service specification for chargeable Grid Services and the Grid Payment System. GESA is not strictly an SLA, it allows service selection and usage based on economic [79] considerations and provides a payment system for Grid services. The architecture includes the use of the Record Usage Service (RUS) [80] and the Grid Payment System. By attaching economic and usage meta-data in the form of service data elements, and a number of service interface definitions, users are able to request Grid service pricing information prior to instantiation.

2.3.6 Service Negotiation and Acquisition Protocol (SNAP)

The Service Negotiation and Acquisition Protocol (SNAP) [72] provides a common protocol for negotiating for the right to consume a set of resources and the subsequent acquisition of the negotiated resources. SNAP provides a framework for three types of SLA: task (TSLA), resource (RSLA) and binding (BSLA). A TSLA is an agreement that specifies the desired performance for an application service. An RSLA specifies the right to use a specific Grid resource. A BSLA associates an RSLA with a TSLA and specifies how that RSLA be applied so that the TSLA can be fulfilled. Whilst SNAP manages the negotiation and acquisition of resources for task submission it does not address the issues of resource management once the SLA exists and the resources are in use.

2.3.7 e-Contracts in e-Commerce

E-contracts [73] governing transaction-based business-to-business (B2B) interactions support value-added service provision. Although this research does not originate from within the Grid community, parallels can be drawn in its goals. The contracts are subject to similar lifecycle procedures; drafting, negotiation and monitoring. The contract specification expresses similar concepts to WSLA and WS-Agreement, such as service obligations, prohibitions and permissions. One novelty of the approach is the expression of obligations, prohibitions and permissions by role. This allows contract negotiations at the organisational level rather than just at the user level. This approach can increase the size of the contract document if many roles have to be considered. An extension [81] to the research considers automatic mechanisms for dealing with e-contract non-compliance. Contracts are deemed to be within one of five levels of compliance whilst they are active. Level 1 reflects total compliance and level 5 total contract failure and the inability to apply corrective measures. This classification of contract violation is more fine-grained than the provisions made for violation in SLA specifications such as WSLA and WS-Agreement. The approach deals more with contracts at the organisation level where a single event or action cannot cause total failure. In WSLA and WS-Agreement, single events or actions can lead to the violation of an SLA, so it becomes meaningless to implement levels of compliance. The SLA used within this thesis is specified at the user level on a task basis, where single actions or events can cause SLA violation, therefore such an approach is not considered.

2.4 SLA Management Systems

The ability to manage and adapt to changes in SLA status is often referred to as SLA management. Management functions include automated mechanisms for SLA generation, monitoring, validation or adaptation. In every case, the primary concern is the ability to monitor actions and verify compliance (or violation, depending on which side of the fence you sit – provider or customer).

SLA management has received increased attention within the Web Service and Grid research communities, but has yet to make an impact on Grid middleware implementations. The major drivers have been within commercial and utility Grid

systems and the Web services industry; domains in which the requirements are driven by profit and efficient utilisation of resources on the side of the providers and provision of application management and contract non repudiation on the side of the customer.

2.4.1 Utility Grid Systems

Utility Grids provide programmable hosting environments for users to deploy applications. Resources can be dynamically requested using the Resource Specification Language (RSL). Once allocated, these resources are used to deploy Grid applications which are executed under best-effort resource provision. The basic building block of QoS provision is accurate monitoring, however, little effort is being made to monitor utilisation, performance degradation or availability. Without such activities, performance guarantees and services such as accounting and auditing cannot be supported. The ability to offer such functionality helps to ensure user confidence and contract non repudiation. Sahai et al [82], propose an SLA specification and monitoring architecture to demonstrate formal modelling and monitoring of SLAs on a utility Grid system.

The SLA specification focuses on specifying and monitoring resources requested by an end-user. Each SLA comprises an ID, the users identity, time period of validity and a collection of Service Level Objectives (SLOs) representing measurement instances and methods. For each SLO there is a dayTime constraint specifying the times over which the SLO is valid. There can be a number of measured items describing the type of object being measured, an address reference and the address of the measuring entity. The measured item is qualified using a number of evaluation functions such as when, what and the function used; for example: must have 99% availability. It is argued that specifying an SLA in such a way provides precise and unambiguous requirements specification. One improvement would include elements which can be set dynamically when the agreement is active to record non compliance. Although the SLA specification used within the work presented in this thesis is inspired by Sahai et al [82], there is a major difference and that is the provision for SLA provenance. To support debugging and auditing, the SLA specification proposed in Chapter 3 allows for dynamic elements which can be set during runtime. Not only is this useful for recording violations, migrations and warnings; for example if the users requirements change during runtime, perhaps to increase resource capacity or tighten agreement parameters – the SLA

specification supports this. If user requirements change dramatically and the SLA fails, this feature may be useful for resolving disputes, if it is found that the failure resulted directly from the changes. These mechanisms strengthen SLA non repudiation by either party, the provider or consumer, because information is recorded which captures provenance data from the Grid application execution. The architecture presented by Sahai et al [82] does keep a record of SLO violation but separates this into a log file. Such information should remain with the SLA as it may still be useful if renegotiation or migration is needed.

By specifying user resource requests in RSL, Sahai et al [82], are separating a key component from their SLA – the job description. This is not important in this instance, as the architecture provides no runtime application management. Runtime adaptation requires that resource requirements be available throughout the lifetime of the SLA in case renegotiation of resources is needed. The infrastructure on which the work is undertaken includes virtualisation support – so there is scope to extend the architecture with a form of runtime adaptivity. SLAs are offered with respect to the hosting environments on which the applications are deployed. The specification provides no support for SLAs on an application basis - the architecture does not contain the granularity to support this. In this respect it could not provide SLO support for time or performance constraints for Grid applications. Another functionality which is missing is support for violation prediction. An example of this is provided by GrADS [83] which uses the NWS [59] to forecast changes in monitoring metrics to predict potential violations – a technique which could be applied to predict future SLO violation.

SLA management functionality centres around a configurable monitoring engine. Emphasis is placed upon collection of measurement data and a new protocol is proposed to achieve this. The justification for this is provided by a requirement for providers from different administrative domains to manage cross-domain SLAs. The protocol invokes a number of limited control actions at each site, independent of how they are implemented at the local level. The measurement protocol consists of five messages sent between the measurer and evaluator. Request is sent by the evaluator and specifies the measurement metrics, such as sampling function, period and reporting parameters. *Init* is sent from measurer and lists the supported measurement metrics at that site. *Agreement* is sent by the measurer when it agrees to the request. *Start* is sent by evaluator to commence measurement. *Report* can be requested to send a measurement report message. The *close*

message terminates the reporting. The protocol provides a common method of requesting and sending measurement information between Grid resources belonging to different administrative domains; however, similar functionality is present within the WS-Notification specification.

2.4.2 Commercial Grid Systems

Users of commercial Grid systems often have to pay for resource usage, therefore expectations are stronger than in users of non-commercial Grid systems. From a Grid resource providers perspective, resource sharing policies and motivation differ greatly between commercial and non-commercial Grid implementations. Within non-commercial Grid systems resource usage is driven by a need to provide shared access for the user base with policies giving equal access for users. This is implemented locally by the network batch queuing systems which controls a sequential schedule of queued jobs which execute according to the access policy. In commercial Grid systems resource usage is driven by profit with policies geared to maximising concurrent utilisation of resources. Such a strategy limits the freedom of the provider to reschedule or deny service to its users. Resources and their utilisation represent the providers costs; they cannot simply overprovision resources in order to meet users expectations in a hope that capacity will never be reached. This approach is too costly and inefficient.

Leff et al [84] examine the effect SLAs have on commercial Grid system usage. The research raises the issues mentioned above and recognises SLAs as a method of controlling resource provision in the context of overall system utilisation. The SLA language used is WSLA, [68] which differs from that used within the utility Grid system presented by Sahai et al [82].

The architecture uses a dynamic offload mechanism to balance load on a provider's resources as demand changes. When demand is high - spare capacity is dynamically allocated onto resources from a pool of Grid resources. This resource pool is made up of resource drawn from other providers. When demand is low, providers can de-allocate their local resources and add them to the pool of Grid resources.

The application used to demonstrate the operation of this offload mechanism is an online brokering web service. The type of guarantees offered within the SLA specification are based on max/min thresholds for concurrent server sessions. Each

service instance represents a sessions on a server. As instances are launched they execute under the control of a specific SLA on a server.

The SLA management engine consists of three modules: monitoring, planning and execution. The monitoring module collects server throughput data for each server. If the number of sessions exceeds the threshold in the SLA, the planning module manages the response. The execution module administers the dynamic offload capability by allocating or de-allocating resources from the Grid resource pool. The results show the system successfully allocates additional resources when demand exceeds SLA levels.

The SLA guarantees offered govern resource access and session management and do not consider execution performance or progress. The system is designed for use within commercial Grid systems deploying transaction based services. Consideration is not given to the use of more CPU intensive applications which would stress the server with a relatively low number of sessions. Requirements for applications with such an execution profile exist within the commercial world, e.g. DAME. Attaching SLOs which define time or performance constraints would require more analytical data measurement than those used in the current model. When specifying SLA guarantees, users are given no indication of acceptable levels of performance. Use of historical information combined with a prediction tool may help to improve resource utilisation. A method of forecasting server utilisation may improve the SLA violation rate by preempting high demand and allocating additional resource before the max session threshold is reached. This represents the difference between a feed forward and feed back control [85].

2.5 SLAs Within UK e-Science Projects

SLAs are being used in many areas of Grid research in numerous projects and applications. An overview of some of the projects is provided below and includes a short discussion on the context of SLA use and why their approach is unsuitable for adaptive SLA Management and the objectives of this thesis.

- The **ICENI** [35] project is a middleware system which gives users the ability to compose and manage custom computational Grids, onto which users can then deploy their applications. ICENI reference implementations have been

produced using Jini [86], JXTA [87] and OGSA. The ICENI service architecture makes use of SLAs for Grid resources exposed to the user. SLAs are used to specify permissions and costs associated with resource usage for a specific duration. Users can instantiate new SLAs to (i) define resource usage by delegation or (ii) restrict application functionality based on identity or role. Automated monitoring and policing of SLAs is not provided. It is seen more as a static usage definition and does not include support for monitoring or adaptation of the deployed applications.

- **GridPP** [88] is developing Grid software and infrastructure to test a prototype architecture for the Large Hadron Collider (LHC) [89]. The system implements network monitoring to identify and differentiate traffic class; this has been developed alongside the MB-NG [90] project. GridPP does not use an automated SLA management system to implement guarantees.

- **Managed Bandwidth – Next Generation (MB-NG)** [90] is a Grid test-bed focusing on advanced networking issues and interoperability. The aim of the project is to address issues arising from multiple administrative domains and their effect on end-to-end quality of service and network throughput. The project does not implement an automated SLA model to formalise QoS requirements. Monitoring is provided by direct measurements taken from network routers.

- The **GRID Resource Scheduler** [91] (GRS) project is investigating a solution to allow Grid users to dynamically request network capacity using advanced reservation. Reservations are between two communicating end systems. QoS specification and management are being investigated but there is no automated SLA provisioning. The project aims to solve the problem of site autonomy by adopting a decentralised approach where QoS management is under local control.

- The **Performance-based Middleware for Grid Computing** [92] project aims to develop a Grid middleware solution based upon the Globus Project; one in which users will be able to automatically use the resources with the minimum of intervention. The project will use existing tools developed at the University of Warwick and integrate them with Globus middleware. It will provide enhancements to **PACE** [93], which is a performance prediction system; **A4**

[94], which is an agent based system for resource advertisement and discovery. The job scheduler **TITAN** [95] will be extended to include QoS metrics and an SLA to formalise QoS requirements. The system uses a code based performance prediction tool to validate QoS guarantees – rather than monitoring tools.

- **Reality Grid** [96] is developing Grid services for modelling and simulation of complex condensed matter structures. Its long-term ambition is to provide generic technology for Grid based scientific, medical and commercial activities. Reality Grid have an interest in achieving performance guarantees through QoS provision. The project requires the simulation to scale to 100s of CPU's and support a high bit rate sustained network bandwidth. Advanced reservation and co-allocation would be important in resource selection. Reality Grid are involved in defining a Grid standard for resource reservation (GRAAP). The project has a plan to develop a resource broker that will address the advanced reservation and co-allocation requirements. Adapting applications to utilise additional resources during run time – through reservation renegotiation is not considered. In addition, there is no formal method of automated SLA negotiation and monitoring within the project.

2.6 Predicting application resource usage

The SLA management systems shown in section 2.4 assume the user has knowledge of their applications resource usage requirements. In Grid systems, users may have no experience of resource usage estimation or may have no time or motivation to gain the experience needed to make more accurate resource usage predictions. Systems which remove this task from the user can help to improve utilisation of Grid resources. Two categories are discussed in detail, both automate the process of resource usage prediction. Learning based approaches [97, 98], use historical information from previous application runs whilst code based approaches, use performance models reflecting application source code to provide performance estimates.

2.6.1 Learning based approach

Providing an estimate of resource usage for a given run of application can help scheduling decisions. Advantages include an improved price/performance trade-off for the user and increased utilisation for the provider. Kapadia et al [97] propose such a technique for PUNCH [99, 100]. PUNCH provides research and commercial applications for users who cannot reliably predict the resource requirements of their specific run of the tool. Users of this system often overestimate the resource requirements of their application run, resulting in an under utilisation of the resource set. The resources used by a specific run of a PUNCH application are dependent on the application input parameters. This dependence is complex but can be represented by a learning technique. Execution statistics from previous runs of the application are collected and stored, the data is used together with a learning technique to estimate resource usage for future application runs.

Prediction of resource usage within a computational Grid is limited by two factors. Prediction must take place in real time between user initiation of the application run and before the application is scheduled. This places a limit on the amount of data used for prediction; too much and the prediction will delay the start of the run, too little and the prediction will be inaccurate. Any technique must account for the variable demand and reliability of resources within computational Grids. The solution proposed within PUNCH employs locally weighted regression [101]. In addition, a cache is used to limit the scope of the data around which a prediction is made. Finally, a two level memory is used to differentiate between short term and long term variations in usage. The additional features provide for faster generation of resource usage predictions and the ability to account for short–term time dependent variations in resource load. The method used does not perform well when the historical dataset is sparse or when there are unusual deviations from typically observed resource utilisation.

Other learning algorithms are discussed by Dushay et al [98] which make use of historical observations to generate predictions. They discuss running averages, single last observations and low pass filters as methods to predict the availability and response time of a web-based search engine. A running average uses all historical observations in the dataset to generate an average value. Last observation uses the most recently

observed measurement to generate the prediction. A low pass filter is an average of recent measurements which decays old observations exponentially.

2.6.2 Code based approach

An example of code based performance prediction is present within the Performance Analysis and Characterisation Environment (PACE) [102]. A language called CHIP^{3}S [103] is used to build performance models for applications run within PACE. These models predict the execution time and resource usage for a given run of an application. Each model consists of subtasks which comprise flow control elements such as loops and conditional statements. Each element contains a set of primitive operations which represent events such as memory accesses, floating point unit multiplies or MPI (Message Passing Interface) communications. Hardware models are created listing the performance costs associated with each primitive operation on a given hardware architecture. Performance predictions are generated by comparing the compiled performance model, the hardware model and the application arguments. Predictions take between 1 millisecond and 1 second to evaluate, so do not suffer the constraints of the learning based approach. It is unclear how the approach would work with code which implements reflection, where the path of the execution is not known before run time. In addition, in environments such as DAME the user does not have access to the application source code, which is subject to Intellectual Property (IP) restriction. The applications would have to be reengineered to add the CHIP^{3}S templates. A further complication involves the production of hardware models in an environment which suffers dynamic resource availability. Resource membership of the Grid is dynamic, a method of evaluating performance models for new resources should be considered.

2.7 Checkpointing

Checkpointing is the process of capturing application or resource state and recording it to secondary storage. It is performed as a fault tolerance exercise to avoid total loss of results in cases of resource failure or a decrease in application performance. It is also performed to support adaptation of Grid based applications during runtime. Grid systems span multiple domains, each with their own resource usage policies and dynamic resource availabilities. With this in mind, checkpointing an applications state

in combination with migration onto another resource can prevent total application failure. Checkpointing as a method of logging application state may be used by the user to debug run specific data. Application checkpointing can be implemented at the system level or user defined, a comparison of which is presented in [104].

- **System level** checkpointing is provided transparently by the operating system or middleware. The application does not have to be specially engineered to enable checkpointing to take place. The operating system does not have any knowledge of the application and simply captures a complete process image.
- **User defined** checkpointing is engineered by the software designer / programmer. It is commonly implemented as specific code within the application.

Both approaches have their merits depending on the situation; broadly, user defined checkpointing is more flexible because checkpoints are taken at logical places within the application code rather than blindly at specific intervals. System level checkpointing is more transparent to the user because checkpoints are handled by the operating system rather than the actual application. The size of the checkpoint is significantly bigger in system level checkpointing because the technique must record memory images and previous system calls. This operation is not only inefficient, but imposes a large performance overhead on the system. In comparison, the checkpoints created by user defined checkpointing are small because application state is known. The performance overhead is significantly lower as a result.

Condor/G [53] is an example of a middleware system which is capable of system level checkpointing. The system is primarily a network batch queuing system (NBQS) which exploits spare CPU cycles from a pool of workstations existing within an organisation. Users can submit jobs to the system which will scavenge spare capacity in order to process the job. Each contributor to the Condor pool can define usage policies for their workstation. If these are violated the system must be able to checkpoint the job and migrate it to another workstation.

2.8 Application Adaptation in Grid systems

Application adaptation [105-108] during runtime involves monitoring aspects of the execution behaviour combined with control policies which determine control actions taken if certain conditions are met. Adaptation involves changing the behaviour of the application process in order to effect its reliability or performability. Process control theory is used within the chemical industry to control process flows, temperatures and reactions [85] in order to maintain product quality and process safety. Multiple-input multiple-output control theory is often used within this field. Diao et al [109], apply this technique to control performance and enforce policies on a web server. However, this is only possible due to the continuous domain of the control action response – in this case maximum permitted sessions and session timeout. Within Grid systems, control theory and control action responses are limited in scope because of problems with site autonomy and policy extensibility. Control actions which are valid at one site may not be valid at others. One control action which is effective is migration, however it does not have a continuous domain. Migration is the action of moving an application execution from one resource to another in order to improve performance. Depending on the control policy in force, this could be dependent on price, performance or risk [110]. In combination with checkpointing, migration can improve the performance or enhance the fault tolerance of applications executing on Grid systems. This requirement is justified by the varying quality, reliability and availability which Grid systems display.

Huedo et al [106, 111] provide a framework for adaptive execution within Grid systems. A Grid application model describes the key attributes needed by an adaptive application. These include resource requirements, a performance profile, input and output files, a restart file and standard out/error specification. The framework uses key components for resource selection, job dispatch, submission and application performance modelling. Resource selection is made against the application resource requirements when it is pending or awaiting scheduling. Job dispatch transfers input files and executables to the remote resource and manages the configuration of standard out/err and other environment variables. Job submission occurs locally on the remote resource. Performance modelling is used to maintain the performance of the application expressed in the performance profile. This is achieved through job migration which is activated if the following conditions are met.

- If the user requests a change of resource
- There is a performance degradation in the application.
- A new more suitable resource is discovered or the job is cancelled / suspended on the remote resource.

Migration feasibility is considered before action is taken, however if the job encounters an error, migration occurs immediately. In the case of new resource discovery, migration only occurs if the new resource is of a higher specification. In the case of performance degradation, migration is evaluated on conditions such as time to finalise [107, 112] or input and restart file transfer costs [106, 111]. The system is tested using an application which can record the time spent on each section of the code in a performance log. The performance profile defines a maximum threshold value for each of these sections. If the value in the performance log exceeds that in the performance profile, the application is migrated. This method of performance monitoring strongly depends on the applications ability to generate its own performance data and the existence of a checkpoint file. The results demonstrate that performance modelling can react to performance degradations and trigger migrations which result in application completion times which are 35% shorter than the same application without migration.

However, speed up due to migration depends on the relative position of the application in its schedule. If the overhead of migration is large compared to the total expected duration of the application, then migration may only provide a speed up in certain circumstances. However, if this is considered in the context of the duration of the resource reservation and the practical assumption that this cannot be extended, then migration may be useful in preventing the total loss of results. If the end time of the scheduling slot is close and there is a possibility that the application may overrun, then migration is a practical control action which can prevent the total loss of results; despite the apparent loss in performance due to the migration overhead. It may also be acceptable to allow the application to finish and then restart manually using the latest available checkpoint. The suitability of either approach is determined by the user or task requirements. Overheads stem from the length of time it takes to stop the application on the current resource and transfer the files to the new resource and begin the execution there. This overhead is lower if the migration is taking place within a single resource provider and the movement is between nodes on a shared memory machine. If the

movement is across a Wide Area Network (WAN) between providers, the overheads are greater.

Overheads due to migration are considered by Vadhiyar and Dongarra [107, 112], by making use of the GrADS [83, 105] architecture to test a migration framework. The framework includes an application, referred to as a *migrator,* a *contract monitor,* which monitors *migrator* progress and a *rescheduler,* which decides when a migration is necessary. The application uses its own monitoring information to record the execution time of different phases of the code. This approach is similar to that of Huedo et al [106]. The *Contract Monitor* compares the recorded time with a predicted time to determine a ratio between the two values. This ratio is averaged and compared with a threshold value, which if reached triggers the *rescheduler* to take some action. The *rescheduler* uses the NWS [59] to determine which other machines are available on the Grid. It then uses the GrADS performance modeller to create a new performance prediction (contained in the *contract* monitor) for the application. The *rescheduler* calculates the remaining execution time for the application if it were executing on the new resource and if it were to stay on the current resource. The overhead due to migration is also taken into account and a value is derived for the rescheduling gain. If this is greater than 30%, the application is migrated. If less than 30%, the threshold values are adjusted to give a greater tolerance. This value varies depending on the resource load, monitoring accuracy and the application.

Both adaptive techniques [106] and [107] use applications which are tightly coupled to the monitoring tool. Both use applications which rely on signature models and code analysis to generate predictions. In addition, the approaches rely on higher level Grid systems, such as GrADS and Grid Way [113]. This limits the scope of migration to providers which have a fully working implementation of the GrADS system. The SLA Management Architecture presented in this thesis differs from that presented by Vadhiyar and Dongarra [107, 112] in two major respects. It does not require a further layer on top of a standard Grid middleware setup in order to function. The SLA Manager is deployed alongside the application server housing the DAME portal and does not need to be resident on any other Grid resource. Additionally, the application used by Vadhiyar and Dongarra [107, 112] is self monitoring and includes code within the application to fulfil this task. The SLA Manager monitors Grid applications which do not need to be re-engineered to perform self monitoring. In a commercial setting this

is often impossible because the IP of the application is owned by a separate organisation.

2.9 Summary

This chapter has examined Grid computing and discussed the architectural requirements needed to allow such systems to function. Initiatives have been presented which define and specify the functional requirements needed to allow such systems to be realised. Examples are given of projects which are implemented as Grid systems; and in addition, the types of Grid applications running within these projects. A description of the DAME project provides a motivational project scenario for this research. The topic of Resource Management and Scheduling provides finer grained scope to the problem of QoS provision within Grid systems; included within this are examples of solutions within this area. There is a description of currently available SLA specifications and management systems; included is a survey of e-Science projects which make use of SLAs or would benefit from their use. Adaptive techniques, prediction and checkpointing are discussed as requirements for application migration within Grid systems.

The next chapter introduces the proposed SLA Management Architecture and an SLA specification. The DAME showcase scenario is used to demonstrate how adaptive SLA Management can be introduced into the DAME business process to enhance application management and contract non repudiation mechanisms. The main actors are identified and requirements assigned by use case definition. From this the system implementation is described, including intended usage scenarios showing the expected use of the system. Finally, an implementation section showcases the SLA Manager and discusses some modifications which were needed to the underlying Grid infrastructure.

Chapter 3
SLA Management Architecture and Specification

This chapter presents a detailed description of deliverables [C1-6]. The DAME showcase scenario is used to demonstrate how adaptive SLA Management can be introduced into the DAME business process to enhance application management and contract non repudiation mechanisms. An application model identifies the requirements of the application needed in order to support adaptive SLA management. The SLA Management Architecture is presented including functional requirements and implementation specifics. System use cases describe in detail the behaviour assigned to each entity in the architecture. The system implementation identifies components, their associations and responsibilities. Intended usage scenarios illustrate the intended actions and behaviour through each system use case. Implementation issues are discussed, along with the SLA specification used within the architecture.

3.1 SLA Management Usage Scenario

The Distributed Aircraft Maintenance Environment (DAME) [2] is a Grid based decision support / expert system for analysing in-flight engine data using a suite of Grid applications to detect component failure. In-flight events include ingress of foreign objects or behaviour which exceeds the manufacturers' tolerances. As part of a routine maintenance schedule this analysis is subject to strict timing constraints. Airline companies operate on the principle of swift aircraft turnaround times within which this analysis has to be completed. The activity diagram in Figure 5 illustrates the DAME

scenario and demonstrates how adaptive SLA management fits within the process. This serves as a motivator for the SLA Management Architecture.

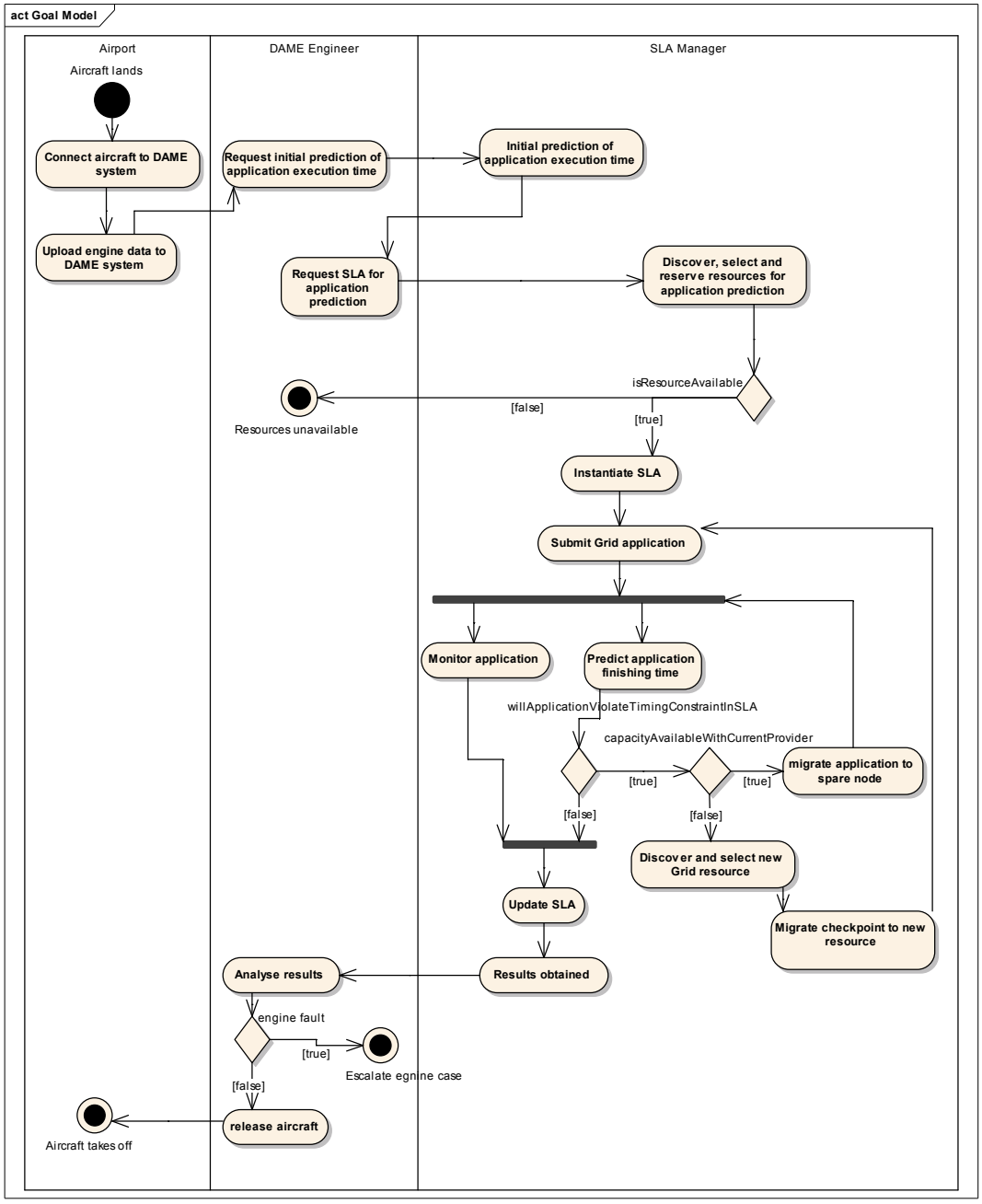

Figure 7 Overall System Usage Scenario

When an aircraft lands at an airport, it is connected to the DAME system and the in-flight engine data is uploaded. The aircraft (DAME) engineer has access to a suite of analysis tools accessed through the DAME portal. Initial analysis results are required during the period of aircraft turnaround so the airworthiness of the engine can be determined and a decision to release the aircraft can be made.

The first stage of the analysis process examines in-flight vibration data recorded from the on-wing QUICK system [114]. This is processed using an application called eXtract Tracked Orders (XTO). The in-flight data consists of a control file and binary data files in proprietary ZMOD format [115]. The number of binary data files depends on the duration of the flight. The control file lists the data files and in which order they should be processed by XTO. From this data, XTO determines if any components have exceeded operational tolerances and records these in an "event-list". The engineer wants a guarantee that these results will be returned within a specified time period so he can specify a deadline by which a release decision will be made.

The engineer can request an initial prediction of the expected duration the results will take. This is based on historical data from previous runs of the application. From this estimate, the engineer can make an SLA request to the SLA Manager. Once the SLA Manager receives the SLA request, it discovers which resources meet the requirements and if they are available for the duration indicated by the prediction. If resources are unavailable to meet the users SLA requirements the request fails. If appropriate resources are available, an SLA for the task is instantiated and the application is submitted. Whilst the application is executing, the SLA Manager must monitor the application state and predict the finishing time. If the application is predicted to violate the finish time specified within the SLA, the application is migrated. If spare resource capacity is available with the resource provider currently executing the application, the migration occurs onto a new node. If no spare capacity is available with the resource provider currently executing the application, the SLA Manager must discover other resources with different providers. Once resources are discovered, the application checkpoint is migrated to the new resource and the application is resumed. Monitoring and prediction are restarted on the new resource.

When the application completes, the SLA is updated and the results are obtained. The engineer analyses the results to determine if an engine fault was present during the flight. If no faults exist, the engineer can release the aircraft for its next flight. If a fault exists, the DAME engineer will attempt to identify the problem whilst the aircraft is grounded. If this is not possible the issue is escalated. The escalation process is not shown but involves the fault case being passed to a maintenance analyst and then a domain expert who possesses more specialist knowledge and has access to other tools within the DAME system.

3.2 Application Model

The type of application which the SLA Management Architecture is targeting are batch processing applications which are CPU intensive and have user requirements which consider the delivery of results to be time critical, an example of which is XTO. The applications are themselves not SLA aware and therefore have to be monitored and adapted externally using functionality from within the SLA Management Architecture.

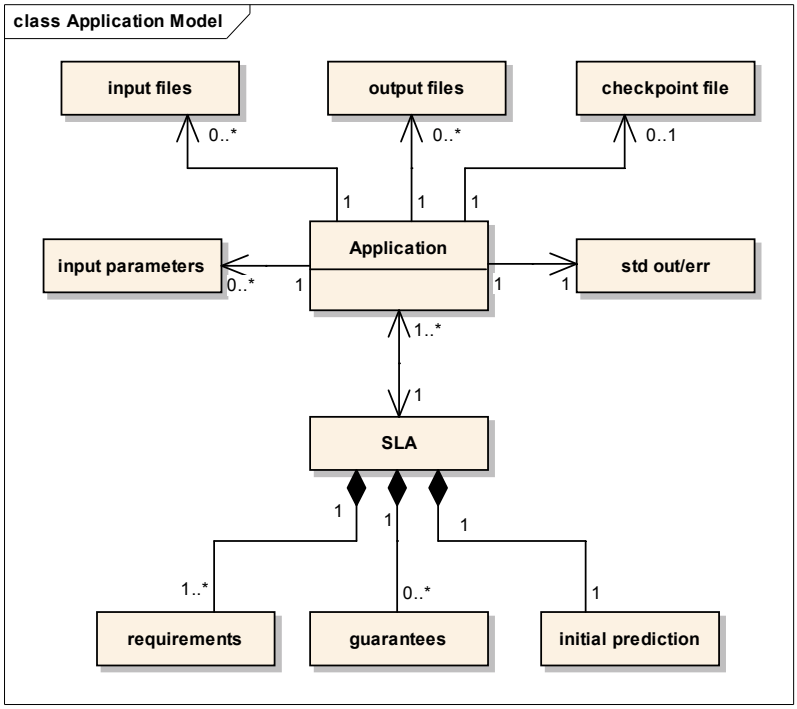

Figure 8 SLA Management Architecture application model

Figure 6 illustrates the application model considered by the SLA Management Architecture. The application may need input files (or data) and parameters to be specified in order for it to perform its task. Standard out and error may be needed to designate the location of output and error files. The application instance is associated with one SLA, which is composed of a set of requirements and guarantees. The requirements are hardware or software related and used during resource discovery and selection. The guarantees describe what should be maintained during the application execution. The SLA Management Architecture is targeting guarantees which are related to resource performance (and as a consequence application performance), for such metrics as system load average; and adherence to execution deadlines. The composition

of guarantees is optional, recognising the possibility that a user may simply want to use resource discovery or selection; rather than reservation, runtime monitoring or adaptation. In addition to requirements and guarantees, the SLA is composed of an initial prediction. This is an approximation of the applications execution time given the input files and parameters and is used in the resource selection process. Grid resources, especially cluster based queuing systems often place restrictions on the duration of time users execute their applications. To support migration and restart, the application should have associated with it a checkpoint which can be used to resume the execution from the current position. User-level checkpointing implemented within the application is favourable because of the heterogeneous resource make-up of Grid systems. The checkpoint file should be architecture independent so that application restart can occur on any resource architecture within the Grid system.

3.3 SLA Management Architecture

The SLA Management Architecture provides the functional requirements needed to support automated SLA management in scenarios such as DAME (Section 3.1). The architecture demonstrates a range of functionalities working together in a single system; something which is novel within a Grid based system where application delivery is constrained by requirements and subject to guarantees. The functionalities have each been demonstrated singularly within SLA systems, (Sahai et al [82], Leff et al [84]), adaptive systems (Huedo et al [106, 111], Vadhiyar & Dongarra [107, 112]), and resource prediction systems (Kapadia et al [97]). The SLA Management Architecture, in addition to demonstrating these functionalities together also exhibits key implementation differences. The adaptive systems presented by Huedo et al [106] and Vadhiyar & Dongarra [107, 112] use Grid applications which are able to monitor themselves by providing timings for sections of computation. The SLA Management Architecture differs by taking responsibility for monitoring away from the application. The application used by Vadhiyar and Dongarra [107, 112] is self monitoring and includes code within the application to fulfil this task. The approach taken by the SLA Manager means the application software does not have to be specifically re-engineered to perform self monitoring. In a commercial setting this is often impossible because the IP of the application is own by a separate organisation. In addition, it uses an external resource broker, the SNAP Three-phase commit broker [116, 117] to discover resources.

Huedo et al [106] make use of the Grid Way framework for resource discovery; whereas in Vadhiyar & Dongarra, the same functionality is provided by the GrADS system. Both Grid Way and GrADS add further layers of complexity on top of the Globus middleware software. The presence and configuration of which adds further to the submission complexity and overhead between Grid resources. From past experience, the reliability and availability of a correctly working standard Grid middleware setup across a VO is hard to achieve. Therefore, achieving a reliable setup which adds an additional layer on top of the standard Grid middleware setup and maintaining this across a VO would be even harder. The SLA Manager does not require a further layer on top of a standard Grid middleware setup in order to function. The SLA Manager is deployed alongside the application server housing the DAME portal and does not need to be resident on any other Grid resources.

Neither Huedo et al [106] or Vadhiyar & Dongarra [107, 112] make provision for an SLA specification for formally expressing requirements and guarantees. In contrast, the SLA management systems presented by Sahai et al [82] and Leff et al [84] do provide formal SLA specifications and methods for monitoring guarantees, however they lack the adaptive capabilities of Huedo et al [106] and Vadhiyar & Dongarra [107, 112]. Leff et al [84] dynamically adapt resource availability with varying demand, but this is applied at the system level rather than the application level. Decisions are taken which ensure efficient resource utilisation and profit, rather than guaranteeing the finish time of applications. Sahai at al [82] offer support for SLA monitoring but this is applied to resource availability and access rather than application performance. Emphasis is placed on reliable monitoring protocols rather than adaptability during runtime.

Kapadia et al [97] provide a method of predicting resource usage prior to application execution. The method removes resource estimation from the domain of the user and places it on an analytical method using statistics from previous application runs. However, the work does not extend to monitoring or adaptivity during application execution, either at the system or application level.

The SLA Management Architecture presented in Figure 9 combines the functionalities listed together in a single system.

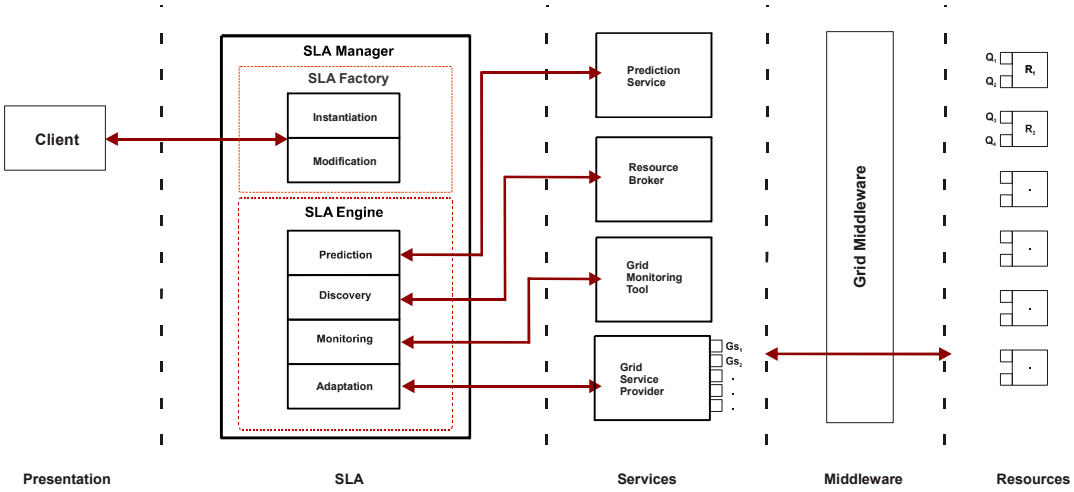

Figure 9 SLA Management Architecture

The architecture illustrates five separate layers; presentation, SLA, services, middleware and resources. The presentation layer contains the client through which the user communicates with the SLA Manager. The SLA layer contains the SLA Manager which executes all SLA management functionality including SLA instantiation and modification. The SLA Engine performs initial prediction of the application execution time, resource discovery, application monitoring and application adaptation. These functionalities are either carried out directly by the SLA Manager or through a service within the services layer. The middleware layer provides access to the resources onto which the Grid application is deployed.

3.3.1 SLA Instantiation and Modification

The SLA Factory provides SLA templates for specifying requirements, guarantees and predicted execution times for the Grid application task. The SLA template and the information it contains is used to select and reserve the resources onto which the Grid application will be deployed. The SLA Factory is able to modify the SLA instance to reflect the actions taken by the SLA Manager during application execution. This includes warnings, migrations and violations signalled as a result of monitoring and adaptive techniques applied to the application during runtime.

3.3.2 Initial prediction of Application execution time

The SLA Management Architecture uses a prediction service to estimate the execution time of an application on a given resource. Initial prediction is a requirement because it demonstrates a step forward from the adaptation systems in the literature [106, 107, 112, 118] which pay little or no attention to the issue of initial prediction, often relying solely on the end-users estimation. To demonstrate the estimation of an application's execution time, experiments will be carried out which make use of a running average [98] of historical observations from previous runs and a learning based [97] local linear regression [101, 119] technique which uses a dataset of historical observations from previous application runs. More details of the methods used are provided in section 4.2 and 5.2.3. The SLA Manager uses initial prediction based on historical observations to achieve a more realistic approximation than relying on an end-user to provide the estimate. Allowing the end-user to estimate the execution time could lead to an inaccurate estimation if they have no experience with the application. The initial prediction is needed during the application execution as a basis for estimating the applications remaining execution. An inaccurate initial prediction will effect the accuracy of this estimate and lead to false positive and negative control actions.

3.3.3 Resource Selection and Reservation

The SLA Manager is able to interface with a resource broker to provide resource selection and reservation. The broker has responsibility for placing reservations with a resource provider. An example of such a broker is the SNAP-based resource broker [116, 117], which provides reservations for compute resources. Work by Padgett et al [7] considered this section of the architecture by demonstrating the selection and reservation of resources.

Figure 10 illustrates the interactions between the user, SLA manager and resource broker during the resource reservation process. When a user requests an SLA, the task requirements and guarantees must be specified in the form of hardware and software specifics. These may be abstracted from some other metrics to insulate the user from making such decisions; the user may not understand the meaning of such low level requirements. The Resource Broker uses the requirements and guarantees to search for matching resources. After resource selection, the broker will attempt to place a

reservation on the resource. If this is successful the broker confirms the reservation and returns the resource details and queue information back to the SLA Manager. Once resources have been reserved the SLA Manager creates an SLA template and presents it to the user. The SLA template can be marshalled into a persistent state (XML) using the Java Architecture for XML Binding (JAXB) [120] API.

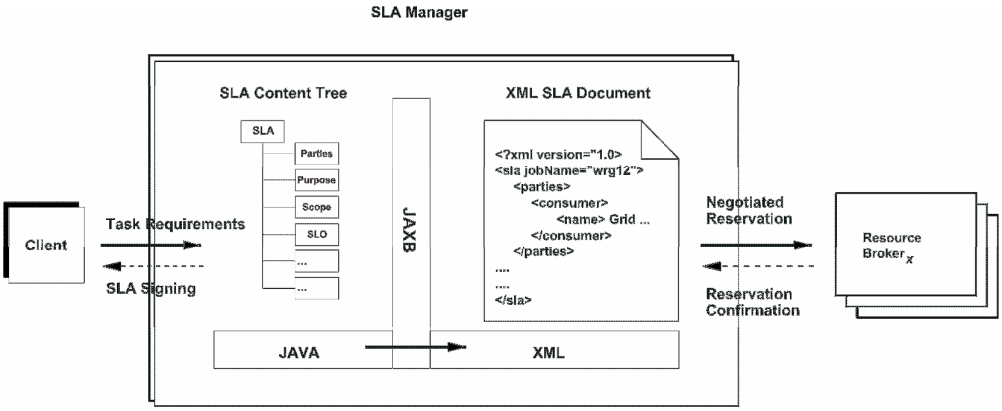

Figure 10 Resource Reservation Process

3.3.4 Resource and Application Monitoring

The SLA Engine can be configured to use different monitoring tools depending on the type of guarantee being monitored. It is the correct matching of guarantee to monitoring tool which enables validation against measurement data. The aim is to dynamically monitor the state of the SLA-bound application during execution. To demonstrate automated monitoring, experiments will be carried out using the MDS and a custom monitoring script. More details of the methods used can be found in section 4.2.1 and section 5.2.2 respectively. The SLA Manager uses monitoring to periodically gather data which can be used to estimate the finishing time of the application. This is then used to determine an appropriate control action which will try to prevent a violation of the timing constraint within the SLA.

3.3.5 Estimating the Applications Remaining Execution Time

Calculating an estimate of the remaining execution time is used as a means of specifying a control variable with which to compare to the scheduled remaining execution time. From this, control actions can be inferred from the status of the two control variables. One of the proposed control variables is an estimate of the application's remaining

execution time. This metric reflects an application's performance as a function of the total amount of computation needed to complete and the current rate of computation on the given resource. The total amount of computation needed to complete is determined through initial prediction against a set of historical observations from previous runs. The rate of computation given by the resource is determined through resource monitoring. From an application perspective, useful computation is performed when the load on the resource is low and the CPU is executing application instructions. Operating Systems (OS) maintain information about system load and processes which represent application executions. To demonstrate runtime estimation of an application's remaining execution time, experiments will be carried out which make use of system load average and application level CPU time. Assuming that each resource maintains accurate records, the CPU time / system load average can be monitored to determine the amount of computation the application has received during a specific monitoring period. Implementation methods are described in greater detail in section 4.2.3 and 5.2.3.

3.3.6 Application Adaptation

The fundamental aspect of application adaptation within the SLA Management Architecture is the ability to steer the execution in the face of changing resource conditions. In systems such as DAME, the SLA Manager does not have authoritative control over all applications submitted to the Grid infrastructure. The reality is that DAME applications will be executing alongside other Grid and non-Grid applications. The ability to affect the amount of CPU time the application receives is controlled through application migration. This could be migration onto a faster resource or one which offers a load average which is less than that of the current resource. Adapting the execution of an SLA bound application in this way has the potential to significantly improve performance and timely completion of an application, in the face of competition from other processes.

Whilst there is a need to dynamically adapt an application during its execution there is a motivation for application checkpointing. During the course of a running application, resource availability and load may change; performance and timing constraints may be affected. In addition, an application may be executed on an initial understanding that there exists a maximum wall time for the execution. If this limit is approached and the application has not completed, it may be necessary to checkpoint, renegotiate access to a

new resource and migrate the application to avoid loss of results. A checkpoint must be taken otherwise full restart of the application is required. It is assumed that application level checkpointing is supported by the applications used with the SLA Management Architecture. Delays due to checkpointing are not taken into account because the application is assumed to support user-level checkpointing at periodic interval whilst the application is executing. If migration needs to take place, the latest available checkpoint is used.

The SLA Manager uses rule based adaptation to adapt applications during runtime. Rule based control offers an effective control algorithm for compute based Grid applications with attached timing constraints. Site autonomy and policy extensibility within Grid systems mean that control actions do not have a continuous domain. The SLA Manager is not designed to have authoritative control over all applications executing within the domain in which it operates. Therefore, a limited number of control actions are available to enforce the SLA bound application. Fuzzy based control [121] is an alternative control strategy which also implements a set of rules to determine the state of a controlled process. This approach is less favourable in this situation because the control actions do not have a continuous domain. A fuzzy approach is more appropriate when the control action can be varied.

The rules within the controller express a series of control variable states and associated control actions reflecting the guarantees (timing constraints) expressed within the SLA. They are represented by **If** premise **Then** consequence. The premise represents the current control variable state and the consequence the control action. Examples of control variables include the scheduled remaining execution time $T_{schedule}$, and the predicted remaining execution time $T_{remaining}$. The rules are triggered when the premise is satisfied, which in turn triggers the control action; limited to migration as already discussed.

If the example of control variables is taken above, the adaptive decisions involve the prediction of the remaining execution time for the SLA bound application. The predicted remaining execution time, $T_{remaining}$ is generated using the monitoring application described in Section 3.3.4 and the technique in Section 3.3.5. The scheduled remaining execution time, $T_{schedule}$ is taken from the SLA and is based loosely on the

predicted wall time which is generated using the initial prediction technique in Section 3.3.4, and includes a buffer depending on the strictness of the timing constraint.

```
1. procedure adaptive_decision (T_remaining, T_schedule,)
2.         if T_remaining > T_schedule then
3.                 control_action := migrate
4.         else
5.                 control_action := zero
6.         return(control_action)
7. end control_action
```

Figure 11 Adaptive Decisions: An Example

An example of the adaptive decision procedure is given in Figure 11. Note the algorithm is illustrative and rules are configured to reflect the stringency of the task requirements. The first rule describes the situation when the predicted remaining execution time $T_{remaining}$ is greater-than the scheduled remaining execution time $T_{schedule}$. In this situation a signal is sent to migrate the application. The second rule describes the situation when the predicted remaining execution time $T_{remaining}$ is less-than the scheduled remaining execution time $T_{schedule}$. In this situation no control action is taken.

3.4 System Use Cases

The system use cases are shown in Figure 12 and describe a set of goals using behaviour and actions from the perspective of the major entities within the system. They also describe goal dependencies to help build a picture of how each action is related to the next. The use cases are divided into boundaries of responsibilities. If a use case falls within a boundary, that entity is responsible for carrying out the action. The key entities considered during the implementation of the SLA Management architecture are the end-user, the SLA Manager and the resource broker. Within each use case the goal and actions are described. Underlined text within each description infers a dependence on another use case.

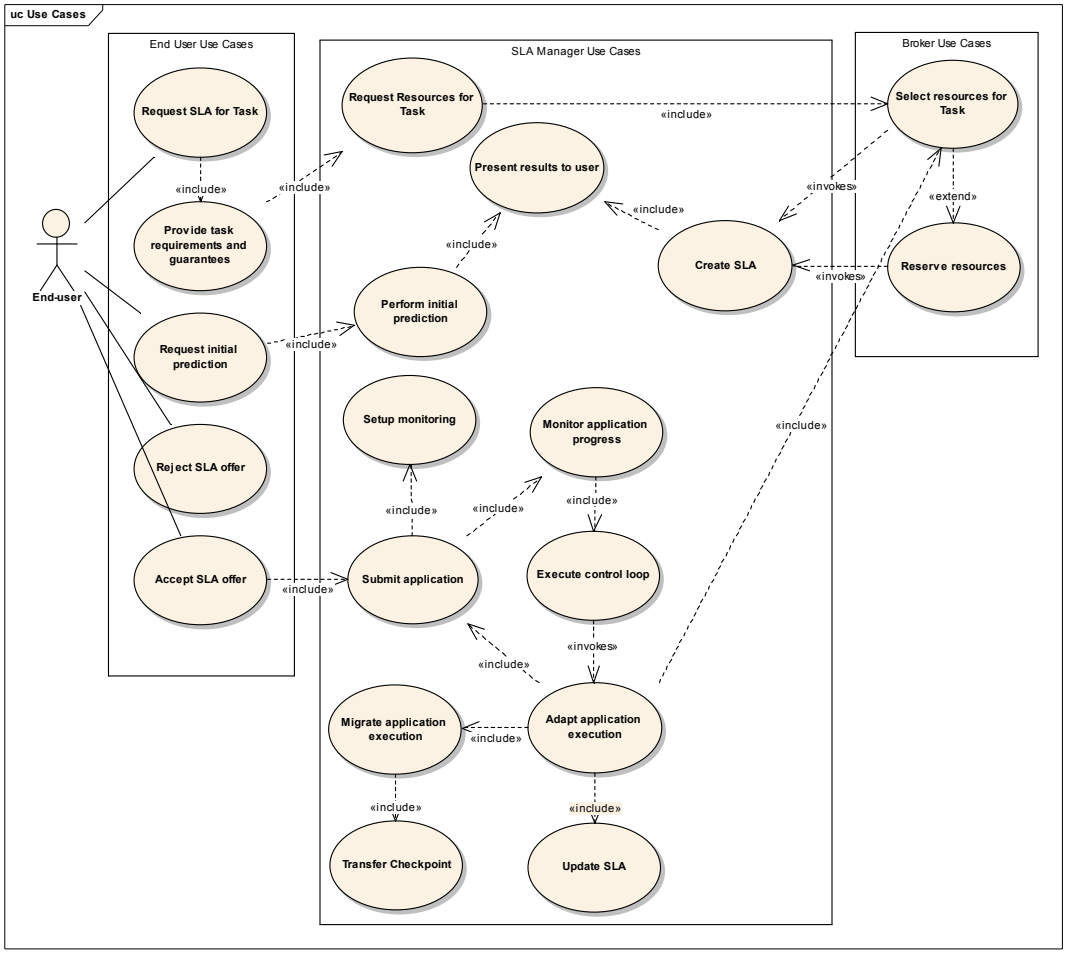

Figure 12 System use case diagram

3.4.1 End-User Use Cases

The end-user use cases describe actions performed by the end-user.

- **Request SLA for task** - The goal of this use case is to send a request to the SLA Manager for the execution of a task representing a run of a Grid application. Most Grid applications are categorised as single or multiple batch submissions, MPI or interactive. The type of tasks targeted by the SLA Management Architecture are single batch applications. They are submitted to cluster based Grid resources using Grid middleware scheduled locally using a network batch queuing system. During the request the end-user will be required to provide task requirements and guarantees.
- **Provide task requirements and guarantees** - The goal is to provide the SLA Manager with a set of task requirements and guarantees. These describe

hardware and software requirements, guarantees and timing constraints. The SLA Manager uses these requirements to create an SLA template and request resources for task.

- **Request initial prediction** - This requests an initial prediction from the SLA Manager for the application execution time. The method of prediction is described in section 5.2.3 and produces an approximation of the execution time based on previous runs of the application. This estimation is used to determine the finishing time of the application whilst it is executing. The SLA Manager performs the initial prediction and presents the results to the user.
- **Reject SLA offer** - The end-user has the option of rejecting the SLA offered by the SLA Manager. If rejected, the end-user has to request another SLA for task.
- **Accept SLA offer** – If the end-user accepts the SLA offered by the SLA Manager, the SLA is approved for execution and the application is submitted. The SLA Manager submits the application which is described by the SLA.

3.4.2 SLA Manager Use Cases

The SLA Manager use cases describe actions which can be taken by the SLA Manager.

- **Request resources for task** – After the end-user has provided the task requirements and guarantees, the SLA Manager must determine if the requirements can be matched to resources. To do this the SLA Manager uses a resource broker such as the Three-phase commit SNAP broker [7] to select resources for task. If resource reservation is needed the SLA Manager can also reserve resources for task using the resource broker.
- **Create SLA** – An instance of a SLA is created which contains the requirements, guarantees and initial prediction according to the specification in section 3.8. This occurs after the SLA Manager has selected resources for the task.
- **Present results to user** – The goal is to allow the end-user to view the result of some internal process; for example the creation of an SLA or the results of an initial prediction.

- **Perform initial prediction** – The goal is to produce an initial prediction for the application execution time. The method of prediction is described in section 5.2.3 and produces an approximation of the execution time based on previous runs of the application.
- **Submit application** – Once the end-user has accepted the SLA, the SLA Manager submits the application in accordance with the SLA.
- **Setup monitoring** – The SLA Manager must be in a position to monitor the applications progress when the Grid application begins execution. The monitoring data describes the performance of the application or the system on which the application is executing SLA Manager.
- **Monitor application execution** – The goal is to monitor changes in performance based on application or system level monitoring data. The SLA Manager executes the control loop in order to determine the best control response for the application given the current state of monitoring. The methods used to monitor application progress are discussed in section 3.3.5.
- **Execute control loop** – The goal is to determine the control response given the Grid applications performance. The SLA Manager implements an adaptive rule based controller described in section 3.3.6. If the Grid application is found to be behind schedule and is predicted to finish after the time deadline, the SLA manager may, depending on the stringency of the rules in the rule base, adapt the application execution.
- **Adapting the application execution** – The SLA Manager will attempt to restart the application on a new resource with the latest checkpoint. If free resources are available with the current Grid resource provider, the application will be migrated to another node. If resources are unavailable with the current Grid resource provider, the application will be migrated to another provider. If this is the case, the SLA Manager must use a Grid resource broker to determine which resources are free by selecting resources for the task. If no resources are free, the Grid application continues to execute on the current resource. Once selected, the SLA Manager must migrate the application execution to a new resource provider and transfer the checkpoint. The SLA Manager is then

responsible for submitting the application to the new resource. The SLA Manager must update the SLA with violations or warnings.

- **Migrate application execution** – The goal is to move the application execution to a new resource. This resource could be with the same resource provider or if no spare capacity is available, resources are selected with a different resource provider.
- **Transfer checkpoint** – The goal is to transfer the latest checkpoint available onto a new resource and restart the application. The applications is assumed to support application level checkpointing. A description of checkpointing techniques is provided in section 2.7.
- **Update SLA** – The goal is to modify those elements of the SLA specification which record warnings, migrations and violations to reflect the changes given the adaptation of the application execution.

3.4.3 Broker Use Cases

The broker use cases describe actions which can be taken by the resource broker.

- **Select resource for task** – The goal is to search for Grid resources which match the task requirements, guarantees and the duration specified by the initial prediction. Once selected the resource broker provides details of the resource onto which the SLA Manager can submit the application.
- **Reserve resources** – The goal is to reserve a node with a given resource provider. This reservation is for a specified period of time and restricts usage so that a users application can start executing at a specified time.

3.5 System Components

Within this section the system components are introduced and assigned responsibilities. In addition, associations show how each component is related to others within the system design. The SLA Manager implementation is illustrated in Figure 13. Each component is described including its functional responsibility and associations.

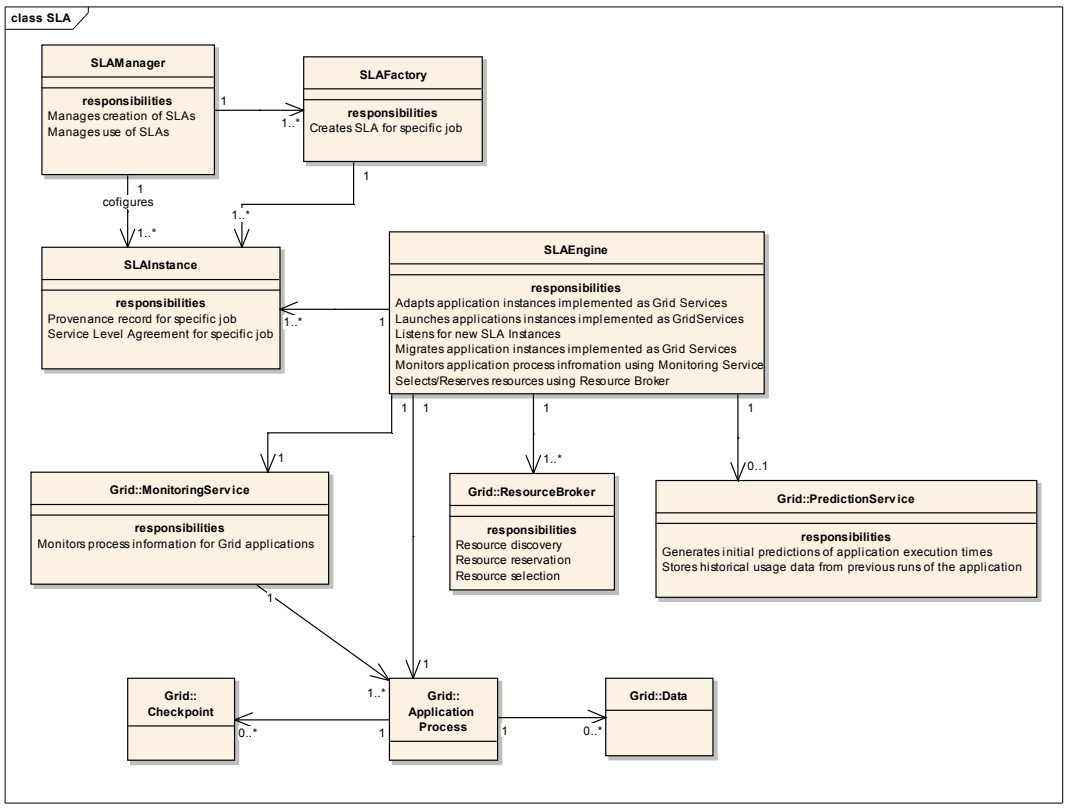

Figure 13 System Implementation Diagram

- **Grid Checkpoint** – saved by the Grid application to record its current state so it can be restarted from that point by the SLA Engine. It is architecture independent so it can be restarted on heterogeneous resource types. Zero or many Grid checkpoints can be associated with one Grid application, recognising that the application may not have checkpointing support, or may not write a checkpoint file until specific points during the execution.

- **Grid Data** - represents the input data (or file(s)) which is submitted to the Grid application. The data has a zero to many association with the Grid application, recognising that a Grid application may not take data (or file(s)) as input.

- **Grid Application** – represents an application instance and the task for which an SLA is created. The Grid application may take Grid Data and process it to achieve results in line with the objectives of the SLA. Periodically, the Grid application may save its current state to one or many Grid Checkpoint files. The application should be deployable onto a number of resource architectures taking account of the heterogeneous makeup of Grid systems.

- **Prediction Service** – uses a historical dataset of previous application runs to predict the execution time of the application given a predictor variable, e.g. the application input parameter.
- **Monitoring Service** – provides monitoring information which reflects the performance of the application. Monitoring can be performed at the system or application level.
- **Resource Broker** – is responsible for resource selection and reservation depending on the task requirements and guarantees.
- **SLA Engine** – is created by the SLA Manager to fulfil a number of SLA management functions. The SLA Engine uses the Prediction Service to generate a prediction for the execution time of the task. Resources needed to meet this prediction are selected and reserved by the SLA Engine using the Resource Broker. After the Grid application is submitted the SLA Engine uses a monitoring service to monitor the Grid application. The SLA Engine's adaptive rule-based controller determines control actions.
- **SLA Factory** – creates new SLA Instances for each task request as they are received from the SLA Manager. Additionally, it is responsible for modifying SLA Instances.
- **SLA Instance** – represents a Service Level Agreement (SLA) for a task. The SLA is specified according to the SLA Specification in Section 3.8. The SLA Instance is updated with actions taken during the application execution.
- **SLA Manager** – is the gateway used by the end-user interface to communicate with the rest of the system. In addition it is responsible for the creation, management and modification of SLA Instances.

3.6 Intended Usage Scenarios

Usage scenarios illustrate component interactions and message flows between major components within the architecture and depict the main success scenario through a use case. The main success scenario is the path through the use case which results in the achievement of the goal. If actions do not result as planned, progression through the use

cases can diverge from the main success scenario and follow alternative paths, for example due to an exception or a faulty condition.

3.6.1 Request Initial Prediction

The intended usage scenario, request initial prediction, is illustrated in Figure 14 together with the intended usage scenarios, perform initial prediction and present results to end-user. The end-user requests an initial prediction using the end-user interface. The end-user interface contacts the SLA Manager to request the launch of the prediction service. The prediction service is launched in a new thread which the end-user can interact with directly. The end-user must select the historical dataset on which the prediction is to be performed. Once selected the predictor variable is used to compute an initial prediction. This is displayed in the initial prediction GUI. The scenario ends after the end-user has viewed the initial prediction and the initial prediction GUI is closed.

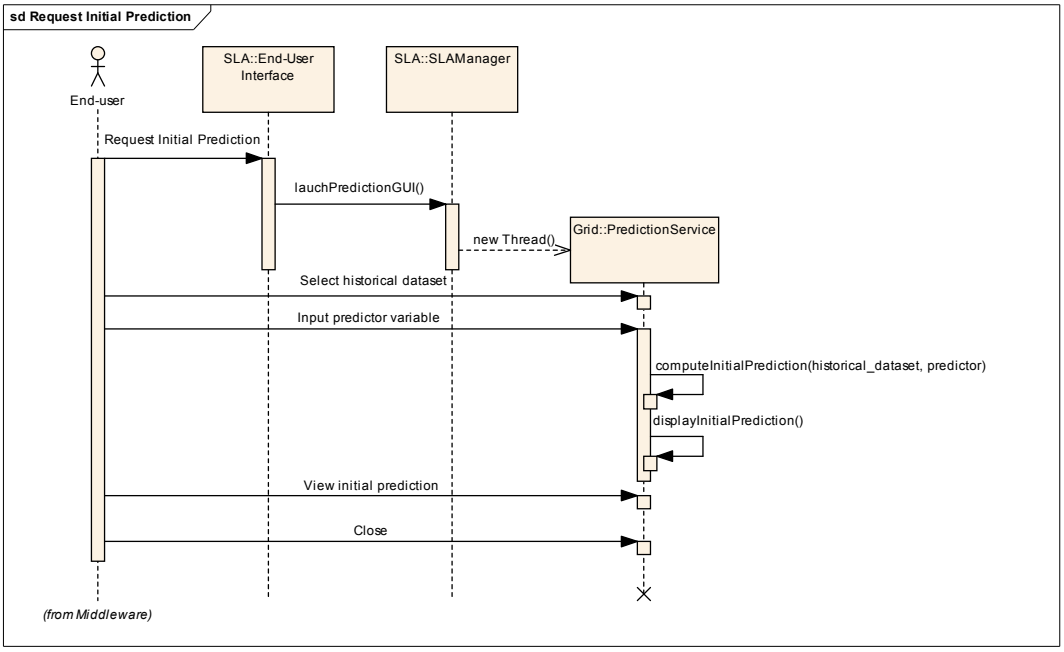

Figure 14 Request Initial Prediction Intended Usage Scenario

3.6.2 Request SLA for task

The intended usage scenario, request SLA for task, is illustrated in Figure 15 together with intended usage scenarios provide requirements and guarantees, request resource for task, select resources for task, reserve resources for task, create SLA and present results to user. The end-user requests an SLA for task using the end-user interface. The end-

user interface contacts the SLA Manager to request an SLA for the task given the requirements, guarantees and prediction. The SLA Manager creates a new SLA Engine instance which contacts the Resource broker to select a resource which matches the requirements and the duration of the prediction. If reservation is part of the requirements, the resources are reserved. The resource broker returns the resource details to the SLA Manager. The SLA Manager creates a new SLA Factory instance which is used to create an SLA instance given the requirements, guarantees, initial prediction and the resource details. The SLA Instance is returned to the SLA Manager and returned to the end-user interface as an SLA offer. The end-user can view the SLA offer using the end-user interface.

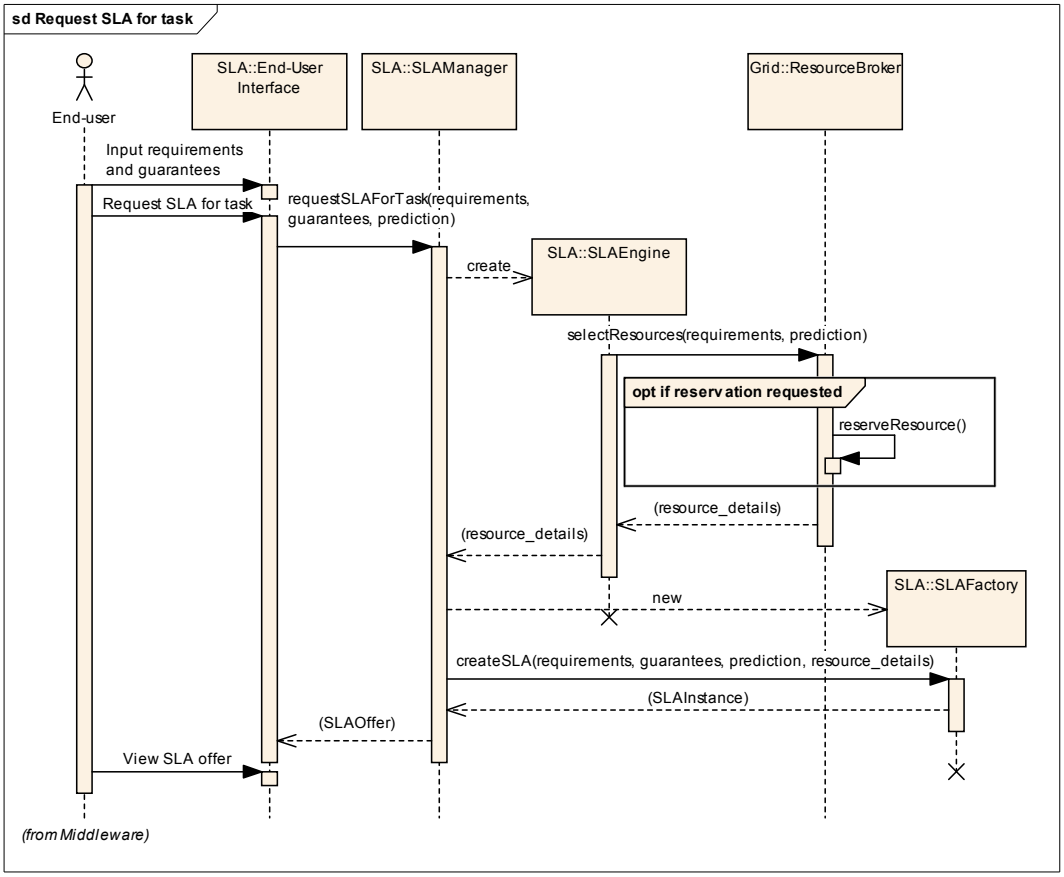

Figure 15 Request SLA for Task

3.6.3 Accept SLA

The intended usage scenario, accept SLA, is illustrated in Figure 16 together with intended usage scenarios submit application and setup monitoring. The end-user accepts

the SLA offer through the end-user interface. The end-user interface contacts the SLA Manager to submit the application according to the terms in the SLA. The application is submitted by the SLA Engine. The submission of the application occurs through a Gatekeeper which is resident on the selected resource. The Gatekeeper creates a job manager which submits the application to the NBQS. The submission of the monitoring script is handled by a separate job manager which executes using the Fork scheduler. The monitoring script gets the CPU time from the application process. This action continues to loop until the application process is killed.

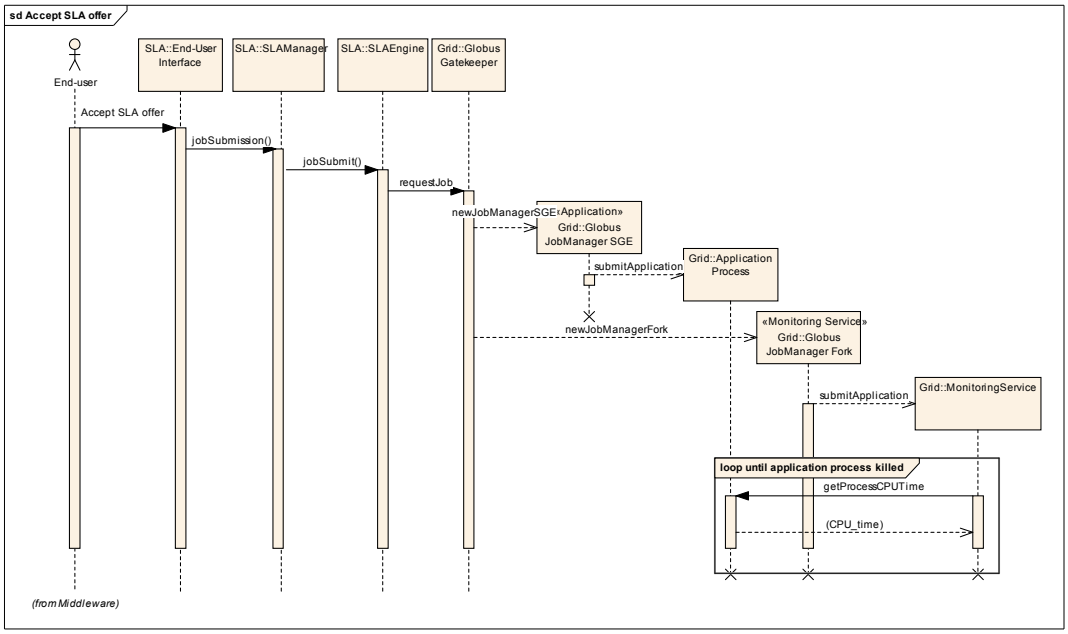

Figure 16 Accept SLA Intended Usage Scenario

3.6.4 Monitor application progress

The intended usage scenario, monitor application progress, is illustrated in Figure 16 together with intended usage scenarios execute control loop, adapt application execution and migrate application execution. The scenario represents the condition when spare resources are available with the current provider. This condition is determined by the job status after a migration script has been executed. The SLA Engine monitors the CPU time of the Grid application by accessing standard output from the job manager assigned to the monitoring script. The SLA Engine uses the CPU time within the execute control loop method to determine if a control action is needed. If migration is needed the SLA Engine sends a migration script to the Grid resource through the Gatekeeper. The

Gatekeeper creates a job manager which executes the migration script using the Fork scheduler. The script suspends the SGE queue on which the application is executing. This triggers SGE to send the Grid application back into the jobs pending queue. If a resource is available, the Grid application will be resumed. The SLA Engine can determine if this has occurred by querying its status using the job manager assigned to the Grid application. If the status remains pending the SLA Engine has to find a free resource with another provider. Assuming a resource is available with the current provider, the SLA Engine continues to monitor the Grid application and execute the control loop until the job manager signals a done signal for the Grid application. Once the job status is done the SLA Engine presents the results to the end-user interface. The SLA instance is updated with provenance from the execution. This is done via the SLA Manager which creates a new instance of an SLA Factory to update the SLA and return the results to the SLA Manager.

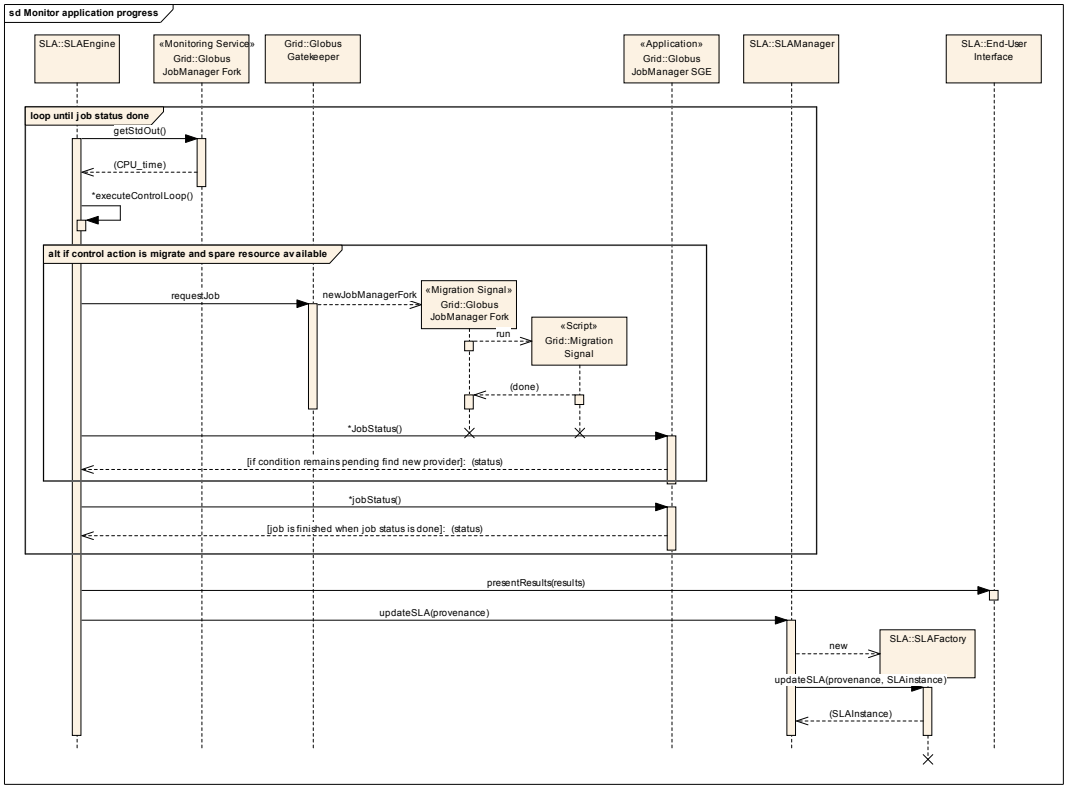

Figure 17 Monitor Application Progress – Single Provider

If the status of the Grid application remains pending after the migration script has been executed, the assumption is that no resources are available at the current provider. In this situation the SLA Engine must contact the resource broker to select a new resource

for task and is depicted in a section of the request SLA for task scenario in Figure 15. Once resource selection is complete, the SLA Engine transfers the checkpoint file to the new Grid resource and restarts the Grid application there. This scenario is illustrated in a section of the accept SLA scenario in Figure 16. A new monitoring script is submitted to monitor the migrated Grid application. The SLA Engine resumes monitoring of the new Grid application taking account of the CPU time received on the previous resource. When the Grid application has completed the job manager status will return. The results are presented to the end-user through the end-user interface and the SLA is updated.

3.7 Implementation

The implementation of the SLA Management Architecture raised a number of issues related to the setup of the Globus Toolkit and SGE 5.3/6.0. The applications targeted by the SLA Management Architecture are executed on HPC resources using the Globus Toolkit 2.4 and SGE 5.3 (or 6.0). In order to support application level checkpointing in Globus 2.4 and SGE 5.3 / 6.0, a number of modifications are needed to both pieces of software.

3.7.1 Updating the Globus Toolkit

Globus Toolkit 2.4 supports the submission of applications as batch jobs to SGE through a GRAM job manager. It supports some basic command line flags for *qsub*, the SGE job submission command. However, it does not allow the specification of SGE checkpointing interface. In addition, there is no support for a job *name* to be passed to SGE in order that the application can be easily identified within the queue. This also helps in the monitoring technique to identify the process assigned to the application. The following changes were needed to the Globus configuration in order to support the features above.

$GLOBUS_LOCATION/share/globus_gram_job_manager/sge.rvf

Figure 18 shows the code which needs adding to schema which regulates what can be specified in the Globus RSL job script.

```
Attribute: name
Description: "Name of the SGE submission; will show in qstat"
ValidWhen: GLOBUS_GRAM_JOB_SUBMIT
Attribute: checkpointenv
Description: "SGE Checkpointing interface associated with job"
ValidWhen: GLOBUS_GRAM_JOB_SUBMIT
```

Figure 18 Changes to Globus RSL Schema

$GLOBUS_LOCATION/lib/perl/Globus/GRAM/JobManager/sge.pm

Figure 19 shows the code which needs to be added to the Globus SGE submission script builder. It writes the –ckpt and –N flags to the SGE submission script. The code takes arguments which are passed by the SLA Manager in the RSL script, (the checkpointing interface and job name) and adds them to the SGE submission script. Without this mechanism, user level checkpointing could not be supported in SGE 5.3/6.0 through Globus Toolkit 2.4.

```
#line 194#
#if checkpointenv rsl attribute exists print -ckpt $checkpointenv
to submission script
if($description->checkpointenv() ne '')
{
$sge_job_script->print("#\$ -ckpt ". $description->checkpointenv()
."\n");
}

#if name rsl attribute exists print -N $name to submission script
if($description->checkpointenv() ne '')
{
$sge_job_script->print("#\$ -N ". $description->name() ."\n");
}
```

Figure 19 Changes to Globus SGE script builder

3.7.2 Extending support in SGE.

SGE 5.3 / 6.0 supports user level checkpointing which will automatically restart applications if they are migrated onto spare resource with the same resource provider. Checkpoints are created by the application; SGE sends no external trigger to create the checkpoint file. This setup is advantageous because SGE will restart the application and write any application logging messages to standard out. This functionality is enabled by the SGE checkpointing interface construct. This specifies where the checkpoints should be written; and allows the application to be restarted after it or the queue in which it is executed is suspended. It also allows for signals or messages from the application to be communicated through SGE to a log file which can be communicated back to stdout or

stderr. This can in turn be communicated back through Globus to the SLA Manager. An SGE checkpointing interface is shown in Figure 20.

```
ckpt name          check userdefined
interface          userdefined
ckpt_command       none
migr_command       none
restart_command    none
clean_command      none
ckpt dir           /home/checkpoint
queue list         all
signal             none
when               xr
```

Figure 20 SGE Checkpointing Interface Construct

3.7.3 Application Migration

Two types of application migration are considered within the SLA Management Architecture. Which is triggered depends on the availability of spare capacity with the current resource provider. If spare capacity is available, the migration is triggered by the SLA Engine and handled by the SGE system. The application is migrated onto another node and restarted with the most recent checkpoint. If no spare capacity is available, the migration is triggered and handled by the SLA Engine. A new provider has to be identified with spare capacity matching the requirements. The checkpoint is transferred to the new provider and the application and monitoring is restarted in a similar way to the original submission.

3.8 SLA Specification

The proposed SLA specification is defined in XML Schema Definition (XSD) [122] making it machine readable and can be updated to reflect actions taken during application management. The components of the specification are inspired by the work of Sahai et al [82], but define in greater detail a job submission description [123] which includes task requirements and guarantees for the Grid application. Support for SLA provenance is a major addition which enhances non repudiation mechanisms by improving agreement traceability and validation. Elements are included in the specification which allow actions taken during application management; violations, migrations or warnings, to be recorded within the SLA document. When the SLA is completed and returned to the end user, the addition of these elements allow the end-

user to trace through the actions which were taken during the execution of their Grid application task. For some commercial customers SLA traceability is a requirement; this is the case for DAME, where the commercial partner wants to know the resources on which the application is executed because the results are the subject of IP restrictions and any breaches have to be traced. In addition, because the results of the Grid application effect health and safety on commercial aircraft, if the application is executed and the results are inaccurate, the resource needs to be traced so that further errors do not occur.

SLA provenance helps to validate the actions taken and their effect on the management of the application. If an SLA is violated, provenance is useful for resolving disputes between provider and customer. If it is found that failure resulted directly from application management actions, provenance can strengthen the non repudiation between the parties. Equally, if the customer and the provider agree that a specific action should always occur if a specific state is reached, SLA provenance helps to validate if these actions were in fact taken.

The specification is motivated by the lack of a single specification within the Grid domain, which can satisfy the requirements of adaptive SLA management for Grid based systems such as DAME. The job submission description elements of the specification are motivated by the Job Submission Description Language (JSDL) [67] which provides a detailed job specification for Grid jobs. The description of parties, purpose and scope is influenced by the WS-Agreement [69] specification. The Service Level Objective (SLO) elements are motivated by the Service Definition element from within the Web Service Level Agreement (WSLA) [68] specification. The provenance record is motivated by the Usage Record (UR) [70] specification.

Job submission elements include a *job description* which identifies the job, user and resource requirements; plus staging out requirements. The SLOs represent the active guarantees within the SLA and are quantified by a corresponding Service Level Indicator (SLI). Elements describing the parties involved in the agreement may support a listing of users or providers.

Provenance elements record the control actions taken during application management. Warnings, migrations and violations are recorded within the SLA to build a historical record of application management for the Grid application task.

Table 1 SLA Specification for a compute service

Component	Observation
Purpose	An application with requirements
Parties	Consumer / provider
Scope	Compute service, data staging
Service Level Objective (SLO)	Ensure job, user and resource requirements are satisfied for the duration of the SLA
SLO Attributes	Time or performance constraints
Service Level Indicators (SLI)	For each SLO attribute, its value is indicated by an SLI
Exclusions	Adaptation / reservation may not be included
Administration	SLOs met through resource brokering / adaptation

The SLO parameter represents a qualitative guarantee such as a time or performance constraint. Time constraints are specified by an acceptable period in which the application must complete. It is a function of the resource speed, the prediction and the strictness of the guarantee.

Performance constraints may be expressed as a desired level of performance in such metrics as system load average and amount of memory (RAM). The SLI parameter represents the quantitative level of guarantee for the SLO. An SLI value may take a number of forms: an upper or lower bound or a mean value to be maintained for the duration of the application.

3.8.1 Overview

The *sla* element (Figure 21) encapsulates a single Service Level Agreement. All specific SLA elements extend from this element. Any structure that contains SLA information should reference the *sla* element so that extensions or restrictions of the element are automatically handled. This element should contain all the information that is generic to an SLA and addressed by this specification.

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:jaxb="http://java.sun.com/xml/ns/jaxb"
jaxb:version="1.0">
<xs:element name="sla" type="slaType"/>
```

Figure 21 SLA Element

3.8.1.1 slaType

SlaType (Figure 22) is a complex type which uniquely identifies the SLA associated with a task. *Parties* represents a global identity of type *partiesType*. There is a requirement for at least one party to be described in the SLA. At least one set of CPU resource requirements can be described by *cpuType*. At least one set of memory resource requirements can be described by *ramType*. At least one set of storage requirements can be described by *hddType*. At least one set of operating system requirements can be described by *osType*. *Completion* describes the timing constraints for the task of type *completionType*. At least one set of timing constraints can be described in the SLA. The attribute *slaID* uniquely identifies the SLA. The attribute *jobName* uniquely identifies the task.

```
<xs:complexType name="slaType">
     <xs:sequence>
           <xs:element name="parties" type="partiesType"
minOccurs="1" maxOccurs="unbounded"/>
           <xs:element name="cpu" type="cpuType"
minOccurs="1" maxOccurs="unbounded"/>
           <xs:element name="ram" type="ramType"
minOccurs="1" maxOccurs="unbounded"/>
           <xs:element name="hdd" type="hddType"
minOccurs="1" maxOccurs="unbounded"/>
           <xs:element name="os" type="osType"
minOccurs="1" maxOccurs="unbounded"/>
           <xs:element name="completion"
type="completionType" minOccurs="1"
maxOccurs="unbounded"/>
     </xs:sequence>
<xs:attribute name="slaID" type="xs:int"
<xs:attribute name="jobName" type="xs:string"
use="required"/>
</xs:complexType>
```

Figure 22 SLA Type

3.8.1.2 partiesType

PartiesType (Figure 23) is a complexType which uniquely identifies the parties involved in an SLA. At least one consumer can be represented within type

consumerType, which represents a global identity for the consumer. At least one provider can be represented within type *providerType*, which represents a global identity for the resourse provider.

```
<xs:complexType name="partiesType">
     <xs:sequence>
          <xs:element name="consumer"
type="consumerType" minOccurs="1"
maxOccurs="unbounded"/>
          <xs:element name="provider"
type="providerType" minOccurs="1"
maxOccurs="unbounded"/>
     </xs:sequence>
</xs:complexType>
```

Figure 23 Parties Type

3.8.1.3 consumerType

ConsumerType (Figure 24) is a complexType which uniquely identifies the consumer side of the parties involved in the SLA. *Name* provides the global identity of consumer. *Address* describes the physical location of the consumer. The *Signed* element is a mutable element representing the acknowledgement of the consumer that the SLA is binding.

```
<xs:complexType name="consumerType">
     <xs:sequence>
          <xs:element name="name" type="xs:string"/>
          <xs:element name="address" type="xs:string"/>
          <xs:element name="signed" type="xs:boolean"/>
     </xs:sequence>
</xs:complexType>
```

Figure 24 Consumer Type

3.8.1.4 providerType

ProviderType (Figure 25) is a complexType which uniquely identifies the provider side of the parties involved in an SLA. *Name* provides the global identity of provider. *Address* describes the physical location of the provider. The *Signed* element is a mutable element representing the acknowledgement of the provider that the SLA is binding.

```
<xs:complexType name="providerType">
     <xs:sequence>
           <xs:element name="name" type="xs:string"/>
           <xs:element name="address" type="xs:string"/>
           <xs:element name="signed" type="xs:boolean"/>
     </xs:sequence>
</xs:complexType>
```

Figure 25 Provider Type

3.8.1.5 cpuType

CpuType (Figure 26) is a complexType which describes the CPU resource requirements for the application/task. *Version* describes the architecture of CPU required. *Count* describes the number of CPU's required. *Speed* describes the speed of the CPU/s required. *Normalised_load* describes a normalised measure of the system load. The requirement for this element is optional in the SLA. The *warnings* element is a mutable element of type *warningType* which identifies the warnings sent by the SLA Engine. The requirement for this element is optional in the SLA as it is set dynamically during task execution. The *violations* element is a mutable element of type *violationType* which identifies the violations sent by the SLA Engine. The requirement for this element is optional in the SLA as it is set dynamically during task execution.

```
<xs:complexType name="cpuType">
     <xs:sequence>
           <xs:element name="version" type="xs:string"/>
           <xs:element name="count" type="xs:int"/>
           <xs:element name="speed" type="xs:int"/>
           <xs:element name="normalised load"
type="xs:int" minOccurs="0" maxOccurs="unbounded"/>
           <xs:element name="warnings"
type="warningType" minOccurs="0" maxOccurs="unbounded"/>
           <xs:element name="violations"
type="violationType" minOccurs="0"
maxOccurs="unbounded"/>
     </xs:sequence>
</xs:complexType>
```

Figure 26 CPU Type

3.8.1.6 ramType

RamType (Figure 27) is a complexType which describes the memory requirements for the task. *Count* describes the amount of memory required for the task. The *warnings* element is a mutable element of type *warningType* which identifies the warnings sent by the SLA Engine. The requirement for this element is optional in the SLA as it is set

dynamically during task execution. The *violations* element is a mutable element of type *violationType* which identifies the violations sent by the SLA Engine. The requirement for this element is optional in the SLA as it is set dynamically during task execution.

```
<xs:complexType name="ramType">
     <xs:sequence>
           <xs:element name="count" type="xs:int"/>
           <xs:element name="warnings"
type="warningType" minOccurs="0" maxOccurs="unbounded"/>
           <xs:element name="violations"
type="violationType" minOccurs="0"
maxOccurs="unbounded"/>
     </xs:sequence>
</xs:complexType>
```

Figure 27 RAM Type

3.8.1.7 hddType

HddType (Figure 28) is a complexType which describes the storage requirements for the task. *Count* describes the amount of storage required for the task. A description of the *warning* and *violation* elements within the *hddType* perform a similar role to those within the *ramType* (Section 3.8.1.6).

```
<xs:complexType name="hddType">
     <xs:sequence>
           <xs:element name="count" type="xs:int"/>
           <xs:element name="warnings"
type="warningType" minOccurs="0" maxOccurs="unbounded"/>
           <xs:element name="violations"
type="violationType" minOccurs="0"
maxOccurs="unbounded"/>
     </xs:sequence>
</xs:complexType>
```

Figure 28 HDD Type

3.8.1.8 osType

OsType (Figure 29) is a complexType which describes the operating system requirements for the task. *Type* describes the operating system required for the task. *Version* describes the operating system version required for the task. *Directory* describes the location of any data staging requirements required for the task.

```
<xs:complexType name="osType">
     <xs:sequence>
           <xs:element name="type" type="xs:string"/>
           <xs:element name="version" type="xs:string"/>
           <xs:element name="directory"
type="xs:string"/>
     </xs:sequence>
</xs:complexType>
```

Figure 29 OS Type

3.8.1.9 completionType

CompletionType (Figure 30) is a complexType which describes the timing guarantees for the task. *Time* describes the timing constraints for the task. The requirement for this element is optional in the SLA. The *warning* and *violation* elements within the *completionType* perform a similar role to those within the *ramType* (Section 3.8.1.6). The *migrations* element is a mutable element of type *migrationType* which identifies the migrations performed by the SLA Engine.

```
<xs:complexType name="completionType">
     <xs:sequence>
           <xs:element name="time" type="xs:int"
minOccurs="0" maxOccurs="unbounded"//>
           <xs:element name="warnings"
type="warningType" minOccurs="0" maxOccurs="unbounded"/>
           <xs:element name="violations"
type="violationType" minOccurs="0"
maxOccurs="unbounded"/>
           <xs:element name="migrations"
type="migrationType" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>
```

Figure 30 Completion Type

3.8.1.10 violationType

ViolationType (Figure 31) is a complexType which describes a violation recorded by the SLA Engine. *TimeStamp* describes the timing of a violation recorded by the SLA Engine. There is a requirement for at least one *timeStamp* within the *violation* element. *Value* quantifies the violation recorded by the SLA Engine. The requirement for *value* is optional.

```
<xs:complexType name="violationType">
     <xs:sequence>
          <xs:element name="timeStamp"
type="xs:dateTime" minOccurs="1" maxOccurs="unbounded"/>
          <xs:element name="value" type="xs:string"
minOccurs="0" maxOccurs="unbounded"/>
     </xs:sequence>
</xs:complexType>
```

Figure 31 Violation Type

3.8.1.11 warningType

WarningType (Figure 32) is a complexType which describes a warning recorded by the SLA Engine. *TimeStamp* describes the timing of a warning recorded by the SLA Engine. There is a requirement for at least one *timeStamp* within the *warning* element. *Value* quantifies the warning recorded by the SLA Engine. The requirement for *value* is optional.

```
<xs:complexType name="warningType">
     <xs:sequence>
          <xs:element name="timeStamp"
type="xs:dateTime" minOccurs="1" maxOccurs="unbounded"/>
          <xs:element name="value" type="xs:string"
minOccurs="0" maxOccurs="unbounded"/>
     </xs:sequence>
</xs:complexType>
```

Figure 32 Warning Type

3.8.1.12 migrationType

MigrationType (Figure 33) is a complexType which describes a migration signal by the SLA Engine. *TimeStamp* describes the timing of an application migration recorded by the SLA Engine. There is a requirement for at least one *timeStamp* within the *migration* element. *Resource* is the host onto which the application was migrated. There is a requirement for at least one *resource* within the *migration* element.

```
<xs:complexType name="migrationType">
     <xs:sequence>
          <xs:element name="timeStamp"
type="xs:dateTime" minOccurs="1" maxOccurs="unbounded"/>
          <xs:element name="resource" type="xs:string"
minOccurs="1" maxOccurs="unbounded"/>
     </xs:sequence>
</xs:complexType>
```

Figure 33 Migration Type

3.9 Summary

The DAME showcase scenario is used to demonstrate how adaptive SLA Management can be introduced into the DAME business process to enhance application management and contract non repudiation mechanisms. The SLA management architecture is introduced and describes the functional requirement which are necessary to support adaptive SLA management within Grid based systems. An application model identifies what is required of the application in order for SLA management to be achievable. Use cases describe system usage and the behaviour of each entity; end-user, SLA manager and resource broker. System components are identified and responsibilities assigned. Implementation issues are identified and an SLA specification discussed.

In Chapter 4, the SLA Manager is tested on a local Grid test-bed using three scenarios. The first tries to violate performance guarantees by manipulating the level of free system memory and the system load average on the resource executing the Grid application. The second and third scenarios demonstrate the benefit provided by the SLA Manager with application adaptation in order to prevent an SLA violation of a timing constraint.

Chapter 4

Prototype testing on a local Grid Test-bed

This chapter contributes to deliverable [C7], by providing a performance comparison of a prototype implementation of the deliverables [C1]-[C2] and [C6] on a local Grid test-bed. The experiment involves the specification of performance and timing guarantees within an SLA and their validation using monitoring data from a Grid resource. Section 4.1 provides an overview of the experimental objectives whilst Section 4.2 describes the experimental design and Sections 4.3 - 4.5 present the results of the experimental scenarios.

4.1 Overview

This experiment is performed on a local Grid test-bed, which is a deployment of 10 computational Grid resources, each configured with a single Intel 32 bit CPU and 256MB of memory. Each resource on the local Grid test-bed is named testgrid*x*.leeds.ac.uk, where *x* is the number of the resource. The operating system is Linux with kernel 2.6. The middleware software is Globus Toolkit 2.4 and the network batch queuing system is Sun Grid Engine 5.3p6 (SGE). Each machine has MDS2 deployed, providing up-to-date resource information via the LDAP [124] protocol. Communication between resources is over a Fast Ethernet 100Mbps LAN network. The local Grid test-bed is a development platform for software testing and debugging prior to deployment onto a large distributed Grid infrastructure such as the WRG. Deploying software in this manner allows for debugging to be undertaken without impacting other Grid and local HPC users; which is more likely on the WRG, with its greater density of

users. However, deployment onto the local Grid test-bed does not expose the software to issues such as site autonomy, heterogeneous substrate and policy extensibility; therefore testing will still need to take place on the WRG.

4.2 Experimental Design

The SLA Manager can specify an SLA for an application, submit the application, monitor the SLA guarantees and signal control actions using the adaptive rule-based controller. The SLA-bound application used is CPU intensive and includes checkpointing support. In this experiment, a learning based technique is not used to generate an initial prediction of application's execution time. Instead, the application described in section 4.2.1 is executed a number of times in order to generate an initial prediction average. The purpose of the experiments in this chapter is to test the SLA Manager, the monitoring technique and the adaptive rule based controller; the initial prediction technique will be introduced in the next experiment. During each experiment competing applications are introduced onto the resource executing the SLA-bound application to degrade its performance. These are other instances of the application described in Figure 34.

Three experimental scenarios are considered:

1. An SLA is created specifying a performance guarantee for a minimum value of the normalised measure of system load described in Section 4.2.3.

2. An SLA is created which specifies a duration based timing guarantee by which the SLA-bound application must complete. The SLA-bound application is executed using best-effort provision which indicates that application migration will not take place.

3. An SLA is specified for a duration based timing guarantee by which the SLA-bound application must complete. In this scenario the SLA-bound application is executed using adaptation which indicates that application migration will be used when signalled by the adaptive controller.

The first scenario tests the SLA Manager's ability to monitor resource information given a performance guarantee. It also tests the adaptive controller's ability to react to this

information and record violations within the SLA. The experiment begins with the creation of an SLA specifying a performance guarantee for a minimum value of the normalised measure of system load (Section 4.2.3) on the resource executing the Grid application. Once the application is submitted, the SLA Manager uses the MDS to monitor the resource. This experiment uses system level monitoring as a method of detecting performance changes to a Grid application. MDS can be configured to monitor the system load average, which can be used to calculate a normalised measure of system load and therefore meets the requirements for use in this experiment; more details of the method are given in section 4.2.2. During execution the adaptive rule-based controller is configured to signal an update in the SLA if the monitored values for the normalised measure of system load falls below the minimum value specified in the SLA. The SLA used for scenario 1 is presented in Section 4.3.

The second and third scenarios test the SLA Manager's ability to monitor resource information given a timing constraint based on an initial prediction of the application's execution time. The adaptive controller is being tested to determine if it reacts to predicted violations in the timing constraint as the normalised measure of system load reduces and the resources ability to execute the application changes. The ability of the SLA Manager to record these events within the SLA is also being tested. The experiment begins with the creation of an SLA specifying a timing constraint for the expected duration of the SLA-bound application. The value is determined by taking an average execution time from the observed values of a number of previous application runs. Once the application is submitted, the SLA Manager uses the MDS to monitor the system load average on the resource executing the application. The variables used in the control loop are the scheduled remaining execution time (specified in the SLA) and the predicted remaining execution time (details are given in section 4.2.3). Resource monitoring is needed to support the adaptive rule-based controller.

Scenario 1 requires that resources are monitored in order that performance guarantees can be validated; the technique used is described in section 4.2.1. In scenarios 2 and 3 timing guarantees are validated using, a method specified in section 4.2.3 to determine the remaining execution time of the application during runtime. Scenario 3 highlights the benefit of the SLA Manager over best effort application execution which is demonstrated in scenario 2.

4.2.1 Grid Application

The application used for these experiments uses a brute force mechanism to approximate the value of π to a set number of decimal places. Figure 34 illustrates the snippet of application code responsible for approximating π; it also shows the code and method for checkpoint creation. Varying the value of r, results in a more accurate approximation of π and a more computationally intensive run. The calculation repeats 1000 times within a for loop. A checkpoint is created every 50 iterations and saves the current position within the loop so that it can be restarted from that position. A full code listing is available in Appendix A.

```
    for ( ; computing.position < 1000; computing.position++)
    {
double r=100000000000, x=1, y=0, s=0, a=0, p=0;

        while(x<=r)
                {
                        y=sqrt(r*r- x*x);
                        s=s+y;
                        x=x+1;
                }
        a=(s*4);
        p=a/(r*r);
        computing.values[computing.position] = p;

        if (checkpoint_enabled && ! just_started)
            if (computing.position % 50 == 0)
                checkpoint_creation();
        just_started=FALSE;
    }
```

Figure 34 Snippet of application code responsible for the approximation of π

The application is submitted by the SLA Manager through the Globus middleware using the JAVA Cog-kit API [46]. RSL is used to pass job submission information which is used to configure the execution environment (Globus and SGE). Figure 35 illustrates the RSL used to submit the SLA-bound application to a resource on the local Grid test-bed.

```
"&" + "(executable=$(GLOBUS_LOCATION)/bin/globus-sh-exec)"
    + "(arguments=$(HOME)/experiments/pi/check_userdefined4.sh)"
    + "(jobtype=single)"
    + "(count=1)"
    + "(name=GlobusTest)"
    + "(checkpointenv=check_userdefined)";
    + "(stdout=/tmp/checkpoint/james.out)"
    + "(stderr=/tmp/checkpoint/james.err)";
```

Figure 35 RSL code for submission of the SLA-bound application

4.2.2 Resource Monitoring

To demonstrate automated monitoring, the SLA Manager will use the Globus Monitoring and Discovery Service (MDS) [23] to monitor resource information for the system load average.

4.2.2.1 Load Calculations

MDS is configured to measure the system load average measured by the SGE scheduler for the resources used during the experiments. At idle, a resource will have a load number of 0, with each competing process adding to this by 1. For the experiments presented in this thesis these processes include the SLA-bound application and all competing applications. The load number is the instantaneous measure of load recorded at the point of observation. Load averages are calculated using a moving average of the load number for the period over which the average is valid for a fixed duration from the point of observation and typically include 1, 5 and 15 minute averages. A 1 minute load average of 2 on a single CPU is interpreted as the CPU being overloaded by 200% during the last minute. If an application were executing in these conditions it could expect to receive a 50% share of the CPU compared with the same application executing under a system load of 1.

4.2.2.2 Motivation for using MDS

The advantage of using MDS for resource monitoring is that it is packaged with the Globus Toolkit and can be easily configured to provide monitoring data at the resource level. For the experiments presented in Chapter 5, the MDS is unsuitable as a monitoring tool because it cannot easily be configured to provide application process information; for that task a monitoring script is used (section 5.2.2). The technique used for this experiment will indicate when the system load average is high and therefore when the performance of the SLA-bound application is likely to suffer because of increased competition for time on the CPU.

The MDS can be configured to operate as a GRIS (Grid Resource Information Service) or a GIIS (Grid Index Information Service). As a GRIS, the MDS provides monitoring information from only the resource on which it is deployed. As a GIIS, the MDS

provides monitoring information from itself and a hierarchy of GRIS from other resources within the Grid system. The GIIS maintains a global view of the Grid system and can access monitoring information from each resource. In order to do this it must contact the GRIS on the resource it wants to monitor, which adds to the measurement overhead and data transmission latency. If a GRIS is contacted directly there is a small measurement overhead and low data transmission latency because the request doesn't have to go through a GIIS. In order to minimise the measurement overhead and data transmission latency a GRIS MDS will be used to monitor the state of the system load average.

As a GRIS, the MDS has cached and non-cached modes of operation. In non-cached mode, MDS does not retain resource information and has to trigger a local resource monitoring process each time the MDS receives a query request. This provides accurate up-to-date resource information, but has the disadvantage of a large measurement overhead and high latency data transmission. The implication is that the measurement process itself may contribute to a slowdown in resource performance and therefore the SLA-bound application and may itself trigger a migration signal. In addition, the high latency data transmission may cause the SLA Manager problems if the measurement query is not answered in a timely manner. When operating in cached mode, MDS retains a store of resource information which is updated only periodically. Query requests are served cached resource information; which results in minimal measurement overhead and low latency data transmission, but has the disadvantage that the information may be stale. The implication is that resource information on which the SLA Manager makes control decisions may be out-of-date and a performance slowdown of the SLA-bound application may go undetected. In order to have access to up-to-date resource information, the MDS is run in non-cached mode, but the query intervals are set to periods which do not increase the measurement overhead on the resource.

4.2.3 Specifying the remaining execution time control variable

To predict the application's remaining execution time whilst it is executing, equation 1 is used. A similar technique is used by Othman et al [118] to predict the remaining execution time of an application when combined with a user estimated completion time.

A similar technique is applied in this experiment to provide a prediction of the remaining execution time of the application based on a normalised measure of the system load average calculated by the SGE scheduler and measured using MDS. This allows an assessment of the remaining execution time which can then be compared with a scheduled remaining execution time. This technique is used only as a means to specify a control variable with which to compare to the scheduled remaining execution time. It should be noted that control variable selection is not limited to the ones chosen in this experiment. They are selected only to demonstrate the adaptive rule-based controller and its ability to execute control actions given the runtime state determined through effective resource monitoring.

A prediction estimate of the application's remaining execution time, $T_{remaining}$, can be made whilst it is executing by using Equation 1. $T_{initial}$ is the wall time needed by the application in order to complete from start to finish in the presence of a mean system load average of $\overline{L_{initial}}$ (in practise in the experiments $\overline{L_{initial}} = 1$, i.e no competing applications).

$$T_{remaining} = \frac{T_{initial} - \sum_{i=1}^{n} N_i T_i}{N_{estimate}}$$

Equation 1 Estimating the application's remaining execution time using a normalised measure of system load

Whilst the application is executing, for each monitoring period from t_0 to n, the normalised measure of system load N_i is calculated using the current system load average L_i (determined by SGE and measured using MDS) and the mean system load average $\overline{L_{initial}}$ observed over the runs to determine $T_{initial}$. N_i is determined using Equation 2.

$$N_i = \frac{\overline{L_{initial}}}{L_i}$$

Equation 2 Calculating the normalised measure of the system load

$\sum_{i=1}^{n} N_i T_i$ indicates the share of wall time the application has received during n monitoring periods compared to the initial run. If the value of N_i remains close to 1, the currently observed system load average L_i, is close to $\overline{L_{initial}}$ and the application will receive a similar share of wall time to the initial run; if this is maintained for the duration of the experiment it will result in a similar completion time. However, if the value of N_i decreases during the execution, the currently observed system load average has increased in the presence of competing applications and the share of wall time will decrease due to other applications competing for the CPU. If maintained for the duration of the experiment this will result in a longer execution time with respect to the initial run.

For each monitoring period, $\sum_{i=1}^{n} N_i T_i$ is deducted from the total needed to complete, $T_{initial}$. A mean value, $N_{estimate}$ of the normalised measure of the system load average, N_i over n samples is used to generate a prediction for $T_{remaining}$. $N_{estimate}$ is calculated from the beginning of the experiment and reset if the SLA-bound application is restarted on a new resource.

$$N_{estimate} = \overline{N_n} = \frac{1}{n} \sum_{i=1}^{n} N_i$$

Equation 3 Calculating the mean of the normalised measure of system load

Othman et al [118] apply the technique differently to its application within this set of experiments. They monitor the CPU utilisation of the application to determine the amount of useful computation received in a given monitoring period. Therefore, the monitoring applies directly to the application and any changes apply specifically to the application. However, the CPU utilisation metric suffers from random noise which can skew the monitoring information causing false positive adaptation actions.

The experiments in this chapter use a normalised measure of the system load to monitor the share of wall time received by the SLA-bound application compared to the initial run. The normalised measure of system load takes account of the number of competing applications and the likelihood that each will demand access to the CPU equally over each monitoring period. As the number of competing applications increase the SLA-

bound application will receive a smaller share of the wall time compared to the initial run. Over a given monitoring period if the share is smaller, the SLA-bound application will require more time to complete when compared to the initial run. This approach does not apply directly to the SLA-bound application, but rather its likely share of the CPU over a given wall time compared to the initial run. This method does not suffer from the same short term fluctuations as the CPU utilisation metric, making it less likely to cause false positive adaptation actions.

4.3 Scenario 1 – Performance Guarantees

In Scenario 1 competing applications are submitted in order to violate a performance guarantee specified within the SLA illustrated in Figure 36. The schema governing the SLAs content is presented in Section 3.8.1. The consumer and provider elements illustrate the SLA consumer, which is an end-user with the username *jamesp* from the University of Leeds. The provider is the administrative service responsible for the Leeds based Grid resource and is also at the University of Leeds. The CPU and RAM requirements specify a node with 1 Intel CPU of 1GHz and at least 256MBs of RAM. The CPU element also specifies one performance guarantee, a value for the normalised measure of system load equal to 0.25. If this metric falls below 0.25 the SLA is violated. OS and storage requirements indicate the node must be Linux based with a kernel version of 2.6 and a temporary directory in which a user can write at least 512MBs data for storage.

```
<?xml version="1.0" encoding="UTF-8 standalone="yes"?>
<sla jobName="job1">
     <parties>
       <consumer>
    <name>jamesp</name>
    <address>University of Leeds, UK</address>
         <signed>true</signed>
  </consumer>
       <provider>
           <name>Informatics Institute</name>
           <address>University of Leeds, UK</address>
           <signed>false</signed>
       </provider>
     </parties>
<cpu>
       <version>Intel</version>
       <count>1</count>
       <speed>1000</speed>
       <normalised_load>0.25</normalised_load>
     </cpu>
     <ram>
       <count>256</count>
     </ram>
     <hdd>
       <count>512</count>
     </hdd>
     <os>
       <type>Linux</type>
       <version>2.6</version>
       <directory>/tmp</directory>
     </os>
</sla>
```

Figure 36 SLA used in scenario 1

The application described in section 4.2.1 is executed with input variables which will allow it to run for approximately 13 minutes. This duration is chosen to allow enough time to submit competing applications and observe the response of the SLA manager. Competing applications are submitted after the SLA-bound application begins executing. These are other instances of the same application described in Figure 34, which will increase the system load average measured by the SLA Engine using the MDS and calculated by SGE.

Figure 37 shows the normalised measure of system load between the start and finish of the execution. Competing applications cause it to decrease but fail to violate the performance guarantee. Another competing application is submitted which reduces the normalised system load below the guarantee threshold causing an SLA violation.

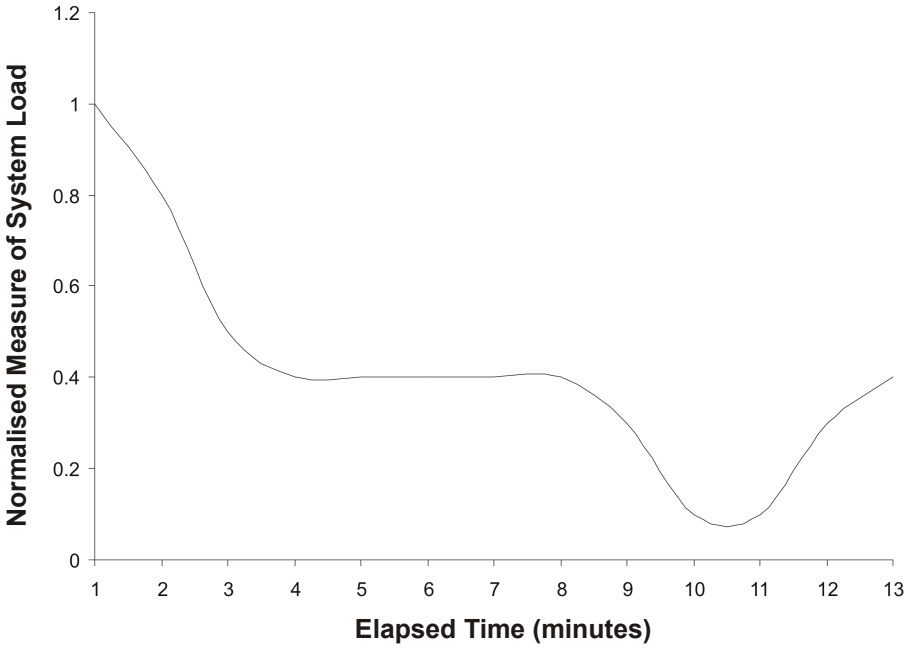

Figure 37 Normalised measure of system load vs. Elapsed Time: Performance Guarantee

Figure 38 shows the resultant changes to the SLA after the run illustrated by Figure 37. The SLA Manager is successful in recording the violation with a timestamp and value whilst the normalised measure of system load remains below the guarantee level.

```
<cpu>
     <version>Intel</version>
     <count>1</count>
     <speed>1000</speed>
     <normalised_load>0.25</normalised_load>
     <violation>
           <timestamp>2004-21-07T10:42:00Z</timestamp>
           <value>0.2</value>
     </violation>
     <violation>
           <timestamp>2004-21-07T10:43:00Z</timestamp>
           <value>0.2</value>
     </violation>
</cpu>
```

Figure 38 Changes to the SLA for experiment in Figure 37

4.4 Scenario 2 – Best Effort

In scenario 2, the SLA-bound application is executed using best effort service. The intention is to demonstrate the SLA-bound application's behaviour when adaptation is not provided. For each run of the scenario the initial prediction is selected to allow time to observe the SLA Manager and the responses of the adaptive controller. The value is determined by taking an average execution time from the observed values of a number of application runs. In each case the application parameters are set and the application is executed a number of times until a stable average is determined. Each run is conducted in the absence of competing applications or heavy weight processes in order to minimise the variance between runs. Initial prediction indicates that the application will take 20 minutes to complete based on an assumption that it executes with a system load average of 1 on Testgrid4 for the duration of the experiment. From this a timing constraint of 25 minutes is used which represents a 25% buffer over the initial prediction. A 25% buffer is used to simulate a strict timing deadline for the SLA-bound application.

The rule base for scenario 2 is shown in Figure 39. The control variables are the scheduled remaining execution time, $T_{schedule}$ and the predicted remaining execution time, $T_{remaining}$.

```
procedure adaptive_decision (T_remaining, T_schedule)
        if T_remaining > T_schedule then
                control_action := warning
        else
                control_action := zero
        return(control_action)
end control_action
```

Figure 39 Rule Base: Scenario 2

The rule base indicates that if the $T_{remaining}$ is larger than $T_{schedule}$, a warning should be recorded within the SLA.

The SLA used for scenarios 2 and 3 is illustrated in Figure 40. Most elements are identical to those specified in Figure 36. However, rather than a performance guarantee, a timing guarantee is specified in the completion element which includes a deadline by which the application must complete. This value will differ for each run which takes place.

```
<?xml version="1.0" encoding="UTF-8 standalone="yes"?>
<sla jobName="job1">
     <parties>
       <consumer>
    <name>jamesp</name>
    <address>University of Leeds, UK</address>
         <signed>true</signed>
  </consumer>
       <provider>
           <name>Information System Services</name>
           <address>University of Leeds, UK</address>
           <signed>false</signed>
       </provider>
     </parties>
<cpu>
       <version>Intel</version>
       <count>1</count>
       <speed>1000</speed>
     </cpu>
     <ram>
       <count>256</count>
     </ram>
     <hdd>
       <count>512</count>
     </hdd>
     <os>
       <type>Linux</type>
       <version>2.4</version>
       <directory>/tmp</directory>
     </os>
     <completion>
       <time>2005-17-08T10:45:00Z </time>
     </completion>
</sla>
```

Figure 40 SLA used in scenario 2 and 3

The scenario returned the following monitoring plots - Figure 41 and Figure 42 which illustrate the normalised measure of system load for the resources on which the SLA-bound application is executing. In Figure 41 competing applications are added early in the execution schedule, in Figure 42 they are added later. In both runs the SLA-bound application is executing under best effort service; migration is not used. In both Figure 41 and Figure 42 the competing applications reduce the normalised measure of system load. With no opportunity for migration, the SLA-bound application continues to execute with reduced performance.

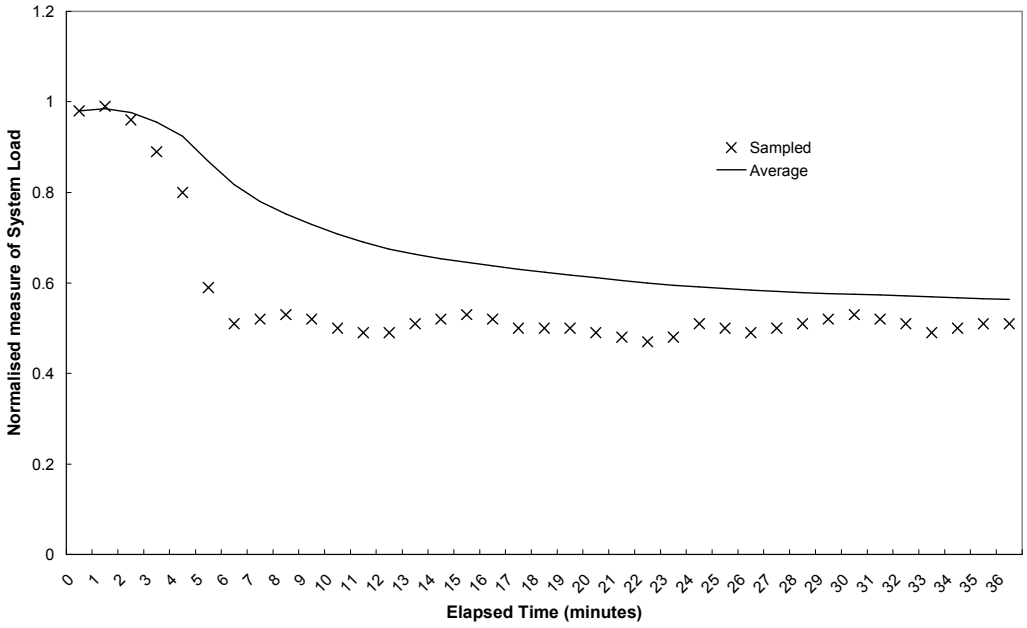

Figure 41 Normalised measure of system load vs. Elapsed Time: Scenario 2 - Disturbance added early

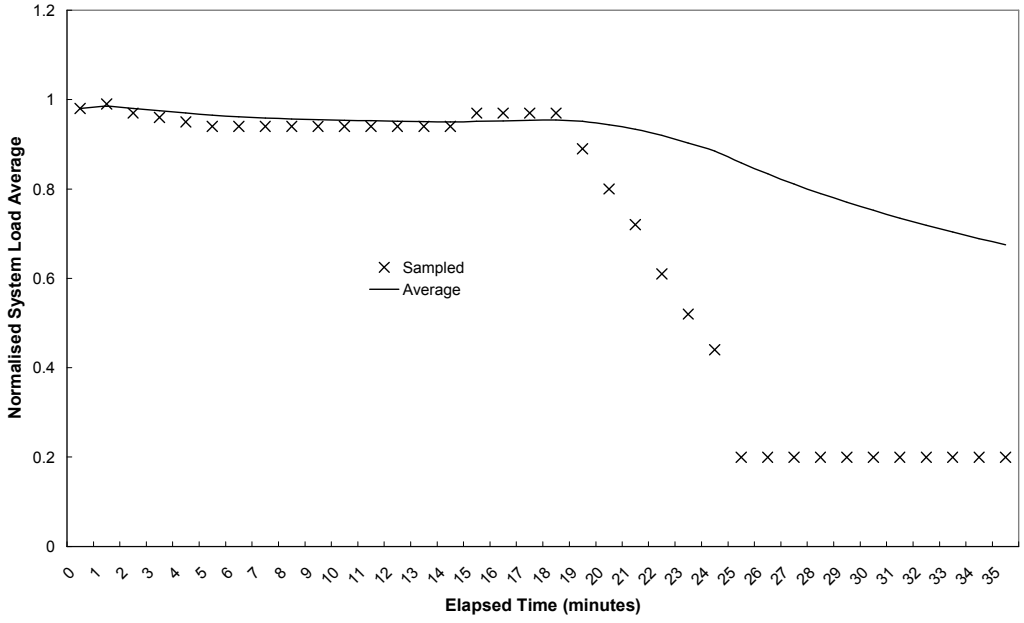

Figure 42 Normalised measure of system load vs. Elapsed Time: Scenario 2 - Disturbance added late

The scenario returned the following plots - Figure 43 and Figure 44 which show $T_{schedule}$ and $T_{remaining}$ for SLA-bound applications executing with normalised system load

described by Figure 41 and Figure 42 respectively. Also shown is $T_{100\%}$, which is the time the application would take to complete if the normalised measure of system load were to remain close to 1 for the duration of the execution.

Both Figure 43 and Figure 44 illustrate the effect of reduced performance on $T_{remaining}$, which in both plots increases above $T_{schedule}$ as the normalised measure of system load decreases. In Figure 43 and Figure 44 the SLA-bound application continues to execute under low normalised system load, preventing it from finishing before the deadline and is terminated.

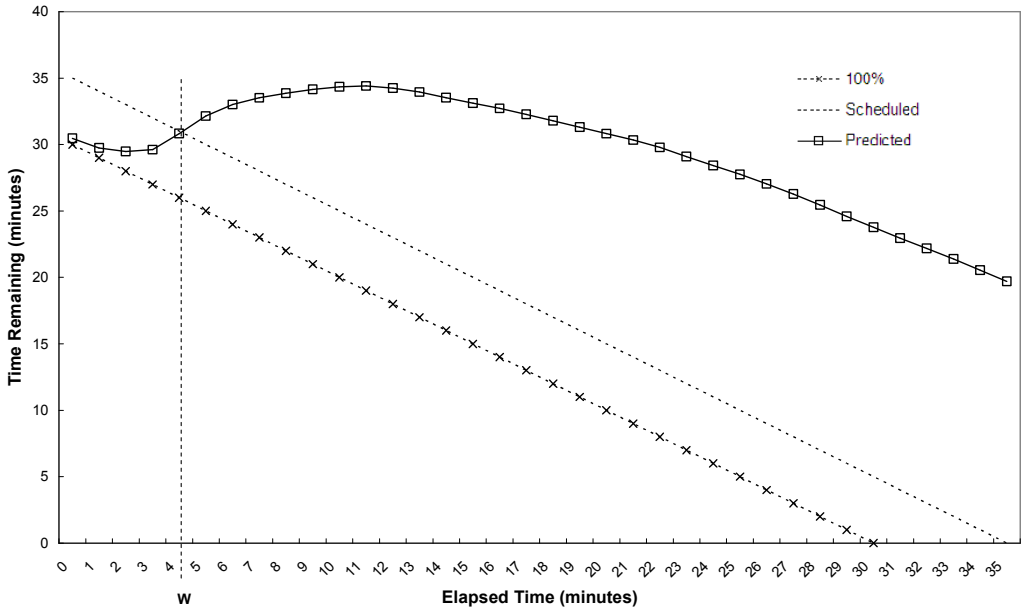

Figure 43 Time Remaining vs. Time elapsed: Scenario 2 - Disturbance added early

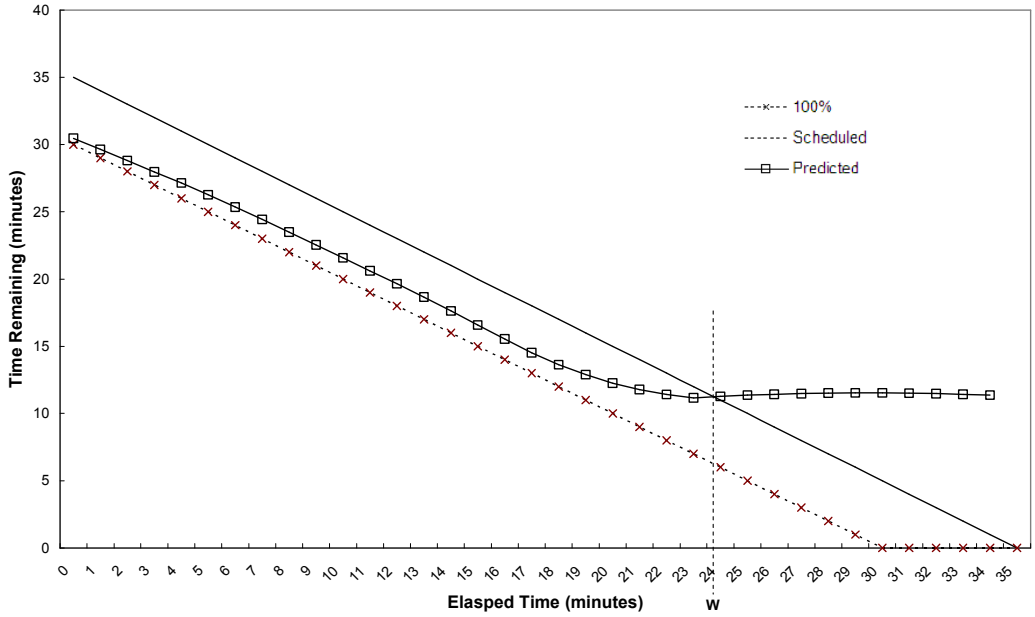

Figure 44 Time Remaining vs. Elapsed Time: Scenario 2 - Disturbance added late

Figure 45 and Figure 46 compare changes made to the SLA by the SLA Manager for both runs – Figure 43 and Figure 44. Although adaptation and migration were not used, the SLA Manager recorded warnings when $T_{remaining}$ increased beyond $T_{schedule}$. In Figure 45 a warning is recorded after 4 minutes and in Figure 46, 24 minutes. This is reflected in Figure 45 and Figure 46 respectively by a timestamp in the warning element.

```
<completion>
     <time>2007-20-01T12:21:00Z</time>
     <warning>
           <timestamp>2007-20-01T11:50:00Z</timestamp>
     </warning>
</completion>
```

Figure 45 Changes to the SLA for experiment in Figure 43

```
<completion>
     <time>2005-17-08T16:32:00Z</time>
     <warning>
           <timestamp>2005-17-08T16:23:00Z</timestamp>
     </warning>
</completion>
```

Figure 46 Changes to the SLA for experiment in Figure 44

4.5 Scenario 3 – SLA Management

In scenario 3, the SLA-bound application is executed using adaptation to demonstrate its added benefit over best-effort provision. Competing applications are submitted during the execution schedule to force the adaptive rule-based controller to migrate the SLA-bound application.

The rule base for scenario 2 is shown in Figure 47. The control variables are the scheduled remaining execution time, $T_{schedule}$ and the predicted remaining execution time, $T_{remaining}$.

```
procedure adaptive_decision (T_remaining, T_schedule)
        if T_remaining > T_schedule then
                control_action := warning, migrate
        else
                control_action := zero
        return(control_action)
end control_action
```

Figure 47 Rule Base: Scenario 3

The rule base indicates that if $T_{remaining}$ is greater-than $T_{schedule}$ a warning be recorded within the SLA and an attempt made to migrate the application. The SLA used for this scenario is the same as that used in scenario 2 and is illustrated in Figure 40. As with scenario 2, an initial prediction of 20 minutes and timing schedule of 25 minutes is used in scenario 3.

The scenario returned the following monitoring plots - Figure 48 and Figure 49 which illustrate the normalised measure of system load for the resources on which the SLA-bound application is executing. In Figure 48 competing applications are added early in the execution schedule, in Figure 49 they are added later. For both runs the adaptive rule-based controller is configured to execute the SLA-bound application using adaptation; application migrations will take place.

In both runs, competing applications reduce the normalised measure of system load until the adaptive controller signals a migration and the application is restarted on a new resource. After the application is migrated from Testgrid4 to Testgrid5, monitoring is resumed on the new resource and the normalised measure of system load returns to the level seen at the start of the experiment as the new resource is free from competing

applications. Testgrid5 is chosen because it has performance which matches that of the initial resource. The overhead due to migration is attributed to the transfer costs associated with moving the checkpoint file onto Testgrid5 and delays due Grid middleware and the SGE queuing system on Testgrid5.

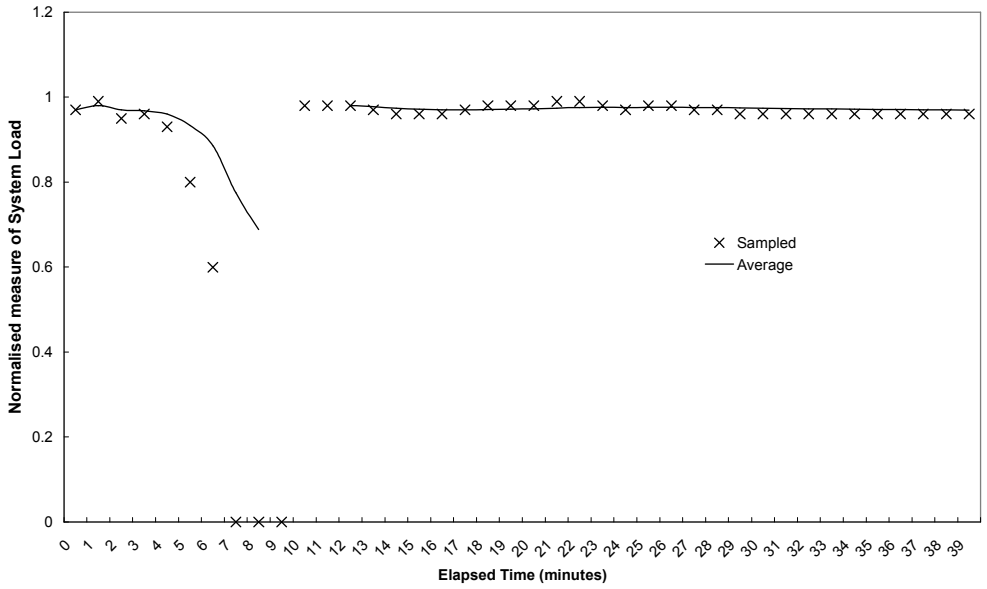

Figure 48 Normalised measure of system load vs. Elapsed Time: Scenario 3 - Disturbance added early

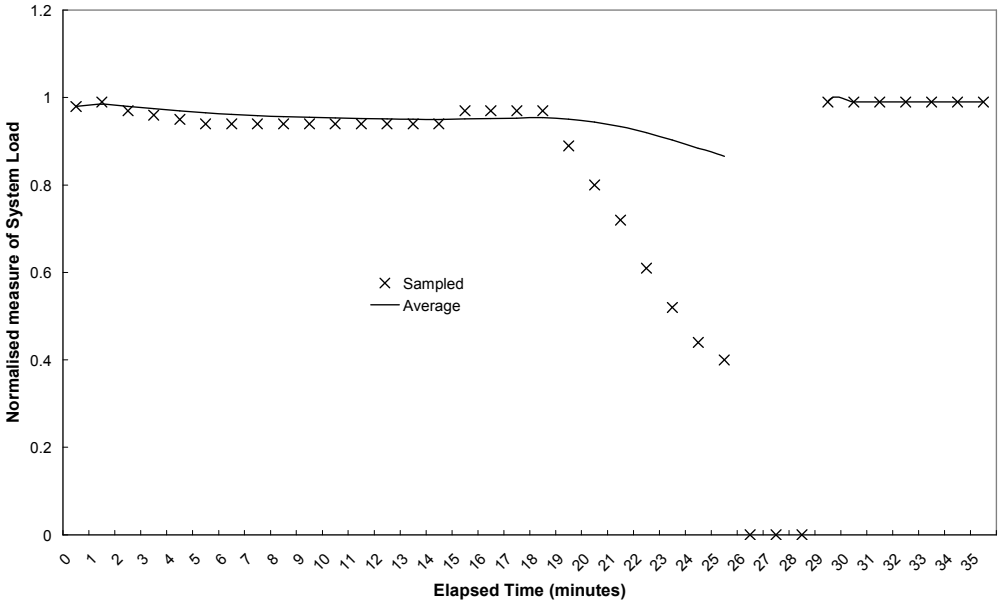

Figure 49 CPU Load vs. Elapsed Time: Scenario 3 - Disturbance added late

The scenario returned the following plots - Figure 50 and Figure 51 which illustrate $T_{schedule}$ and $T_{remaining}$ for the SLA-bound applications executing with a normalised measure of system load described by Figure 48 and Figure 49 respectively.

Both Figure 50 and Figure 51 demonstrate the affect of reduced performance on $T_{remaining}$, which in both cases increases above $T_{schedule}$ as the normalised measure of system load reduces. In both plots, the SLA-bound application executes with a low normalised measure of system load until the SLA Manager migrates the application onto a new resource – testgrid5. The remaining execution time is maintained by the SLA Manager and used as a starting value on the new resource.

In both Figure 50 and Figure 51, when the application is restarted, the normalised measure of system load is closer to the levels seen at the start of each experiment, causing $T_{remaining}$ to decrease below $T_{schedule}$. The normalised measure of system load remains stable in both plots until the end of the execution, allowing the application to finish before the deadline specified within the SLA.

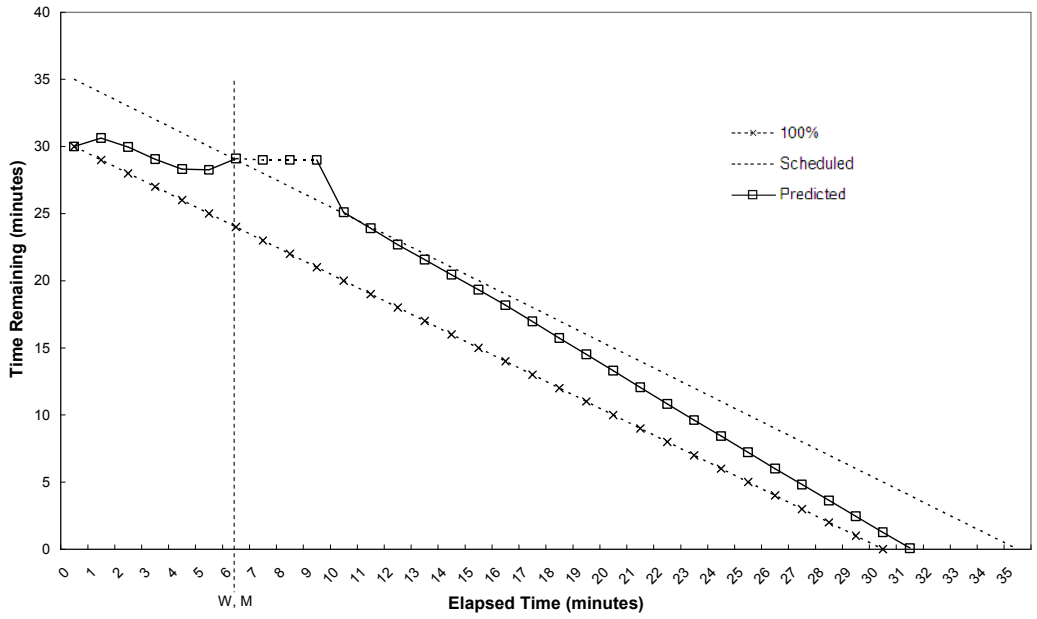

Figure 50 Time remaining vs. Elapsed Time: Scenario 3 - Disturbance added early

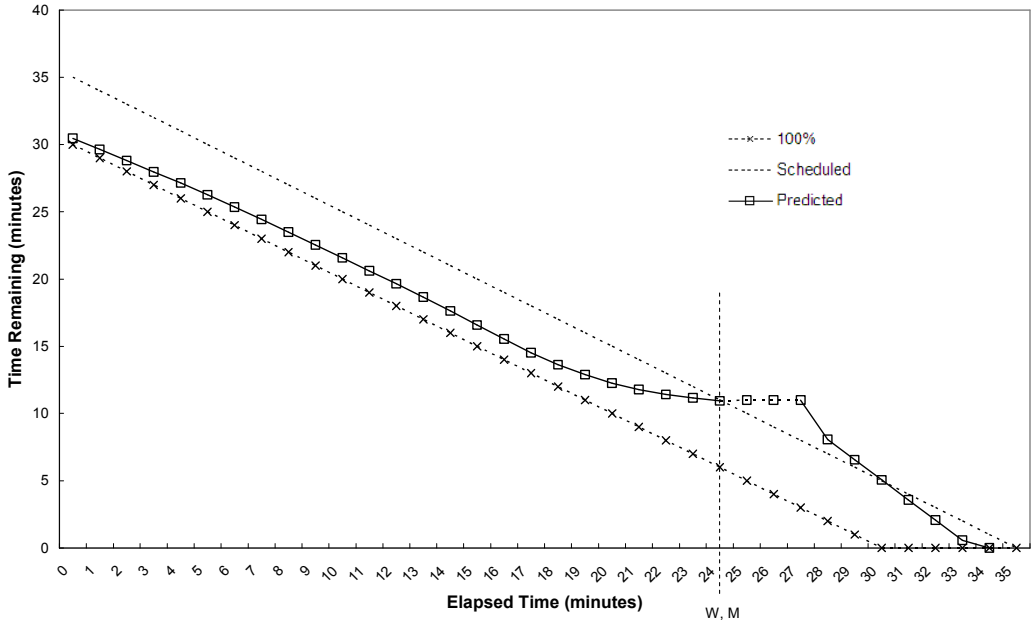

Figure 51 Time remaining vs. Elapsed Time: Scenario 3 - Disturbance added late

Figure 51 and Figure 52 show the resultant changes to the SLA after the experimental runs illustrated in Figure 50 and Figure 51 respectively. Figure 50 indicates a warning and migration signal is triggered by the adaptive rule-based controller after 6 minutes and in Figure 51 the same action occurs after 24 minutes. In both cases the SLA Manager successfully updates the SLA with a warning and associated timestamp as well as a migration and associated timestamp and details of the resource onto which the application was migrated. The resource element indicates that for both runs, testgrid5 was the resource onto which the application was migrated.

```
<completion>
     <time>2007-20-01T15:37:00Z</time>
     <warning>
          <timestamp>2007-20-01T15:09:00Z</timestamp>
     </warning>
     <migration>
          <timestamp>2007-20-01T15:09:00Z</timestamp>
          <resource>testgrid5</resource>
     </migration>
</completion>
```

Figure 52 Changes to the SLA for experiment in Figure 50

```
<completion>
     <time>2005-23-08T11:13:00Z</time>
     <warning>
           <timestamp>2005-23-08T11:02:00Z</timestamp>
     </warning>
     <migration>
           <timestamp>2005-23-08T11:02:00Z</timestamp>
           <resource>testgrid5</resource>
     </migration>
</completion>
```

Figure 53 Changes to the SLA for experiment in Figure 51

4.6 Summary

This chapter has demonstrated the SLA Manager on a local Grid test-bed. The motivation for the experiment is a need to test and debug the SLA Manager prior to deployment on a large distributed Grid infrastructure, the WRG. The scenarios confirm that the SLA Manager and the adaptive rule-based controller in combination with effective monitoring can detect and record warnings, violations and migration signals within the SLA. For the three scenarios tested, the same components are successful in preventing violation of a timing constraint when an SLA-bound application suffers a performance degradation caused by competing applications. The resulting SLAs illustrate that warnings, migrations and violations signalled by the adaptive controller can be captured and added to the SLA by the SLAManager . The results are evaluated in detail in Chapter 7, where they are considered in the context of the results from Chapter 5, the experiments on the WRG.

In Chapter 5, the SLA Manager is tested on the WRG using three scenarios. The first is the execution of an SLA-bound application with a timing constraint with application adaptation enabled. This demonstrates the enhancements brought by the SLA Manager from the perspective of an end-user. The second is a Grid application execution with a timing constraint with no adaptation. It is expected to show the disadvantages of executing a Grid application with current (best-effort) Grid middleware provision. The third scenario uses the DAME XTO application with the SLA Manager to evalute the migration strategy for an application which relates more to the motivating scenario discussed in Section 3.1.

Chapter 5

Testing adaptive SLA management on the White Rose Grid

This chapter contributes to deliverable [C7], by providing a performance comparison of deliverables [C1]-[C2] and [C6] on a large distributed Grid infrastructure, the White Rose Grid. The experiments involve the execution of a CPU intensive Grid application, bound to an SLA specifying a timing guarantee for its completion. During the experiments, competing applications are introduced in an attempt to reduce the performance of the SLA-bound application causing it to violate the timing guarantee. The SLA Manager, through application level monitoring and an adaptive rule-based controller must react to the disturbance to ensure that the timing constraint and the SLA is not violated. This is achieved by migrating the SLA-bound application onto resources with spare capacity; two scenarios are considered: (1) sufficient resource capacity is available with the intial resource provider[1] and (2) the resource provider has insufficient resource capacity and migration must take place onto resources owned by a different provider. Section 5.2 provides an overview of the experiment and its objectives. Additionally, there is a description of deliverable [C5]; a method for monitoring the amount of CPU time received by a Grid application, which can be used to monitor timing guarantees specified within an SLA. Deliverable [C4] describes an initial prediction method based on historical usage statistics, which can approximate the amount of CPU time needed by a Grid application in order for it to complete. A method for specifying the remaining execution time is also described. Section 5.3 presents the

[1] A resource provider is the owner of resources on which the application is currently executing. Different resource providers belong to different administrative domains and only have admin rights over the resources owned by them.

results from scenario 1, single provider migration and Section 5.4 presents the results from scenario 2, multiple provider migration. Section 5.5 introduces a third scenario which involves the use of the SLA Manager with an application closely associated with the motiviating scenario in Section 3.1. The chapter is summarised in Section 5.6.

5.1 Overview

This experiment is performed on the WRG, which is a deployment of Grid resources distributed between the Universities of Leeds, Sheffield and York. The WRG is a multi domain Grid deployment and offers a more realistic environment in which to test Grid research software. Demand for resources is more intense because usage is shared not only between a greater density of Grid users but in addition, local HPC users. This is in contrast to the local Grid test-bed used in Chapter 4, which is a single domain deployment, with few users and not subject to the resource management issues such as site autonomy, heterogeneous substrate or policy extensibility.

The WRG (Figure 54) consists of 4 clusters: Maxima (Leeds #1), Snowdon (Leeds #2), Iceberg (Sheffield) and Pascali (York).

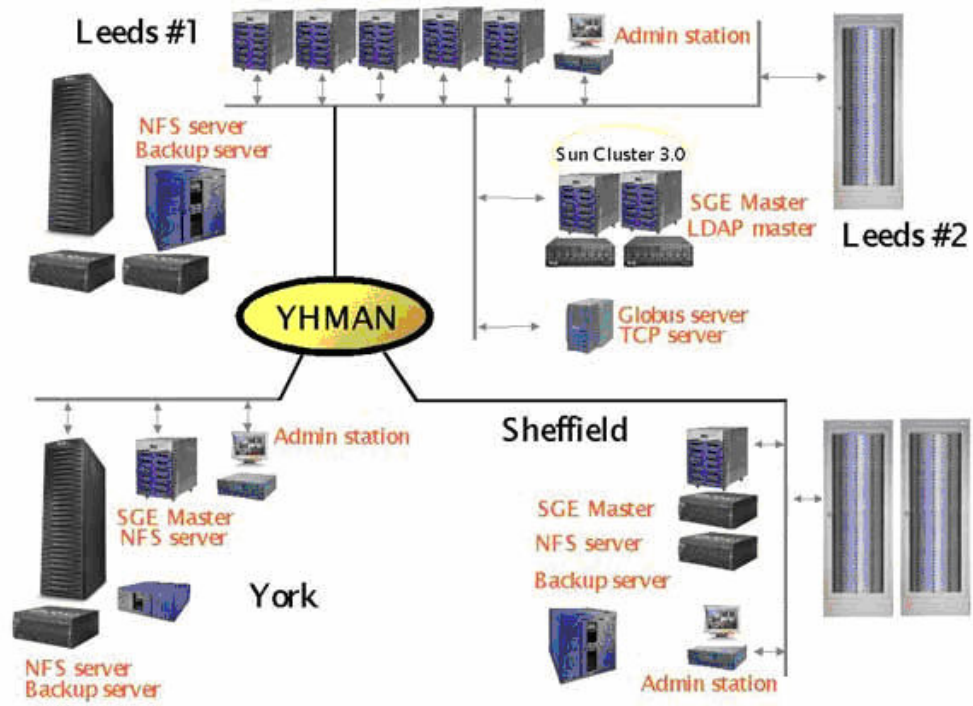

Figure 54 The White Rose Grid (WRG) Architecture

- Maxima incorporates five 8 node Sun V880 servers each with 24GB of shared memory and one 20 node SunFire 6800 server with 44 GB of shared memory; all with Sun UltrasparcIII CPUs. The operating system is 64bit Solaris.
- Iceberg incorporates a 160 node cluster, eighty of which are quad AMD Opteron 64bit CPUs each with 16GB of shared memory and a further eighty dual AMD Opteron 64bit CPUs each with 4 GB of main memory. The operating system is 64 bit Solaris.
- Pascali incorporates five 8 node Sun V880 servers each with 24GB of shared memory and one 20 node SunFire 6800 server with 44 GB of shared memory; all with Sun UltrasparcIII CPUs. The operating system is 64bit Solaris.
- Snowdon incorporates a cluster of 128 nodes each with dual Intel 32 bit CPUs and 2 GB of shared memory. The operating system is 32 Bit Solaris.

Each has installed Globus Toolkit 2.4 Grid middleware and Sun Grid Engine 5.3/6.0 network batch queuing system. MDS2 is deployed at each site which can be accessed using the LDAP protocol to provide resource information. Communications between resources is over the Yorkshire and Humberside Metropolitan Area Network (YHMAN).

The WRG represents a large distributed Grid infrastructure drawing resource from three separate organisations. Testing Grid research software in such an environment will highlight problems and offers a more realistic testing environment with a high volume of users and applications. The software will be exposed to issues of site autonomy, heterogeneous substrate and policy extensibility.

5.2 Experimental Design

As with the experiments on the local Grid test-bed in Chapter 4, the SLA Manager and its ability to prevent violations to SLA timing guarantees is under investigation. To recap, the SLA Manager provides enhancements (deliverables [C1]-[C2] and [C6]) to current Grid middleware by combining a number of techniques (deliverable [C4] and [C5]) which have been demonstrated separately within other Grid systems. Initial prediction using historical observations allows an approximation of the execution time

of the application. The resulting prediction is more realistic than relying on an end-user to provide the estimate. Allowing the end-user to estimate the execution time could lead to an inaccurate estimation, especially if they have no experience with the application. An adaptive rule-based controller in combination with application level monitoring provides a control loop for adaptation of the SLA-bound application. Results are expected to show that SLA violations can be avoided as a direct result of intervention by the SLA Manager and the rule based controller. Updating the SLA during run-time with modifications to key elements to reflect warnings, migrations and violations demonstrate that SLAs need not be static entities which are useful only as a means to negoatiate usage.

The SLA Manager can specify an SLA for a Grid application, submit the application, monitor the SLA guarantees and signal control actions. The Grid application is CPU intensive and includes checkpointing support; it is described in more detail in Section 5.2.1. In order to test the SLA Manager, two experimental scenarios are considered:

1. An SLA is created which specifies a guarantee for the completion of a specific run of a Grid application by a deadline. The duration is derived from the total amount of CPU time needed in order for that specific run to complete. This is determined using the initial prediction technique in section 5.2.3. The migration is handled by the SLA Manager using Globus and SGE. This scenario demonstrates availability of resources with the current provider.

2. As with scenario 1, an SLA is created which specifies a guarantee for the completion of a specific run of a Grid application by a deadline. This scenario demonstrates the unavailability of resources with the initial provider and migration onto a Grid resource with a new provider.

In both scenarios, a comparison is made with a Grid application executing without application adaptation support. This demonstrates the performance enhancement provided by the SLA Manager when compared to a Grid application executing on a standard Grid infrastructure[2]. For this comparison, the adaptive rule based controller is

[2] Where a standard Grid setup is defined as the bespoke installation of a Grid middleware such as the Globus Toolkit connected to a network batch queuing system such as SGE.

configured to signal warnings and violations; no migration actions are specified. Competing applications are submitted during the execution and the SLA Manager is used only to monitor and predict the remaining execution time of the Grid application; adaptation is not provided.

The scenarios test the SLA Manager's ability to monitor application process information given a timing constraint based on an initial prediction of the application's execution time. The adaptive controller is being tested to determine if it will react to a predicted violations in the timing constraint when the Grid application process has to compete for CPU time with another application. The ability of the SLA Manager to record these as warnings within the SLA is also being tested. The experiments simulate two expected usage scenarios; one in which the resource provider has additional spare capacity onto which the Grid application can be migrated and one in which resources from a new provider have to be used. The experiment makes use of the WRG, with the first experiment making use of Snowdon and the second making use of Snowdon and Iceberg. The experiment begins with the creation of an SLA specifying a duration based timing guarantee for a specific run of a Grid application. This is generated using the initial prediction method described in section 5.2.3, which uses information from previous runs. Once the application is submitted, the SLA Manager uses a monitoring script to monitor the CPU time of the Grid application process (details are given in section 5.2.2). The control variables used are the scheduled remaining execution time (specified in the SLA) and the predicted remaining execution time (details are given in section 5.2.4). The control action is a signal to migrate the application; in addition to this the SLA Manager will update the SLA with warnings, violations and migrations when they are performed or detected.

As with the experiments in chapter 4, both resource monitoring and prediction of the application execution time during runtime are needed to support the adaptive rule-based controller. The performance of this technique is not under consideration and is used only as a means to specify a control variable with which to compare the scheduled remaining execution time. Both scenarios use initial prediction and a theoretical maximum available CPU time for the selected resource and a buffer to select a suitable timing deadline. Initially predicting the total amount of CPU time needed for a Grid application to complete is important because the calculation of the predicted remaining execution time during execution depends upon it. For this experiment, the method of predicting

the remaining application execution time during runtime has changed from the previous chapter, details are provided in Section 5.2.2.

5.2.1 Grid Application

The Grid application used in scenario 1 and 2 uses the same computationally intensive application to approximate the value of π as is used in Chapter 4 (Figure 34). The Grid application used in scenario 3 is the DAME XTO application used in the motivating scenario in Section 3.1.

The application is submitted by the SLA Manager in the same way as it was it Chapter 4. However, the RSL used to pass job submission information and configure the execution environment (Globus and SGE) is different. Figure 55 illustrates the RSL used to submit the SLA-bound application to Snowdon at the Leeds WRG site.

```
"&" + "(executable=$(GLOBUS_LOCATION)/bin/globus-sh-exec)"
    + "(arguments=$(HOME)/experiments/pi/check_userdefined4.sh)"
    + "(jobtype=single)"
    + "(count=1)"
    + "(queue=00short1)"
    + "(project=Computing)"
    + "(name=GlobusTest)"
    + "(checkpointenv=check_userdefined)";
    + "(environment=(LD_LIBRARY_PATH /usr/local/sge/lib/glinux))”
    + "(stdout=/tmp/checkpoint/james.out)"
    + "(stderr=/tmp/checkpoint/james.err)";
```

Figure 55 RSL code for submission of the SLA-bound application

Argument specifies the location of the shell script (check_userdefined4.sh) from which the application is executed. Check_userdefined4.sh (Figure 56) tells SGE where to write the checkpoint files and checks that the directory has been created. The script checks whether the application is a restart, if so the –r flag is used to tell the application to restart using the checkpoint file in the checkpoint file directory, which is specified using the –d flag. If this is not the case, the application is started without the –r flag.

```
SGE_CKPT_JOB=$SGE_CKPT_DIR/$USER

if [ \! -e "$SGE_CKPT_JOB" ] ; then
    mkdir $SGE_CKPT_JOB
fi

if [ \! -d "$SGE_CKPT_JOB" ] ; then
    echo "Checkpoint subdirectory couldn't be found."
    exit 1
fi

if [ "$RESTARTED" -eq "1" ] ; then
    $HOME/experiments/pi/checkpoint_program4 -r -d $SGE_CKPT_JOB
else
    $HOME/experiments/pi/checkpoint_program4 -d $SGE_CKPT_JOB
fi

rm -rf $SGE_CKPT_JOB
exit 0
```

Figure 56 Grid application submission script

In scenario 1, the single provider experiment, the migration is signalled by the SLA Manager, but it is handled by SGE. The application is suspended and moved from the current execution node to another spare node. This makes use of the shared memory setup on Snowdon and carries with it a smaller overhead than scenario 2. In scenario 2, the multiple provider experiment, the migration is signalled and handled by the SLA Manager. The application remains active on Snowdon in Leeds, whilst the checkpoint file is transferred by the SLA Manager using GridFTP to Iceberg. This carries with it a greater overhead because of the latency costs associated with a GridFTP transfer and resubmission on a new resource. Two instances of the application are active after the migration has taken place, one on Snowdon, the other on Iceberg.

5.2.2 Resource Monitoring

To monitor the progress of the application during a specific time period, the CPU time of the process is measured periodically. From an application perspective, useful CPU cycles are performed when the CPU is in user mode; i.e. when the CPU is executing application instructions. Most operating systems maintain this information on a per process basis. A monitoring script has been written to monitor the CPU time of the application process as it is executing. It does this by reading from the /proc directory structure within UNIX type operating systems. The proc directory maintains information describing each OS process including state, usage and performance. Within the stat file, the process CPU time is recorded in the 14th field. This is shown in bold in Figure 57,

along with other fields which describe the process. A field key is provided in Appendix B.

32406 (checkpoint_prog) R 32404 32404 32404 0 -1 1048576 17 0 71 0 **12720** 0 0 0 15 0 0 0 451525747 1540096 72 4294967295 134512640 134517172 3221222320 3221221696 1073888283 0 0 0 0 0 0 0 17 1

Figure 57 Information maintained by the OS for each process in the /proc file system

The monitoring script provides accurate up-to-date application information without a large measurement overhead or high latency transmission. The monitoring information is recorded in /proc regardless of its use by the monitoring script, which implies that the chance of a false positive migration signal is smaller compared to the MDS used in Chapter 4. High latency transmission is less of a problem with the monitoring script because the monitoring tool is itself a process reading information from the file system, rather than a C application triggering a process; as is the case with MDS.

Further difference between the use of MDS and the monitoring script include the level at which the monitoring is applied. The monitoring script used in Chapter 5 monitors at the application level, providing accurate up-to-date information which is pertinent to the Grid application, rather than the system. This does not invalidate the use of MDS, which monitors the CPU load at the system level, so long as it is understood that performance effects on individual Grid applications will be felt less sharply in this metric.

5.2.3 Initial Prediction

The initial prediction method uses linear regression to predict the CPU time, J_{CPU} needed in order for the application to complete. This represents the amount of CPU time (jiffies) needed by the application in order for it to complete. A dataset of previous runs is built by executing the application 100 times on Maxima and Snowdon and Iceberg. A shell script is used to record the input variable and amount of CPU time (jiffies) of each specific run. For the input variable the value of r in the Grid application (Section 5.2.1) is modified.

The response variable represents the CPU time, J_{CPU}, and the predictor variable represents *r*, the accuracy with which π is calculated. A low number results in a quick calculation and low CPU intensity; as the number increases, so does the accuracy of the π calculation and the CPU intensity.

Research into similar learning techniques [97] have shown dataset sizes of 100 produce prediction errors in the order of 10%. The prediction error decreases as the number of runs increases, but so does the computation overhead. In [97], additional techniques are used to minimise the computation overhead, such as selecting upper and lower bounds to the dataset around the area of interest, a technique known as caching. In addition, runs which have been consistently used to make incorrect predictions are selectively dropped from the dataset, a technique known as instance editing. Neither of these techniques are used here to optimise the computation overhead of the prediction; again a performance optimisation of the technique is outside the scope of this thesis, therefore the use of 100 runs represents an optimum error/overhead trade-off.

During periods when a Grid resource is executing more than one heavy-weight process, for example when two or more applications are competing for the CPU, a Grid application will take longer to execute because it is forced to compete for CPU time. Therefore, it is important to know the approximate hardware specification with which the runs were made. Within this experiment, it is assumed that migrations take place between resources which exhibit similar CPU time totals for identical runs of the Grid application. The experiment does not consider the affect of CPU architecture on the amount of CPU time an application takes to complete. A more efficient CPU architecture may take less CPU time to complete the same run of the Grid application compared to a less efficient one.

5.2.4 Specifying the remaining execution time control variable

To predict the applications remaining execution time whilst it is executing, Equation 4 is used. The technique is derived from Equation 1 in Chapter 4, with the exception that it makes use of application process CPU time, rather than system level CPU load. Performance optimisation of this technique is not within the scope of the thesis, however comparisons of the two techniques will be made in Chapter 6. Calculating the predicted remaining execution time using Equation 4 is used only as a means to specify

a control variable with which to compare the scheduled remaining execution time. It should be noted that control variable selection is not limited to those choosen in this experiment. They are selected merely to demonstrate the adaptive rule-based controller and its ability to execute control actions given the runtime state described using the control variables and verified through effective application monitoring.

An applications workload profile represents its CPU usage trend over the course of the execution. The application used for these experiments has a linear workload profile and consistently accesses the CPU at the same rate from start to finish and does not show a reduced CPU usage during I/O or system calls.

A prediction estimate of the application's remaining execution time, $T_{remaining}$, is made using Equation 4. J_{Total}, represents the total CPU time (jiffies) needed by the application in order to complete and is obtained using the initial prediction technique. A jiffy is a basic packet of CPU time and is related to the speed of the CPU. It is used here as a measure of the amount of processing given to the application during a monitoring period.

$$T_{remaining} = \frac{J_{Total} - \sum_{i=1}^{n} J_{CPU}}{\overline{J_{CPU}}}$$

Equation 4 Estimating an application's remaining execution time using CPU time

Whilst the application is executing, for each monitoring period from t_0 to n, the amount of CPU time, J_{CPU} received by the Grid application (process) is sampled. For each monitoring period, the amount of CPU time is deducted from the total needed to complete, J_{Total}. An exponentially weighted moving average of the amount of CPU time received by the Grid application over the last 5 samples is taken, $\overline{J_{CPU}}$ and used to generate a prediction of $T_{remaining}$. An exponentially weighted moving average is used because it gives greatest significance to the latest monitoring information and therefore reacts faster to measurement changes than a simple moving average. Up-to-date measurements allow the information on which the adaptive rule-based controller makes decisions to more accurately represent the current state of the execution. Exponentially weighted moving averages are an accepted tool used within the financial services industry to most accurately track recent stock market price changes. The method is

applied here because it closely matches the requirements of the prediction technique in Equation 4.

The prediction remains valid if the system state remains largely unchanged for the rest of the execution. It is assumed that all resources are using the same definition of CPU time and that each resource can be quantified on a standard scale.

Although outside the scope of this thesis, for applications which are I/O dependent or when the workload profile of the Grid application is piecewise; due to I/O or system calls, this technique could be extended to include application kernel time.

Studies have shown [125] that long running applications executing alongside other competing applications can expect large variations in the per process CPU time available to them. The variation falls within three normal distributions (or modes) dependent on the number of competing applications. In the event that the process CPU time changes significantly in successive samples, the application is assumed to be executing within a different mode and the predicted time remaining is invalid. This is countered by using an exponentially weighted moving average. Othman et al [118] use a moving average of fractional CPU utilisation to generate a worst case estimate of future CPU utilization. The average is reset after the CPU utilisation of the Grid application falls in a predetermined number of successive samples to account for the change in execution mode. This approach potentially masks short term changes in the application execution mode, resulting in an under estimation of $T_{remaining}$.

If application migrations are necessary the SLA manager uses a checkpoint to restart the application on the new resource. The CPU time of the application is recorded when the checkpoint is taken. On the new resource, Equation 4 is again used to calculate values for $T_{remaining}$, however, J_{Total} is taken to be $J_{Total} - \sum_{i=1}^{migration} J_{CPU}$ from the calculation of $T_{remaining}$ on the first resource.

5.2.5 Rule Base and SLA

The rule base used for the experiments in this chapter is shown in Figure 58. In this experiment the adaptive controller compares the predicted and scheduled remaining execution times. If the predicted remaining time is greater than the scheduled, a further

test is applied. This tests if the SLA Manager has previously triggered a migration. This is designed to prevent migration signals every time the control loop is executed and the predicted time exceeds the scheduled time. If migration has taken place the control action is zero. This branch of the rule base allows time after a migration in order for the application to catch up with the schedule. If no migration has taken place a further test is applied to determine if the time currently is less than the timing constraint. If the time currently is less than the timing constraint, a warning and a migration is signalled and the migration triggered flag is set true. This prevents a further migration on the new resource when the control loop is first executed and the predicted time remaining is greater than the scheduled time remaining. If the time currently is not less than the timing constraint then a violation is recorded within the SLA.

```
procedure adaptive_decision (T_remaining, T_schedule)
        if T_remaining > T_schedule then
                if migration_triggered = false then
                        if T_i <  T_constraint then
                                migration_triggered = true
                                control_action := warning, migrate
                        else
                                control_action := violate
                else
                        control_action := zero
        else
                migration_triggered = false
                control_action := zero
        return(control_action)
end control_action
```

Figure 58 Rule Base experiments in Chapter 5

The SLA used for the experiments in this chapter is illustrated in Figure 59. The consumer and provider elements are specified identically to those in Chapter 4. The CPU and RAM requirements have been altered for the purposes of this experiment and now specify a node with 2 Intel or AMD64 CPUs of 1.5GHz and at least 512MBs of RAM. OS and storage requirements indicate the node must be Linux based with a kernel version of 2.4 and a temporary directory in which a user can write at least 512MBs data for storage. Finally, a timing guarantee is included in the completion element which specifies the time by which the application must complete. This value will differ for each scenario and run which takes place.

```
<?xml version="1.0" encoding="UTF-8 standalone="yes"?>
<sla jobName="job1">
     <parties>
       <consumer>
    <name>jamesp</name>
    <address>University of Leeds, UK</address>
         <signed>true</signed>
  </consumer>
       <provider>
           <name>Information System Services</name>
           <address>University of Leeds, UK</address>
           <signed>false</signed>
       </provider>
     </parties>
<cpu>
       <version>Intel, AMD64</version>
       <count>2</count>
       <speed>1500</speed>
     </cpu>
     <ram>
       <count>512</count>
     </ram>
     <hdd>
       <count>512</count>
     </hdd>
     <os>
       <type>Linux</type>
       <version>2.4</version>
       <directory>/tmp</directory>
     </os>
     <completion>
       <time>2006-08-02T13:17:00Z</time>
     </completion>
</sla>
```

Figure 59 SLA for scenarios 1 - 3

5.3 Scenario 1 – Single Provider

In this scenario competing applications force the adaptive controller to migrate the SLA-bound application onto resources with the current provider. This demonstrates a scenario where sufficient resource capacity is available with the current resource provider and migration takes place within the current Grid node. Using resources from the current provider incurs a smaller migration overhead than migration to a new resource provider. Initial prediction using the application parameters intended for use in the experiment indicates that it will take 26 minutes to complete. This is based on the assumption that it receives the maximum potential CPU time that Snowdon can provide for the duration of the experiment. From this a timing constraint of 34 minutes is used

which represents a 30% buffer over the initial prediction. A 30% buffer is used to simulate a strict timing deadline for the SLA-bound application.

5.3.1 Run 1

The first run of scenario 1 produced the following plots - Figure 60 and Figure 61 which illustrate the CPU time available to the SLA-bound application. In Figure 60 adaptation is used to migrate the SLA-bound application when triggered by the adaptive controller, in Figure 61 it is not. In both plots competing applications can be seen to affect the amount of CPU time available to the SLA-bound application after 4 minutes. This action simulates other Grid users executing further instances of the application described in Section 5.2.1. Where adaptation is used, migration allows the CPU time received by the application to increase. Where no adaptation is used the CPU time remains low due to competition from competing applications.

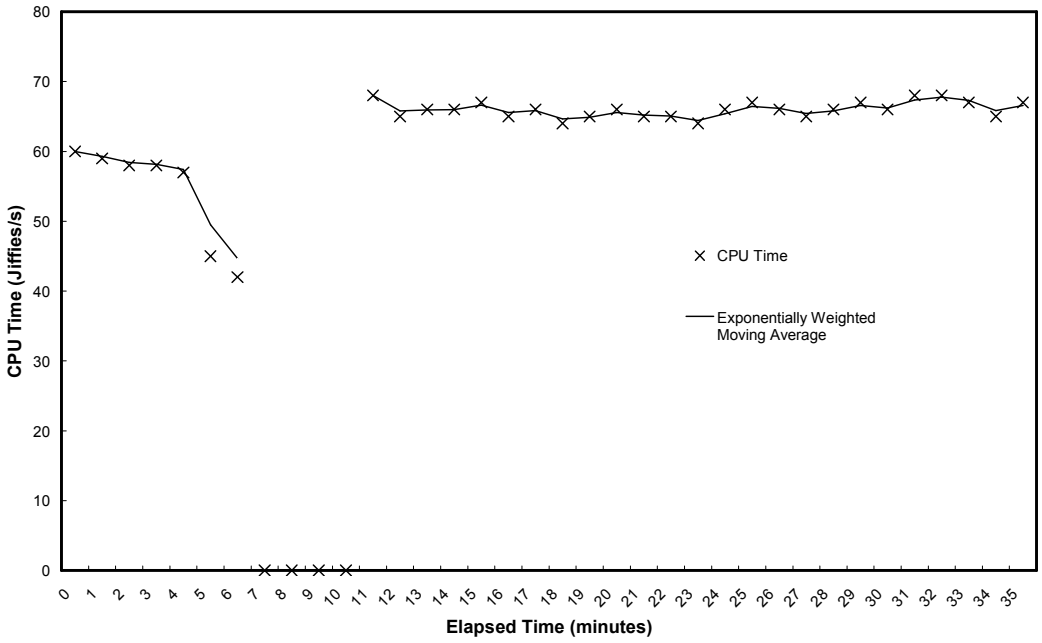

Figure 60 CPU Time (jiffies/s) vs. Elapsed Time (minutes) Single Provider with adaptation

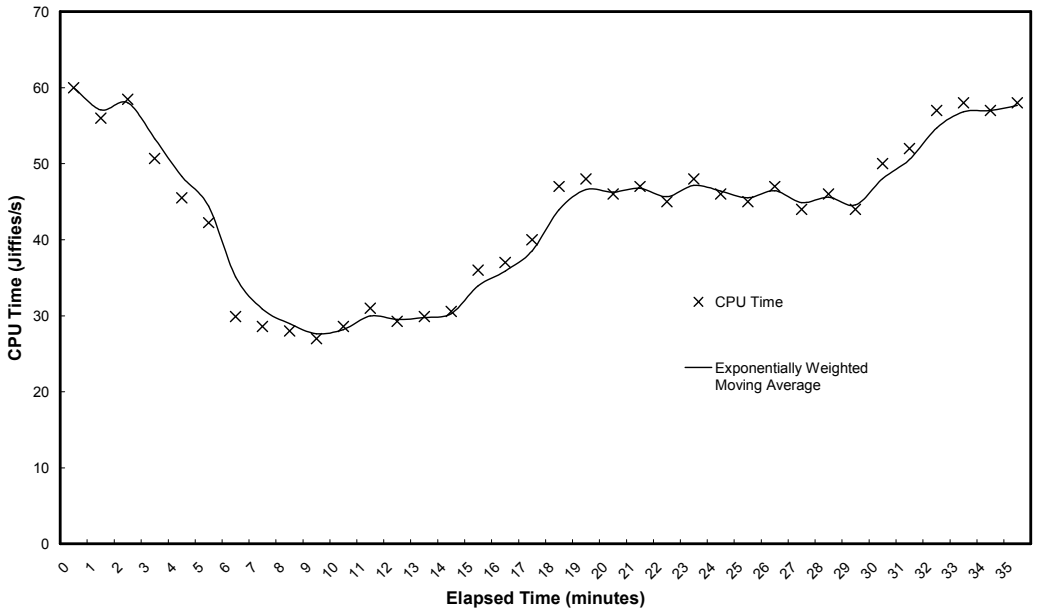

Figure 61 CPU Time (jiffies/s) vs. Elapsed Time (minutes) Single Provider without adaptation

The scenario also returned Figure 62 and Figure 63 which illustrate $T_{schedule}$ and $T_{remaining}$ for SLA-bound applications executing with CPU time described by Figure 60 and Figure 61 respectively. Also shown (100%), is the time the application would take to finish if it were to receive the maximum potential CPU time that the resource can provide for the duration of the experiment. In both plots the effect of reduced performance causes $T_{remaining}$ to increase beyond $T_{schedule}$. In Figure 62, where adaptation is used the adaptive rule-based controller triggers a warning and migration, which moves the SLA-bound application onto another node within the Snowdon cluster. After migration the SLA-bound application restarts it receives increased CPU time which causes $T_{remaining}$ to fall below $T_{schedule}$.

In Figure 63, no migration takes place and the SLA-bound application is forced to share the CPU with the competing applications. It continues to receive reduced CPU time and the difference between $T_{remaining}$ and $T_{schedule}$ continues to increase until the SLA timing constraint is violated.

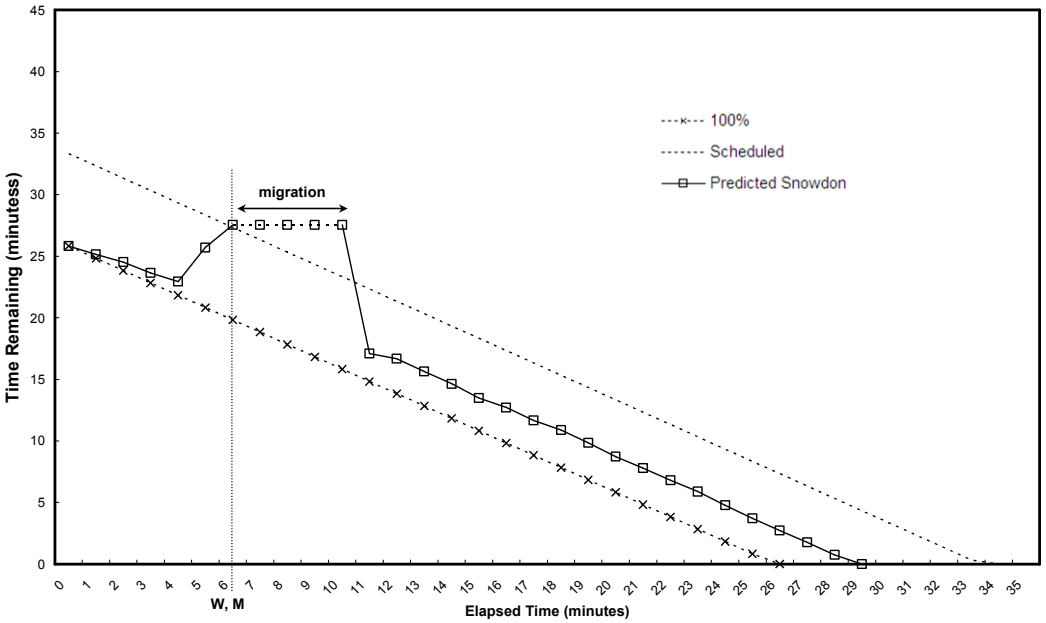

Figure 62 Time Remaining (minutes) vs. Elapsed Time (minutes) Single Provider with adaptation

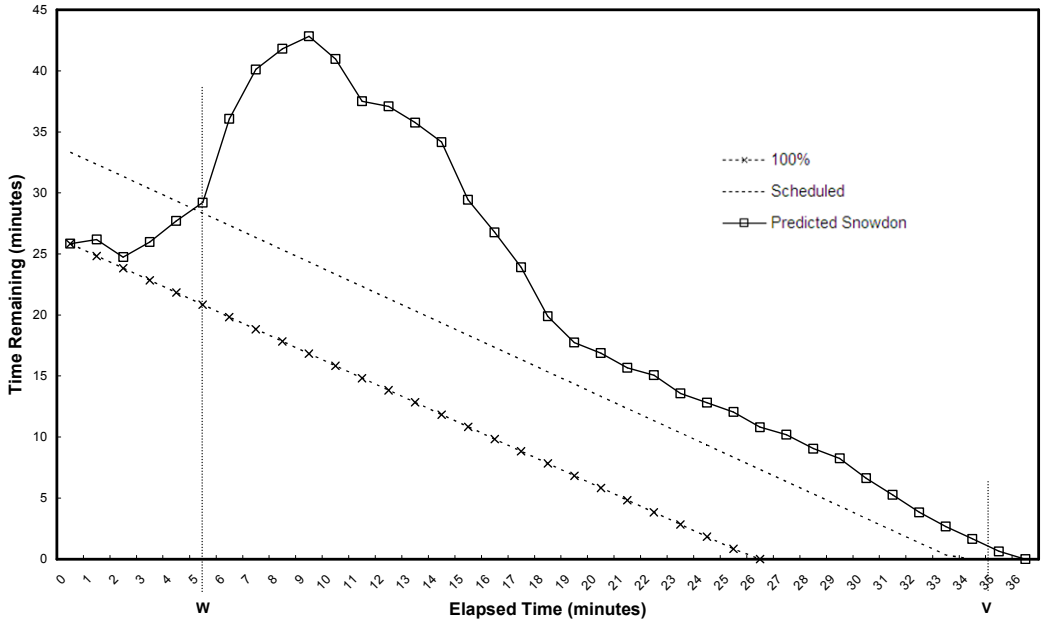

Figure 63 Time Remaining (minutes) vs. Elapsed Time (minutes) Single Provider without adaptation

Figure 64 and Figure 65 show the changes made to the SLA after the experimental runs illustrated in Figure 62 and Figure 63 respectively. In Figure 62, where adaptation is used the adaptive controller triggers a warning and migration after 6 minutes. The SLA

Manager updates the SLA with a warning and associated timestamp and a migration and associated timestamp in addition to details of the resource used after migration.

In Figure 63, where adaptation is not used the adaptive rule-based controller triggers a warning after 5 minutes. In addition to recording a warning, the SLA Manager also records a violation when the application fails to finish before the deadline is reached.

```
<completion>
     <time>2007-22-01T19:51:00Z</time>
     <warning>
           <timestamp>2007-22-01T19:23:00Z</timestamp>
     </warning>
     <migration>
           <timestamp>2007-22-01T19:23:00Z</timestamp>
           <resource>snowdon.leeds.ac.uk</resource>
     </migration>
</completion>
```

Figure 64 Changes to the SLA for experiment in Figure 62

```
<completion>
     <time>2006-06-02T11:53:00Z</time>
     <warning>
           <timestamp>2006-06-02T11:23:00Z</timestamp>
     </warning>
     <violation>
           <timestamp>2006-06-02T11:53:00Z</timestamp>
     </violation>
</completion>
```

Figure 65 Changes to the SLA for experiment in Figure 63

5.3.2 Run 2

Scenario 1 is conducted again with the submission of competing applications occurring in the second half of the execution. This run verifies the results from the first run and will determine if the SLA Manager implementation can maintain monitoring and adaptation for the full duration of an execution. It also represents a potential usage scenario in which an additional guarantee prevents the loss of results due SLA violation. This type of clause may be linked with pricing policy, with each migration attracting a price or penalty.

The results from the second run are illustrated in Figure 66 and Figure 67 and show the CPU time available to the SLA-bound application. In Figure 66 adaptation is used to migrate the SLA-bound application when triggered by the adaptive controller, in Figure 67 it is not. In both plots competing applications are introduced after 18 minutes and

immediately cause the CPU time received by the SLA-bound application to decrease. Where adaptation is used (Figure 66), the application is migrated on a another node within the Snowdon cluster and receives increased CPU time on the new resource. The CPU time available to the application which is not migrated (Figure 67) - remains low.

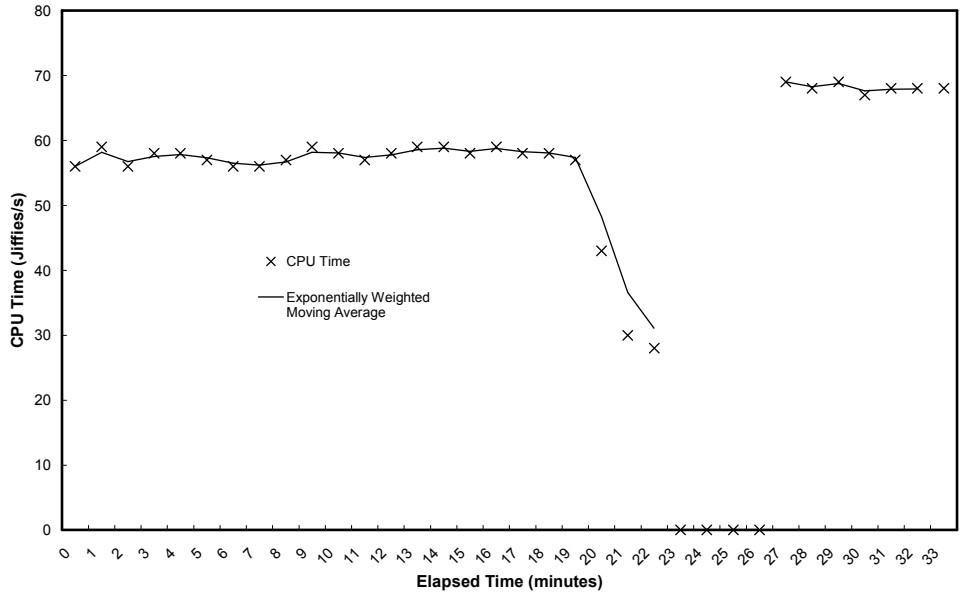

Figure 66 CPU Time (jiffies/s) vs. Elapsed Time (minutes) Single Provider with adaptation

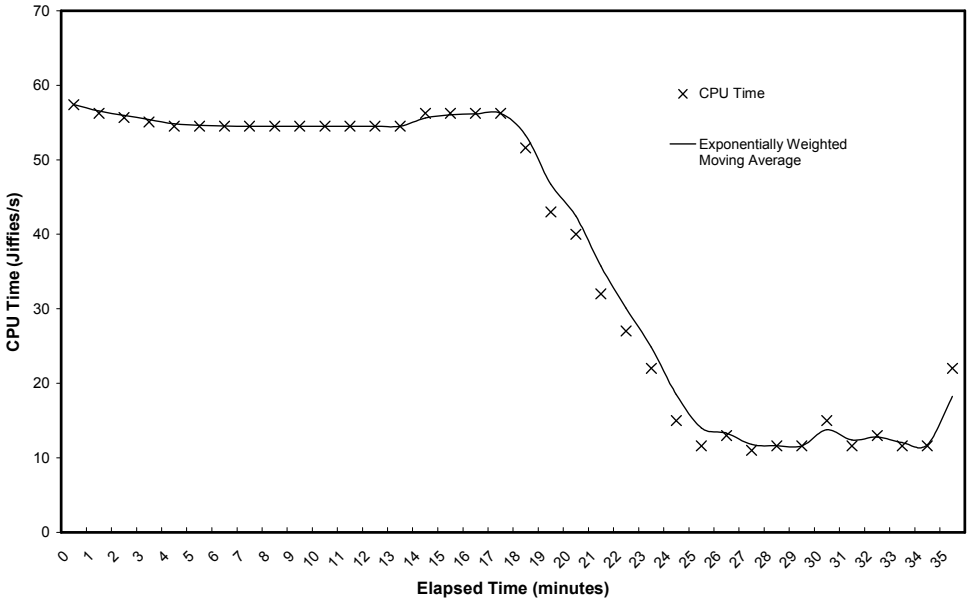

Figure 67 CPU Time (jiffies/s) vs. Elapsed Time (minutes) Single Provider without adaptation

The run also produced Figure 68 and Figure 69 which verify the results from the first run (Figure 62 and Figure 63). Where adaptation is used (Figure 68), the SLA Manager successfully migrates the SLA-bound application when $T_{remaining}$ exceeds $T_{schedule}$. The improved CPU time it receives on the new resource quickly decreases $T_{remaining}$ enabling the SLA-bound application to finish before $T_{schedule}$. Where no adaptation is used (Figure 69), the SLA-bound application shares the CPU with the competing applications. It continues to receive reduced CPU time and the difference between $T_{remaining}$ and $T_{schedule}$ increases until a violation is recorded when the deadline is reached.

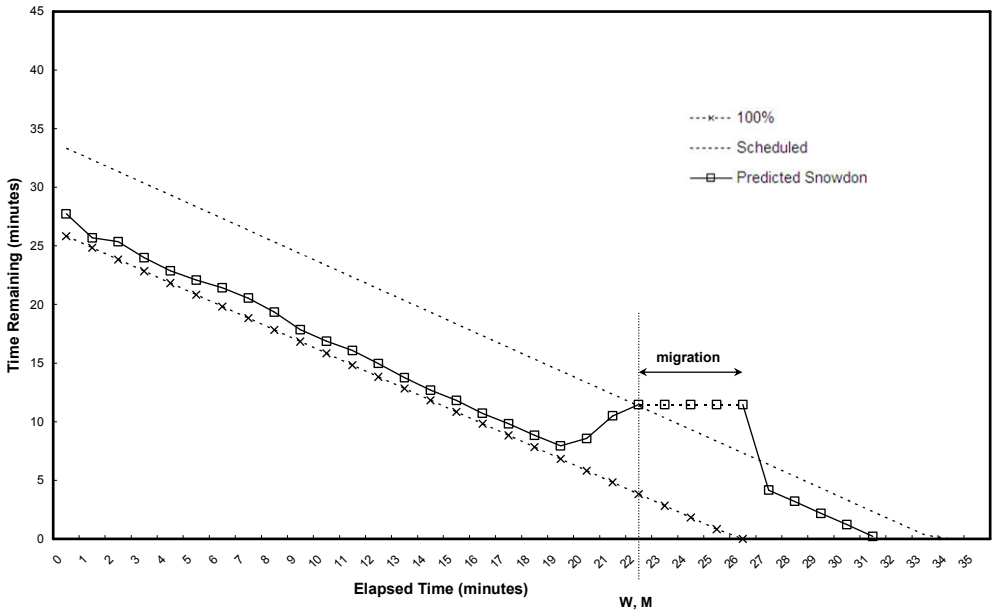

Figure 68 Time Remaining (minutes) vs. Elapsed Time (minutes) Single Provider with adaptation

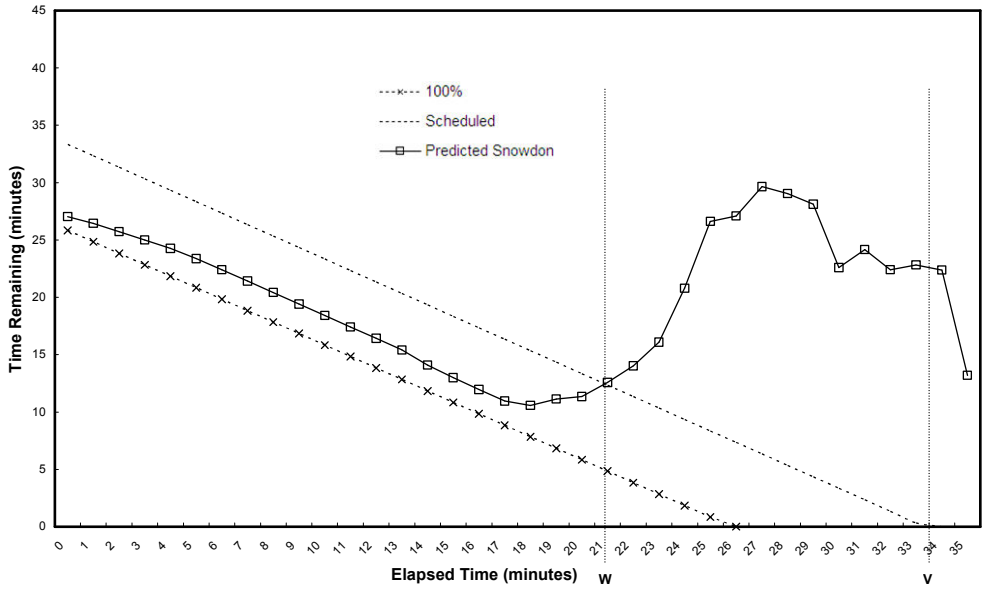

Figure 69 Time Remaining (minutes) vs. Elapsed Time (minutes) Single Provider with no adaptation

Figure 70 and Figure 71 illustrate the changes made to the SLA for Figure 68 and Figure 69 respectively. A warning and migration signal is once again successfully recorded after 22 minutes when $T_{remaining}$ increases beyond $T_{schedule}$. Where no adaptation is used a warning is recorded after 21 minutes and a violation after 34 minutes when the deadline is reached.

```
<completion>
     <time>2007-24-01T18:43:00Z</time>
     <warning>
          <timestamp>2007-24-01T18:32:00Z</timestamp>
     </warning>
     <migration>
          <timestamp>2007-24-01T18:32:00Z</timestamp>
          <resource>snowdon.leeds.ac.uk</resource>
     </migration>
</completion>
```

Figure 70 Changes to the SLA for experiment in Figure 68

```
<completion>
     <time>2006-06-02T15:42:00Z</time>
     <warning>
           <timestamp>2006-06-02T15:28:00Z</timestamp>
     </warning>
     <violation>
           <timestamp>2006-06-02T15:42:00Z</timestamp>
     </violation>
</completion>
```

Figure 71 Changes to the SLA for experiment in Figure 69

5.4 Scenario 2 – Multiple Providers

In this scenario, it is assumed that competing applications will again force the adaptive rule-based controller to migrate the SLA-bound application. However, spare resource capacity will be unavailable with the current provider forcing migration onto a resource with a new resource provider. Such a scenario is highly likely given the high demand usage pattern of the WRG. Using resources from a new resource provider from a different site and administrative organisation incurs a larger migration overhead than migration within a single provider. Additionally, the application will continue to execute on the resource used initially to execute the application - Snowdon, whilst the checkpoint file is transferred by the SLA Manager using GridFTP to Iceberg. Two instances of the application are active after migration, both will be illustrated in the results. The experiment will determine if the SLA Manager can function remotely as it functioned within the same domain. The rule base (Figure 58), initial prediction (26 minutes) and timing constraint (34 minutes) used in scenario 2 are identical to those used in scenario 1.

This scenario returned Figure 72 which shows two CPU time traces, one for Snowdon, the resource onto which the application is initially submitted; and Iceberg, the resource onto which the application is migrated. Competing applications are submitted to reduce the amount of CPU time available to the SLA-bound application. After migration is signalled the application restartes on Iceberg. Monitoring continues on Snowdon for the remainder of the experiment and is started on Iceberg after migration. The CPU time received by the application on Iceberg is close to the level it received on Snowdon at the start of the execution. The monitoring trace shows that this level is maintained until the end of the experiment; on Snowdon the CPU time continues at a reduced rate.

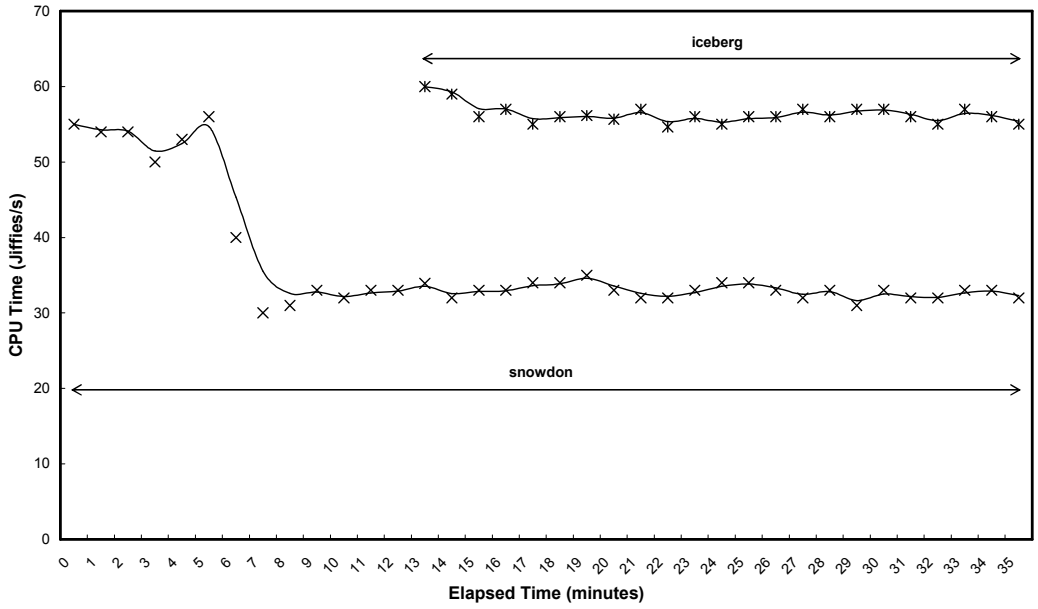

Figure 72 CPU Time (jiffies/s) vs. Elapsed Time (minutes) Multiple Providers

Also returned is Figure 73 which shows $T_{remaining}$ on Snowdon and Iceberg given the CPU time available to the SLA-bound application is Figure 72. When the SLA-bound application begins on Snowdon, $T_{remaining}$ is below $T_{schedule}$. The competing applications cause $T_{remaining}$ to increase beyond $T_{schedule}$, which triggers a migration signal on Snowdon. Unlike scenario 1, migration takes place onto a resource belonging to a different provider; Iceberg. Migration improves the CPU time available to the SLA-bound application wich enables it to complete prior to the deadline. On Snowdon, the original application continues to execute after the checkpoint is migrated and receives a reduced amount of CPU time which results in an SLA violation.

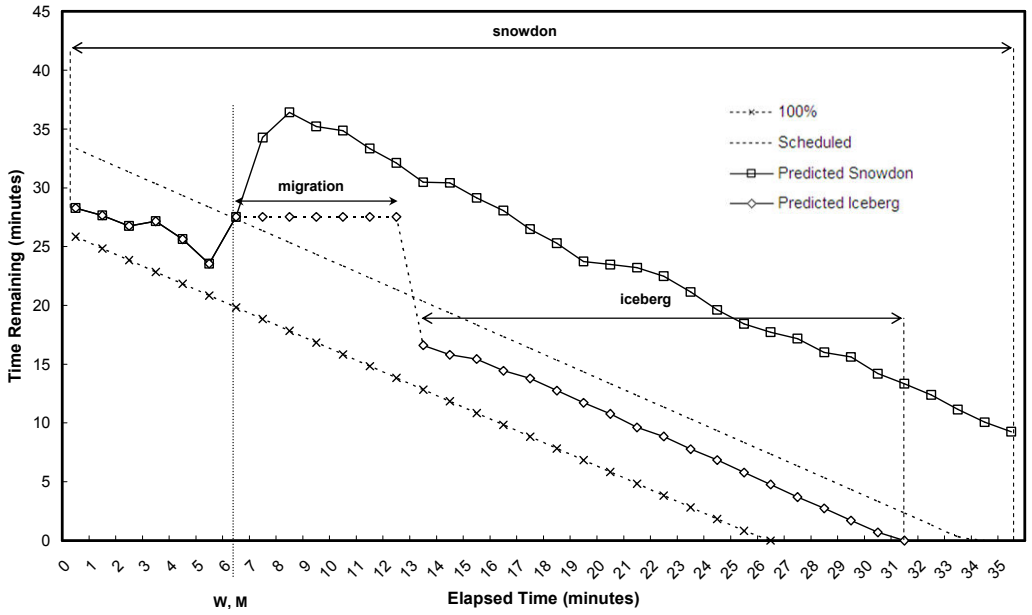

Figure 73 Time Remaining (minutes) vs. Elapsed Time (minutes) Multiple Providers

Figure 74 shows the changes made to the SLA during the run illustrated in Figure 73. A warning and migration signal is successfully recorded by the SLA Manager after 6 minutes. The migration element also indicates that Iceberg was the resource onto which the SLA-bound application was migrated.

```
<completion>
     <time>2006-30-07T15:27:00Z</time>
     <warning>
          <timestamp>2006-30-07T14:58:00Z</timestamp>
     </warning>
     <migration>
          <timestamp>2006-30-07T14:58:00Z</timestamp>
          <resource>iceberg.shef.ac.uk</resource>
     </migration>
</completion>
```

Figure 74 Changes to the SLA for experiment in Figure 73

5.5 Scenario 3 – DAME XTO Application

In this final scenario the SLA-bound application is replaced with the DAME XTO application; a description of which is provided in Section 3.1. Use of the DAME XTO application will demonstrate the SLA Manager working with an application closely related to the motivating scenario described in Section 3.1. It will also determine if

modifications are needed to the migration strategy when a different application is bound to an SLA. Unlike the application used to determine π, described in Section 4.2.1 and used in scenarios 1 and 2, XTO does not support application level checkpointing. Therefore, after migration XTO will need to be restarted from scratch on the new resource. The XTO application and dataset do not need to be migrated before XTO is restarted, they are resident on each of the WRG resources used for this experiment.

As with previous scenarios within this chapter, competing applications are introduced to the resource on which XTO is executing, in order to degrade its performance. It is hoped that this will trigger a rule within the adaptive controller which will restart the SLA-bound XTO application. Restarting XTO from the beginning and finishing before the deadline will rely more on the performance of the resource than it did in the experiments presented in Chapters 4 and 5. Therefore, XTO will be restarted on two WRG machines – Maxima and Iceberg, to demonstrate the affect of performance on the ability of the resource to finish the restarted XTO application. The rule base used is identical to the one used in scenario 1 and 2 and is presented in Figure 58.

In order to test the SLA Manager with XTO an SLA^3 is created which specifies a guarantee for the completion of a specific run by a deadline. The duration is derived from the total amount of CPU time needed in order for that specific run to complete. This is determined using a similar technique to that used in Chapter 4. A mean value is obtained from previous runs of XTO using identical input parameters and size of dataset. This technique is preferred for the purposes of testing XTO because it has a fixed configuration, dataset size and parameter value set which cannot be changed. Initial prediction indicates that XTO would take 21 minutes to complete. This is based on the assumption that it will receive the maximum potential CPU time on Snowdon for the duration of the experiment. A timing constraint of 29 minutes is used to simulate a strict timing deadline.

The scenario returned Figure 75 which shows the CPU time traces for Snowdon, the resource on which XTO is initially executed, as well as Maxima and Iceberg, the resources on which XTO is restarted. As with the application used in scenario 1 and 2, the competing applications decrease the CPU time available to XTO whilst it is

executing on Snowdon. Figure 75 illustrates that monitoring on Maxima and Iceberg begins after XTO has been restarted on those resources.

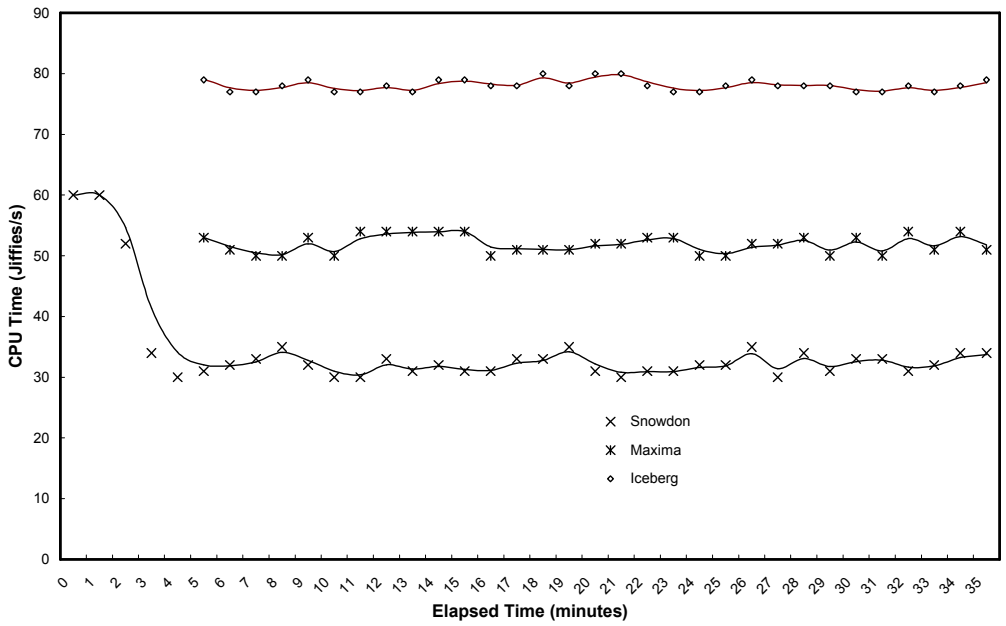

Figure 75 CPU Time (jiffies/s) vs. Elapsed Time (minutes) XTO

Also returned is Figure 76 which shows $T_{remaining}$ on Snowdon, Maxima and Iceberg given the CPU time available on each of those resources illustrated in Figure 75. The SLA Manager restarts XTO on Maxima when $T_{remaining}$ increases beyond $T_{schedule}$. The time taken to restart XTO is significantly less than the time taken to migrate the application used in scenarios 1 and 2 because the restart process does not involve the movement of a checkpoint file from the initial to the new resource. This is because XTO does not have application level checkpointing support and is restarted from scratch rather than from a position described in the latest checkpoint. When XTO starts on Maxima the CPU time available is insufficient to reduce $T_{remaining}$ below $T_{schedule}$; as a result XTO fails to finish before the deadline specified within the SLA. A second instance of XTO is started at the same time on Iceberg, a resource which is able to provide a greater amount of CPU time. The increased performance is sufficient for $T_{remaining}$ to reduce below $T_{schedule}$ and allow XTO to finish before the deadline.

---

[3] The SLA used is identical to the one used in scenario 1 and 2 and is presented in Figure 59.

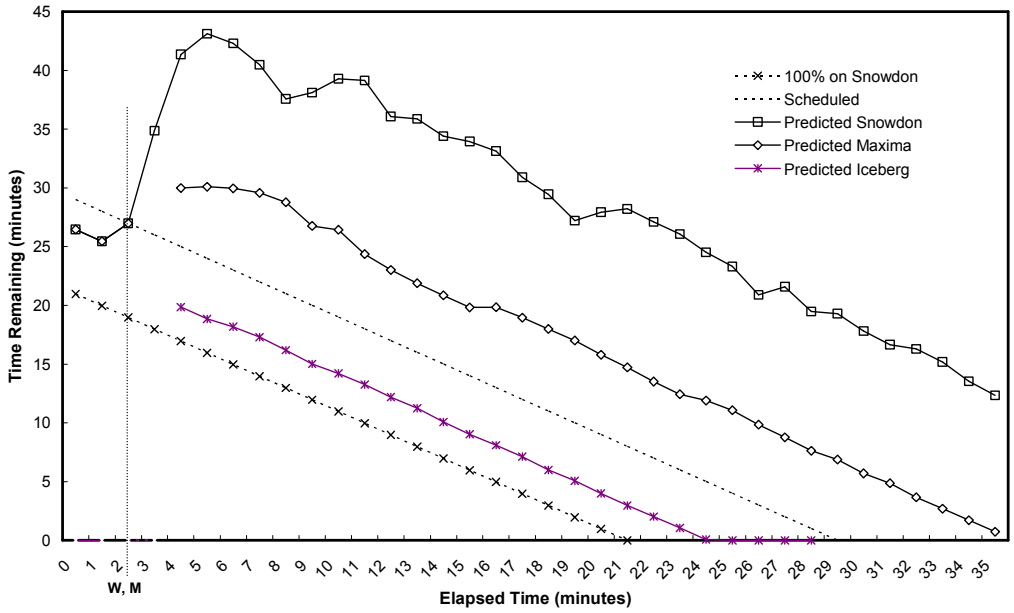

Figure 76 Time Remaining (minutes) vs. Elapsed Time (minutes) XTO

Figure 77 shows the changes made to the SLA during scenario 3. A warning and migration signal is successfully recorded by the SLA Manager after 2 minutes. The migration element also indicates that Maxima and Iceberg were the resources onto which XTO was restarted.

```
<completion>
     <time>2007-28-01T16:11:00Z</time>
     <warning>
          <timestamp>2007-28-01T15:44:00Z</timestamp>
     </warning>
     <migration>
          <timestamp>2007-28-01T15:44:00Z</timestamp>
          <resource>iceberg.shef.ac.uk</resource>
          <resource>maxima.leeds.ac.uk</resource>
     </migration>
</completion>
```

Figure 77 Changes to the SLA for scenario 3

This scenario demonstrates the importance of application checkpointing support within the SLA Management Architecture. Where an application has to be restarted from scratch, the SLA Manager must rely on the availability of more powerful Grid resources onto which it can migrate the application in order for it to meet strict timing guarantees. On the WRG this requirement limits the number of resources available to the SLA Manager. One way of mitigating the need for powerful resources is to only offer SLAs with slack deadlines for applications which have no checkpointing support. This would

allow for the increased amount of computation needed when restarting an application from the beginning. The need for more powerful Grid resources after migration places greater importance on good resource selection; a process which must take resource performance into account before an application is restarted.

5.6 Summary

This chapter has demonstrated the SLA Manager on a large distributed multiple domain Grid infrastructure, the WRG. This experiment is motivated by a need to test the software on resources from a large distributed Grid infrastructure which are subject to the resource management issues mentioned in Chapter 2 and a larger number of users and applications. The scenarios confirm that the SLA Manager and the adaptive rule-based controller in combination with application level monitoring can detect and record violations, warnings and migrations within the SLA. For the scenarios tested, the same components are successful in preventing violations to timing constraints when a Grid application experiences a performance degradation. This is the case for both single and multiple resource provider migration; although the latter displays a larger migration overhead. In both scenarios the resulting SLA demonstrates that the SLA Manager can successfully provide updates when they are detected. The third scenario demonstrates the SLA Manager with the DAME XTO application. The results show that applications with no checkpointing support must be restarted on powerful Grid resources if they are to finish before strict deadlines. This places more emphasis on resource selection based on performance prior to application restart.

In Chapter 6, an evaluation discusses the significance of the results in greater detail and considers them in the context of those from Chapter 4 and the related work.

Chapter 6

Evaluation

This chapter evaluates the performance of the deliverables and compares the results obtained in Chapters 4 and 5. Section 6.1 presents an overview of Chapters 4 and 5 and describes the methods used to compare the deliverables. Section 6.2 presents the evaluation methodology for the experimental results and in the context of the related work. Section 6.3 evaluates the performance of the solution for the experiments in Chapters 4 and 5. Section 6.4 evaluates the solution in the context of the related work. A summary of the chapter is provided in section 6.5.

6.1 Overview

Chapter 4 examines the performance of prototype SLA Manager implementation which uses system level CPU load to detect changes in Grid application performance. The experimental scenarios make use of a local Grid test-bed and examine the benefit of executing a Grid application with adaptive SLA Management compared with best-effort execution using a standard Grid middleware installation. An adaptive rule based controller reacts to changes in control variable state in order to determine if the Grid application should be migrated onto a new resource, for example in response to a performance slowdown.

In Chapter 5, the experimental objectives remain the same; to assess the added benefit of the SLA Manager when executing a compute intensive Grid application bound to an SLA expressing a timing guarantee. Monitoring takes place at the application level, rather than the system level and makes use of the CPU time received by the Grid application, as opposed to the CPU load of the resource. In order to quantify the predicted remaining execution time control variable, changes are made to Equation 1.

6.2 Methodology

The evaluation discusses the solution and its performance in the experiments and with respect to the related work. It also considers a non functional requirements evaluation.

- The experimental evaluation is structured to justify the solution by comparing the benefit of adaptive SLA Management over best-effort execution, migration using single or multiple resource providers and a comparison of early / late migration of Grid applications
- The solution is evaluated with respect to the related work within adaptive systems and SLA management and highlight the benefits and system drawbacks.
- The non functional requirements evaluation is structured according to the ISO-9126 [126] standard, which defines categories of software qualities (or non-functional requirements) with which to evaluate the solution.

6.3 Experiments

Timing guarantees are used to prove that the adaptive rule-based controller and SLA Manager can prevent a Grid application from violating a timing constraint when competing with other applications. The experiments in Chapter 4 are conducted on a local Grid test-bed with competing applications introduced to disrupt the system level CPU load. This metric is measured by the SLA Manager and used to derive a predicted remaining execution time, which in-turn is used by the adaptive rule based controller to determine if application migration is needed. In Chapter 5, the experiments are conducted on a large distributed Grid infrastructure and the measured metric is the application level CPU time. The method used to predict the remaining execution time, is altered to account for the change in measurement. The objective in both experiments is to react to slowdowns in system (or application) performance and prevent a Grid application from violating a timing guarantee specified within an SLA.

Figure 78 illustrates the role of the SLA Manager in scenarios where adaptive SLA Management is being used to manage the Grid application.

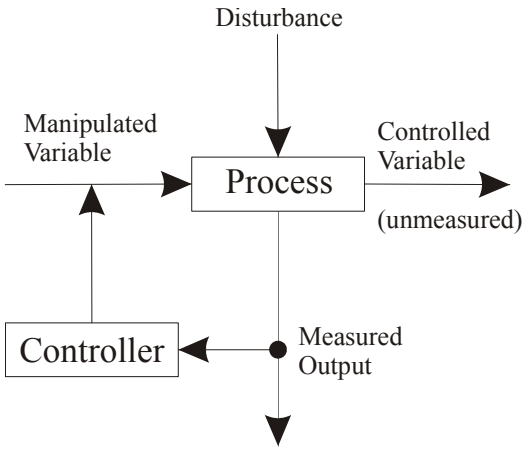

Figure 78 Inferential Controller

The adaptive rule-based controller forms a full control loop, with the ability to signal application migrations. The manipulated variable represents the amount of computation received by the application. The only action available to the SLA Manager which can manipulate the amount of computation received is migration of the application onto another resource. The measured output is the system CPU load (Chapter 4) or the application CPU time (Chapter 5). The disturbance is the amount by which the competing applications affect the system CPU load (Chapter 4) or the application CPU time (Chapter 5). The controlled variable is the predicted remaining execution time and the set point is the scheduled remaining execution time. The manipulated variable is altered by the controller through migration of the Grid application onto another resource. This manipulation itself imposes a disturbance on the process, however, it is assumed that migration will benefit the application.

Figure 79 illustrates the role of the SLA Manager in scenarios where no adaptive SLA Management is being used to manage the Grid application.

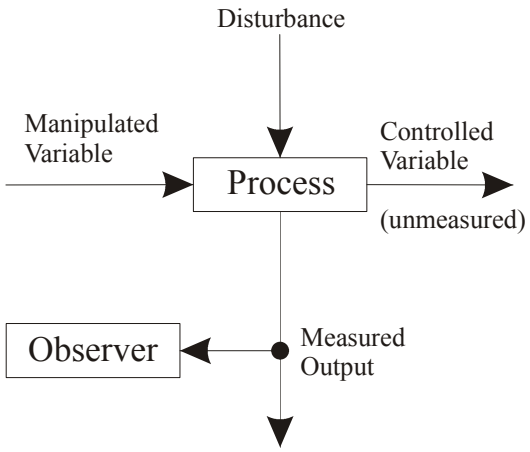

Figure 79 Inferential Observer

The Grid application is executing under best-effort QoS provision on a standard Grid middleware configuration. In this scenario, the controller is operating in the role of observer - there is no control loop. The SLA Manager simply observes the measured output and has no ability to change the manipulated variable, i.e. the resource on which the Grid application is executing.

The effectiveness of the SLA Manager can be seen clearly from the results illustrated in Figure 43 and Figure 50. Both scenarios showcase the execution of a Grid application on a local Grid test-bed, with competing applications submitted in the early stages of the execution schedule. Figure 43 highlights execution under best effort conditions with the SLA Manager operating as an observer. The monitoring technique, which samples a normalised measure of system load, successfully detects changes given the disruption caused by the competing applications. The adaptive controller implemented inside the SLA Manager triggers a warning which signifies that the Grid application will not meet its timing constraint. The increased system load average produced by the competing applications causes the performance of the Grid application to suffer. The lack of a full controller means a control action which can affect the manipulated variable is absent. The Grid application continues to execute under increased system load and the controlled variable, the predicted remaining execution time remains greater than the set point, the scheduled remaining execution time.

The presence of a full controller can be seen in Figure 50, by the response of the predicted remaining execution time of the SLA-bound application. The controller changes the manipulated variable by migrating the application onto another resource.

When the predicted remaining time exceeds the scheduled time the controller triggers a migration. Despite the initial overhead, the assumption is that migration will lead to a performance increase, which will enable the SLA-bound application to finish sooner compared to the same run executing under best-effort conditions. The migration overhead is attributed to additional middleware contacts with the new resource, the transfer of the checkpoint file and application restart costs associated with the middleware and the NBQS on the new resource. Despite this overhead, the intervention of the SLA Manager prevents the Grid application from violating the SLA timing constraint because of its ability to react to changes in the system load average.

The results presented in Figure 62 and Figure 63 illustrate the same scenario executed on the White Rose Grid. As with the results presented in Chapter 4, the presence of the SLA Manager and adaptive controller benefit the SLA-bound application; migration prevents it from violating the SLA timing constraint. As with the experiments on the local Grid test-bed the migration overhead observed, is approximately the same. The similarity is attributed to the corresponding communication setup between the SLA Manager, the local Grid test-bed and the WRG. In both experiments, communication occurs over a Fast Ethernet LAN network; therefore the overhead is likely to be the same.

This is in contrast to the results observed when a new resource provider is used for the migration. Differences between single and multiple provider migrations can be seen by comparing Figure 62 and Figure 73. The migration overhead is larger for migrations between resources belonging to different providers..In scenario 2 of Chapter 5 communication between the SLA Manager and Iceberg (section 5.1) occurs over a WAN. The increased latency and bandwidth of the WAN (compared with a LAN) contributes to the additional overhead observed in Figure 73. Despite this, migration benefits the Grid application and an SLA violation is prevented. The original application on Snowdon (section 5.1) continues to execute with reduced performance and fails to complete before the timing guarantee.

The addition of competing applications in the later stages of the execution schedule demonstrates that the solution can maintain monitoring and control for the duration of the experiment. It proves that migration late in the application schedule, can be justified to prevent the total loss of results due to SLA violation. This is a potential usage

scenario in which an additional guarantee is made to prevent the total loss of results due to the expiry of the timing guarantee. An end-user could specify a clause within the SLA in which the results of the application must be returned at all costs and to renegotiate access to new resources until this is achieved. This type of clause may be linked with a pricing policy, with each migration attracting a price or penalty.

The effectiveness of the SLA Manager when used with the DAME XTO application can be seen in Figure 76. The results show Maxima was unable to finish executing the restarted XTO instance before the deadline, despite being free from competing applications. Iceberg, which is a more powerful computational resource, was needed to complete the execution before the deadline. Using the SLA Manager with an application without checkpointing support means that the application must be restarted from scratch whenever a migration is needed. Therefore, the SLA Manager must rely on the availability of more powerful computational Grid resources in order to meet strict timing guarantees. On the WRG this limits the choices available to the SLA Manager. One way of mitigating this is to offer SLAs with deadlines which are less strict and allow for the additional computation needed if restarts are required. Promoting the uptake of such SLAs would be possible if the SLA Management Architecture formed part of an economic usage model which could offset the disadvantages against cost. The result places greater importance on resource selection based on performance; a process which must be taken into account before an application is restarted.

In the experiments on both the local Grid test-bed and the WRG, the SLA specification provided mutable elements which record the actions of the adaptive rule-based controller. Changes to the SLA were made automatically by the SLA Manager and reflected the warnings, migrations and violations. In each experiment the SLA Manager successfully updated the SLA document and recorded a timestamp reflecting the time at which the action was signalled. In the case of migration between resource providers, the resource name is recorded along with a timestamp. This successful demonstration of automatic updates, allows an end-user to trace through the actions which were taken during the execution of their Grid application task.

6.4 Related Work

The SLA Management Architecture and its functionalities have been demonstrated in the context of the DAME business process. These functionalities introduce adaptive SLA management to Grid based systems to improve Grid application QoS support. The results show the added benefit brought by the SLA manager and its ability to prevent violations to SLAs which specify timing constraints. In the experimental scenarios, the presence of adaptive SLA Management lead to a performance improvement which was brought about through application migration; using either resources from the same or a different providers. The SLA specification not only to specifies requirements and guarantees, but also SLA provenance, recording application management decisions during the execution.

If the solution has achieved its set objectives, how does it compare with related work within this area. The areas of contribution for this work are SLA Management and adaptive application executions. Other smaller contributions have been made to the areas of Grid monitoring, SLA specification, prediction of application execution times and control theory for applications executing on Grid systems. The solution draws together each functionality into a single system; something which is novel within Grid based systems. Solutions in many of these areas have already been proposed by Sahai et al [82] and Leff et al [84] (SLA Management), Huedo et al [106, 111] and Vadhiyar & Dongarra [107, 112], (adaptive systems) and Kapadia et al [97] (resource prediction systems).

The SLA Management Architecture and the system implementation, the SLA Manager have brought these functionalities together to demonstrate adaptive SLA management for Grid based systems. When compared with each of the solutions mentioned above, the SLA Management Architecture and the implementation offers a simpler solution to achieve similar functionality or has enhanced the solution by combining functionality.

The adaptive systems presented by Huedo et al [106] and Vadhiyar & Dongarra [107, 112] work with Grid applications which self monitor by providing timings for sections of computation during the execution. The approach taken by the SLA Management Architecture removes responsibility for monitoring away from the application and implements its own external system. This approach means that all computation received

by the Grid application is used solely for the purpose the application was intended, rather than the secondary task of monitoring. The measurement overhead associated with self monitoring is the communication of information from within the application to the adaptive decision maker. This is likely to carry a premium, especially if system or I/O calls are necessary. The SLA Manager has been demonstrated using a monitoring script which uses the /proc file system to observe changes in application performance. In comparison to self monitoring the measurement overhead is small because the information is recorded in /proc regardless of whether it is used by the SLA Manager. A secondary, more practical advantage of external monitoring is that application re-engineering does not have to be performed. In commercial Grids this is often impossible because the application code may be subject to IP restrictions which prevent re-engineering. Finally, in order for application self monitoring to be possible, an application must perform the type of computation which can be predicted prior to runtime.

The SLA Manager implements an adaptive rule-based controller to infer control actions given changes to control variables describing an applications performance, relative to an execution schedule. This approach is inspired by process control theory [85] and involves the implementation of a control loop which reacts to performance degradations by attempting to migrate the application onto another resource. Vadhiyar & Dongarra [107, 112] rely on Autopilot [127, 128] to provide adaptive decisions. Autopilot uses rule based control to compare application monitoring information with a performance prediction in order to infer control actions based on rule based control. Huedo et al [106] implement a performance monitor which compares application monitoring information with a performance profile to infer rescheduling actions.

The SLA Manager is designed to make use of an external resource broker, the SNAP Three-phase commit broker [116, 117] to discover and select resources. Huedo et al [106] make use of the Grid Way framework for resource discovery; whereas Vadhiyar & Dongarra propose the use of the GrADS system to achieve the same functionality. This decision limits the scope of migration to providers which have a fully working implementation of the GrADS or Grid Way systems. The SLA Manager's use of an external resource broker means that resource discovery and SLA Management are not tightly coupled. This allows the SLA Manager to use any resource broker, so long as the API remains the same. Use of the SNAP Three-phase commit broker [116, 117]

provides rapid and efficient selection of resources due to the three phase commit protocol.

The presence and configuration of GrADS [83] and Grid Way [113] add further to the submission complexity and overhead between layers. The experience gained by deploying the SLA Management Architecture on the WRG has shown it is difficult to maintain a correctly working standard Grid middleware setup across a VO. Each contributing organisation, Leeds, Sheffield and York have separate administration departments and implement slight differences to middleware setup. These differences are most apparent in the NBQS configuration which specifies access permissions for resources and users. Achieving a reliable setup which adds an additional layer on top of a standard Grid middleware setup and maintaining this across a VO would be even harder. The SLA Manager is deployed in a single location to perform remote monitoring and control and does not rely on additional layers above the standard Grid middleware. In the DAME example, the SLA Manager is deployed alongside the application server housing the DAME portal and does not need to be resident on any other Grid resources. The only requirement is the use of SGE as the local resource manager. The reason for this is not compatibility, only practicality. SGE is the local resource manager on both the WRG and the local Grid test-bed, the opportunity to test other resource managers on a large scale on either the local Grid test-bed or the WRG was not available. However, the functionality of local resource managers (SGE, PBS and LSF) which are compatible with the Globus middleware is broadly the same. The task of enabling the SLA Manager to function with a different local resource manager could be achieved through modification of the Grid middleware – local resource manager interface

The use of SLAs is one area of functionality which is absent from the adaptive systems presented by Huedo et al [106] or Vadhiyar & Dongarra [107, 112]. Provision for an SLA specification for formally expressing requirements and guarantees is absent from both. In contrast, the SLA management systems presented by Sahai et al [82] and Leff et al [84] do provide formal SLA specifications and methods for monitoring guarantees. Sahai et al [82] do not attempt to implement the adaptive capabilities which are present within the SLA Manager. Instead they focus on the collection and aggregation of monitoring information as methods for reliable verification of guarantees within the SLA. In contrast, Leff et al [84] make provisions for dynamically adapting resource availability with varying demand, but this is applied at the system level rather than the

application level. Decisions are taken from the providers' perspective, ensuring efficient resource utilisation and profit, rather than from a customer perspective, ensuring application finishing time.

The SLA specification used within the SLA Manager allows updates to reflect actions taken during application management. Support for SLA provenance is a major addition which enhances non repudiation mechanisms by improving agreement traceability and validation.

In DAME, both traceability and validation are a priority for two reasons:

- the commercial partner places IP restrictions on the results obtained from the Grid application. In order to trace potential breaches in security, knowledge of the resources on which the application has executed is required.
- the results obtained, affect health and safety on commercial aircraft. If results are inaccurate, it is important that the resource on which the execution took place can be traced to prevent further errors.

The SLA specification is motivated by the lack of a single specification within the Grid domain which can satisfy the requirements of adaptive SLA management for Grid based systems such as DAME. Job submission description elements are motivated by the Job Submission Description Language (JSDL) [67]. The description of parties, purpose and scope is influenced by the WS-Agreement [69] specification. Service Level Objective (SLO) elements are motivated by the Web Service Level Agreement (WSLA) [68] specification. The provenance record is motivated by the Usage Record (UR) [70] specification.

6.5 Non Functional Requirements

The non functional requirements are evaluated below according to 4 criteria from the ISO-9126 [126] standard: usability, efficiency, reliability and portability.

The **usability** of the SLA Manager can be evaluated by examining the GUI (Figure 80) through which a user specifies an SLA request and launches a Grid application. The area bounded by box A in Figure 80 illustrates fields which take application task

requirements. The user can initiate a number of SLA templates for the same task, perhaps with slightly different requirements, and these will be displayed within box B. By selecting one of the SLA templates in box A, and clicking the negotiate resource button, they can discover the resources which match the request. The resources matching the request are displayed in Box C. In order to submit the Grid application using a specific resource, the user selects the resource in box C and clicks the submit job button. Box E represents a console which captures execution logging information from the Grid application. The GUI improves usability by displaying the SLAs and jobs which the user has running. This is an improvement over command line interface interaction, which does not present a global view of the system.

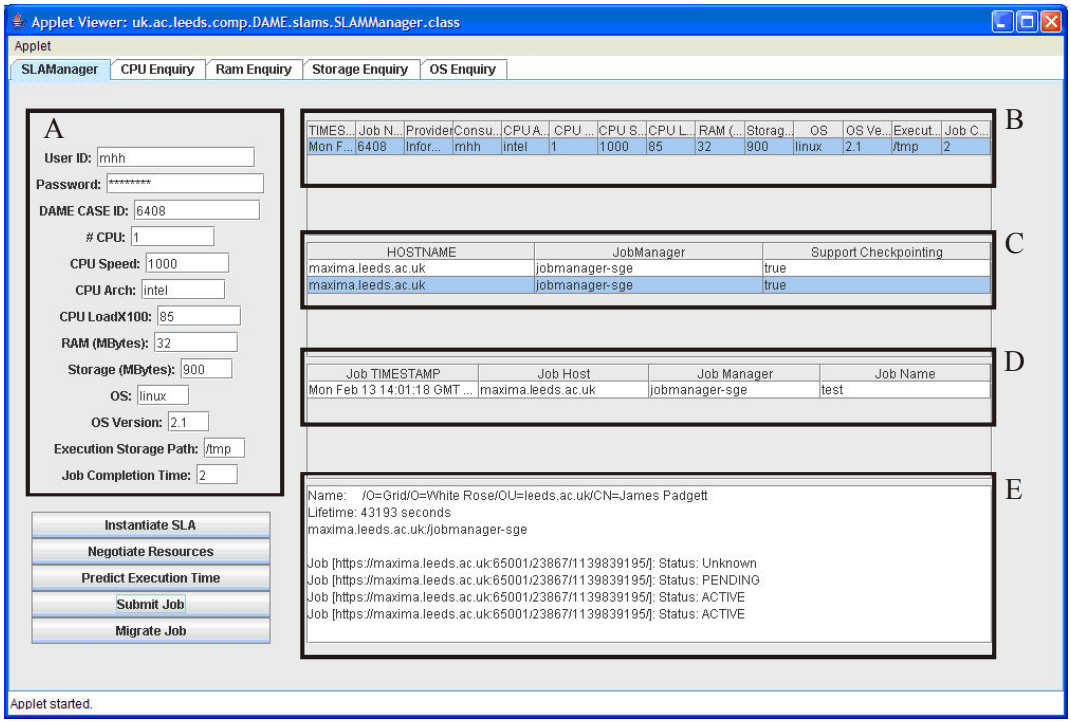

Figure 80 SLA Manager GUI

The **efficiency** of the solution can be evaluated using a number of criteria. From the perspective of application monitoring, the SLA Manager is more efficient than those systems [106, 107] which use application self monitoring. This is down to the use of the /proc file system to monitor Grid application performance. The move from system to application level monitoring improved efficiency by targeting a metric which responds instantly to performance degradations. When using system level monitoring to detect

application performance, a measurement latency delays the affect of the performance degradation on the metric being measured.

As a consequence of changes in monitoring, the method used to estimate the remaining execution time also changes. This contributes to an increase in efficiency by basing the calculation on monitoring observations which do not suffer from monitoring lag.

The addition of execution time estimation using historical observation improves efficiency by taking the decision away from the end-user. Historical observations based the estimate on previous applications runs, whereas user estimation relies on human inference.

The relative position of decision maker and monitored process effects the data transmission latency between the two entities. A higher latency can prevent real-time monitoring and control because once a control action has been decided the state of the process may have moved on. The control action may not be appropriate for the up-to-date state of the process. The SLA Manager is designed to use remote monitoring and control and is always situated remotely from the Grid application. As a result, the data transmission latency is larger than in systems which house the decision maker and process together [118].

Using migration to implement application adaptation carries with it a performance overhead prior to any performance gain. This is an inefficient method of controlling the manipulated variable; but it is the only control action available to the SLA Manager. When compared with other adaptive systems [106, 107], the solution is no more or less efficient for its use of migration because they too use migration. Efficiency is improved with the use of checkpoints to restart the application after migration; a technique which is overlooked by other adaptive systems [118].

The **reliability** of the solution has been proved both on a local Grid test-bed and a large scale distributed Grid infrastructure, the WRG. The experiments in which migration is performed late in the execution schedule, demonstrate that the monitoring and control used within the solution is reliable for the full term of a Grid execution.

The **portability** of the SLA Manager is assured because it is implemented using an interoperable programming language, JAVA. This means that the SLA Manager can be

deployed in any location which supports a Java Virtual Machine (JVM). The SLA Manager uses standard API's from the Grid research community to interact with Grid end systems, an example includes the JAVA Cog Kit [46]. The SLA Manager does not require installation on any Grid resource in order to function; therefore the software does not have to co-exit with other Grid middleware components.

6.6 Summary

This chapter presents a reminder of the experimental procedures in Chapters 4 and 5 and an evaluation methodology which describes how the experimental results are compared. The benefits of the solution are considered in the context of the experimental results and justified with respect to the related work. Four non functional requirements are evaluated including usability, efficiency, reliability and portability.

In Chapter 7, conclusions are reached including a summary of the work presented in each chapter and the results. The direction of future research is discussed, including ways in which the work could be refined or the effectiveness improved.

Chapter 7
Conclusion

Section 7.1 presents a summary of the thesis - chapter by chapter. A synopsis of the thesis results in Section 7.2 includes a discussion of the effectiveness of the methodology. System drawbacks are considered in Section 7.3 and future work in Section 7.4. An overall conclusion is presented in section 7.5.

7.1 Summary of Work

The work presented in this thesis demonstrates an adaptive SLA Management Architecture supporting execution of Grid applications with SLA based timing constraints in Grid based systems such as DAME.

Chapter 2 introduces Grid computing as the broad area in which this research is conducted. Architectural philosophies, including OGSA and WSRF are defined including Grid interoperability, integration, virtualisation and management.

A discussion of Grid resource management and scheduling narrows the research onto non-trivial QoS for Grid applications. To highlight resource and application monitoring/scheduling, a number of brokers, schedulers and monitoring tools are described.

Service Level Agreements are presented as languages which formalise QoS requirements. A number of specifications are actively used within the Grid research domain, these are discussed. A survey of SLA Management systems identifies the functionalities and weaknesses of a number of implementations.

Prediction of application execution times, checkpointing and methods for adapting Grid applications during runtime are also discussed as technologies needed to support adaptive SLA Management Architecture.

Chapter 3 presents a DAME scenario as motivation for the SLA Management Architecture. An application model indicates the requirements of applications in order to enable them to be used adaptively. The SLA Management Architecture is introduced and describes key functional requirements needed for adaptive SLA Management in Grid based systems. Use cases develop the requirements, assigning functionality to different parts of the system implementation. System components are derived and the intended usage of the system is presented. Implementation issues are discussed and the SLA specification is presented.

In Chapter 4, a prototype SLA Manager is tested on a local Grid test-bed. Monitoring occurs at the system level using the CPU load of the resource. The first scenario tests the SLA specification using a performance guarantee. Its purpose is to prove the ability of the SLA manager to verify, through effective monitoring, the metric specified in the SLA. The second and third scenarios test the ability of the adaptive rule-based controller to signal warnings, migrations and violations. The control loop uses an estimate of the remaining execution time and an execution time schedule to determine the performance of a Grid application. The controller tries to prevent the Grid application from violating an associated timing constraint within the SLA. The experiment is run with a full controller and an observer to simulate a Grid application executing with and without adaptive SLA Management adaptive. The resulting SLA fragments are shown to demonstrate SLA provenance.

Chapter 5 tests adaptive SLA Management on a large scale Grid infrastructure, the White Rose Grid. A different method of monitoring and estimating the remaining execution time is used. Monitoring occurs at the application level; performance is measured using CPU time. The purpose of the experiment is to prove the ability of the SLA Manager on a large scale distributed Grid infrastructure. The scenarios test the ability of the adaptive rule-based controller to signal warnings, migrations and violations. The control loop uses an estimate of the remaining execution time and an execution schedule to determine the performance of a Grid application. The experiments highlight the benefit of adaptive SLA Management over ‘best-effort’ execution using a

standard Grid middleware setup. In addition, a comparison is made between single and multiple provider migration as well as use of the DAME XTO application.

Chapter 6 presents an evaluation of the SLA Manager performance on both the local Grid test-bed and a large scale distributed Grid infrastructure, the WRG. Comparison is made with the related work and an evaluation of the non functional requirements is performed.

7.2 Results

The objective of the work presented in this thesis is to enhance Grid application QoS support within Grid based systems using adaptive SLA Management. The method is demonstrated using the DAME system and the benefits tested on a local Grid test-bed and the WRG. During the development it became apparent that a number of other contributions were needed to support this objective. These contributions included an SLA specification which maps application requirements into a formal, machine readable format, including elements which can be used to record adaptive activities such as warnings, migrations and violations automatically within the SLA document. Methods of monitoring were needed to allow precise measurement of CPU time for a given Grid application process. An adaptive rule-based controller was needed to infer control actions given the state of two control variables. A method was needed which based application execution time prediction on historical observations rather than end-user estimations.

A system implementation was developed and experiments conducted in two Grid environments, a local Grid test-bed and the White Rose Grid. They show:

- In the scenarios tested, the SLA Manager system implementation is successful in preventing a CPU intensive Grid application from breaking a timing guarantee specified in an SLA. Adaptation through migration has proved useful in reducing the execution time of an application when the performance of the system and application is reduced.
- The monitoring techniques employed in both experiments can detect Grid application performance changes, however the method which uses application

level monitoring provides information which more accurately reflects the performance of the Grid application.

- The adaptive rule-based controller, functioning either as a full controller or an observer, can signal warnings, migrations and violation as based on the control variables: the predicted and scheduled remaining execution time.
- The SLA specification records application management decisions as SLA provenance to improve traceability and validation for the end-user.
- A learning based initial prediction technique is demonstrated which decreases the chance of large estimation errors, typical of end-user estimation. Whilst a performance comparison of the technique is not performed, it can be concluded that the technique improves upon other implementations which implement adaptive Grid application provision [118].
- Where the SLA Manager is used with an application without checkpointing support, more emphasis is placed on the selection of resources prior to migration. Selection must indicate a resource which will be powerful enough to complete the application run from scratch before the deadline.

Application migrations between different providers suffer a greater migration overhead than between resources of a single provider. This is attributed to an increased number of Grid middleware contacts during migration and communication over a WAN rather than a LAN.

Application migrations which take place late in the execution schedule demonstrate that for some long running applications this action can benefit from the adaptive decision process. This is despite the fact that time delays due to migration represents a bigger percentage of the remaining execution time in the latter stages of the execution schedule. With this in mind, if an application enters the latter stages of its execution schedule and a migration is needed, it may be more appropriate to use resources with the current provider in order to minimise the migration overhead.

7.2.1 Methodology

Methodologies were introduced in section 1.4 to indicate the approach used to solve the research objectives. Each one is discussed to assess its merits.

- *Capturing Requirements*. The requirements were drawn from the DAME project, allowing them to be tightly specified around a business process being developed within that project. Use of DAME requirements lead to a clearer justification of the research objectives. However, the solution could be easily applied in other domains and is not specific to DAME.
- *UML Modelling and Experimental design*. The Unified Modelling Language (UML) [3-6] provided a concise and clear method for taking requirements through to solution. In addition, it allowed for the solution to be documented according to accepted software development practices.
- *Gathering historical data*. The requirement for accurate initial predictions is the basis for accurate estimation of the remaining execution time during runtime. The use of end-user estimations was insufficient for use within the DAME project, instead a method had to be based on historical observations from previous runs. The only method of achieving this was to obtain a set of previous application runs in order to build up a historical dataset.
- *Performance comparison* examined the thesis contributions with reference to a standard Grid middleware deployment which represented current best-effort application management. The comparison also considered the contributions using single and multiple resource providers and early vs. late migration. Such comparisons allowed the benefits of the solution to be illustrated but also determines if it can be applied in different scenarios.
- *Performance evaluation* allows the contributions to be clearly justified by evaluating the result of the comparisons. Once the justification has been established, the work can be evaluated in a wider context against related work.

7.3 System Drawbacks

After testing and evaluating the SLA Management Architecture and the system implementation, a number of system drawbacks have been identified.

- The use of historical observations to generate execution time estimates requires a dataset of previous values in order to be effective. In order to populate the

dataset with samples, the application has to be run many times using different input parameters.

- When a decision is made to migrate an application, the overheads due to file transfer and queuing are not considered. The dominant overhead is assumed to be that associated with the Grid middleware.
- When a decision is made to migrate an application, no assessment is made of the current status or performance of the resource onto which the migration will take place. It is assumed that because the new resource is free of competing applications it will offer a performance gain over the current resource.
- If the NBQS is to kill or suspend the Grid application whilst it is executing, the SLA Manager will not automatically trigger a migration, and the execution will stall. Listening for the NBQS suspend signals would allow the SLA Manager to initiate a migration of the Grid application if it were suspended by SGE.
- The only control action available to the SLA Manager is migration to another resource. This control action carries with it a performance overhead before any performance enhancement is felt by the application. The assumption is that the SLA Manager does not have administrative authority to suspend competing applications.
- The system has only been tested with a specific type of application, a compute intensive batch job. Other types of applications are run within Grid systems, examples include interactive visualisations and MPI jobs.

7.4 Future Work

The following ideas represent future work to refine or extend the research presented in this thesis.

- The SLA Management Architecture could be extended to work with an ensemble of applications where many instances need to be run and covered by a single SLA. For each application instance within the ensemble an instance of the SLA Engine could be used to monitor and adapt to performance degradations – functionality which is currently supported. However, a new

component would be needed, a master SLA Engine, responsible for coordinating all application instances within the ensemble. This component would take on a metascheduling role with the power to suspend and restart instances to achieve the optimum performance for the SLA rather than a specific application instance. Metascheduling decisions could be implemented in a rule base controlled by the adaptive controller. For example, an instance which is executing with a predicted finish time well before the deadline may be suspended and replaced with an instance which is behind schedule or one which may not have started yet. This may occur if the number of application instances is large and the Resource Broker is unable to reserve enough resources for the entire job set to start simultaneously. This approach is counter-intuitive at the instance level but may optimise the performance of the SLA.

In addition to these changes, the SLA specification proposed in Section 3.8 would need to be altered to include elements which could describe the number of instances within the ensemble. The SLA would need finer grained warning, migration and violation elements in order to describe which instances the warnings, migrations and violations applied to.

The SLA Management Architecture would still rely on an initial prediction for the length of time each application instance would take to execute. The deadline offered within the SLA would be a function of these individual times and other factors such as availability of resources before the start and average queuing times for instances not scheduled at the start.

The monitoring techniques demonstrated in Chapter 4 and 5 could be applied to monitor each application instance, however the master SLA engine would need information regarding SGE (or other LBQS) queue status in order to make metascheduling decisions. Most LBQSs support this information but not across domains within a Grid Infrastructure. Therefore, the master SLA Engine would have to rely on a higher level monitoring service such as MDS or a Resource Broker which would have access to information from across the domains.

- Currently, it is assumed that the resource onto which the migration occurs will offer the application greater performance. Use of the DAME XTO application

during scenario 3 in Chapter 5 demonstrated the importance of assessing resource performance before migration. Assessment of the performance gain offered by the new resource prior to migration is one method which would achieve this. Comparison of the predicted remaining execution time on the current and new resource and the overhead due to migration would provide a measure of performance gain. The overheads due to migration would include potential variation in checkpoint transfer time and NBQS queuing time on the new resource. A simple method of estimating the rescheduling gain is shown in Equation 5 and used by Vadhiyar and Dongarra in [107] to decide whether application adaptation should proceed.

$$gain = \frac{\left(T_{remaining}^{current} - \left(T_{remaining}^{new} + T_{roverhead}^{migration}\right)\right)}{T_{remaining}^{current}}$$

Equation 5 Calculating the potential gain of migration

- An extension to the SLA Management Arcitecture may place greater emphasis on predicting overheads due to file transfer or queuing. A solution may employ the learning based initial prediction technique described in Section 5.2.3 to predict delays using historical observations. Sampling the average bandwidth between the current resource and the new resource and the average job queuing time from the NBQS on the new resource metrics which could indicate possible overheads.

- An extension to the adaptive controller would enable fuzzy rather than rule based control. The use of fuzzy control is useful when the control domain is continuous. In the experiments in Chapters 4 and 5 the control domain is not continuous because the ability of the SLA Manager to affect the performance of the Grid application is controlled by the resource on which the Grid application is executing.

  If Grid resources were to support a priority based execution policy, the SLA Manager could manipulate the priority given to the Grid application, enabling it to receive a greater amount of CPU time compared to the competing applications. Nice values are a way of controlling process priority on POSIX operating systems. An experiment might manipulate this value in order to

affect the amount of CPU processing received by the Grid application. If monitoring suggests the application will meet the timing guarantee, the application could be given a lower priority, which would decrease the amount of CPU time received. Equally, if performance is low and the predicted remaining execution time is higher than the schedule, the application could be given a higher priority which would increase the amount of CPU time received by the application.

- Extending the technique to include a broader range of applications would enhance the capability. The current solution is tested using a CPU intensive application and rule based adaptation. The solution should be tested with other application types (interactive or MPI) to determine if it is suitable in other fields.
- The architecture may benefit from a resource acquisition protocol which implements an economic model to control resource usage. An implementation based on GESA [71] could be applied to allow resource usage to be linked with usage.

7.5 Overall Conclusions

The work presented in this thesis provides one route to achieve the objectives and answers the research questions presented in Chapter 1. The SLA Management Architecture provides a blueprint for the functional requirements needed to support adaptive SLA management in Grid based systems. In the scenarios tested, the system implementation successfully prevents violation of an SLA with timing constraints given performance degradation of a Grid application caused by competing applications. The SLA specification contains elements which record provenance within the SLA, to reflect actions taken by the SLA Manager during application management. This ability improves non repudiation mechanism by improving SLA traceability and validation.

Adaptive SLA Management is a potential route to commercialisation of Grid computing. For users, it represents an opportunity to share resources and services, outsource parts or all of their process or avoid the high costs of investing in an in-house computing infrastructure. However users may intend to take advantage of Grid

computing, assurances that their processes will complete according to their requirements is a priority. Current Grid middleware solutions leave users with nothing if their process fails. Adaptive SLA Management is one solution which can meet their requirements. The work in this thesis has demonstrated that users requiring timely execution of Grid applications can use adaptive SLA Management to monitor and control their processes in order to prevent missed deadlines. A situation which could not be achieved using currently available Grid middleware.

Appendix A

Figure 81 shows the full source code listing for the Grid application used within the experiments presented in Chapter 4 and 5.

```
#include <stdio.h>
#include <unistd.h>
#include <sys/stat.h>
#include <math.h>

#define TRUE 1
#define FALSE 0

static char        checkpoint_directory[255];
static char        checkpoint_name[]="/checkpoint_4";
static char        checkpoint_last[]="/checkpoint_4.old";

static int         just_started=TRUE;

struct checkpoint_struct {

    short    position;
    float    values[1000];
} computing;

/*******************************
* Initialization of the program *
*******************************/

void init_program(struct checkpoint_struct *value)
{
    value->position=0;
}

/*******************************
* Here we write the checkpoint *
*******************************/

void checkpoint_creation()
{
    char    checkpoint_old[255];
    char    checkpoint_new[255];

    struct stat sbuf;

    FILE    *checkpoint_file;

    fprintf(stderr, "Checkpoint creation initiated.\n");

    strcpy(checkpoint_old, checkpoint_directory);
    strcat(checkpoint_old, checkpoint_last);
    strcpy(checkpoint_new, checkpoint_directory);
    strcat(checkpoint_new, checkpoint_name);
```

```
    if (stat(checkpoint_new, &sbuf) == 0)
        if (link(checkpoint_new, checkpoint_old) != 0)
            fprintf(stderr, "Couldn't rename old checkpoint.\n");
        else if (unlink(checkpoint_new) != 0)
            fprintf(stderr, "Couldn't unlink old checkpoint.\n");

    checkpoint_file=fopen(checkpoint_new, "wb");

    if (checkpoint_file == NULL)
        fprintf(stderr, "Couldn't open checkpoint for
writing.\n");
    else if (fwrite(&computing, sizeof computing, 1,
checkpoint_file) != 1)
        fprintf(stderr, "Couldn't write checkpoint.\n");
    else if (fclose(checkpoint_file) != 0)
        fprintf(stderr, "Couldn't close checkpoint.\n");
    else
        fprintf(stderr, "Checkpoint created to restart with
%d.\n", computing.position);

    if (stat(checkpoint_old, &sbuf) == 0)
        if (unlink(checkpoint_old) != 0)
            fprintf(stderr, "Couldn't remove old checkpoint.\n");
}

/**************************
* Main program starts here *
**************************/

int main(int argc, char **argv)
{
    int         option;
    static char option_string[]=":d:r";
    int         checkpoint_enabled=FALSE;
    int         restarted=FALSE;

    int         i;

    struct stat sbuf;

    char        checkpoint_read[255];

    FILE        *checkpoint_file;

/******************************
* Process the command line args *
******************************/

    while ((option = getopt(argc, argv, option_string)) != -1)
        switch (option)
        {
            case 'r':
                restarted=TRUE;
                break;
            case 'd':
                strcpy(checkpoint_directory, optarg);
                if (stat(checkpoint_directory, &sbuf) != 0)
                {
                    fprintf(stderr, "Couldn't stat %s.\n",
checkpoint_directory);
                    abort();
                }
                if (S_ISDIR(sbuf.st_mode))
                    checkpoint_enabled=TRUE;
                break;
            default:
```

```
                fprintf(stderr, "Unknown option: %s.\n",
argv[optind-1]);
                abort();
        }

/*****************************************
* Test for restart without given directory *
*****************************************/

    if (restarted && !checkpoint_enabled)
        fprintf(stderr, "Can't restart without given directory.
Request ignored.\n");

/*********************************
* Access the last checkpoint file *
*********************************/

    if (restarted && checkpoint_enabled)
    {
        fprintf(stderr, "I will try to restart...\n");

        strcpy(checkpoint_read, checkpoint_directory);
        strcat(checkpoint_read, checkpoint_name);

        if (stat(checkpoint_read, &sbuf) != 0)
        {
            fprintf(stderr,"No checkpoint file written up to now.
Restart from the beginning.\n");
            init_program(&computing);
        }
        else
        {
            checkpoint_file=fopen(checkpoint_read, "rb");
            if (checkpoint_file == NULL)
            {
                fprintf(stderr,"Couldn't open checkpoint for
reading.\n");
                abort();
            }
            else if (fread(&computing, sizeof computing, 1,
checkpoint_file) != 1)
            {
                fprintf(stderr,"Couldn't read checkpoint.\n");
                abort();
            }
            else if (fclose(checkpoint_file) != 0)
            {
                fprintf(stderr,"Couldn't close checkpoint.\n");
                abort();
            }
            else
                fprintf(stderr, "Checkpoint file read.
Recalculation starts at %d.\n", computing.position);
        }
    }
    else
        init_program(&computing);

/************************
* Now start the program *
************************/


    for ( ; computing.position < 1000; computing.position++)
    {
double r=100000000000, x=1, y=0, s=0, a=0, p=0;
   while(x<=r)
         {
```

```
                y=sqrt(r*r- x*x);
                s=s+y;
                x=x+1;
          }
    a=(s*4);
    p=a/(r*r);
    computing.values[computing.position] = p;
        //sleep(1);
        if (checkpoint_enabled && ! just_started)
            if (computing.position % 300 == 0)
                checkpoint_creation();
        just_started=FALSE;
    }


/*********************
* End of the program *
*********************/

    return(0);
}
```

Figure 81 Grid application source code

Appendix B

Figure 82 is the field key for the stat file within the proc filesystem on POSIX type OSs.

```
pid - Process id
comm - The executable filename
state - R (running), S(sleeping interruptable), D(sleeping),
Z(zombie), or T(stopped on a signal).
ppid - Parent process ID
pgrp - Process group ID
session - The process session ID.
tty - The tty the process is using
tpgid - The process group ID of the owning process of the tty the
current process is connected to.
flags - Process flags, currently with bugs
minflt - Minor faults the process has made
cminflt - Minor faults the process and its children have made.
majflt
cmajflt
utime - The number of jiffies (processor time) that this process
has been scheduled in user mode
stime - in kernel mode
cutime - This process and its children in user mode
cstime - in kernel mode
counter - The maximum time of this processes next time slice.
priority - The priority of the nice(1) (process priority) value
plus fifteen.
timeout - The time in jiffies of the process's next timeout.
itrealvalue - The time in jiffies before the next SIGALRM is sent
to the process because of an internal timer.
starttime - Time the process started after system boot
vsize - Virtual memory size
rlim - Current limit in bytes of the rss of the process.
startcode - The address above which program text can run.
endcode - The address below which program text can run.
startstack - The address of the start of the stack
kstkesp - The current value of esp for the process as found in the
kernel stack page.
kstkeip - The current 32 bit instruction pointer, EIP.
signal - The bitmap of pending signals
blocked - The bitmap of blocked signals
sigignore - The bitmap of ignored signals
sigcatch - The bitmap of catched signals
wchan - The channel in which the process is waiting. The "ps -l"
command gives somewhat of a list.
```

Figure 82 Stat field key

References

1. Foster, I., C. Kesselman, and S. Tuecke, *The Anatomy of the Grid: Enabling Scalable Virtual Organizations.* International Journal of High Performance Computing Applications, 2001. **15**(3): p. 200-222.
2. Austin, J., T. Jackson, M. Fletcher, M. Jessop, P. Cowley, and P. Lobner, *Predictive Maintenance: Distributed Aircraft Engine Diagnostics*, in *The Grid 2: Blueprint For A New Computing Infrastructure*, I. Foster and C. Kesselman, Editors. 2004, Elsevier Science: Oxford, UK.
3. Fowler, M. and K. Scott, *UML distilled : a brief guide to the standard object modeling language*. The Addison-Wesley object technology series. 2000, Reading, Mass. ; Harlow: Addison-Wesley. xxi, 185.
4. Larman, C., *Applying UML and patterns : an introduction to object-oriented analysis and design and the unified process*. 2nd ed. 2002, Upper Saddle River, NJ: Prentice Hall PTR. xxi, 627.
5. Stevens, P. and R.J. Pooley, *Using UML : software engineering with objects and components*. Addison-Wesley object technology series. 2006, Harlow: Addison-Wesley. xxii, 248 p.
6. *Unified Modeling Language*, Object Management Group: http://www.uml.org/.
7. Padgett, J., M. Haji, and K. Djemame. *SLA Management in a Service Oriented Architecture*. in *Proceedings of the 2005 International Conference on Computational Science and its Applications*. 2005. Singapore: Springer-Verlag: p. 1182-1191.
8. Foster, I. and C. Kesselman, *The Grid 2: Blueprint for a new computing infrastructure*. 2004: Morgan Kaufmann.
9. Buyya, R., D. Abramson, and J. Giddy. *Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid*. in *International conference/exhibition on high performance computing in the Asia-Pacific region*. 2000. Beijing: IEEE Computer Society: p. 283-289.
10. Abramson, D., R. Buyya, and J. Giddy, *A computational economy for grid computing and its implementation in the Nimrod-G resource broker.* Future Generations Computer Systems, 2002. **18**(8): p. 1061-1074.
11. *CORBA*, Object Management Group: http://www.corba.org/.
12. *DCOM,* 2003. Microsoft. [accessed August 5 2006] Available from: http://www.microsoft.com/com/tech/dcom.asp.
13. Postel, J. *RFC791 Internet Protocol,* 1981. Internet Engineering Task Force.
14. Foster, I., C. Kesselman, J.M. Nick, and S. Tuecke, *Grid Services for Distributed System Integration.* The Computer Journal, 2002. **35**(6): p. 37-46.
15. Czajkowski, K., D. Ferguson, I. Foster, J. Frey, S. Graham, I. Sedukhin, D. Snelling, S. Tuecke, and W. Vambenepe. *The WS-Resource Framework (WSRF),* 2004. Available from: http://www-128.ibm.com/developerworks/library/ws-resource/ws-wsrf.pdf.

16. Tuecke, S., K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maquire, T. Sandholm, D. Snelling, and P. Vanderbilt. *Open Grid Services Infrastructure (OGSI),* 2003. draft-ggf-ogsi-gridservice-29. Global Grid Forum. Available from: http://www.gridforum.org/ogsi-wg/drafts/draft-ggf-ogsi-gridservice-29_2003-04-05.pdf.
17. Foster, I., J. Frey, S. Graham, S. Tuecke, K. Czajkowski, D. Ferguson, F. Leymann, M. Nally, I. Sedukhin, D. Snelling, T. Storey, W. Vambenepe, and S. Weerawarana. *Modeling stateful resource with Web services,* 2004. Available from: http://www-128.ibm.com/developerworks/library/ws-resource/ws-modelingresources.pdf.
18. Christensen, E., F. Curbera, G. Meredith, and S. Weerawarana. *Web Services Description Language (WSDL) 1.1,* 2001. World Wide Web Consortium. [accessed 16 September 2003] Available from: http://www.w3.org/TR/wsdl.
19. *Open Grid Services Infrastructure Working Group*, Global Grid Forum: http://forge.gridforum.org/projects/ogsi-wg.
20. Bosworth, A., D. Box, E. Christensen, F. Curbera, D. Ferguson, J. Frey, and M. Handley. *Web Services Addressing,* 2004. Available from: http://www-128.ibm.com/developerworks/webservices/library/specification/ws-add/.
21. Foster, I., C. Kesselman, G. Tsudik, and S. Tuecke. *A Security Architecture for Computational Grids*. in *Computer and communications security*. 1998. San Francisco; CA: ACM Press: p. 83-92.
22. Czajkowski, K., I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke, *A Resource Management Architecture for Metacomputing Systems.* Lecture Notes in Computer Science, 1998: p. 62-82.
23. Czajkowski, K., S. Fitzgerald, I. Foster, and C. Kesselman. *Grid Information Services for Distributed Resource Sharing*. in *High performance distributed computing*. 2001. San Francisco, CA: IEEE Computer Society Press: p. 181-194.
24. Allcock, B., J. Bester, J. Bresnahan, A.L. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, and S. Tuecke, *Data management and transfer in high-performance computational grid environments.* Parallel Computing, 2002. **28**(5): p. 749-771.
25. *Grid Resource Allocation Agreement Protocol Working Group*, Global Grid Forum: https://forge.gridforum.org/projects/graap-wg.
26. Erwin, D.W., *UNICORE - A Grid computing environment.* Concurrency and Computation, 2002. **14**(13/15): p. 1395-1410.
27. Streit, A., D. Erwin, T. Lippert, D. Mallmann, R. Menday, M. Rambadt, M. Riedel, M. Romberg, B. Schuller, and P. Wieder. *UNICORE - From Project Results to Production Grids*. in *International advanced research workshop on high performance computing; Grid computing the new frontier of high performance computing*. 2004. Cetraro, Italy: Amsterdam: p. 357-376.
28. *JAVA*, Sun Microsystems, Inc: http://java.sun.com/.
29. Henderson, R.L. *Job Scheduling Under the Portable Batch System*. in *Job scheduling strategies for parallel processing*. 1995. Santa Barbara; CA: Springer-Verlag: p. 279-294.
30. Keller, A. and A. Reinefeld. *CCS Resource Management in Networked HPC Systems*. in *Heterogeneous computing workshop*. 1998. Orlando, FL: Los Alamitos CA: p. 44-56.
31. Looker, N., J. Xu, T. Wo, and J. Huai. *Application of Fault Injection to Globus Grid Middleware*. in *Proceedings of the 2006 UK e-Science All Hands Meeting*. 2006. Nottingham, UK: NeSC: p. 265-272.

32. *Open Middleware Infrastructure Institute UK*, OMII: http://www.omii.ac.uk/.
33. *Grid Enabled Remote Instrumentation with Distributed Control and Computation,* 2004. GRIDCC. [accessed 20th September 2006] Available from: http://www.gridcc.org/.
34. *Enabling Grids for E-science,* 2006. EGEE. [accessed 20th September 2006] Available from: http://www.eu-egee.org/.
35. Furmento, N., A. Mayer, S. McGough, S. Newhouse, T. Field, and J. Darlington, *ICENI: Optimisation of component applications within a Grid environment.* Parallel Computing, 2002. **28**(12): p. 1753-1772.
36. Haya, G., F. Scholze, and J. Vigen. *Developing a Grid-Based Search and Categorization Tool,* 2003. High Energy Physics Libraries Webzine, issue 8. [accessed 20th September 2006] Available from: http://library.cern.ch/HEPLW/8/papers/1/.
37. *Grid Protein Sequence Analysis,* 2005. Pôle Bioinformatique Lyonnais. [accessed 20th September 2006] Available from: http://gpsa.ibcp.fr/.
38. Kallos, G., S. Nickovic, D. Jovic, and O. Kakaliagou. *The ETA Model Operational Forecasting System and its Parallel Implementation*. in *Large-scale scientific computations*. 1997. Varna; Bulgaria: Braunschweig: p. 176-188.
39. Hobson, P., S. McGough, and D. Colling. *GRIDCC - Providing a real-time Grid for distributed instrumentation*. in *to appear in the Proceedings of the 2006 Computing in High Energy and Nuclear Physics Conference*. 2006. Mumbai, India.
40. *gPTM3D,* 2005. EGEE. [accessed 20th September 2006] Available from: http://egee-na4.ct.infn.it/biomed/gPTM3D.html.
41. Thomas, N., *Editorial: Grid Performability.* Computer Journal, 2005. **48**(3): p. 323-324.
42. Buyya, R., S. Chapin, and D. DiNucci, *Architectural Models for Resource Management in the Grid.* Lecture Notes in Computer Science, 2000(1971): p. 18-35.
43. Czajkowski, K., I. Foster, and C. Kesselman. *Resource Co-Allocation in Computational Grids*. in *High performance distributed computing*. 1999. Redondo Beach; CA: IEEE Computer Society Press: p. 219-228.
44. *The White Rose Grid,* 2002. White Rose Consortium. [accessed 15 January 2007].
45. Karonis, N.T., B. Toonen, and I. Foster, *MPICH-G2: A Grid-enabled implementation of the Message Passing Interface.* Journal of Parallel and Distributed Computing, 2003. **63**(5): p. 551-563.
46. Von Laszewski, G., I. Foster, J. Gawor, and P. Lane, *A Java commodity grid kit.* Concurrency and Computation, 2001. **13**(8/9): p. 645-662.
47. *GT 2.4: The Globus Resource Specification Language RSL v1.0,* 2003. Globus Alliance. [accessed 20th September 2006] Available from: http://www.globus.org/toolkit/docs/2.4/gram/rsl_spec1.html.
48. Anjomshoaa, A. *Analysis of Globus Toolkit V2.0,* 2002. EPCC. Available from: http://www.epcc.ed.ac.uk/sungrid/PUB/gtII-ooa.pdf.
49. Brucker, P., *Scheduling algorithms*. 2004, London: Springer-Verlag.
50. Zhou, S. *LSF: Load sharing in large-scale heterogeneous distributed systems*. in *Proc. Workshop on Cluster Computing*. 1992.
51. Jackson, D.B., *Grid Scheduling with Maui/Silver.* International Series in Operations Research and Management Science, 2003(64): p. 161-170.

52. Tannenbaum, T. and M. Litzkow, *The Condor Distributed Processing System.* Doctor Dobbs Journal, 1995. **20**(2; 226): p. 40.
53. Frey, J., T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke, *Condor-G: A Computation Management Agent for Multi-Institutional Grids.* Cluster Computing, 2002. **5**(3): p. 237-246.
54. Chapin, S.J., D. Katramatos, J. Karpovich, and A.S. Grimshaw, *The Legion Resource Management System*, in *Job Scheduling Strategies for Parallel Processing*, D.G. Feitelson and L. Rudolph, Editors. 1999, Springer Verlag Kg. p. 162-178.
55. Foster, I. and C. Kesselman, *Globus: A Metacomputing Infrastructure Toolkit.* International Journal of Supercomputer Applications and High Performance Computing, 1997. **11**(2): p. 115-128.
56. Chapman, B., B. Sundaram, and K. Thyagaraja, K. *EZ-Grid: Integrated Resource Brokerage Services for Computational Grids,* 2002. Department of Computer Science, University of Texas, TX. Available from: http://www.cs.uh.edu/~ezgrid/doc.html.
57. Aloisio, G., M. Cafaro, E. Blasi, and I. Epicoco, *The Grid Resource Broker, a ubiquitous grid computing framework.* Scientific Programming, 2002. **10**(2): p. 113-120.
58. Berman, F., R. Wolski, H. Casanova, W. Cirne, H. Dail, M. Faerman, S. Figueira, J. Hayes, G. Obertelli, and J. Schopf, *Adaptive Computing on the Grid Using AppLeS.* Ieee Transactions on Parallel and Distributed Systems, 2003. **14**(4): p. 369-382.
59. Wolski, R., N.T. Spring, and J. Hayes, *The network weather service: a distributed resource performance forecasting service for metacomputing.* Future Generations Computer Systems, 1999. **15**(5-6): p. 757-768.
60. Tierney, B., R. Aydt, D. Gunter, W. Smith, M. Swany, V. Taylor, and R. Wolski. *A Grid Monitoring Architecture,* 2002. Global Grid Forum. Available from: http://www.ggf.org/documents/GFD/GFD-I.7.pdf.
61. Gunter, D., B. Tierney, B. Crowley, M. Holding, and J. Lee. *NetLogger: A Toolkit for Distributed System Performance Analysis*. in *Modeling, analysis and simulation of computer and telecommunication systems*. 2000. San Francisco, CA: IEEE Computer Society: p. 267-273.
62. *Information Services (MDS) : Key Concepts,* 2005. Globus Alliance. [accessed 20th September 2006] Available from: http://www.globus.org/toolkit/docs/4.0/info/key-index.html.
63. *Hawkeye,* 2003. UW-Madison. [accessed 20 September 2006] Available from: http://www.cs.wisc.edu/condor/hawkeye/.
64. Massie, M.L., B.N. Chun, and D.E. Culler, *The ganglia distributed monitoring system: design, implementation, and experience.* Parallel Computing, 2004. **30**(7): p. 817-840.
65. Sacerdoti, F., M. Katz, M. Massie, and D. Culler. *Wide Area Cluster Monitoring with Ganglia*. in *Cluster computing*. 2003. Hong Kong: IEEE Computer Society: p. 289-299.
66. Tierney, B., W. Johnston, B. Crowley, and G. Hoo. *The NetLogger Methodology for High Performance Distributed Systems Performance Analysis*. in *High performance distributed computing*. 1998. Chicago; IL: IEEE Computer Society Press: p. 260-267.

67. Anjomshoaa, A., F. Brisard, M. Drescher, D. Fellows, A. Ly, S. McGough, D. Pulsipher, and A. Savva. *The Job Submission Description Language Specification,* 2005. Global Grid Forum.
68. Ludwig, H., A. Keller, A. Dan, and R. King. *A Service Level Agreement Language for Dynamic Electronic Services*. in *Advanced issues of E-commerce and web-based information systems*. 2002. Newport Beach, CA: IEEE Computer Society: p. 25-32.
69. Andrieux, A., K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu. *Web Services Agreement Specification (WS-Agreement),* 2004. Global Grid Forum.
70. Mach, R., R. Lepro, S. Jackson, and L. McGinnis. *The Usage Record Specification,* 2005. Global Grid Forum.
71. *Grid Economic Service Architecture Working Group*, Global Grid Forum: http://www.doc.ic.ac.uk/~sjn5/GGF/gesa-wg.html.
72. Czajkowski, K., I. Foster, C. Kesselman, V. Sander, and S. Tuecke. *SNAP: A Protocol for Negotiating Service Level Agreements and Coordinating Resource Management in Distributed Systems*. in *Job scheduling strategies for parallel processing*. 2002. Edinburgh: Berlin: p. 153-183.
73. Marjanovic, O. and Z. Milosevic. *Towards Formal Modeling of e-Contracts*. in *Enterprise distributed object computing*. 2001. Seattle, WA: IEEE Computer Society: p. 59-68.
74. *Distributed Resource Management Application API Working Group*, Global Grid Forum: https://forge.gridforum.org/projects/drmaa-wg.
75. *Web Services Architecture*, World Wide Web Consortium: http://www.w3.org/TR/ws-arch.
76. Oldham, N., K. Verma, A. Sheth, and F. Hakimpour. *Semantic WS-Agreement Partner Selection*. in *15th International World Wide Web Conference, WWW2006*. 2006. Edinburgh, UK: ACM Press: p. 697-706.
77. Jackson, S. and R. Lepro. *Usage Record -- XML Format,* 2003. Usage Record - XML Format ver 1.7. Global Grid Forum. Available from: mailing list.
78. Newhouse, S. *Resource Usage Service Specification,* 2003. draft-ggf-rus-service-1. Global Grid Forum. Available from: http://www.doc.ic.ac.uk/~sjn5/GGF/draft-ggf-rus-service-1.pdf.
79. Buyya, R., D. Abramson, J. Giddy, and H. Stockinger, *Economic models for resource management and scheduling in Grid computing.* Concurrency and Computation, 2002. **14**(13/15): p. 1507-1542.
80. *Resource Usage Service Working Group*, Global Grid Forum: http://www.doc.ic.ac.uk/~sjn5/GGF/rus-wg.html.
81. Milosevic, Z., A. Josang, T. Dimitrakos, and M.A. Patton. *Discretionary Enforcement of Electronic Contracts*. in *Enterprise distributed object computing*. 2002. Lausanne, Switzerland: IEEE Computer Society: p. 39-50.
82. Sahai, A., A. Graupner, V. Machiraju, and A. van Moorsel. *Specifying and Monitoring Guarantees in Commercial Grids through SLA*. in *3rd IEEE/ACM International Symposium on Cluster Computing and the Grid*. 2003. Tokyo: IEEE Computer Society.
83. Berman, F., A. Chien, K. Cooper, J. Dongarra, I. Foster, D. Gannon, L. Johnsson, K. Kennedy, C. Kesselman, and J. Mellor-Crummey, *The GrADS Project: Software Support for High-Level Grid Application Development.* International Journal of High Performance Computing Applications, 2001. **15**: p. 327-344.

84. Leff, A., J.T. Rayfield, and D.M. Dias, *Service-Level Agreements and Commercial Grids.* Ieee Internet Computing, 2003. **7**(4): p. 44-50.
85. Stephanopoulos, G., *Chemical process control : an introduction to theory and practice*. Prentice-Hall international series in the physical and chemical engineering sciences. 1984, Englewood Cliffs ; London: Prentice-Hall. xxi, 696p.
86. *JINI*, Sun Microsystems, Inc: http://www.sun.com/software/jini/.
87. *JXTA*, Sun Microsystems, Inc: http://www.sun.com/software/jxta/.
88. *GridPP*, PPARC e-Science Project: http://www.gridpp.ac.uk.
89. *LHC - THE LARGE HADRON COLLIDER*, CERN: http://lhc.web.cern.ch/lhc/.
90. *MB-NG Managed Bandwidth Next Generation*, PPARC e-Science Project: http://www.mb-ng.net.
91. *GRID Resource Scheduler*, PPARC + e-Science Core Project: http://www.cs.ucl.ac.uk/research/grs/.
92. *Performance-based Middleware for the Grid*, High Performance Systems Group, University of Warwick: http://www.dcs.warwick.ac.uk/research/hpsg/.
93. *PACE*, High Performance Systems Group, University of Warwick: http://www.dcs.warwick.ac.uk/research/hpsg/html/htdocs/public/pace.html.
94. *A4*, High Performance Systems Group, University of Warwick: http://www.dcs.warwick.ac.uk/research/hpsg/html/htdocs/public/a4.html.
95. *TITAN*, High Performance Systems Group, University of Warwick: http://www.dcs.warwick.ac.uk/research/hpsg/.
96. *RealityGrid*, EPSRC e-Science Project: http://www.realitygrid.org/.
97. Kapadia, N., J. Fortes, and C. Brodley. *Predictive Application-Performance Modeling in a Computational Grid Environment*. in *High performance distributed computing*. 1999. Redondo Beach; CA: IEEE Computer Society Press: p. 47-56.
98. Dushay, N., J.C. French, and C. Lagoze. *Predicting Indexer Performance in a Distributed Digital Library*. in *Research and advanced technology for digital libraries*. 1999. Paris: Springer: p. 142-166.
99. Kapadia, N.H. and J.A.B. Fortes. *On the Design of a Demand-Based Network-Computing System: The Purdue University Network-Computing Hub*. in *High performance distributed computing*. 1998. Chicago; IL: IEEE Computer Society Press: p. 71-80.
100. Kapadia, N.H., J.A.B. Fortes, and M.S. Lundstrom, *The Purdue University Network-Computing Hubs: Running Unmodified Simulation Tools via the WWW*. Acm Transactions on Modeling and Computer Simulation, 2000. **10**(1): p. 39-57.
101. Atkeson, C.G., A.W. Moore, and S. Schaal, *Locally Weighted Learning.* Artificial Intelligence Review, 1997. **11**(1/5): p. 11-73.
102. Nudd, G.R., D.J. Kerbyson, E. Papaefstathiou, S.C. Perry, J.S. Harper, and D.V. Wilcox, *PACE - A Toolset for the Performance Prediction of Parallel and Distributed Systems.* International Journal of High Performance Computing Applications, 2000. **14**: p. 228-251.
103. Papaefstathiou, E., D.J. Kerbyson, G.R. Nudd, and T.J. Atherton. *An Overview of the CHIP'S Performance Prediction Toolset for Parallel Systems*. in *Parallel and distributed computing systems*. 1995. Orlando; FL: Raleigh: p. 527-533.
104. Silva, L.M. and J.G. Silva. *System-Level Versus User-Defined Checkpointing*. in *Reliable distributed systems*. 1998. West Lafayette; IN: IEEE Computer Society: p. 68-74.

105. Kennedy, K., M. Mazina, J. Mellor-Crummey, K. Cooper, L. Torczon, F. Berman, A. Chien, H. Dail, O. Sievert, and D. Angulo, *Toward a Framework for Preparing and Executing Adaptive Grid Programs.* International Parallel and Distributed Processing Symposium, 2002: p. 171.
106. Huedo, E., R.S. Montero, and I.M. Llorente, *A framework for adaptive execution in grids.* Software Practice and Experience, 2004. **34**(7): p. 631-652.
107. Vadhiyar, S.S. and J.J. Dongarra, *Self adaptivity in Grid computing.* Concurrency and Computation, 2005. **17**: p. 235-258.
108. Allen, G., D. Angulo, I. Foster, G. Lanfermann, C. Liu, T. Radke, E. Seidel, and J. Shalf, *The Cactus Worm: Experiments with Dynamic Resource Discovery and Allocation in a Grid Environment.* International Journal of High Performance Computing Applications, 2001. **15**: p. 345-358.
109. Diao, Y., J.L. Hellerstein, S. Parekh, N. Gandhi, and D.M. Tilbury. *Using MIMO Feedback Control to Enforce Policies for Interrelated Metrics With Application to the Apache Web Server*. in *IEEE/IFIP network operations and management symposium*. 2002. Florence, Italy: IEEE: p. 219-234.
110. Djemame, K., I. Gourlay, J. Padgett, G. Birkenheuer, M. Hovestadt, O. Kao, and K. Voß. *Introducing Risk Management into the Grid*. in *to appear in the Proceedings of the 2nd IEEE International Conference on e-Science and Grid Computing*. 2006. Amsterdam.
111. Montero, R.S., E. Huedo, and I.M. Llorente, *Grid Resource Selection for Opportunistic Job Migration.* Lecture Notes in Computer Science, 2003(2790): p. 366-373.
112. Vadhiyar, S. and J. Dongarra. *A Performance Oriented Migration Framework for The Grid*. in *Cluster computing and the grid; CCGrid 2003*. 2003. Tokyo: IEEE Computer Society: p. 130-137.
113. *Gridway Metascheduler,* 2006. Universidad Complutense de Madrid. [accessed 20th September 2006] Available from: http://www.gridway.org/.
114. Nairac, A., N. Townsend, R. Carr, S. King, P. Cowley, and L. Tarassenko, *A System for the Analysis of Jet Engine Vibration Data.* Integrated Computer Aided Engineering, 1999. **6**(1): p. 53-66.
115. King, S. *Corporate Standard for Dynamic Analysis - Data Files,* Rolls-Royce Electronic Instrumentation Report Series, Issue 3, EIR01315. Rolls-Royce plc.
116. Haji, M., P. Dew, K. Djemame, and I. Gourlay. *A SNAP-based Community Resource Broker using a Three-Phase Commit Protocol*. in *Proceedings of the 18th IEEE International Parallel and Distributed Processing Symposium*. 2004. Santa Fe, USA.
117. Haji, M.H., I. Gourlay, K. Djemame, and P.M. Dew, *A SNAP-Based Community Resource Broker Using a Three-Phase Commit Protocol: A Performance Study.* The Computer Journal, 2005. **48**(3): p. 333-346.
118. Othman, A., P. Dew, K. Djemame, and I. Gourlay. *Adaptive Grid Resource Brokering*. in *Cluster computing*. 2003. Hong Kong: IEEE Computer Society: p. 172-179.
119. Loader, C., *Local regression and likelihood*. Statistics and computing. 1999, New York, USA: Springer-Verlag.
120. *JAXB*, Sun Microsystems, Inc: http://java.sun.com/webservices/jaxb/.
121. Passino, K. and S. Yurkovich, *Fuzzy control*. 1998, Menlo Park, Calif. ; Harlow: Addison-Wesley.
122. *XML Schema Definition,* 2001. W3C. [accessed 20th September 2006] Available from: http://www.w3.org/XML/Schema.

123. *Job Submission Description Language Working Group*, Global Grid Forum: http://www.epcc.ed.ac.uk/~ali/WORK/GGF/JSDL-WG/.
124. Yeong, W., T. Howes, and S. Kille. *RFC177 Lightweight Directory Access Protocol,* 1995. Internet Engineering Task Force.
125. Schopf, J. and F. Berman. *Performance Prediction in Production Environments*. in *International parallel processing symposium*. 1998. Orlando; FA: IEEE: p. 647-653.
126. Jung, H.W., S.G. Kim, and C.S. Chung, *Measuring Software Product Qualities: A Survey of ISO/IEC 9126.* IEEE Software, 2004. **21**: p. 88-92.
127. Ribler, R.L., H. Simitci, and D.A. Reed, *The Autopilot performance-directed adaptive control system.* Future Generations Computer Systems, 2001. **18**(1): p. 175-187.
128. Ribler, R.L., J.S. Vetter, H. Simitci, and D.A. Reed. *Autopilot: Adaptive Control of Distributed Applications*. in *High performance distributed computing*. 1998. Chicago; IL: IEEE Computer Society Press: p. 172-179.