n-Dimensional Prediction of RT-SOA QoS

by

David Wesley McKee

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

UNIVERSITY OF LEEDS

The University of Leeds
School of Computing

July 2017

Declarations

The candidate confirms that the work submitted is his own, except where work which has formed part of jointly authored publications have been included. The contribution of the candidate and the other authors to this work has been explicitly indicated below. The candidate confirms that appropriate credit has been given within the thesis where reference has been made to the work of others.

The core work in the following articles is the candidate's own work:

McKee, DW; Webster, D.; Townend, P., Xu, J.; Battersby, D.; Towards a Virtual Integration Design and Analysis Enviroment for Automotive Engineering. in *2014 IEEE 17th International Symposium on Object/Component/ Service-Oriented Real-Time Distributed Computing (ISORC)*

McKee, DW; Webster D; Xu J; Enabling Decision Support for the Delivery of Real-Time Services. in *2015 IEEE 15th International Symposium on High-Assurance Systems Engineering (HASE)*

McKee, DW; Webster D; Xu J; Battersby D; DIVIDER: Modelling and Evaluating Real-Time Service-Oriented Cyberphysical Co-Simulations. *2015 IEEE 18th International Symposium on Object/Component/ Service-Oriented Real-Time Distributed Computing (ISORC)*

McKee, DW; Clement SJ; Almutairi J; Xu J; Massive-Scale Automation in Cyber-Physical Systems: Vision & Challenges *2017 IEEE 13th International Symposium on Autonomous Decentralized Systems (ISADS)*

McKee, DW; Clement SJ; Ouyang X; Xu J; Romano R; Davies J; The Internet of Simulation, a Specialisation of the Internet of Things with Simulation and Workflow as a Service (SIM/WFaaS) *2017 IEEE Symposium on Service-Oriented System Engineering (SOSE)*

McKee, DW.; Clement SJ; Xu J; Battersby D; n-Dimensional QoS Framework for Real-Time Service-Oriented Architectures *2017 IEEE International Symposium on Real-time Data Processing for Cloud Computing (RTDPCC)*

Some parts of the work presented in this thesis have also previously appeared in the following additional papers:

Dickerson CE; Clement SJ; Webster D; **McKee DW**; Xu J; Battersby D; A service oriented virtual environment for complex system analysis: Preliminary report. in *IEEE 10th System of Systems Engineering Conference (SoSE)*

Dickerson CE; Ji S; Clement SJ; Webster D; **McKee DW**; Xu J; Battersby D; Bevan N; Turner N; Suart W; A Demonstration of a Service Oriented Virtual Environment for Complex System Analysis. *IJCS-Computing, Sensing and Control*

Garraghan P; **McKee D**; Ouyang X; Webster D; Xu J; SEED: A Scalable Approach for Cyber-Physical System Simulation. *IEEE Transactions on Service Computing*.

Garraghan P; Perks S; Ouyang X; **Mckee DW**; Moreno IS Tolerating Transient Late-timing Faults in Cloud-based Real-time Stream Processing in *2016 IEEE 19th International Symposium on Object/Component/ Service-Oriented Real-Time Distributed Computing*

Clement, SJ; **McKee DW**; Xu J; Service-Oriented Reference Architecture for Smart Cities *2017 IEEE International Symposium on Service-Oriented System Engineering (SOSE)*

Clement, SJ; **McKee DW**; Romano R; Xu J; Lopez JM; Battersby D; The Internet of Simulation: Enabling Agile Model Based Systems Engineering for Cyber-Physical Systems *2017 IEEE International Conference on System of Systems Engineering (SoSE)*

Acknowledgements

I would like to thank my supervisor Prof. Jie Xu for his guidance, support and advice throughout my studies at the University of Leeds. Additionally I would like to thank my thesis examiners Prof. John McDermid and Dr. Karim Djemame for their excellent comments and feedback when examining this Thesis.

I would also like to acknowledge the help of my colleagues Dr. Paul Townend, Dr. Stephen Clement, Xue Ouyang and Prof. John Davies for the advice given throughout my research. Particularly I would like to thank Dr. David Webster for his help and direction during the first two years of my research. I also want to extend my gratitude to the rest of my colleagues and friends within the Distributed Systems and Services Research Group and many others within the University of Leeds who have supported me on this journey.

Finally I would like to thank my wife Sophie for her continuous support and never-ending patience.

Abstract

Service-Orientation has long provided an effective mechanism to integrate heterogeneous systems in a loosely coupled fashion as services. However, with the emergence of Internet of Things (IoT) there is a growing need to facilitate the integration of real-time services executing in non-controlled, non-real-time, environments such as the Cloud. As such there has been a drive in recent years to develop mechanisms for deriving reliable Quality of Service (QoS) definitions based on the observed performance of services, specifically in order to facilitate a Real-Time Quality of Service (RT-QoS) definition. Due to the overriding challenge in achieving this is the lack of control over the hosting Cloud system many approaches either look at alternative methods that ignore the underlying infrastructure or assume some level of control over interference such as the provision of a Real-Time Operating System (RTOS). There is therefore a major research challenge to find methods that facilitate RT-QoS in environments that do not provide the level of control over interference that is traditionally required for real-time systems.

This thesis presents a comprehensive review and analysis of existing QoS and RT-QoS techniques. The techniques are classified into seven categories and the most significant approaches are tested for their ability to provide QoS definitions that are not susceptible to dynamic changing levels of interference. This work then proposes a new n-dimensional framework that models the relationship between resource utilisation, resource availability on host servers, and the response-times of services. The framework is combined with real-time schedulability tests to dynamically provide guarantees on response-times for ranges of resource availabilities and identifies when those conditions are no longer suitable. The proposed framework is compared against the existing techniques using simulation and then evaluated in the domain of Cloud computing where the approach demonstrates an average overallocation of 12%, and provides alerts across 94% of QoS violations within the first 14% of execution progress.

Contents

List of Figures

List of Tables

List of Equations

List of Acronyms

AVC	Absolute Violation Count	73
BPEL	Business Process Execution Language	26
C2	Command and Control	
DDS	Data-Distribution Service	14
FaaS	Function-as-a-Service	125
FMI	Functional Mock-up Interface	148
GA	Genetic Algorithm	46
HIL	Hardware-in-the-Loop	48
HLA	High Level Architecture	146
HPC	High Performance Computing	23
IoS	Internet of Simulation	145
IoT	Internet of Things	54
IaaS	Infrastructure as a Service	128
MAE	Mean Absolute Error	72
MAV	Mean Absolute Violation	73
MPE	Mean Percentage Error	72
MPV	Mean Percentage Violation	73
MPW	Mean Percentage Waste	72
μS	Micro-Service	24
NECTISE	Network Enabled Capability Through Innovative Systems Engineering	47
PaaS	Platform-as-a-Service	128
PCC	Pearson's Correlation Coefficient	49
QoE	Quality of Experience	48
QoS	Quality of Service	16
REST	REpresentational State Transfer	16
RTOS	Real-Time Operating System	59
RT-QoS	Real-Time Quality of Service	49
RT-SOA	Real-Time Service-Oriented Architecture	40
SaaS	Software-as-a-Service	10
SIMaaS	Simulation-as-a-Service	146
SLA	Service Level Agreement	10
SOA	Service-Oriented Architecture	9
SOAP	Simple Object Access Protocol	16
UDDI	Universal Description Discovery and Integration	11
WCET	Worst-Case Execution Time	40
WCRT	Worst-Case Response Time	41
WFaaS	Workflow-as-a-Service	146
WS-Agreement	Web Services Agreement Specification	18
WSDL	Web Service Definitions Language	11
WSLA	Web Service Level Agreement	18

Chapter 1

Introduction

1.1 Research Motivation

Modern computer systems are comprised of many integrated components which are composed together to provide some global function. Many of these systems adopt the Service-Oriented Architectural (SOA) style in order to improve system dependability [1; 2]. By utilising service-orientation components are represented as services, whereby if a single service fails alternatives can be provided to either individually or collectively provide the equivalent or degraded capability.

These system's components are typically represented as Web Services which rely on XML and Web-related standards (Including SOAP, JSON, and REST) [3; 4]. Each web service is hosted by a provider with some level of Quality of Service (QoS). The QoS properties capture the non-functional aspects of a service, such as performance, reliability, scalability, and availability [5]. Users can then "consume" the services to achieve a particular requirement or purpose. The user is able to evaluate the QoS of a service to select which will meet their quality and performance requirements [6].

Services may be comprised of single atomic components or a composition of other services where QoS limitations of any individual service will have a negative impact on the others. In an ideal scenario where the components execute without faults, and also in a fault-free environment with no resource contention, the individual and composite services will perform according to the specified QoS. However, in the real world the behaviour of services is adversely affected by internal component errors as well as by faults in the hosting environment, often in the form of resource contention.

By increasingly supporting the publication of components as services with reliable QoS information the complexity of managing large-scale systems is reduced. However, in order to do so the challenges of guaranteeing QoS must be addressed, particularly in the context of integrating real-time components without the guaranteed support of Real-Time Operating Systems (RTOSs). Research in the automated integration of components as services needs to address the complexity of the relationships between heterogeneous components, systems, and infrastructure and the corresponding impact of execution behaviour in the context of faults. In turn the methods developed in this thesis will facilitate further research in real world scheduling techniques, resource management, energy-efficiency, security, and dependability.

This thesis reviews the state-of-the-art in service QoS and finds in the context of real-time operational requirements it to be lacking. Therefore a new approach which aims to capture the nuances of the performance vs. execution environment relationship is presented. Further the importance of understanding QoS is highlighted not just through literature and discussion but by considering a range of computing applications which require robust QoS methods.

1.2 Research Context and Scope

Research in service-oriented systems has focussed primarily on facilitating loose-coupling, modularity, location transparency, and fault tolerance. With regards to QoS modelling, previous research has concentrated on techniques for performing service composition under time constraints or creating compositions that will meet specified deadlines. There has been limited work on modelling the relationships between services and the underlying infrastructure. Regarding service composition most techniques focus on service performance heuristics as well as the algorithms for calculating compositions in time-constrained environments. Alternatively the QoS modelling techniques have focussed on modelling the relationships between users and services whilst ignoring much of the relationship between the underlying infrastructure and the service's execution behaviour.

This research focusses on the gaps in current research in service-oriented systems with a particular focus on QoS modelling. The general concepts proposed in this research should be applicable to the domains of Internet of Things (IoT), stream processing, and smart cities. This work will concentrate on atomic services, known as Micro-Services (μSs), and not consider in depth workflows, as many complex workflows require extensive further study in their own right. However, it is anticipated that the techniques proposed in this thesis can built upon to explore various workflow patterns. Instead the focus of the research will be the modelling of execution performance of individual atomic services (μS) with respect to their host environments. Specifically the work will focus on modelling the behaviour of micro-services, which can also be referred to as tasks or individual processes.

1.2.1 Research Sponsor

One particularly interesting area, that derives from needs from the manufacturing industries as well as the evolving areas of research on automation for both smart cities and autonomous vehicles, is that of simulations as services. This work has formed part of the Programme for Simulation Innovation (PSI), sponsored by Jaguar Land Rover and the UK's Engineering and Physical Sciences Research Council (EPSRC), and has formed the basis for understanding the QoS needs for simulation integration in the development of a Virtual Integration Environment (VIDAE) that brings together heterogeneous simulations from across the automotive sector. Applying this research to the domain of simulations, and simulation integration, is the proposal of a new paradigm for simulation integration: an Internet of Simulation (IoS) which is introduced at the end of this thesis.

1.3 Aims and Objectives

The aim of this research is to improve the accuracy of QoS timing definitions and facilitate the reliability of response-time and execution behaviour predictions for individual services. This is to address the urgent need to provide Service-Oriented Architecture (SOA) with the capability of integrating services which have strict timing requirements. This research will identify the existing techniques for QoS prediction in both online and offline contexts as well as evaluating their effectiveness in real-world imperfect environments. The findings will be used to provide a feature classification of techniques representing their robustness to specific faults. Subsequently this research will extend the state-of-the-art in Real-Time Quality of Service (RT-QoS) prediction with traditional real-time systems techniques and provide a generic theoretical framework for modelling the performance of services with respect to any and all environmental resources, such as CPU or memory, that affect their execution performance.

Specifically, the main objectives of this research are:

i. *To provide an in-depth analysis and classification of existing techniques for service QoS prediction.* This work will empirically analyse the effectiveness of existing approaches to the online prediction of response-times of executing services in the context of environmental faults. Further it will provide a method for classifying the existing approaches as well as future techniques against real-world environmental faults. This will facilitate the evaluation of both the methodologies developed in this work as well as identifying future related work.

ii. *To provide a theoretical mechanism for efficient and accurate prediction of Real-Time Quality of Service (RT-QoS).* This work will focus on developing a method for characterising service's execution behaviour with respect to their environment. The framework will be defined mathematically and analysed against real-time systems Schedulability tests. It will be designed to be applicable to various domains, beyond service-orientation, and cope with multiple environmental factors.

iii. *To provide efficient scalable algorithms for the prediction and management of Real-Time Quality of Service (RT-QoS).* The mathematical framework will be converted to an algorithmic representation which is both fast enough to operate in a real-time environment and scales with respect to storage space, number of services, and service execution instances.

iv. *To provide an empirical evaluation of the proposed techniques.* The research will also be evaluated using both simulation as well as experimentation in VIDAE deployed within a Cloud environment.

1.4 Research Methodology

There are three primary methods for the development and evaluation of computational research: mathematical modelling, simulation, and prototypes. The use of mathematical modelling allows for formal reasoning using mathematical symbols, operators, and techniques to demonstrate the effectiveness of the research. A prototype allows the research to be developed into an α-product for the concepts to be tested without the controlled assumptions of mathematical modelling or simulation. Alternatively simulation allows the imitation and emulation of system behaviour in a controlled manner allowing for the evaluation of hypotheses or specific scenarios which may not be feasible to evaluate using mathematical methods or a prototype.

The research methodology of this work builds on each of the three methods and consists of the following core elements:

1. *A thorough literature review* on techniques for QoS prediction. This review covers the foundational concepts behind service-orientation and real-time systems.

2. *A classification of QoS techniques* providing both an approach for feature-based classification and an analysis of the existing approaches. The analysis itself uses both mathematical modelling to demonstrate the theoretical limitations of the approaches whilst simulation is used to evaluate their accuracy and performance at runtime.

3. *A framework for QoS prediction* is designed and developed using purely mathematical modelling bringing together the concepts from the domains of Service-Oriented Architecture (SOA) QoS and real-time systems schedulability. The framework is then theoretically evaluated against the alternative approaches.

4. *Simulation of QoS prediction* is then used to demonstrate the effectiveness of the developed framework under specific scenarios. The results of which are benchmarked against the simulation evaluation of existing techniques for both increased prediction accuracy as well as increases in performance and efficiency.

5. *The design and development of a prototype* with VIDAE is used to evaluate the actual effectiveness of the solution against the outcome expected by both simulation and the mathematical methods. This technique is applied to specific case studies whereby simulations are provided as services in the automotive domain.

1.5 Major Contributions

The major contributions within this thesis are:

- *An analysis and classification of existing QoS techniques.* This looks at eighty existing approaches, groups them, and then analyses the effectiveness of each group theoretically.

- *Simulation of existing QoS techniques.* From the seven identified categories the most relevant are experimentally evaluated using simulation for their capability in predicting QoS with non-static interfering workloads.

- *A mathematical n-dimensional framework capturing the relationship between Micro-Service (μS) execution performance and the execution environment.* This framework explicitly models the relationship between Micro-Services (μSs) and their host server's resources in terms of utilisation and availability. The framework is then proven using real-time schedulability techniques.

- *Extensive simulation analysing the effectiveness of the QoS framework under various interfering Cloud workloads.* The mathematical framework is evaluated against existing approaches against periodic and random interfering workloads. The results are analysed using metrics for accuracy, wasted time, and QoS violation.

1.6 Thesis Organisation

The thesis is comprised of seven chapters, with the remaining structured as follows:

Chapter 2. Presents an introduction to the topics related to service-orientation and specifically Quality of Service (QoS). It describes the foundational concepts of Service-Oriented Architecture (SOA), dependability, and real-time systems. The advantages and limitations of the state-of-the-art are explored with regards to Real-Time Service-Oriented Architectures (RT-SOAs).

Chapter 3. Presents a classification of the current state-of-the-art in Real-Time Service-Oriented Architecture (RT-SOA) QoS techniques. The classifications are analysed using both mathematical modelling and simulation techniques to identify their strengths and limitations. Finally, the research challenges in Real-Time Quality of Service (RT-QoS) are presented providing the scope for this thesis.

Chapter 4. Presents a mathematical framework for the online construction of a QoS model capturing the relationship between the execution environment and the service performance. The framework is progressively developed using core principles from real-time systems in the context of service-orientation. All aspects of the approach are defined explicitly in mathematical symbolic notation which can be used to show its theoretical capabilities and limitations. Finally, the framework is converted into an algorithmic representation and analysed for it's performance, efficiency, and scalability in contrast to alternative techniques.

Chapter 5. Presents the architecture of the QoS prediction system along with the its simulation. A detailed set of use cases are outlined comprising of environmental configurations as well as individual services to be evaluated. The simulation results are then evaluated using a defined set of measures and metrics against the simulation results from Chapter 3.

Chapter 6. Describes a series of case studies from three domains: Cloud computing, simulation, and human tasks. Within the former two areas the generic architectures that support the proposed QoS framework are presented. Then experimental results from the Cloud and human task domains are discussed to provide an overall assessment fo the proposed Real-Time Service-Oriented Architecture (RT-SOA) QoS technique.

Chapter 7. Presents a summary of the findings and contributions as well as exploring potential future research directions, either not covered in or, building upon this thesis.

Chapter 2

Service-Orientation

This chapter describes the topics relevant to the context of this research, providing the building blocks used in this thesis. The concepts of Service-Orientation are discussed and Service-Oriented Architectures (SOAs) are explored in detail. Dependability and the related topics are defined and then in the context of Service-Oriented Architectures (SOAs). Subsequently an overview of Real-Time Systems theory is presented relating particularly to measuring execution time and predicting process schedulability. Finally this chapter presents the state-of-the-art in Real-Time Service-Oriented Architecture (RT-SOA) research and techniques for predictive Quality of Service (QoS) modelling before outlining the challenges which these research areas have yet to address.

2.1 Service Orientation

Service Orientation is an architectural paradigm [1; 7] designed to increase the reliability, availability, and maintainability (see Section Section 2.2.2) of large-scale systems through features such as loose-coupling and modularity (see Section Section 2.1.1). Specifically it provides either the programming model, approach or even the business process whereby processes operate independently as services, and can be collectively organised into a workflow to provide some higher function. "[Service-Orientation] *promotes the idea of assembling application components into a network of services that can be loosely coupled to create flexible, dynamic business processes and agile applications that span organisations and computing platforms*" [8].

Service-Oriented Architectures (SOAs) provide the architectural style, or template, for building service oriented systems in which individual software solutions can be provided as services and combined into business processes, known as workflows (see Section Section 2.1.3). SOA dictates that the underlying architecture of a system should consists of three core participants: a *service provider*, a *service registry*, and a *service consumer*. This separation of concerns allows for SOAs to support: technology neutrality, loose coupling, and location transparency [1] without sacrificing solution simplicity or capability [9]. Conceptually SOA is designed to support an open marketplace where a consumer can source a service providing the functionality, desired at the current moment in time, agnostic of it's provider or location, and invoke it as many or as few times as required [10]. Additionally a service's functionality can be a composition of the functionality provided by subcontracted services [11], discussed further in Section 2.1.2.

Due to its flexibility, SOA lends itself to designing, developing, managing, and organising services within complex computing and business environments [12]. The remainder of this section outlines the detail of Service-Oriented Architectures (SOAs) as well as the state-of-the-art research relevant to this thesis, covering the fundamental principles of SOAs (Section Section 2.1.1), the architectural structure and use of abstraction (Section Section 2.1.2), as well as the concepts of workflows (Section Section 2.1.3).

2.1.1 The Principles of Service-Orientation

The central theme of SOAs is the Software-as-a-Service (SaaS) paradigm which separates "*the possession and ownership of software from its use*" [13]. Therefore services can be outsourced, under a Service Level Agreement (SLA), allowing developers to focus on and respond to changing and evolving needs [14]. SOAs can increase the maintainability of software by increasing the modularity, re-usability, loose coupling, and simplicity of systems [9]. The remainder of this section outlines these key features of SOAs, in addition to introducing Quality of Service (QoS) in the context of SLAs.

SOA W3C Triangle

In order to facilitate the access and utilisation of software beyond it's traditional siloed single system deployment, SOA adopts a tri-party and multi-layered approach (discussed in Section Section 2.1.2). As shown in Figure 2.1 an SOA must consist of at least one service provider, registry, and consumer where [1; 15]:

Providers must publish their service to the registry. They are responsible for generating a service definition that is compatible with the registry and can be understood by consumers and contains all the necessary information required for the service to be integrated into a process or workflow. The service definition may be in the form of a Web Service definition using the Web Service Definitions Language (WSDL) [16].

Registry stores the reference to the service in the form of its service definition. The registry may also actively seek out new services using a discovery mechanism to retrieve definitions for publicly available services [6]. In the case of web services the registry may follow the Universal Description Discovery and Integration (UDDI) specification [17]. The registry may also act as a broker or autonomous manager providing further assistance to the consumer for the selection of services and supporting other wider system function such as scheduling and workload balancing [18; 10].

Consumers are then able to request particular functionality from the system. The service registry returns the service definitions of those services which meet the specified criteria to the consumer. The consumer and provider then interact directly in a binding process whereby a Service Level Agreement (SLA) is agreed (discussed further on page 18).

Modularity

The SOA triangle (Figure 2.1) by its very nature supports a modular and decoupled system structure. By facilitating increased modularity, SOAs enable increased application abstraction, in-

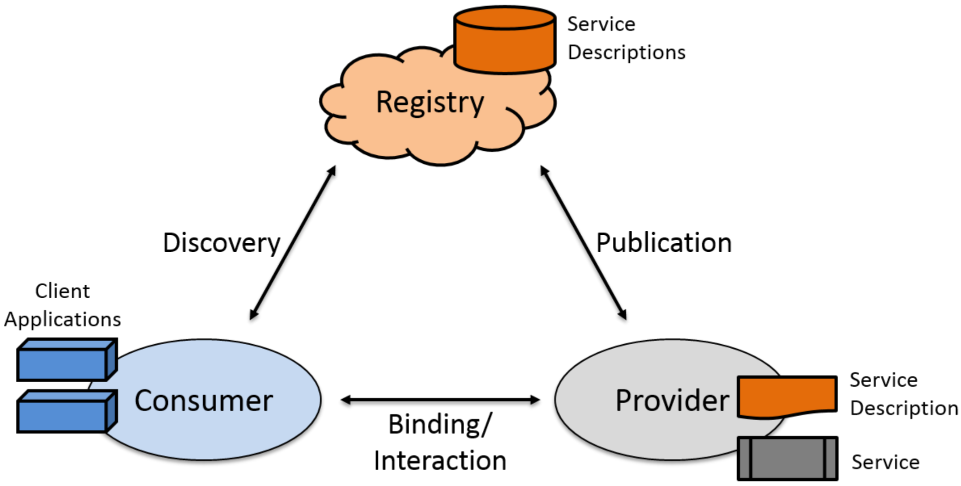


Figure 2.1: W3C SOA Architecture Triangle

frastructure virtualisation and management, service composition, reusability, and granularity [9]. Individual applications can focus on completing specific tasks reliably rather than attempting to deliver a larger feature set. Additionally this increase in the granularity of the software allows for better alignment with the logic in business processes and allows for increased individualisation by composing services together in different fashions.

Application abstraction forms one of the central tenets of SOAs allowing the description of the functionality of a service to be abstracted away from the implementation and the physical location of the service provider. This facilitation of location transparency [1] enables consumers to utilise services without any understanding of the location of providers and therefore utilise services potentially located on opposite sides of the globe as long as the supply network allows the delivery of the advertised service. The location transparency also enables the concealment of the complexity of the potentially physically distributed system components [19] and as such is also technology neutral allowing for them to be implemented using the most appropriate technologies and tools [1]. It is generally accepted that SOA must be invokable on the *"lowest common denominator technologies that are available to almost all IT environments"* such as through web browsers with web services [20]. (Further discussion on the layers of abstraction in SOA is found in Section Section 2.1.2.)

In order to facilitate application abstraction and reap the resulting benefits from SOAs the infrastructure itself must be managed. Specifically the service-oriented systems must manage the infrastructure so as to maintain the supply network's availability [10]. Depending on the scenario this may range from ensuring up-to-date registry information through to managing software

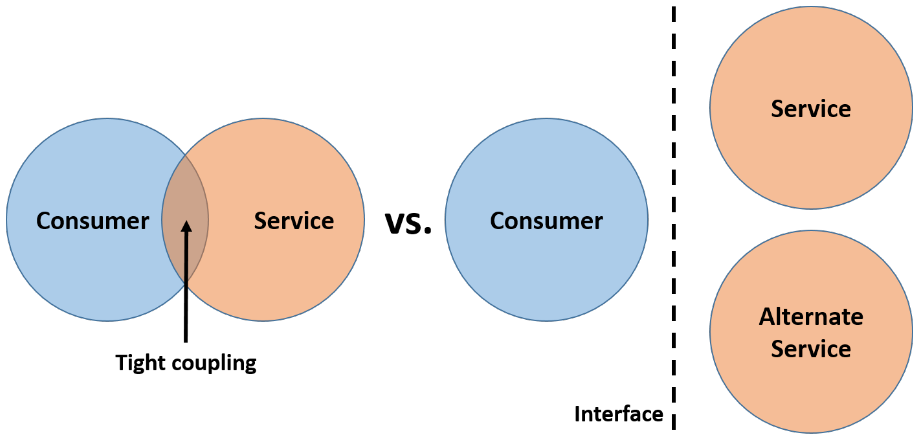


Figure 2.2: Visual representation of loose coupling

defined networks and scheduling server workloads so as to guarantee service delivery.

Additionally as mentioned above the modular nature of SOA implies a granular software design where individual services focus on performing specific functions rather than suites of operations. By using standard interfaces, through the likes of WSDL, and allowing services to be reused by various consumers, services can be composed to provide more complex functionality (see workflows Section Section 2.1.3). Furthermore, individual services may also subcontract specific functionality to other more specialised services. This structure allows SOAs to adapt to the changing requirements and the environment in which they operate as well as reducing the dependency of a consumer or a service on a specific service vendor.

Loose Coupling

The modular nature of SOA relies on there being only a loose coupling between services and system components. Utilising this modular nature, SOA aims to minimise the dependencies between systems and their constituent components [21], allowing a single Service Implementation to service, or provide functionality to, multiple consumers or other services. Loose Coupling is regarded as *"a feature of software systems that allows those systems to be linked without having knowledge of the technologies used by one another"* [22; 23]. It allows the abstraction of Service Descriptions from their implementations and the sharing of schemas and contracts between services rather than the sharing of classes as would be the case in an Object-Oriented system [24; 25]. In turn consumers are able to source functionality from various providers through a service registry or a marketplace (see Figure 2.1). Although reducing the coupling between components can result in an increase in upfront design and implementation work, it can enable organisations to change and adapt IT systems with minimal impact reducing the long-term system maintenance costs [7].

There are however multiple dimensions to be considered in the context of loose coupling within SOA, specifically: functional, temporal, and transactional coupling [7]:

Functionally SOA extends interface-based design [26] by enforcing the interfaces to be specified using enterprise-wide semantics, such as XML-based WSDL, as well as a common data model which allows services to exchange data and interoperate. As such individual services are assumed to be responsible for handling the transformation between the specified data model and their internal models. This is similar to the Data-Distribution Service (DDS) approach [5] in which processes publish or subscribe to data according to a particular globally tagged type (DDS is discussed further in Section Section 2.3.3).

Transactional coupling refers to the handling of data and state where multiple services access and update the data. The SOA implementation must therefore trade-off the properties of atomicity, consistency, isolation, and durability (ACID) [27] against tighter coupling between services, data, and resources. Where atomicity ensures that all changes or made or no changes are made in an all or nothing manner. Consistency refers to guaranteeing any constraints on the system and isolation ensures that concurrent execution has the same result as sequential execution. And durability ensures that the committed results are stored permanently in a fault tolerant fashion. Therefore a basically available, soft state, eventual consistency approach is often adopted whereby greater importance is given to the availability and performance of services rather than the total accuracy and consistency of the data between individual processes [28]. This of course would restrict the domains which SOAs could be utilised. As outlined in Section Section 2.1.2 and Section Section 2.1.3 SOA as an archtectural style leaves this issue to be addressed by particular modules of the system by separating the concerns of data processing and data management, such that services operate on data which is supplied to them by the workflow engine.

Temporal coupling relates to understanding, defining, and managing the complexity of the temporal properties of services. Typically service-orientation assumes an event-based perspective where a workflow engine is responsible for managing all concepts of time synchronisation and state [29]. Section Section 2.1.3 discusses the detail and the challenges surrounding temporal coupling.

In order for services to be loosely-coupled they, and the service-oriented system as whole, must support the late binding of services, also known as dynamic binding [30]. As depicted in Figure 2.1 in order for services to be utilised by a consumer the two must be bound together. Service binding is therefore the method, or protocol, by which an individual instance of a service is identified, or created, and using the service definition the consumer is able to connect and interact with the specified instance [31]. This may occur at either design or at runtime. In the former case the specific service interface details are statically encoded into a client's implementation [32]. However, binding may also be performed at runtime, as would be the case where the provider's service is unavailable at the required point in time therefore requiring an alternative service to complete the required function. SOA therefore supports the concept of dynamic binding whereby in the event of a service being unavailable an alternative functionally equivalent service can be used [33] resulting in an increase in the dependability of the system. This can be taken further with *"ultra-late binding"* where service binding is not specified or performed until the latest possible moment before the functionality is required by the consumer [34]. By using ultra-late binding a service-oriented system can maintain a high level of flexibility and greatly reduce a consumer's reliance on any individual service provider, as long as alternative services exist. In order for a system to be fully loosely-coupled it must support ultra-late binding and therefore be able to guarantee the availability of a service at a potentially arbitrary binding time. To some degree this can be controlled in the context of workflows by the workflow engine which is discussed in some detail in Section Section 2.1.3.

Standardisation and Interoperability

In order for SOAs to function in the modular and loosely-coupled fashion as described above, the service interfaces must conform to a standard in order that consumers can dynamically switch between functionally equivalent services provided by different providers [35]. As long as the interfaces are standardised, and functionally equivalent services subscribe to the same service definitions, the heterogeneity of the services themselves as well as their providers is transparent [9]. Most approaches to SOA use Web Services [8] as the *de-facto* set of industry standards. Interfaces

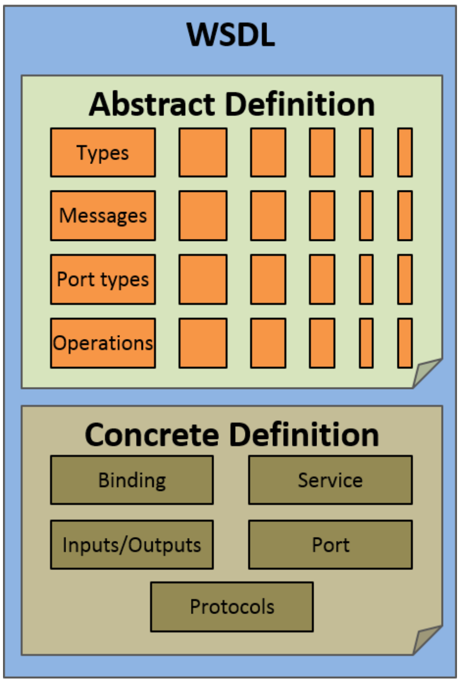


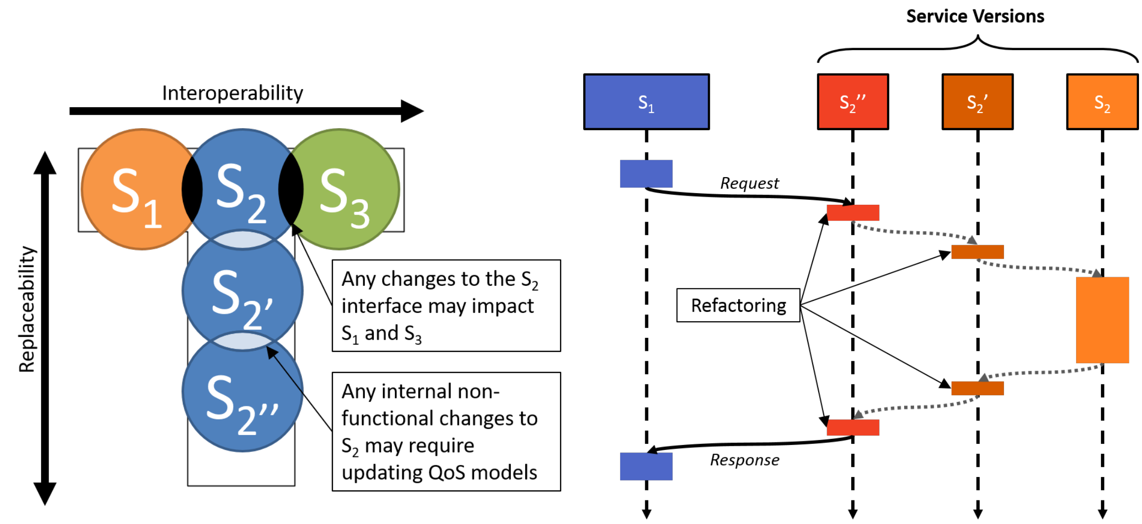
Figure 2.3: Web Service Definitions Language (WSDL)

are specified using Web Service Definitions Language (WSDL), communication protocols such as Simple Object Access Protocol (SOAP) or REpresentational State Transfer (REST) are also used and other aspects such as security, resource management, and Quality of Service (QoS) can be specified using WS standards, or alternatives, such as WS-QoS [36] which is described in the next section [37].

The Web Service Definitions Language (WSDL) provides one of the most commonly adopted standardised methods for specifying service interfaces. A WSDL definition is specified in XML with various components as outlined in Figure 2.3 [38; 31]. Particularly it must detail the data types, messages, and operations which a service requires and provides. A definition may be split into the abstract and concrete parts separating the detail of the network configuration from the functional description of the service [24; 38].

Even with strict adherence to standards, one of the interesting challenges with regards to service interoperability is the management of service evolution over time. If managed correctly it can be regarded as the *"continuous process of development of a service through a series of consistent and unambiguous changes"* [39]. These changes may be [40]:

Structural affecting the data types, messages, operations, syntax and semantics of the services and their definitions.



(a) T-Changes, adapted from [40], showing the horizontal and vertical dimensions of service evolution.

(b) Refactoring of services adapted from [41].

Figure 2.4: Maintaining service interoperability under evolution

Behavioural which could include any changes in system protocols.

Policy-induced which may introduce operational constraints on service operation and require stricter, or looser, enforcement of QoS.

These types of changes refer to those which are externally observable and therefore have a direct impact on their interfaces and therefore the consumers, services, and other system components which interact with them. The changes can also be categorised as either *shallow* or *deep* where the former refers to those which affect an individual service and those system components which interact with it directly. The latter refers to those large-scale changes which affect multiple system components. In terms of shallow changes, and beyond using satisfactory versioning of services, in order to manage the interoperability of services the compatibility between service versions must be controlled. One such method is defining the changes in terms of *T-Changes* with either horizontal or vertical change [40]. A horizontal change is one that impacts the interfaces of a service whilst a vertical chance is an internal non-functional change to a particular service as depicted in Figure 2.4a. For example the removal of operations or the changing of data types are horizontal changes and in this case are not backwards compatible, whereas altering implementations without changing the interfaces or adding optional new data types are vertical changes that are backwards compatible alterations. As services are iteratively evolved with new interfaces, definition changes, new data types, and new performance information the *substitutability* and *replaceability* in the vertical dimension can be formally analysed using Liskov substitution principle [40].

Where the evolution of a service does not maintain compatibility in both dimensions, and therefore does not sustain *strict* compatibility, it may be necessary to refactor the service in order to maintain the service's interoperability with consumers, other services, and system components. Refactoring is the *"controlled technique for improving the the design* [by] *applying a series of small behaviour-preserving transformations"* [42]. One approach, shown in Figure 2.4b is to wrap a system or software component in order to reuse it in an incompatible system [24]. This particular refactoring process is often adopted as it can provide a cost-effective way of integrating services with significantly complex business logic into the system. As can be seen in Figure 2.4b the wrapper transforms the requests between versions, which may include data transformations or reordering of communications due to protocol changes [41]. Moreover as a service evolves the understanding of its operational behaviour must also evolve and the QoS models must adapt appropriately. A detailed discussion on QoS and on the impact of evolving operational behaviour can be found below and in Section Section 2.3.

Quality of Service

Quality of Service (QoS) is a concept that underpins the modular and loosely-coupled nature of SOAs and provides confidence that services will interoperate in a timely and correct fashion. QoS is typically defined using a Service Level Agreement (SLA) and must be managed by the service-oriented system as a whole. Specifically QoS exists in order to improve a consumer's trust in both the system as a whole as well as in individual providers. Therefore a service must demonstrate that it can consistently behave as expected and that it will continue to do so [43; 44].

A SLA provides the formally defined contract of the relationship between the service provider and the consumer. It explicitly states the expectations and obligations, often defined as Service Level Objectives, existing in the relationship [45; 46]. The SLA is associated with the relevant service definition and encapsulates: *"What to measure, How to measure it, Who does what,* [and any] *Guarantees"* [47]. The specification of the SLA must also allow for automatic service provisioning and monitoring. Typically SLAs are administered independently of the provider or consumer, often by a brokering system, but are associated with the specific services. In the domain of Web Services, SLAs are often defined using the Web Service Level Agreement (WSLA) standard or the Web Services Agreement Specification (WS-Agreement) [48]. WS-Agreement

```
<wsag:Agreement AgreementId="xs:string">
   <wsag:Name>
       xs:string
   </wsag:Name> ?
   <wsag:AgreementContext>
      wsag:AgreementContextType
   </wsag:AgreementContext>
   <wsag:Terms>
       wsag:TermCompositorType
   </wsag:Terms>
</wsag:Agreement>
```

(a) WS-Agreement general format

```
<wsag:ServiceProperties
   wsag:Name="xs:string" wsag:ServiceName="xs:string">
   <wsag:VariableSet>wsag:VariableSetType</wsag:VariableSet>
</wsag:ServiceProperties>
```

(b) WS-Agreement property format

Figure 2.5: WS-Agreement Example

```
<wsla:SLAParamter name="Performance" type="double">

<Metric name="ResponseTime" type="double" unit="seconds">
```

Figure 2.6: WSLA structure for response-time in seconds

is a protocol that uses XML notation to establish agreements between entities and also allows the formation of agreement templates using the form shown in Figure 2.5. For the purposes of this research, these would specifically consist of service properties corresponding to response and execution times which is shown in Figure 2.6 using WSLA formatting.

The need for QoS derives partially from the evolution of autonomic computing which aims to allow computing systems to become self-manageable in the same way as a biological human system [49]. The drive for autonomic systems stems, as with many technological advances, from the defence industry with the Situational Awareness System (SAS) by the Defence Advanced Research Projects Agency (DARPA) in 1997 which aimed to provide a communication system enhanced with location information for battlefield situations [50]. Autonomic computing continued to be developed with: NASA working on Autonomous Agent systems for long range space missions [51]; DARPA working on the architectural Dynamic Assembly for Systems Adaptability, Dependability, and Assurance approach which started to deal with the complexity of large distributed systems with the notion of using monitoring agents and adaptation engines to optimise system performance [50]. The autonomic computing journey then moved into full swing with

IBM's autonomic computing initiative in 2001 [52]. In their autonomic blueprint IBM introduced the four core concepts of autonomic computing: self-optimisation, self-healing, self-protection, and self-configuration [53]. The concepts of autonomic computing continued to evolve in the domain of cloud computing with additional QoS parameters aimed at improving the reliability and scalability of services [54]. A more detailed discussion on specific QoS parameters can be found in Section Section 2.3.3.

Work by Al-Kalbani et al. [55] and also Singh and Chana [49] looked at how QoS techniques can be classified. Specifically [49] provides a taxonomy using the QoS parameters of: scalability, availability, reliability, security, cost, time, energy, resource utilisation, and SLA violation where methods are qualitatively marked as either true or false for each of those elements. (A detailed analysis of the features of individual QoS techniques is presented in Chapter 3.) Alternatively Al-Kalbani et al. [55] identify different perspectives on how to classify QoS methodologies: Nature, Form, Process, or Objective based:

Form View Describes how the approach is represented in terms of notation and mathematical formalisms.

Process View Considers the tools and techniques used by the approach and also how it was evaluated.

Objective View Outlines what the methodology was trying to achieve, considering the stakeholders and how decisions are made to distinguish between *"good"* and *"bad"* services.

Nature View Describes the QoS model in terms of the properties which the methodology considers and the architectural level which it focusses on.

The *"Nature View"* in [55] and the taxonomy in [49] provide a strong basis upon which QoS techniques can be evaluated for their potential effectiveness in providing QoS models. Al-Kalbani et al. [55] identify two levels of classification within this view: service level and system level. The service level considers the aspects of: response-time, throughput, availability, accessibility, and reliability, whilst the system level considers interoperability, security, and maintainability.

One example of a QoS approach by Liu et al. [56] who define the QoS of an individual service as the probability of successful execution given some response-time deadline and accounting for execution failures within the service. The authors then propose a mechanism for improving the

QoS by using the concepts of N-versioning [57] and N-copy [58]:

$$\underbrace{\prod^{s}}_{\text{N-versions}} \underbrace{(1 - p_s^{r_s})}_{\text{N-copy}} = \underbrace{x}_{\substack{\text{target} \\ \text{reliability}}} \tag{2.1}$$

Where p is the probability of service s failing, r the level of redundancy for s, and x the target level of reliability for which to solve the equation [56].

The fundamentals discussed in this section so far provide the foundation for service orientation and Service-Oriented Architectures (SOAs). The modular nature and principles of loose-coupling have been presented on page 12 with respect to both functional and temporal aspects of individual services. The importance of standardisation and the challenges of evolution at both a small and large scale have been discussed. Then finally the concept of Quality of Service (QoS) was introduced along with Service Level Agreements (SLAs) as the mechanism for guaranteeing a level of performance. The concepts of QoS will be explored further later in this chapter (see Section 2.3.3). First the architectural and workflow aspects of SOAs will be considered.

2.1.2 Layers of Abstraction

As discussed in the previous section the modular nature of SOA allows services to be loosely-coupled together with the consumers being agnostic of the physical location of the provider. One of the key elements of the SOA paradigm are the clearly defined layers of abstraction that separate the service implementations from their definitions.

The layers of SOA can be defined either from a business perspective or with respect to the technical structure of the software architecture, as shown in Figure 2.7 which is adapted from [59; 56; 60; 61; 1; 34; 62]. In the former, the operational nature of the individual services becomes a business capability which can be supplied in a marketplace. In the technical domain the abstract definitions of the services are mapped through to the infrastructure on which specific software, or hardware, instances are invoked. Vertically, there are therefore three layers [1]:

Operational layer encapsulates the basic SOA functionality with the base components of services, their definitions, execution instances, as well as the methods for performing publication, discovery and runtime binding.

Integration layer hides the *"how"* of the operational layer. It exposes the service inter-

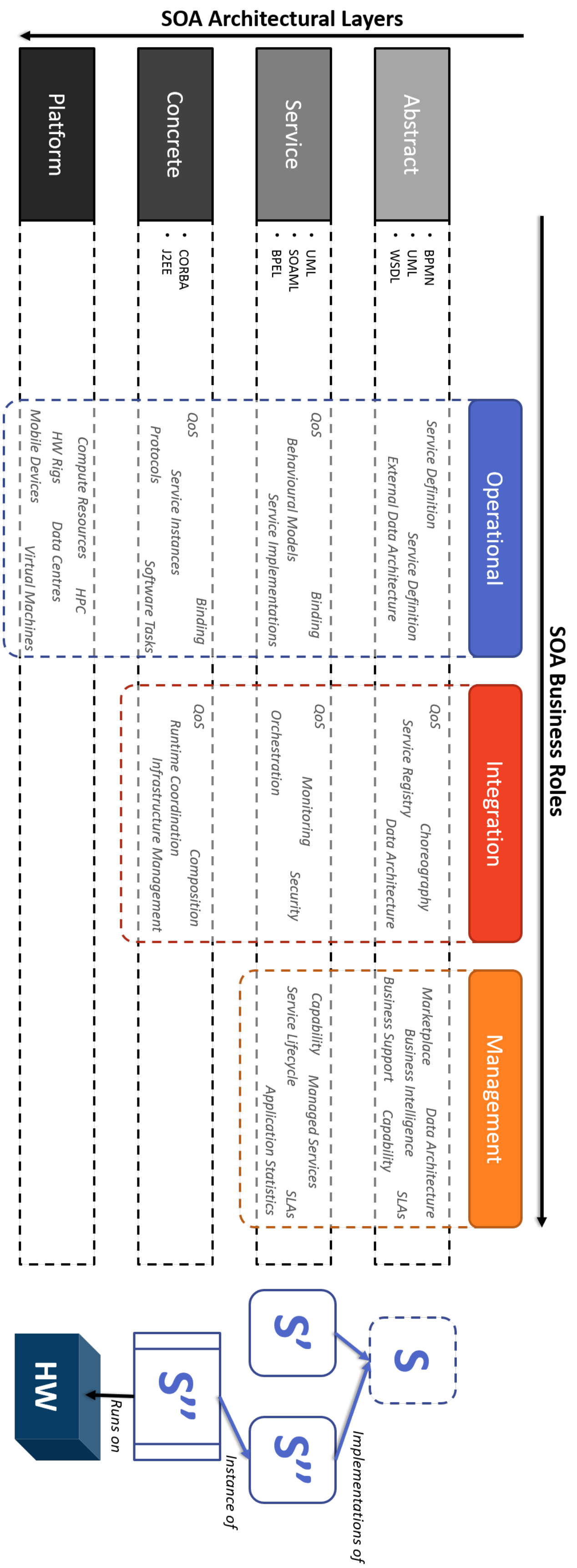


Figure 2.7: SOA Layers of Abstraction in both business and architectural/technical dimensions

faces, as well as methods for managing runtime performance, QoS, and the infrastructure. The concepts of binding are also encapsulated as part of coordinating workflows with choreography, orchestration, and composition which are explained in Section Section 2.1.3 on Page 26.

Management layer further abstracts the technical detail exposing the function of the services as business capabilities. At this level the service registry is represented as a marketplace and SLAs can be negotiated and monitored.

As shown in Figure 2.7 crossing these layers of business logic are the fundamental technical layers of abstraction in SOAs: the abstract, service, concrete, and the platform layers [59]. The remainder of this sub-section explains the upper three layers in more detail. The platform layer provides the base resources and computational infrastructure to host the running services. The platform layer itself can therefore be decomposed into the types of infrastructure such as High Performance Computing (HPC) or data-centres and then decomposed into the individual virtual and physical compute platforms servers and hardware configurations that are deployed to run the services.

Concrete Layer: Services vs. Micro-Services

The lowest two layers of SOA encapsulate the physical computational infrastructure on which the services execute, the software instances, as well as relevant communication protocols. At the platform level in the context of cloud computing this would encapsulate the entire infrastructure and would itself be comprised of multiple layers of abstraction. Alternatively it may also be integrated with the concrete layer above where the services themselves are hardware devices such as sensors, motors [63], robots [64], or other cyberphysical systems [65; 66].

The concrete layer can be viewed either in the operational or integration context (see Figure 2.7). From an integration perspective it focusses on the technologies for managing the connected services as workflows, which will be explored further in Section Section 2.1.3. Operationally this system layer must mange the deployment and runtime execution of services. This can control the communication protocols and is responsible for performing the binding between services, regardless of the type of binding that is being utilised.

At the concrete level only the instances of the services which are executing are represented, rather than all existing service implementations. However, in order to correctly manage the execu-

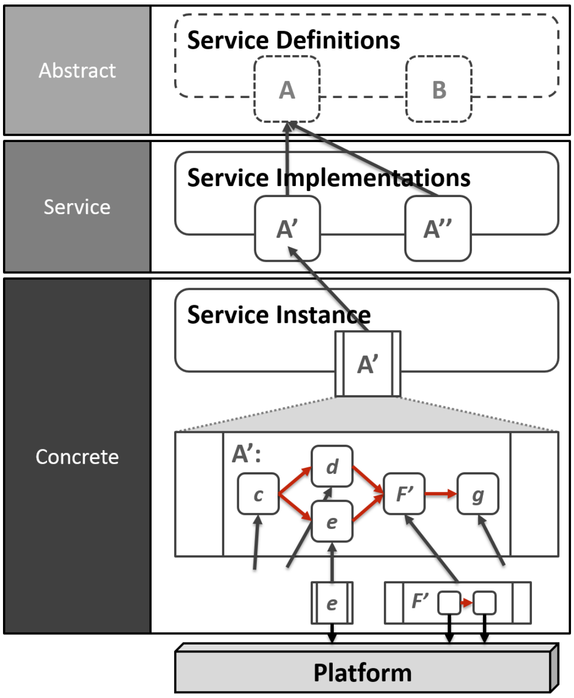


Figure 2.8: SOA technical layers of abstraction

tion of a given service its structure must be at least partially understood. Specifically, as mentioned on Page 10 services may subcontract functionality to other services [11] which implicitly forms an execution graph [67], or workflow, within the service (see Figure 2.8). The resulting constituent parts of a service may be further services, which can also be decomposed further (E.g. service $F \in A'$ in Figure 2.8), known as Micro-Services (μSs). μSs are defined as *"minimal independent processes interacting via messages"* [68] or more simply a μS is [69]: *"a functional element for which it is not practical to decompose into smaller components."* Estévez-Ayres et al. [70] in a similar fashion distinguish services from their implementations, and from the individual tasks which must be executed to provide the functionality of the service. In the example shown in Figure 2.8, c, d, e and g are all μSs which can be directly instantiated as processes on the platform.

In the context of cyberphysical systems hardware devices are likely to be considered as μSs. For software, distinguishing between services and μSs is left to the engineer to adopt best practice for modular software design without going too far and creating nano-services which are regarded as *"an anti-pattern where a service is too fine grained...whose overhead (communications, maintenance etc.) outweighs its utility"* [71].

A μS, as with any software process, will have: parameters, utilise computational resources, and a QoS. In the context of cloud computing μSs are described as tasks being the basic processing elements and have been analysed extensively with respect to [72; 73]:

Duration - the time between the submission event and successful completion.

Length - a function of the duration and CPU utlisation, being measured in Millions of Instructions.

Disk usage - the distribution pattern of disk utilisation over the execution duration.

Priority - characterising tasks into different task types for: low, medium, and high.

Service Layer

The central service layer spans across each of the business roles. At the operational level this provides the set of all implementations of a given service definition (Figure 2.7 and Figure 2.8) alongside their respective behavioural models [59]. It is from here that the system must select which services to utilise, and therefore bind, at runtime. Each of the services in the set must be fully interoperable (see Standardisation and Interoperability on page 17 and Figure 2.4a), implementing all of the functionality specified in the service definition. Different implementations will likely have different QoS and may therefore be appropriate for different situations.

At the integration layer the behavioural information can be integrated with infrastructure management in order to expose the management layer to a set of managed services. As will be discussed in Section Section 2.1.3 the different combinations of services can be considered during orchestration (which is defined in the next section) to provide a workflow with an overall acceptable QoS.

The service layer also introduces the management roles in SOA where the service implementations can be described in terms of their capability to perform a function [34]. At this point the required QoS is agreed upon using a SLA. This layer can introduce functions for managing service lifecycle and for performing analytics on system performance.

Abstract Layer

The topmost layer of SOA moves away from any reference to individual service implementations and instead deals entirely with the set of service definitions which reflects only the functional nature of the services [59]. The methods and appropriate data structures may be exposed using the likes of WSDL. At the integration level the service registry is exposed storing references to all the service definitions. QoS is also introduced at this level not as non-functional feature of the services, but as a non-functional requirement to be used in selecting an appropriate implementation. Often the QoS may be specified at the workflow level rather than for an individual service.

From the management perspective the service definitions can be exposed in a marketplace in terms of their functional capability. Additionally any domain specific information, such as data architectures which will be used across a set of services would be defined here. This layer can also be used to provide additional business functionality to the system by encapsulating the management functions from the service layer into business intelligence and support functions.

2.1.3 Workflows

Throughout the previous section reference was made to the concept of workflows. This section briefly explores what they are; how they fit into the SOA paradigm; and the types of workflows.

A workflow can be defined as the *"executable business process that can interact with both internal and external services"* [74]. A workflow specification, using the likes of Business Process Execution Language (BPEL) or Business Process Modelling Notation, should capture the relationships between services, defining how they collaborate to provide an overall capability [75]. It can be considered as the arrangement of services' execution to form a complete process in the same way that an arrangement of musical scores can form an orchestral performance. Regardless of the technology that is used to define a workflow they are normally represented and studied using Petri nets [76] which can represent the majority of workflows [77; 78].

Layers of Abstractions

There are three categories of workflow formation: choreography, orchestration, and composition which correlate to the SOA abstract, service, and concrete layers respectively, all within the integration business role:

Choreography occurs at the abstract layer to *"define the constraints and requirements"* of the workflow using just the service definitions [79]. This creates an application graph which consists of the set of service definitions S, the directed relationships between them R, and the set of QoS constraints Q [59]: $AG = \{S, R, Q\}$.

Orchestration introduces the protocols for communication and message exchange. The SOA service layer is aware of all the possible implementations of the service definitions and therefore an expanded graph is formed from the set of all service implementations SI, directed relations R' between the service implementations as informed by the application graph, and the QoS constraints from the application graph [59]: $XG = \{SI, R', Q\}$

Composition which is the process responsible for selecting and binding services together and to consumers. If there are multiple service implementations, and therefore multiple possible workflows, this process must choose the most appropriate solution to reduce the expanded graph to the execution graph consisting of only the required service implementations and respective relations such that the QoS will meet the required level [59].

The execution graph EG is a tuple $\{SI', R'', Q'\}$ such that SI' is a subset of set of service implementations from the expanded graph XG. Therefore R'' is also a subset of the relations from XG such that the final QoS Q' is at least as good as the user specified QoS. In this context, and the context of this research, the QoS is assessed purely in terms of timing properties, for example response-time.

The majority of solutions to workflow formation focus on the problem of composition which is computationally complex due to potential size of the search space and the need to compute compositions at runtime in the context of ultra-late binding. Compositions are also often recomputed at runtime by management systems in order to try to avoid SLA violations when the observed QoS is degrading [59; 80; 81].

Much of the challenges of choreography and orchestration relate to the semantics of expressing the workflows [82]. Aalst et al. [83–89] discuss in comprehensive detail the advantages and disadvantages of the various technologies that can be adopted to represent workflows.

Workflow Patterns

Workflows can be viewed from different perspectives that capture different aspect of how parallelism, synchronisation, information handling, etc. are handled. Each perspective has its own set of patterns, as outlined by Aalst et al. [83–89]. The five perspectives are:

Control-flow patterns are comprised of several categories: basic control flow; branching and

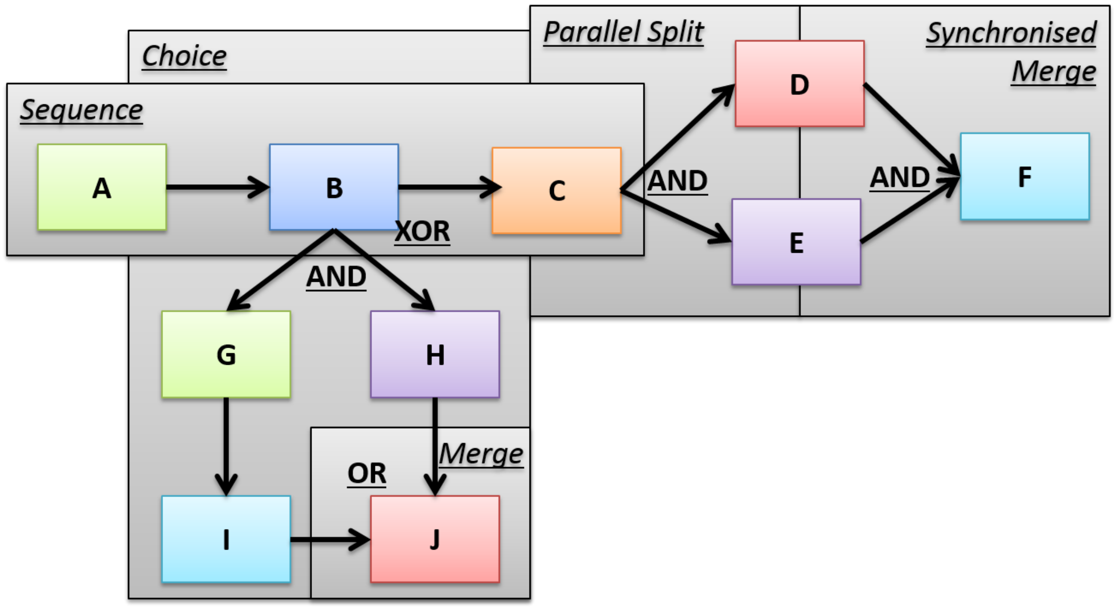


Figure 2.9: Sample workflow patterns

synchronisation; multiple instance; state-based; cancellation and forced completion; iteration; termination; and trigger patterns [84]. In total these categories are comprised of forty-three patterns each of which requires a different strategy to manage its execution as well as different semantics to represent it. As such different technologies, including BPEL and YAWL [90], handle different subsets of the patterns, but none facilitate all of them. Eight example patterns are depicted in Figure 2.9 including sequential operation of services A, B, and then C; parallel operation of services D and E involving splitting and merging before and after; as well as decision making.

Two particular patterns that pose difficult challenges in terms of QoS, and are not supported by workflow management systems, are *"Arbitrary Cycles"* and the *"Transient Triggers"* [65]. Arbitrary cycles are a form of the halting problem [91], which is known to be NP-hard to solve, whereby there is either no clear definition of a halting condition or no clear indication of when that condition would be met. Arbitrary cycles become further challenging by facilitating multiple entry and exit points from the cycle. In the context of SOAs solving these pattern and therefore providing an exact solution to the QoS of the workflow would require an understanding of the exact behaviour of each of the services with respect to their QoS as well as complete understanding of the data value flow through the workflow.

Data-flow [86] patterns are used to capture how data is represented and utilised in workflows. Aalst et al. identify forty key patterns which are categorised into: visibility; interaction (internal and external); transfer; and routing patterns. Relating back to the transient trigger control flow pattern are the routing data and event based task triggers. The event-based task trigger extends the transient trigger pattern discussed previously be defining the external data event and condition which is required in order to begin or resume execution. The data-based pattern facilitates the arbitrary cycle control-flow pattern and refers specifically to initialising or resuming execution of specified services dependent on the value of a given data object within the workflow context. In this case the value of the data object would be changed by operations of other services within the workflow.

In the context of QoS these patterns, and the technologies which support them, do not facilitate the prediction of QoS without requiring a formal analysis of how the workflow will perform under specific conditions [92]. Alternative methods to solving the data-based pattern have been proposed using probabilistic techniques and evaluations of the rate of change of data values [65].

Resource-flow [85] patterns are used to complement the control and data-flow patterns from the resource perspective in terms of how individual services and processes require and interact with resources. As before they are divided into several categories for: creation; allocation (in either a push or pull fashion); detours for handling interrupted execution; auto-starting; visibility; and multiple resource patterns for concurrent utilisation. These patterns extend the basic understanding of resource management consisting of three key phases: acquisition, lifecycle, and release [93].

Figure 2.10 depicts the resource phases from creation through to their utilisation and release by processes. Beginning with creation, in the context of SOAs resources are defined as either a *human* or *non-human* and can be related under an organisational structure with privileges affecting their accessibility and visibility. From the perspective of computational processes a service's resource can

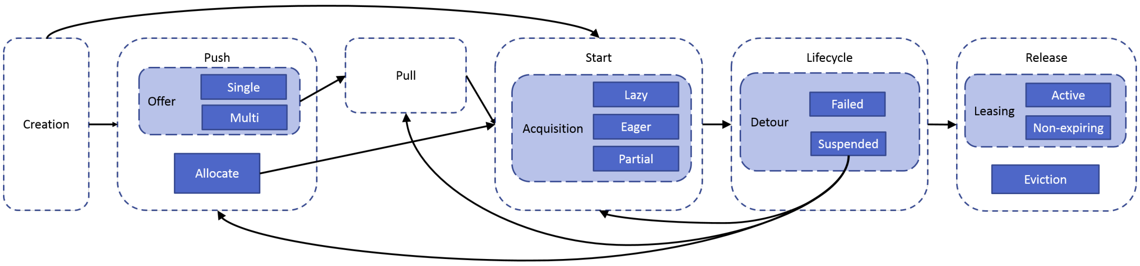


Figure 2.10: Resource Lifecycle, adapted from Russell et al. [85] and Kircher & Jain [93]

refer to memory, processor time, persistent storage, or other network or physical resources. Alternatively their may refer to specific items such as files. As each resource is created it may be assigned properties relating to how it can be distributed and made available to processes for utilisation. The creation of a resource may also *Trigger* a process to begin.

Once created a resource must be allocated to one or more processes. Firstly resources are made available to a set of processes which then pull them ready for use at runtime. Resources can be acquired by a given process in several fashions: lazy, eager, or partially. Using eager acquisition resources are fully acquired at start-up and suspension of their access blocks the process' execution. Lazy acquisition the process acquires required resources at runtime, either at start-up, upon user-request, or at a given point during execution. Alternatively a process may stage the acquisition of resources over the execution duration rather than requesting all resources at start-up [93].

At runtime suspension or failure of the resource can result in blocking or failure of the process. Where multiple processes require the same resources, they may be repeatedly suspended and reallocated to alternative processes. Finally processes may release resources upon termination or throughout their execution. From a workflow perspective in order to reliably schedule the execution of the services, and therefore predict QoS the system must understand: levels of resource contention (Figure 2.18 on Page 51), when resources will be allocated and released, and the impact of resource suspension or failure on other QoS.

Exception-flow [88] refers to how exceptions are handled by a workflow and specifically extends the resource flow patterns whereby: resource are unable to be offered; services are unable to acquire resources; detours occur requiring reallocation of resources; or how resources are released from failed states. In the first case the workflow system must manage how resources are re-offered to services and how to withdraw any multiple-offering once a service has acquired the resource. A failed allocation may be the result of a withdrawal or may occur due to multiple processes requiring the resource concurrently. The workflow must manage the release of resources if a service has failed, or may be required to forcibly release resources to allow for their reallocation. Each of these exceptions may cause another exception type to occur and methods for how these can be handled are discussed further in Section Section 2.2.2.

Interaction-flow [87] refers to the basic elements for communication and interaction between services allowing workflows to be constructed. These describe aspects relating to sending and receiving messages in sequential and concurrent fashions. Mechanisms for describing selective communication to decide which messages to send or receive as well as handling multiple instances of services.

Different technologies adopt differing paradigms for communication, for example DDS adopts a publish/subscribe [94] approach where services are persistent and run concurrently pushing and/or pulling data based on specific tags. Alternatively the majority of approaches are event-based where the messages are sent at the beginning or end of service instance execution. This becomes more complex when specific functionality is sub-contracted to other service or μSs creating a flow of interaction within the service [95].

2.2 Dependable Real-Time Systems

Service-Oriented Architectures (SOAs) are a distributed system paradigm specifically designed to increase dependability. This section explores the central concepts around system structure, dependability, and specifically dependability in SOAs.

2.2.1 System Model Definition

This research is focussed on exploring a specific part of the SOA paradigm, however in order to understand the central concepts particularly of dependability a brief introduction to the definition of systems is required. This section outlines the differences between systems, components, and architectures.

Systems, Components, and Architectures

A system is *"a group of related hardware units or programs or both, especially when dedicated to a single application."* [96]. In principle it is an entity that consists of one or more components which are designed to cooperate and interact in order to perform a specific task or tasks [97]. An individual system may itself be a component in a larger system of systems.

The system and all its elements are defined by its architecture which describes how the components are composed together and captures the relationships between them. Specifically a software architecture can be defined as *"the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them"* [11]. A system architecture can therefore be described structurally, functionally, and in terms of non-functional requirements such as QoS.

SOA is however not a system architecture, but rather is an architectural style which guides the design and development of SOA system architectures. An architectural style is the *"family of architectures related by common principles and attributes"* [7] which in the case of SOA have been described in Section Section 2.1.1. As a result depending on from which business role (see Section Section 2.1.2) a SOA is described a different perspective is presented. At the management level a SOA provides the *"set of services that constitute IT capabilities and can be used for building solutions"*. Conversely at the operational level it provides the *"programming model complete with standards, tools, and technologies"* [7].

Environment

When defining any system architecture it is necessary to distinguish between the system itself, the environment in which it operates, and the system's operational context. The environment is the collection of all the elements that form the surroundings in which the system is utilised, developed,

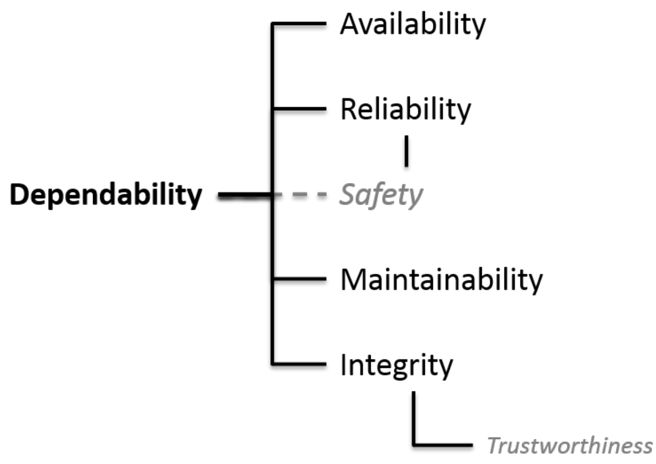


Figure 2.11: Attributes of dependability [98; 2].

produced, or retired. Conversely the context is the subset of elements of the environment which are required to defining the specific states under which the system must operate.

Considering SOAs and for this thesis, the environment of a running service is the workflow in which it is executing as well as its host machine. These are elements that will directly impact the service's execution and therefore must be taken into consideration at design time. A workflow however must consider as its environment the wider system elements including workflow engines, service registries, and the service consumers.

2.2.2 Dependability

Underpinning any research relating to Quality of Service (QoS) is the concept of dependability which is the *"ability to avoid service failures that are more frequent and more severe than is acceptable"* [2]. This concept provides metrics for measuring how *"dependable"* a system is with respect to specific attributes and defines the basic constructs for understanding system faults and methods for handling them.

The concept of dependability was initially explored by J. Laprie [99] and was formalised into a taxonomy by Avizienis et al. [2; 98] and is shown in Figure 2.11. The concepts of availability and reliability are central to defining and evaluating QoS:

Availability refers to the ability of the system to provide the correct service when required. It is considered as a measure of the frequency and consistency of the delivery of the correct capability [100]. In terms of service delivery it captures the ratio of alternation between correct, incorrect, and no service.

Reliability provides the measure of the continuous duration for which the system is able to maintain correct service delivery [100]. A service's reliability is therefore a measure or a distribution of the mean time between failures.

Faults, Errors, and Failure

The need to specify the attributes of dependability arises from the need to tolerate threats, where a threat may be either a fault, error, or failure. There are four means by which threats may be tolerated [2; 100]:

Fault Prevention also known as dependability procurement, which attempts to prevent faults occurring through robust system design and development.

Fault Removal or mitigation occurs once faults have been identified, this would ideally be during development but is often after execution.

Fault Forecasting refers to the process of evaluating the system behaviour with regards to fault existence and their activation (discussed below). These methods attempt to predict the number of faults present in the system and their likelihood of becoming activated and causing the system to enter into error states.

Fault Tolerance specifically refers to techniques designed to increase the system dependability in the presence of faults during execution. Such mechanisms are designed to inhibit the development of faults into failures, as discussed below. In the context of SOA, and generally in distributed systems, techniques such as recovery blocks [101], N-versioning [57], and N-copy [58] are common place. The former two refer to embracing design diversity with multiple implementations of the same service which can operate either sequentially or concurrently to handle failures. The latter refers to utilising multiple instances of the same service to mitigate failures caused by either data or the operational context of a given service. These approaches directly inspire the modular and loosely-coupled nature of SOA and are therefore inherently part of it.

The above method categories are designed to mitigate faults, errors, and failures which must be understood as a chain of activations, as shown in Figure 2.12a:

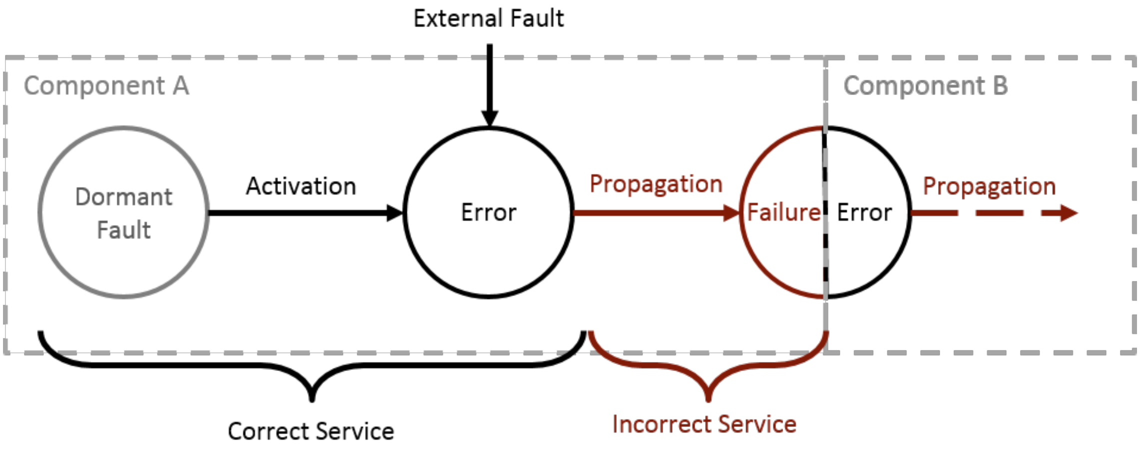
Faults can either be dormant or active:

- **Dormant** where it merely exists but has yet to be activated. Dormant faults may never be activated and can continue unnoticed.
- **Active** faults are those where an input to a component transitions it to an erroneous state. Fault activation transforms a fault into an error.

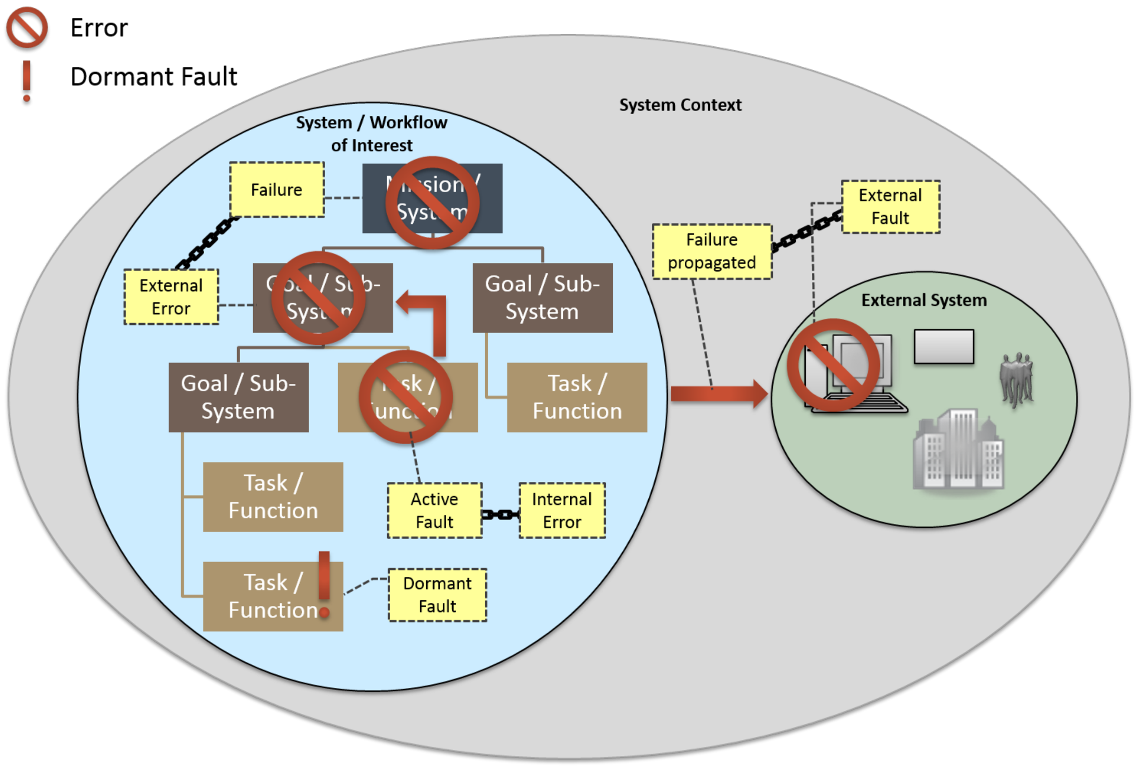
Errors can be either internal or external:

- **Internal** errors are handled internally and do not reach the component interface and so do not propagate to other components.
- **External** errors result from a propagation of the error to the component's interface and may therefore propagate into another component. In the context of SOA this may be an incorrect result from one service being passed on to the next service in the workflow.

Failures occur due to the propagation of errors to the system boundary resulting in the system deviating from correct service operation. Depending on the system and failure type they may be permanent or transient failures. In a system of systems a failure of a system can be regarded an activated fault within the wider operational context. Figure 2.12b depicts an example of a C2 system, with a mission and goals, where an internal error occurs, activating a fault which propagates up the tree to cause an external error and consequently mission failure. That failure propagates as an external fault to the wider system. In Figure 2.12b there is one dormant fault in a task and one fault that has been activated in another task which is regarded as an error. That error can propagate up the hierarchy of the system



(a) Fault-Error-Failure chain, adapted from [2]



(b) Fault chain propagation through a workflow or C2 system

Figure 2.12: Fault propagation

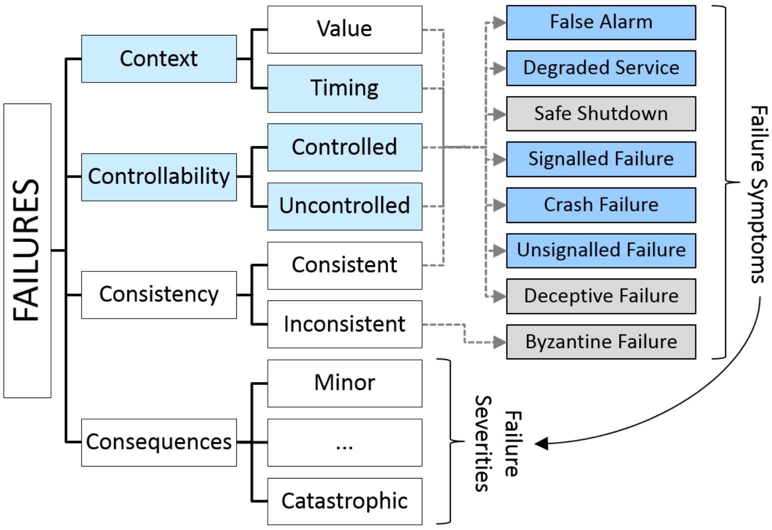


Figure 2.13: Failure modes adapted from Avizienis et al. [98] highlighting the modes of interest to this research.

to cause another error and failure at the system level. When this propagates outwards to another system it becomes a fault within that system.

Faults and Failure Types

Faults, errors, and failures can be of various types and can be permanent or transient in nature where transient faults are caused by either external interactions or by the operational context of the system [2]. For example software flaws or production defects are permanent whilst input mistakes are transient. Figure 2.13 depicts a set of failure modes caused by activated faults and their propagation across system boundaries.

This research will be focussing specifically on physical SOA operational faults that are non-malicious and not caused by human interactions. These are highlighted in Figure 2.13 and relate specifically to fault activations caused by physical deterioration and interference. Specifically this thesis considers the failures relating to timing and can result in the highlighted symptoms.

2.2.3 Dependability in SOA

As mentioned previously Service-Oriented Architectures (SOAs) are designed to enhance a distributed system's level of dependability through features like loose-coupling, modularity, and standardisation. However all systems remain prone to failures and this section explores in further detail some of the methods that have been adopted in SOAs to increase system dependability. First a study of the types of faults that can occur in SOAs is discussed which will provide the foundation for much of this thesis.

Faults in Services

For any given system type there are a specific set of faults that specifically pertain to it. Bruning et al. [102] present a taxonomy of SOA faults under five categories: publishing, discovery, composition, binding, and execution. Figure 2.14 depicts each of these categories and their respective faults, except discovery as it is outside the scope of this thesis. Under each category are specific faults or fault groups that can be decomposed further down to specific faults such as service description formatting or input errors such as values out of range.

Highlighted in the figure are specific faults and fault categories of particular relevance to this research in QoS. With regards to service publishing faults the service descriptions should present the expected level of QoS which can be used to define a SLA. Many technologies do not facilitate this in the service description semantics. Further unless there are mechanisms to guarantee that the described QoS is accurate there may be a mismatch between the service implementation and the advertised description. A fault of this type may result in a faulty workflow composition that is unable to meet the specified SLA. During binding if the service description is incorrect the system may bind to the wrong service. Each of these may result in the workflow producing an incorrect result.

Additionally most prevalent to individual service QoS are the challenges relating to execution timing due to server crashes and communication failures. The Bruning et al. [102] taxonomy shown in Figure 2.14 depicts potential propagation of service description publishing faults through to composition, binding, and execution faults, as well as a range of other fault types under each of those categories. However this taxonomy does not provide the categories required to understand and satisfactorily explore timing challenges. Specifically the *Service/Description Mismatch* does not have a category for response-time, and correspondingly the execution timing category assumes complete failure rather than degradation of service. It therefore provides a strong basis to be extended with the relevant categories to explore and analyse QoS timing faults.

The next section focusses on the domain of Real-Time Systems and scheduling theory which aim to mitigate these fault types.

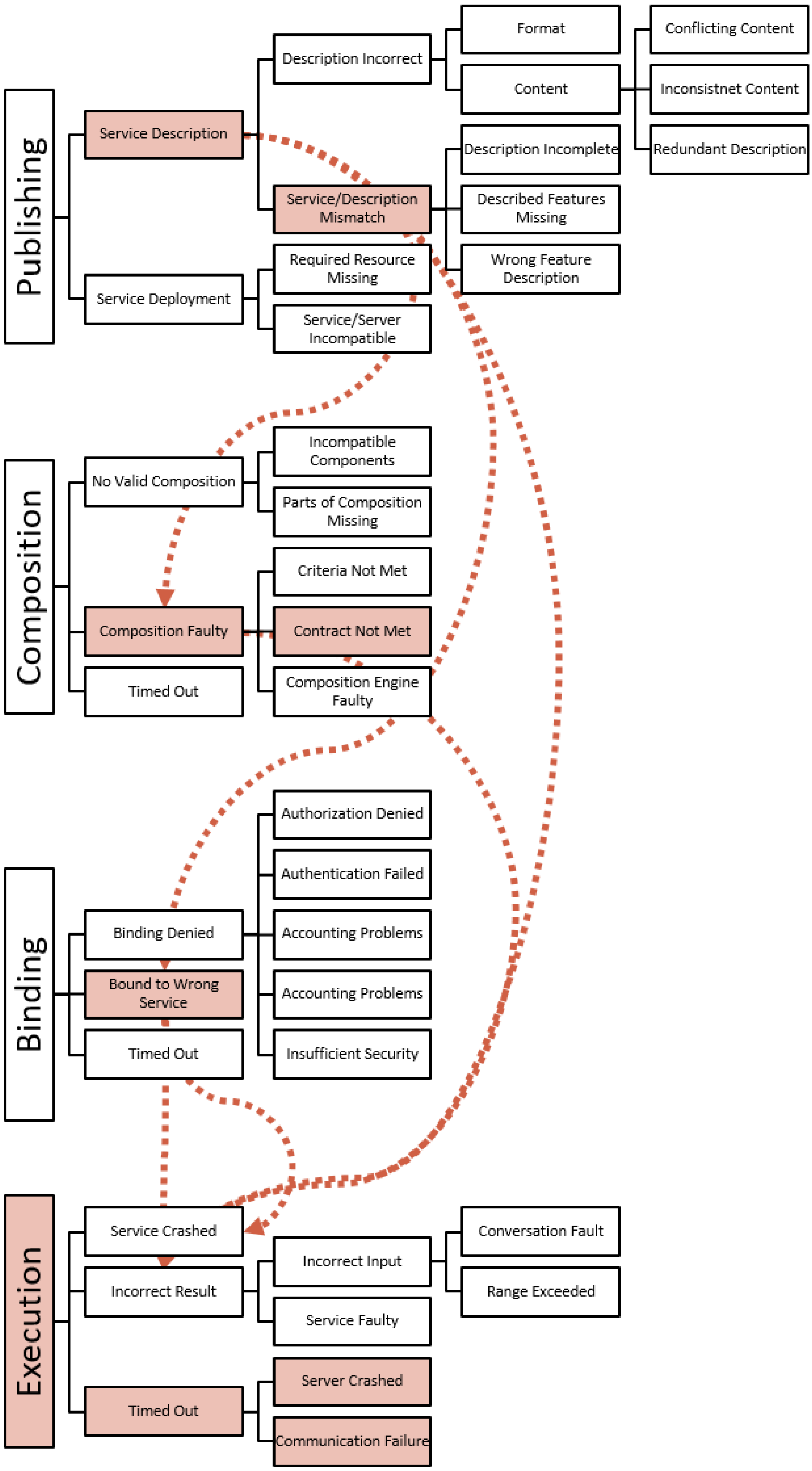


Figure 2.14: SOA taxonomy of faults adapted from Bruning et al. [102] highlighting the faults of most interest to this research.

2.3 Real-Time Service-Orientation

The research in this thesis is focussed on advancing the state-of-the-art in Real-Time Service-Oriented Architectures (RT-SOAs). The previous sections of this chapter have introduced core concepts of SOAs (Section Section 2.1) along with the fundamental principles of dependability (Section Section 2.2). This section now focusses on the Real-Time aspects of SOAs, firstly introducing the central concepts of Real-Time Systems which provides the foundations for RT-SOAs (Section Section 2.3.2). Then a detailed discussion around QoS in the context of RT-SOA is presented (Section Section 2.3.3) before the challenges which this research addresses are considered (Section Section 2.4). A detailed analysis of existing approaches for RT-SOA is presented in Chapter 3.

2.3.1 Real-Time Systems Schedulability

Although SOA embraces the concepts of dependability in order to build robust real-time systems using service-oriented principles there are additional requirements with regards specifically to ensuring the schedulability of the system with respect to service and workflow deadlines. Real-time systems are distinguished as those where the *"correctness of the system behaviour depends not only on the logical results of the computations, but also on the physical time when these results are produced"* [103]. This section outlines the standard notation and terminology that is used to specify a Real-Time System followed by distinguishing between hard, firm, and soft systems. Further core concepts relating to resource management and schedulability are then discussed.

Standard Notation

In order to explore real-time systems in SOAs the underlying concepts must be introduced. In this thesis the formal notation for modelling real-time processes used by Burns and Wellings [104] is adopted:

B *Worst-case blocking time for the task:* This defines the maximum time that a task may be blocked from starting by other tasks, typically of higher priorities.

C *Worst-Case Execution Time (WCET) of the task:* This defines the maximum computation time required to complete the task. It does not take into account blocking, jitter, or interference times. In traditional real-time systems this is assumed to be fixed and known for any given task.

D *The deadline of the task, relative to the release time:* Depending on the type of real-time system the significance of the deadline may vary. In many real-time systems the deadline is assumed to be less than or equal to the task period with implicit assumption that only a single instance of a task can execute on a given processor at a time.

I *The interference time of the task:* Defines the amount of time the task spends in a waiting state due to other tasks executing.

J *The release jitter of the task:* Provides a measure for delays between releasing a task and it starting execution, without any blocking.

N *Number of tasks in the system*

P *Priority assigned to the task:* typically in ascending order.

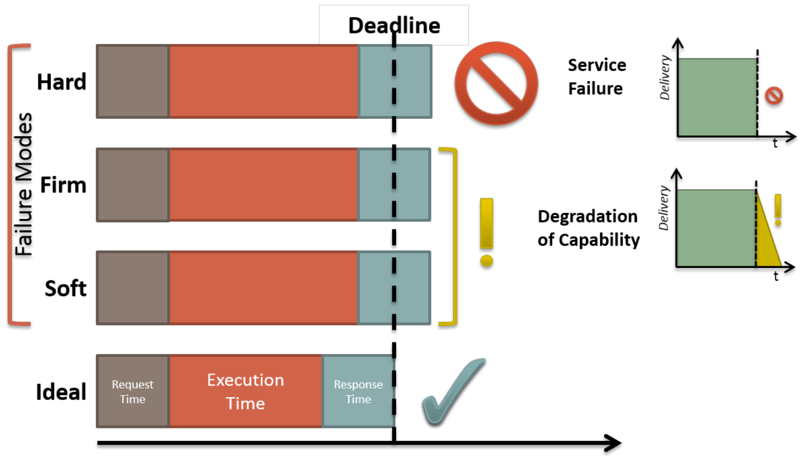
R *Worst-Case Response Time (WCRT) of the task:* Provides a measure of the response time of the task from request to completion, taking into account blocking, WCET, interference, and jitter.

T *Minimum time between task release, i.e. the task period:* In a system where tasks execute periodically this provides the minimum time between releases and as such can be used to understand the maximum load of the system comprised of *periodic* tasks. If the request times of tasks are not known, they are regarded as *sporadic* tasks and to be analysed need to have minimum frequency or period. If there are no constraints on their frequency they are known as *aperiodic* tasks.

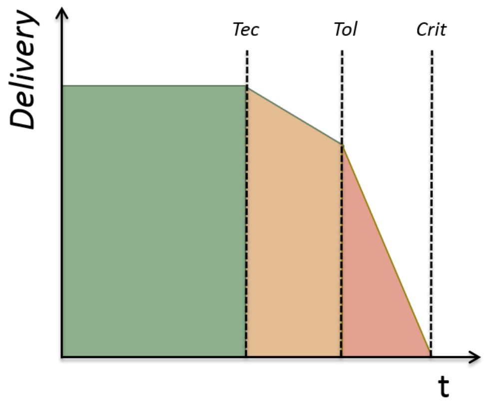
U *The utilisation of each task:*
This is the CPU utilisation of each task and is calculated as a function of the WCET and period: C/T

Hard $\rightarrow$ Firm $\rightarrow$ Soft

A real-time system can be classified as having: hard, firm, or soft deadlines (see Figure 2.15a). Soft real-time systems are those where correct functionality is still accepted even if deadlines are



(a) Hard, Firm, & Soft deadlines in a Real-Time System



(b) Technical, tolerated, and critical deadlines as introduced by R. Kirner [105]

Figure 2.15: Depiction of Real-Time deadline

occasionally missed. Conversely firm real-time dictates that there is no benefit from late service delivery. Finally hard real-time dictates that there will be consequences resulting in system failure if deadlines are not met.

Soft and firm deadlines can provide a sliding scale of degraded capability with intermediate deadlines [106] and this is explored by Kirner [105] who introduces the concept of using intermediate deadlines to tolerate performance degradation with technical, tolerated, and critical deadlines (see Figure 2.15b). The former provides a definition of expected normal timely operation of the process. The tolerated deadline provides an additional safety margin which sits between the expected and the critical deadlines.

Real-time systems are primarily comprised of two types: reactive and time-triggered systems. The first of these must respond within a specified time from an input and is particularly concerned with managing input and output jitter caused by communication latencies over networks in a distributed system. Time-triggered systems perform activities in accordance with pre-specified timing information, typically executing on a periodic basis. In reality most real-world systems are mixtures of both categories with tasks being required to adhere to different deadline types.

Resource Adequacy

Real-time systems typically focus on managing processor utilisation and ignore other constraints on resources [104]. However in order to guarantee dependable execution systems must be designed in a *resource adequate* manner. *Resource Adequacy* refers to ensuring that *"there are enough computing resources available to handle the specified peak load and fault scenario"* [103]. In that context resources can be defined as either: computational or system resource. The former refers specifically to the utilisation of resources, such as computational time or memory space and is typically quantified using *Big O* notation [107; 108]. In contrast system resources are the physical and virtual components that can be consumed by processes, such as: CPU, memory (physical and virtual), hard disk drives, network throughput, power, and files.

There are various approaches for expressing resource models and utilisation models many using either probabilistic [109; 110] or fuzzy-logic methods [111], both of which are explored later in the context of RT-SOAs. Some detailed work has been explored in modelling the power utilisation by servers, virtual machines, and processes in the context of dynamic voltage and frequency scaling [112] which look at managing the execution speeds through voltage control. In the context of energy management some techniques look at managing the efficiency and cost of virtual machine placement whilst adhering to SLAs [113–115]. In a similar vein, more closely related to real-time systems , some research has considered cost and energy modelling of CPU utilisation taking into account the cost of context switching [116].

Much of the remaining work in the field of RT-SOA relates to optimising the bin packing problem to take into account the different resource types, specifically CPU and memory [117]. Some approaches consider dynamically recomputing the bin packing problem in online situations for virtual machine or process scheduling in cloud environments [118; 119]. However these approaches are limited in that they consider resource utilisation to be a static known value about each process and also do not take into account the formalisms used in real-time systems to capture behaviour due to interference and blocking.

Schedulability

In the context of real-time systems the primary objective is to ensure that all tasks complete before their respective deadlines. *Schedulability tests* are used to calculate whether a static set of periodic or sporadic tasks will be schedulable [103]. Schedulability tests are typically either based on utilisation or response-time analysis and can be: necessary, sufficient, or exact. If a necessary schedulability test is negative then the set of tasks is definitely not schedulable. Whilst a positive sufficient test indicates that the tasks are schedulable. It is important to bear in mind that the necessary test cannot indicate schedulability and similarly the sufficient test cannot indicate non-schedulability. Exact schedulability tests are both necessary and sufficient.

The approach of utilisation-based analysis, which provides a necessary condition rather than a sufficient or exact condition for schedulability, comprises of a processor utilisation factor, which defines the percentage of processor time spent executing the task set.

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \tag{2.2}$$

and also a utilisation bound which is dependent on the chosen scheduling algorithm. For example Rate-Monotonic Scheduling [120], which assigns priorities based on their periods, has a utilisation bound of: $n(\sqrt[n]{2} - 1)$ which tends to 69.3% as the number of tasks tends to infinity. Alternatively Earliest Deadline First scheduling specifies a utilisation bound of 100% [121].

Response-time analysis provides an exact schedulability test, in the specified execution context, and calculates whether all the response-times of the tasks will be less than their respective deadlines. This can take into account task priorities, interference, blocking, and jitter. For some task i:

$$R_i = C_i + I_i + B_i + J_i \tag{2.3}$$

Calculating WCRT therefore comprises of the WCET of the task of interest, interference from higher priority tasks, and blocking time (Eqn. 2.4).

$$\begin{aligned} I_i &= \textstyle\sum_{j \in hp(i)} \left(\left\lceil \frac{R_i}{T_j} \right\rceil C_j \right) \\ B_i &= \textstyle\sum_{k=1}^{K} usage(k, i) C_i(k) \end{aligned} \tag{2.4}$$

where *usage* is a 0/1 function; $usage(k, i) = 1$ is resource k is used by at least one process with a priority less than P_i, and at least one process with a priority greater or equal to P_i. However each

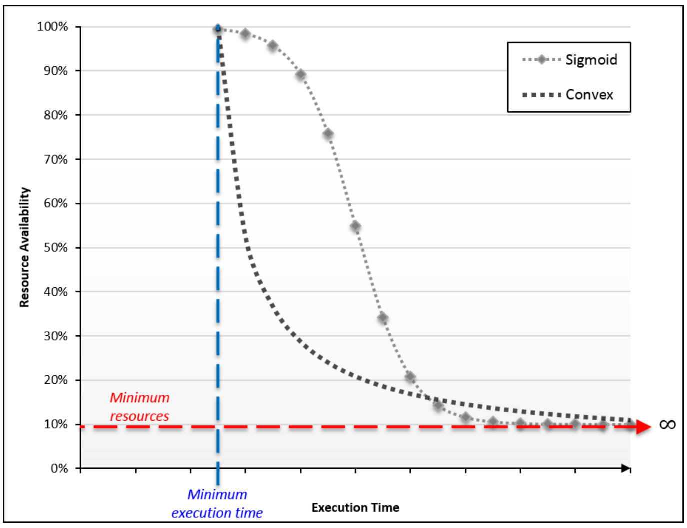


Figure 2.16: Depiction of sigmoid and convex task shape

interfering or blocking task may itself also suffer from interference or blocking, thus requiring a recurrence relation:

$$R_i^{n+1} = C_i + B_i + \sum_{j \in hp(i)} \left(\left\lceil \frac{R_i^n + J_i}{T_j} \right\rceil C_j \right) \tag{2.5}$$

These schedulability conditions provide the basis for several streams of work in RT-SOAs. For example the work by Lee et al. [106] use the utilisation bound from Eqn. 2.2 replacing the period with the deadline to guarantee schedulability of tasks on a given machine. Similarly the work by Estévez-Ayres et al. [70] evaluate their proposed RT-SOA solution using response-time schedulability analysis from Eqn. 2.5 where they assume the services are non-preemptable. Importantly in order for the system to provide any level of guarantee of its ability to adhere to deadlines it must provide a method for performing schedulability analysis.

Scheduling Theory

Moving towards general scheduling theory for distributed systems there are several further concepts to be introduced of relevance to this work [122]. The terminology and notation used in this context differs from that presented previously from the real-time systems domain. Firstly computation or *processing time* is defined as $p(i)$ where i is the number of processors servicing the task. Notably this definition, unlike the definition for WCET, allows for capturing the impact of multiple processor systems on execution. In this case $p(1)$ represents total computation to be performed and therefore the WCET running on a single processor. $p(i)$ is typically greater than $p(1)$ due to there being some cost $c(i)$ for using parallel processing. Therefore *work* W performed by a task on i processors is $ip(i)$.

Although there is cost attributed to parallelising a process there is normally a *speedup* ς that provides the measure of how much an application can be accelerated by using more processors: $\varsigma(i) = \frac{p(1)}{p(i)}$. It is normally assumed that there will be a non-decreasing *speedup* by using more processors: $\varsigma(i) \leq \varsigma(i+1) \Rightarrow p(i) \geq p(i+1)$. The concept of *slowdown* or *stretch* s provides a mechanism to represent the perceived speed of a host machine by a particular task: $\frac{1}{s}$ where $s = \frac{f}{p}$ where f is the response-time of the process. The *slowdown* factor can be used in conjunction with resource adequacy to indicate whether a given task will complete by the required deadline. The nature of this *slowdown* factor is often assumed to be sigmoidal [123] of the form $\frac{1-\alpha}{1+e^{x-\beta}} + \alpha$, where α is the minimum required resources and β the best possible execution time, and by others to be merely convex [124] as depicted in Figure 2.16. However dependent on the resource patterns, discussed in Section 2.1.3, different functions may apply to different phases of the executing tasks.

Cloud Scheduling

Moving briefly away from real-time scheduling, approaches to scheduling in Cloud computing must be considered for completeness. Nuaimi et al. [125] provide a review of Cloud scheduling techniques where the primary focus is on the load balancing. Techniques are categorised as either static or dynamic, in the former case tasks are assigned to servers, or virtual machines, based on their ability to process new requests. These approaches typically monitor resource utilisation and number of tasks loaded on any given server. One such algorithm is the MapReduce approach to partitioning a job into tasks, executing them across a set of virtual machines, and then combining the results [126]. Alternatively dynamic methods may take into account prior knowledge of task and server performance to optimise the scheduling. One such approach by Zhong et al. [127] uses Genetic Algorithms (GAs) to optimise the allocation of virtual machines as part of an online adaptive scheduler.

Traditional Cloud scheduling approaches are however not suited for real-time systems as the concept of deadlines does not exist. They can however be adapted by providing a cost function that penalises a potential deadline miss. An example by Liu et al. [128] takes this technique and then sorts the tasks based on the expected gain and does not accept those whose cost is too great. The next section will explore this area of real-time services in further detail to understand some of the major challenges in facilitating real-time service-orientation.

2.3.2 Real-Time Services

Real-Time Service-Oriented Architectures (RT-SOAs) were popularly defined by Tsai et al. [129] as an extension of SOAs that supports real-time processing which would adhere to timing constraints and function in a necessarily predictable manner with regards to composition, orchestration, deployment, and runtime management. The core ability of SOA to rapidly integrate heterogeneous systems is a fundamentally attractive proposition for many industries but is currently limited in its level of support for real-time systems. Although there are numerous application domains for RT-SOAs, the majority of research focusses on using Web Services in domains with real-time constraints and are typically either related to: decision support, modelling & simulation, or cyberphysical systems:

Decision Support Systems form a particular subset of service-oriented systems where QoS is essential to managing the time frames in which decisions are made and which actions are carried out. One such example is the use of service-orientation for coordinating fire-fighting in a forest fire scenario to facilitate the decisions made by fire-fighters with regards to their deployment and movement [130]. In a similar vein the Network Enabled Capability Through Innovative Systems Engineering (NECTISE) project [131; 34; 56; 24; 132] considered the use of SOA for decision support in a military context for C2 systems. The NECTISE project proposed managing service dependability based on using service redundancy for those services with lower QoS values.

Modelling & Simulation considers simulations as services where different simulations can interact as part of a co-simulation. The system must manage the integration of the simulations with respect to their interfaces, data structures, and timing requirements [65; 133]. In the case of automotive simulation consisting of a driving simulator and simulations of various vehicle systems the different elements will operate at different speeds, for example driving simulators typically run at 60Hz whilst complex models of fluid dynamics can take hours to run a single simulation step. Further, the data structures within the models typically differ due to tool differences and the infrastructure required to run any given simulation may vary with some simulators requiring specific hardware or HPC.

Cyberphysical Systems forms a specific category of SOAs in which there are both software as well as human or hardware services. Both decision support and modelling & simulation systems can themselves also be cyberphysical systems. With regards to human-in-the-loop services QoS is often referred to as Quality of Experience (QoE) [130] as it normally refers to the perceived system performance and is often less time-critical in nature. QoE feedback is often scored using the Mean Opinion Score [134] notation: bad, poor, fair, good, and excellent.

Hardware-in-the-Loop (HIL) systems involve hardware devices such sensors, actuators, or embedded systems where there may often be strict real-time timing constraints that will effect the system outcome. In the context of embedded systems the iLand project by Estevez-Ayres et al. [59; 67; 70; 135–144] focussed on developing a soft real-time middleware to enable the connection of embedded systems and services. The specific domain of interest considered the processing of video streams as part of a surveillance system and their solution focussed on monitoring QoS to inform online re-composition using Genetic Algorithms (GAs).

In another domain Zhou et al. [64] apply service-orientation for controlling swarm robotics [145] where there is both a software and hardware services layer. The software layer dictates the function that the swarm must perform and the hardware layer, comprising of multiple individual robots, must coordinate to achieve that goal. In this context the QoS for inter-robot communication as well as between the swarm and the users are vital.

The majority of research focusses on prediction of Web Service QoS with various authors considering techniques for updating the QoS of Web Service definitions and using those predictions to inform online re-composition. The most prominent analysis with regards to Web Service QoS was performed Zheng et al. [6; 33; 123; 146–148] which analysed over 16 thousand web services with respect to their definitions and response-times. Their work demonstrated specifically the variable nature of response-times due to the changeable nature of the Internet [6]. Further they introduce the use of Pearson's Correlation Coefficient (PCC) and matrix factorisation to predict QoS for different services and and users [147] which has since been adopted by several other approaches.

Beyond Web Services, many approaches including the work by Tsai et al [129], Schneider [149], the iLand project [140], and Perez & Gutierrez [150] make use of the OMG Data-Distribution Service (DDS) middleware [5]. DDS is designed to support publishing and subscribing of data by services. Unlike traditional SOA it does not natively support workflows but instead workflows can be inferred by controlling which services subscribe to which data types and the relevant update frequencies . Further although DDS has an extensive QoS definition, comprising of twenty-one parameters, it does not enforce compliance and subscription to data is based on periodically receiving the most recently published value.

2.3.3 Real-Time Service QoS

In order for a SOA to guarantee Real-Time behaviour the QoS definition must accurately represent the executing service's performance with respect to the execution environment over the duration of execution. There are numerous approaches to estimating, monitoring, and adapting Real-Time Quality of Service (RT-QoS) in various environments for various purposes. Appendix A provides a comprehensive list of approaches with a short summary on each of them. Further QoS methods relating specifically to the domain of Cloud computing and not applicable to RT-SOAs are reviewed by Abdelmaboud et al. [151]. These approaches each have limitations and are constrained in their capability to reliably guarantee response-times for real-time services, particularly in a context where there is not absolute control over every part of the system.

QoS Parameters

The various approaches to QoS take different factors into account as parameters in their models. The most extensive set of parameters by a single model is the DDS QoS model which consists of 21 parameters incorporating [150; 5]:

Data Availability: history, durability, lifespan, and lifecycle

Maximum Resources: resource and time limits

Data Delivery: reliability, ownership, presentation, ordering, and partitioning

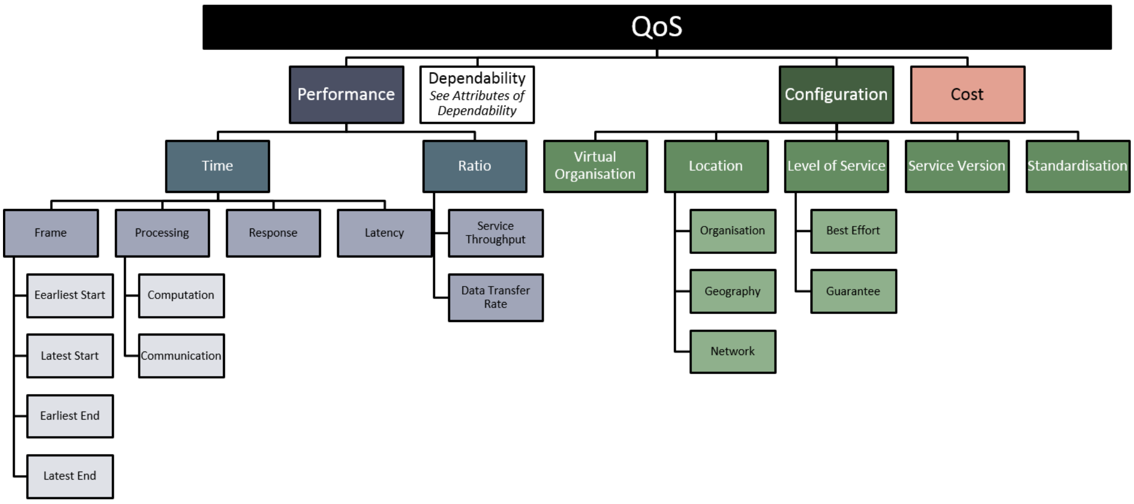


Figure 2.17: Taxonomy of QoS parameters, adapted from Truong et al. [152]

User Configuration: user, topic, and group data tags

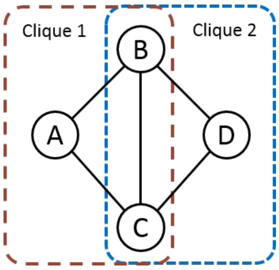
Data Timeliness: deadline, latency budget, and transport priority

System run-time configuration: entity factory and liveliness

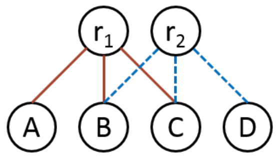
Alternatively the work by Estévez-Ayres et al. [70] could be considered with the parameters of: WCET, period or request frequency, deadline, an offset, and service priority. They also include the physical resource requirements and data requirements [142]. The work by Wang et al. [153] specifies QoS attributes with respect to response-time as well as price (or computational cost), reputation, availability and appropriate weight factors for each of them. Truong et al. [152] provide a taxonomy of QoS metrics which is shown in Figure 2.17. Other than the attributes of dependability that are discussed on Page 33 the taxonomy comprises of three categories: performance, configuration, and cost. The former considers execution times and latencies. The configuration category defines aspects of the SLA such as where service will be geographically distributed and what level of guarantees are made about the service provision. The variance and limitations of parametrisation techniques across the methodologies will be discussed in detail in Chapter 3.

Impacting Factors

Many of the QoS approaches for RT-SOAs assume fully controlled systems where the overall schedule of processes across the infrastructure can be controlled. However in practice the infrastructure may be shared with other non-real-time services and the underlying operating systems



(a) Interference graph



(b) Resource Contention graph

Figure 2.18: Sample Interference and Resource Contention graphs adapted from [154].

may not be real-time operating systems. As a result resource contention and interference by other co-located processes becomes a significant issue that can reduce a service's ability to adhere to deadlines.

One approach to understanding resource contention and interference is provided by Huang and Peng [154] where an interference graph identifies the processes which can interfere with each other and a resource contention graph uses the cliques from that graph to identify the contention regions. The example shown in Figure 2.18 shows the forming of two cliques in the interference graph ($\{A, B, C\}$ and $\{AB, C, D\}$) where all the nodes of the graph (representing processes) can interfere with each other. The cliques can then be identified as competing over either resource r_1 or r_2 in the contention graph. A resource in this case would typically be an area of memory, CPU registers, or a specific device. In order to guarantee timely completion of processes, each process associated with a contention region must be considered as part of a set of competing processes.

In practice, in a distributed system it is likely that the majority, if not all, of the processes executing on a given server will interfere with each other. This graph based approach only captures references to specific resources and not aspects such as CPU time or amount of physical memory allocated. In this way resource interference and contention in general computing must refer to the adequacy of the resources provided to individual processes. Under-provisioning of resources leads to performance degradation and slower response-times. Conversely over-provisioning leads to wastage of time and resources that could be used by other processes [155; 156]. One example of the impact of under-provisioning of resources is the *"long-tail"* problem where specific processes may exhibit abnormally long response-times [157].

Resource Patterns

Outside of traditional Real-Time Systems and the many RT-SOA systems which can be fully controlled with respect to the running processes, the system workload is inherently non-deterministic. Analysis in the context of Cloud computing workloads provides an insight into modelling and predicting what and how many processes may interfere with any given process. Fehling et al. [158] document a wide range of Cloud computing patterns to facilitate the design of Cloud systems[a] and specifically identify five workload patterns:

Static workloads where the resource utilisation over time is constant. This can be extended to consider the workload as static within a variance and can therefore be guaranteed to not exceed a given threshold.

Periodic where the resource utilisation peaks at reoccurring time intervals.

Once-in-a-lifetime workload refers to general workload that is predictable disturbed by a peak utilisation which only occurs once. This is a particular case of the periodic workload pattern where the timeframe is particularly long.

Unpredictable refers to a random utilisation and can be considered as a generalisation of periodic workloads.

Random (Continuously Changing) workload is where the utilisation is either continuously growing or else continuously shrinking.

In that context the work by Moreno et al. [73] provides a detailed analysis of the behaviour of individual processes executing in the Cloud with respect to their: execution times, CPU and memory utilisation, and their computation length. It is however necessary to note that workloads patterns for Cloud computing may not be pervasive in other domains such as High Performance Computing, private Clouds, or other time critical systems [155]. Further any workload pattern should also be considered in the context of the resource lifecycles within each process (see Figure 2.10).

[a]http://www.cloudpatterns.org

QoS Learning Techniques

The final aspect with respect to RT-SOA and specifically QoS are the methods for learning and adapting the QoS values over time. A detailed analysis of specific methodologies will be presented in the next chapter however there are popular techniques that form the basis for the majority of the existing metholodiges:

Static techniques which do not learn and do not change over time and are therefore insensitive to variations in the execution environment.

Periodic Updating such as the work by Bosman et al. [159] where a probe is used to periodically check the response-time of the services.

Continuous Historical updates where all, or a sample of, previous response-times for service execution instance are stored and aggregated to provide an anticipated response-time. For example the work by Zou et al. [81] adopt the observed worst-case as the new QoS value for each new execution instance.

Genetic Algorithms such as the work by Canfora et al. [160] where a GA is used to generate a genome representing the QoS of services and composed workflows. For example these approaches will look at different service implementations of the same service definition and seek to optimise the selection of services based on a range of criteria specified as a fitness function.

2.4 Challenges in Real-Time Service-Orientation

This chapter has introduced the Service-Oriented Architectures (SOAs) and many of its key features (Section 2.1). The concept of dependability (Section 2.2) has been discussed in detail followed by an overview of Real-Time Systems theory. This chapter has discussed many of the concepts that Real-Time Service-Oriented Architectures (RT-SOAs) must address, specifically through the use of appropriate Quality of Service (QoS) mechanisms. However in order to guarantee the correct behaviour of a RT-SOA three areas must be considered:

1. **Execution management** to ensure that service adhere to their advertised QoS. This can be achieved either through the use of a Real-Time Operating System or else my managing the services at the workflow level.

2. **RT-QoS monitoring and prediction** to alert the system when a service is likely to violate timing constraints.

3. **Workflow management** to ensure that overall QoS of the workflow is guaranteed and that the system remains capable of delivering it's functionality.

Workflow Management

Workflow management in a RT-SOA specifically refers to the challenge of monitoring execution progress, predicting potential failures, and consequently performing a re-composition of the workflow. Several approaches endeavour to do this and address the timeliness of the re-composition process itself. The limitation with most of these RT-SOA approaches is the assumption that they are isolated and not sharing resources with other systems and can therefore be fully controlled.

A further and more interesting challenge is the management and prediction of workflow behaviour where the workflows are comprised of complex patterns as discussed in Section 2.1.3 on Page 27. These challenges start to move away from the fully controlled environments that are typically assumed by introducing levels of non-determinism into the system. However in order to address these issues significant work must be done on the modelling of QoS in uncontrolled environments which is this focus of this research.

QoS Monitoring and Prediction

As outlined in this chapter Quality of Service (QoS) can be described at both the workflow, service, and the Micro-Service (μS) level. This research is concerned specifically with exploring RT-QoS at the μS level as this forms a foundation upon which to build further RT-QoS models.

As with workflow management a key limitation of the majority of the research on QoS in RT-SOAs has assumed full knowledge of the system and full control of its environment. However, in practice this is not the case, particularly with Cloud computing and Internet of Things (IoT). In these instances services may have temporal constraints, either self-imposed or imposed by other services within a workflow, and running with interference and resource contention are commonplace.

A further limitation on the majority of existing techniques is the lack of application of schedulability theory. Where very few exceptions the approaches do not consider the schedulability conditions of their work and further their approaches do not consider the resource implications to service performance.

The following chapter will consider in detail the existing methodologies for managing and predicting QoS as part of a RT-SOA. It will present an analysis of the limitations in existing work from both a theoretical and experimental perspective and provide the benchmark against which this research can be compared.

Chapter 3

Classification of Real-Time SOA QoS Techniques

This chapter reviews in detail the existing methods for Quality of Service (QoS) in Real-Time Service-Oriented Architectures (RT-SOAs). First seven categories of QoS approaches are identified and eighty existing techniques are then reviewed. Subsequently a theoretical analysis of selected approaches is discussed in Section 3.2 before the approaches are analysed using simulation to provide the benchmark against which this research can be compared (Section 3.3). In Section Section 3.4 a discussion of the limitations identified by the theoretical and simulation analyses is presented before the case for this research is outlined.

3.1 Review of RT-QoS Methods

Each of the 80 reviewed QoS approaches is summarised in this section. Each approach is considered with respect to:

Resource Awareness where the majority of approaches do not consider aspects such as CPU, memory, or network implications on QoS. And many of those that do, presume to have full control over resource allocation and system scheduling.

Real-Time focus as claimed by the authors and whether the approach *actually* considers real-time services and considers techniques such as schedulability testing.

Service or Workflow level QoS where the majority of approaches focus on the selection of services for composing workflows in order to guarantee an overall workflow QoS.

3.1.1 Approach Categories

The majority (84%) of the reviewed approaches fall into the following seven categories shown in Figure 3.1: correlation, optimisation, containment, middleware, fuzzy-logic, cost, and tolerance.

Correlation

Approaches within the first category are based on identifying the similarity between services and users. In this context correlation is utilised to build a model representation of the expected performance of the specified service, typically using Pearson's Correlation Coefficient (PCC) to achieve this with a *user-service* matrix. QoS predictions are made by identifying the most similar users. These approaches are some of the most generically applicable supporting any infrastructure and treating services as black boxes. Table A.1 outlines eight approaches based on correlation, which are mostly variants of the approach by Zibin Zheng et al. [146] which will be considered in depth in Section 3.2.1.

As can be seen in Table A.1 most of these approaches are not resource-aware or provide any method for calculating real-time guarantees. Some of them, such as Sandhu and Sood [161] provide a probabilistic model to provide a likelihood of response-time and likely level of resources required. As a result these approaches are not suited to RT-SOAs.

Optimisation

The second group of approaches use optimisation to parametrise a QoS definition to maximise the likelihood of the service meeting certain conditions. Many of these use Genetic Algorithms (GAs) but also techniques such as linear programming. They mostly focus on optimising service selection for guaranteeing overall workflow QoS but are sometimes used for the calculation of individual service QoS parameter values [160], however they are not used for real-time systems due to the computation time required to run the algorithms. Table A.2 summarises optimisation based QoS approaches.

Similar to the correlation based approaches the majority of these techniques are not resource-aware and except those applied to robotics [64] do not provide real-time support. Further these techniques are almost exclusively applied at the workflow level to the problem of service selection. Although there may be a use for these approaches at the workflow level in RT-SOAs they are not suited to providing RT-QoS definitions that are adaptive. Theoretically if there was sufficient data collected around a service's execution performance with respective to resources and other factors optimisation and machine learning approaches may provide effective RT-QoS definitions. That however introduces an elongated data collection phase which may not be feasible in the majority of situations and deployments.

Containment and Virtualisation

Container based methods are not all strictly part of SOA approaches but assume that all services or processes execute within a containerised environment such as a virtual machine. These approaches then use the resource control the containers provide to control and predict the execution performance. These approaches are limited to those domains, such as Cloud computing, where containerisation can be adopted. For prediction alone, within a Cloud environment, these approaches require minimal interference or understanding of the hosting system [162; 163]. The approaches listed in Table A.3 all utilise containers or virtualisation in some form.

These techniques differ from those already considered in that they can be used to calculate real-time guarantees using the resource utilisation information. However they generally require full system control, including process and virtual machine scheduling and many require an underlying Real-Time Operating System (RTOS). Given those constraints the approaches use traditional real-time scheduling techniques to provide appropriate guarantees. Without those constraints the approaches provide best-effort or soft guarantees.

Middleware

Many approaches to managing QoS look below the services themselves to the underlying communication middleware. In many ways these aim to achieve the same as the previously discussed container-based approaches however require a greater level of control of the underlying computational infrastructure. DDS is one technology that is adopted by numerous approaches as the underlying network infrastructure upon which to build a RT-SOA. This claims to be a real-time infrastructure using a Publish/Subscribe approach with several parameters but does not actually provide any real-time guarantees relating to the timeliness of data publication by any given service [144]. Table A.4 provides a list of middleware-based approaches which are largely based on the work by Tsai et al. [129] and most comprehensively developed by Estévez-Ayres et al. [144].

Similar to the containment and virtualisation approaches these techniques are based on traditional real-time scheduling and require full system control. These approaches however go further and manage the entire infrastructure including computation and communication.

Fuzzy-Logic

The fifth category is the use of qualitative terms, such as *good* or *bad* rather than distinct quantities, to describe QoS. Two of the methods in this group are shown in Table A.5 with the first being explored in further detail in Section 3.2.6. The primary challenge with these is the ability to map back to real-time systems scheduling which would have use some probabilistic method. Also they are primarily used at the workflow level rather than for individual defining at a fine-grained level the QoS of individual services.

Cost

The pricing of service execution is one of the most common parameters of QoS in addition to response-time. Considering *cost* facilitates a trade-off against performance and can consider aspects such as: power or energy; usage of resources such as memory, CPU and storage; as well as the infrastructure provider's pricing. Most of the approaches listed in Table A.6 build on the work by Kaur et al. [164].

Although cost is a significant consideration it does not provide any measure of whether a service will be able to meet a given deadline. It could potentially be used to provide cost categories such that services are charged based on deadline guarantees.

Tolerance and Probability

Within many of the previous approaches the likelihood of a given response-time is often considered. However there are several approaches that only measure the probability of a given response-time. Table A.7 provides a summary of many of these approaches. The primary comment on these is the lack of real-time support and the lack of methods for mapping probabilistic response-time models to real-time guarantees. Additionally it is noted, with one exception, that none of these approaches consider the underlying resources such as CPU or memory.

Probabilistic measures can also be used to tolerate or avoid QoS violations. One such approach by Russell et al. [131] (listed in Table A.8) uses redundancy to improve overall reliability. The formalism from their approach is shown in Eqn. 2.1. The recorded QoS is compared against the required QoS and is used to determine how many copies or versions of the service are required in order to increase the likelihood of providing that level of service. This is not however satisfactory for RT-SOA.

Within this category are also those approaches that actively monitor the response-times of services. Table A.9 outlines those approaches that periodically monitor service performance.

A further 13 approaches are listed in Table A.10 including hardware solutions [165] and several approaches targeted specifically at workflow configuration.

3.1.2 Significant Contributions

From the range of methods and approaches outlined in the previous section there are specific key contributions within each category. These specific approaches, listed below and shown in Figure 3.1, form the most comprehensive approaches within each category when considering the scale and detail of their work as well as their adoption by other similar techniques. Each of the listed approaches will then be considered in more detail in Section 3.2.

Zibin Zheng et al. [146] provides the most comprehensive analysis of Web Service QoS analysing several thousand unique services with over thirty million cumulative invocations. They introduce the use of Pearson's Correlation Coefficient (PCC) to correlate *users* with *services* and utilise collaborative filtering to predict missing values for new *User/Service* combinations.

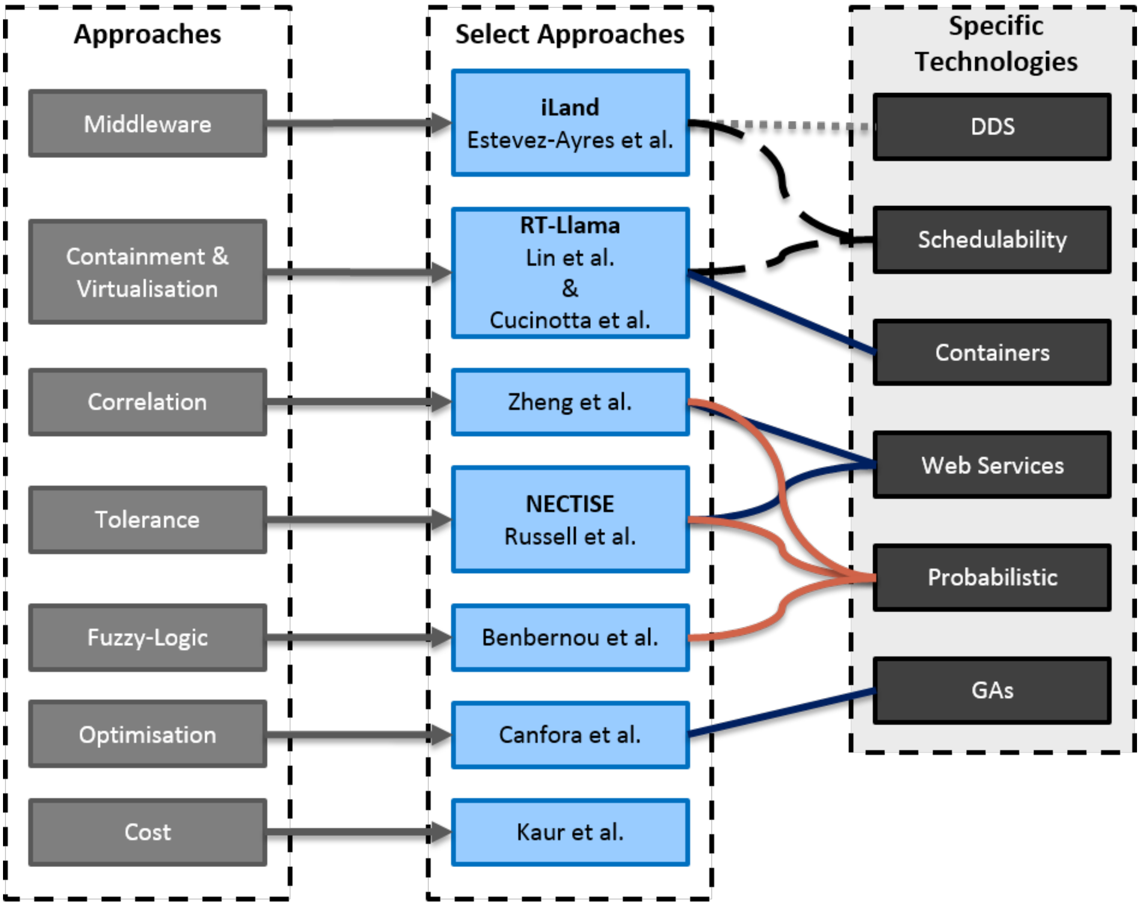


Figure 3.1: Core approaches to RT-SOA and the most significant works

Estévez-Ayres et al. [144] uses the Data-Distribution Service (DDS) as the infrastructure for a RT-SOA implementing the concepts of Tsai et al. [129]. Their approach does consider resource adequacy and uses traditional real-time schedulability to verify that services will meet deadlines.

Russell et al. [131] presents the NECTISE project as one of the first real-world uses of SOA adopting the use of service redundancy to improve the likelihood of adhering to the deadlines.

Lin et al. [162] provides a comprehensive approach with RT-Llama to use virtual machines to contain and manage service execution. Their approach also introduces the use of intermediate deadlines during service execution to provide monitoring points. This approach is similar to that adopted by Cucinotta et al. [63] which uses traditional schedulability analysis to verify the service execution performance.

Canfora et al. [160] presents an approach that uses constrained Genetic Algorithms (GAs) to perform workflow composition where the *fitness function* considers the dependability attributes of availability and reliability as well as response-time and cost.

Benbernou et al. [111] presents a real-time approach for service-orientation using fuzzy-logic to describe the response-times and memory utilisation for the purposes of workflow composition.

The next section considers each of the above QoS techniques in detail presenting a summary of the algorithm adopted providing the basis for the remainder of this chapter.

3.2 Key QoS Techniques

This section explores each of the previously identified key approaches in more detail. Firstly exploring the overall approach followed by any mathematical formalisms and algorithmic considerations.

3.2.1 Technique: User-Service Correlation (*Zibin Zheng et al. [146]*)

An overview of the approach presented by Zibin Zheng et al. [146] is depicted in Figure 3.2 and is comprised of three major components:

1. Identifying similar users and services using Pearson's Correlation Coefficient (PCC) which can be computed in linear time with respect to the number of users or services respectively. The similarity is then updated to apply a *significance weighting* representing the density of invocations. The *significance weighting* acts to reduce the influence of small numbers of invocations and outliers by applying greater weight to those with more instances.

2. The QoS values can then be predicted by selecting the *Top-K* similar neighbours, either users or services. By using collaborative filtering missing or *null* values from the model can be predicted.

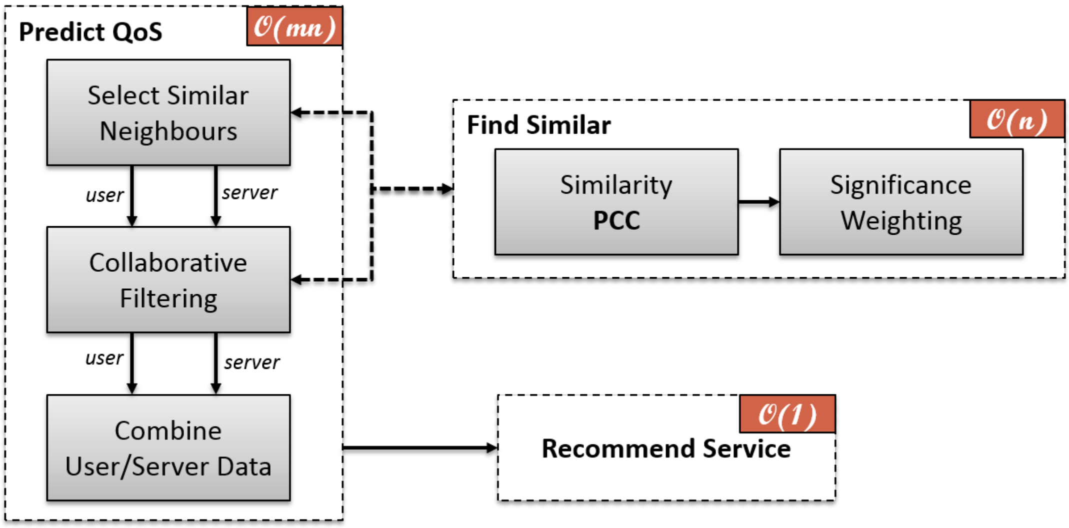


Figure 3.2: Overview of approach proposed by Zibin Zheng et al. [146] for predicting QoS using similar neighbours.

3. Finally the predictions can used to recommend the top *k* services to users and vice versa recommend the top *k* users to service providers as potential customers.

The proposed approach as outlined in Table A.10 does not consider the impact of resources on execution performance and it is therefore not possible to perform traditional real-time schedulability analysis. It is noted however that the approach does facilitate adaptive QoS performing re-computation in $\mathcal{O}(m^2n + n^2m)$ where m is the number of services and n the number of users. In Section 3.3 the practical implications and limitations of this approach will be considered.

3.2.2 Technique: iLand with DDS (*Estévez-Ayres et al. [144]*)

The work by Estévez-Ayres et al. [144] on the iLand project is the most extensive in the domain of RT-SOAs and makes use of the Data-Distribution Service (DDS) middleware to facilitate the real-time network that is required to provide real-time guarantees in a distributed infrastructure [140]. Critically, as discussed on page 18 DDS does not provide guarantees regarding individual service performance. Therefore any DDS-based approach must consider the service executions themselves.

In that context the authors do consider the schedulability of individual services, taking into

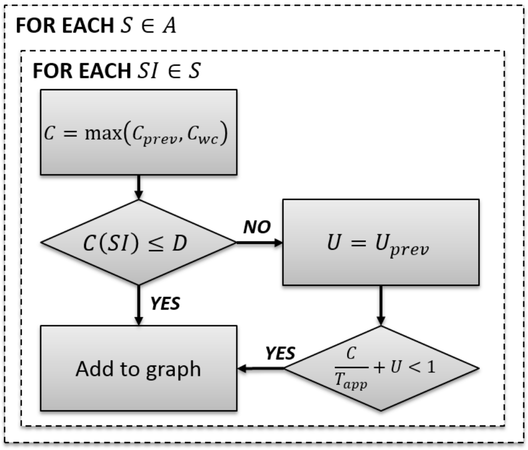


Figure 3.3: Overview of approach proposed by Estévez-Ayres et al. [144] focussed primarily on *real-time* service composition.

account CPU utilisation, application periods, and observed worst-case execution time [70]:

$$\begin{aligned} U =&\ U_{serv} + \frac{C}{T_{app}} < 1 \\ U =&U_{serv} + U_{worst} \end{aligned} \tag{3.1}$$

It can also be noted that in the context of the iLand project where the entire system workload is controlled and known it is possible to perform traditional schedulability analysis using both the response-time equation (Eqn. 2.5) and utilisation analysis (Eqn. 2.2). The proposed approach can update the utilisation and WCET properties in constant time and is then subject to the schedulability test that is used, which may be linear $\mathcal{O}(m)$ in the case of utilisation analysis. As shown in Figure 3.3 the WCET is calculated based on the previous response-time C_{prev} and previously observed WCET C_{wc}. This is then used to calculate the schedulability by looking at the current server utilisation U_{serv}, before the μS is deployed, as shown in Equation Eqn. 3.1. The overall QoS is then a function of the WCET and a slowdown factor of $\frac{U_{wc}}{1-U_{serv}}$. In this case if the server utilisation is 90%, the maximum historical utilisation of the service is 20%, and the historical WCET is $5s$ the slowdown factor will double resulting in an estimated WCET of $10s$.

The remainder of their work focusses on workflow composition to provide overall guarantees of schedulability. A detailed analysis of the feasibility of their approach for more general RT-SOA QoS modelling is presented later in this chapter.

3.2.3 Technique: NECTISE with Probabilistic Redundancy (*Russell et al. [131]*)

The probabilistic techniques vary primarily with respect to how the historical data is collected. In many instances probes are used to periodically test the response-time of the services. In these cases the QoS model is defined using a sliding window of data, taking account of the most recent x observations. Alternatively approaches record each and every response-time to build a population, rather than sample, based model.

The Network Enabled Capability Through Innovative Systems Engineering (NECTISE) approach takes a population-based probabilistic model of response-times. Each response-time is measured against the QoS that was expected of it producing a likelihood of the service not adhering to it's advertised QoS. The approach then uses this to improve the percieved reliability of the service through the use of redundancy. As shown in Section Eqn. 2.1 on Page 21 the likelihood of QoS violation is used to inform the number of required replicas and can rearranged for solving for single service implementation (i.e. ignoring workflows) as follows:

$$\begin{gathered} 1 - p^r = x \\ r = \frac{\log(1-x)}{\log(p)} \end{gathered} \tag{3.2}$$

3.2.4 Technique: RT-Llama (*Lin et al. [162]*)

This approach is similar to the iLand project described earlier and does present a real-time system and uses traditional analysis to demonstrate the schedulability of services. The authors consider a service which is comprised of smaller functional elements, which can be considered as Micro-Services (μSs) where the Worst-Case Execution Time (WCET) is known a priori.

A service request can be either: immediate, reserved, or best-effort where the latter results in best-effort execution on an unmanaged infrastructure. Immediate and reserved requests are executed on a real-time CPU as shown in Figure 3.4. The reserved requests are executed on a reserved portion of the RT-CPU in virtual containers. For immediate processes the system endeavours to find a *Feasible Sub-Process* (FSP) set which will meet the required deadline and if found schedules it using Earliest Deadline First (EDF) scheduling. The basis for immediate tasks however is the remaining CPU space that is not reserved on the server for the execution window defined by

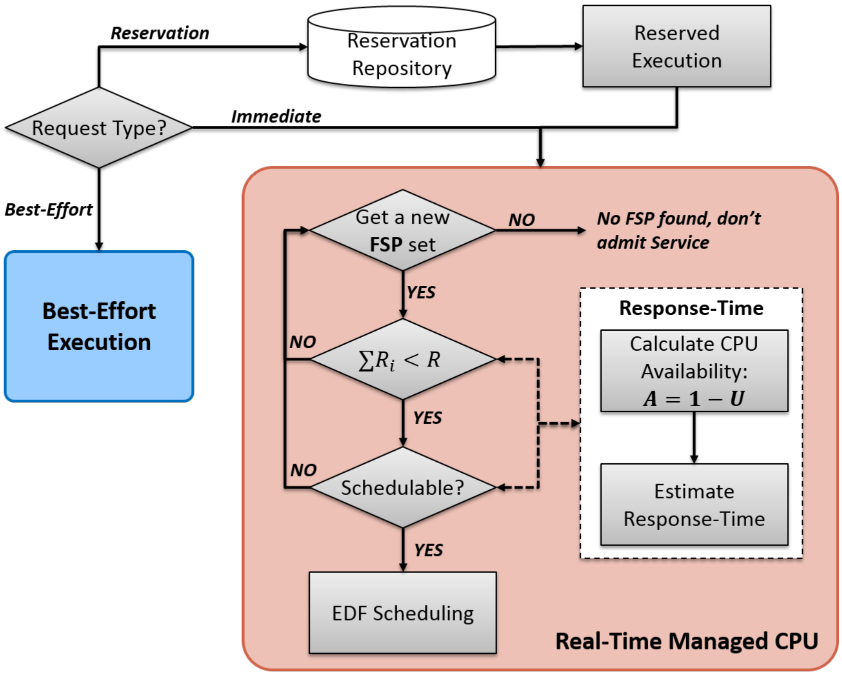


Figure 3.4: Overview of the RT-Llama approach by Lin et al. [162] which treats services as either best-effort, immediate, or reserved.

the request-time and relative deadline:

$$1 - U_{[r,d]} \tag{3.3}$$

This approach requires known WCETs and resource utilisation patterns for each of the μSs and requires a RT-OS as the underlying infrastructure of a RT-SOA. In terms of time complexity, if a service is schedulable it will be found within $\mathcal{O}(m^s)$.

3.2.5 Technique: Optimisation (*Canfora et al. [160]*)

Many approaches focus on optimising the selection of services for workflows so as to improve the overall response-times. The approach by Canfora et al. [160] however seeks to also optimise the specification of QoS parameters to more accurately reflect the services themselves.

As depicted in Figure 3.5, the proposed approach uses a Genetic Algorithm (GA) to for optimisation. A population of 100 individuals is chosen where the genome represents either the QoS parameters of an individual service or the selection of services for a workflow. Each individual is evaluated according to a fitness function that considers the the QoS parameters of: cost, response-time, availability, and reliability. The approach iterates, up to a maximum generation (100 in the authors experiments), until the constraints are met at which point the algorithm may iterate a max-

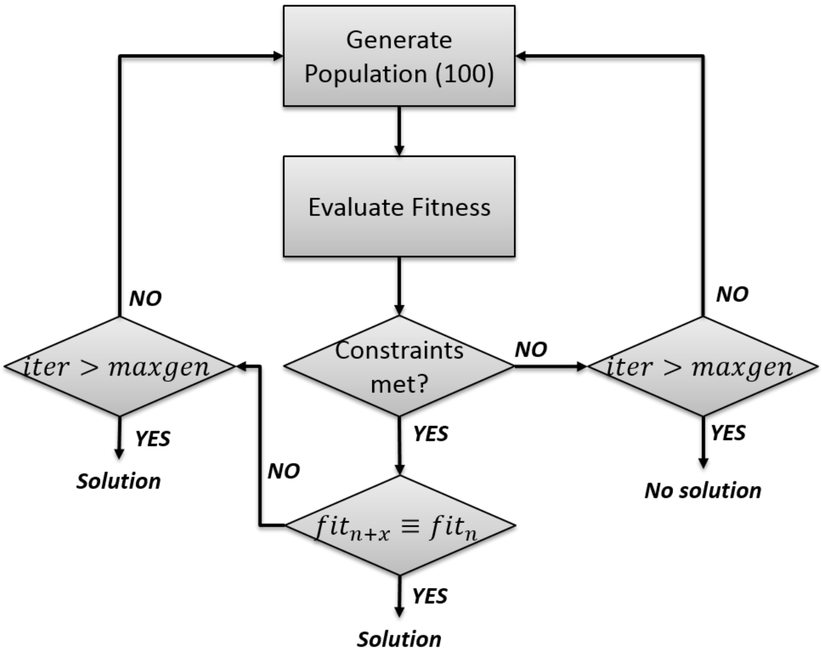


Figure 3.5: Overview of approach by Canfora et al. [160] using GAs to optimise QoS parameters.

imum of x further times before producing the best found solution. Each new generation keeps the best two individuals and mutates or combines/crosses-over the remaining population using roulette-wheel selection.

Although this approach potentially calculates the QoS or composition in $\mathcal{O}(gp)$, where each generation requires space $\mathcal{O}(p)$, it is not generally appropriate for real-time systems as there is no guarantee that a solution will be found.

3.2.6 Technique: Fuzzy-Logic (*Benbernou et al. [111]*)

The final technique for describing and adapting QoS considered in this thesis is the use of fuzzy logic and probabilistic modelling. In this fashion *fuzzy* terms such *good* or *bad* are applied to ranges of values based on pre-specified thresholds.

One such approach is presented by Benbernou et al. [111], shown in Figure 3.6, where they model response-time as: Good, Medium, or Bad and memory consumption as either Good or Bad. The authors monitor the executing service and encode the behaviour using the fuzzy logic rules. The previous execution behaviour is then used to calculate the probability of a particular behaviour

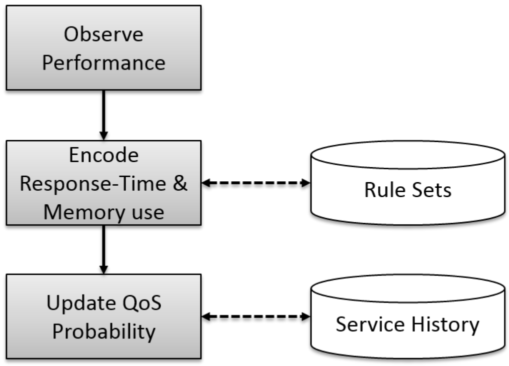


Figure 3.6: Overview of fuzzy-logic approach by Benbernou et al. [111].

reoccurring:

$$P(SymbQoS|S_i) = \frac{P(SymbQoS \cap S_i)}{P(S_i)} = \frac{|SymbQoS \cap S_i|}{|S_i|} \tag{3.4}$$

Where the probability of observing a given performance, denoted by a QoS symbol, is the number of occurrences of that symbol for a given service as a proportion of the total number of execution instances of that service. This can then be used for future iterations in selecting services with acceptable QoS.

The following section details the practical evaluation of the presented techniques using simulation of various workloads and service types.

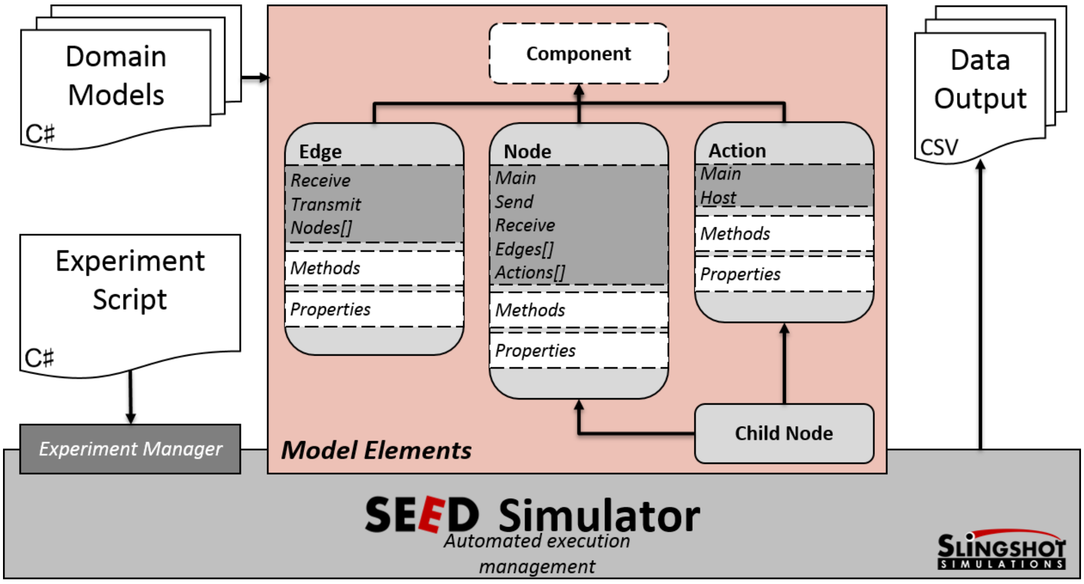
3.3 Simulation of Approaches

This chapter has so far presented an overview of several of key approaches to modelling QoS and managing real-time services. This section presents the simulation and analysis of those approaches. Specifically the simulation considers the aspects, presented in Chapter 2, of workloads and service execution types with respect to their resource utilisation behaviour. In Section Section 3.3.1 the simulation design will be presented followed by the results in Section Section 3.3.2.

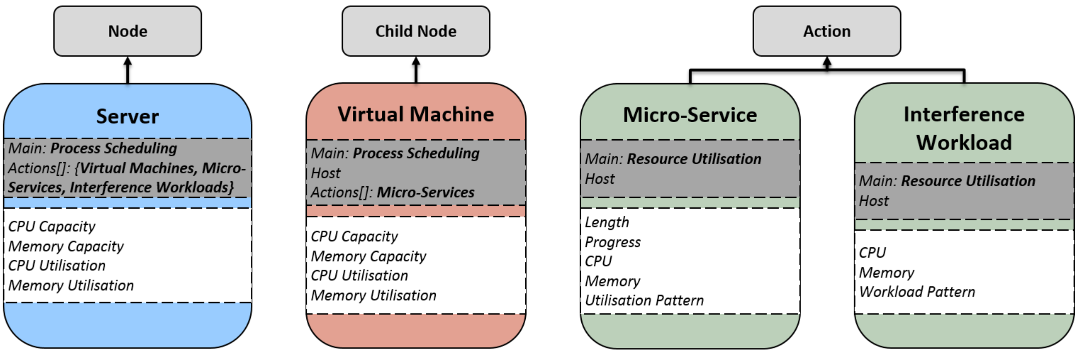
3.3.1 Simulation Design

There are numerous simulation tools available specifically for the purposes of modelling computing behaviour. This research adopted the SEED simulator[a] which has been used to accurately

[a]Provided by Slingshot Simulations Ltd., `www.slingshotsimulations.co.uk`



(a) High level architecture of the API provided by the SEED simulator.



(b) Core components for simulating interference experience by Micro-Services and Virtual Machines.

Figure 3.7: The SEED simulator

and efficiently model task and job behaviour in large data-centres [166–170]. Specifically Garraghan et al. [166] modelled a Google data-center with 2000 servers and verified the results against Moreno et al. [156]. Subsequently Ouyang et al. [168] used the simulation tool and developed models of servers and virtual machines to explore longtail behaviour in data-centers. For this research the verified models from those works were used for the core components of the simulation as detailed below. Unlike other simulation technologies, such as CloudSIM [171], SEED separates the simulation execution management from the domain modelling. The simulator provides an API for implementing: experiments and domain components where the components can be integrated in the form of a graph. In practice this means that simulation components such as servers are modelled as nodes, network connections are represented as edges, and then iteratively nested graphs provide layers of virtualisation.

As shown in Figure 3.7a individual components are specified as either: graph nodes; edges; actions which operate within a node; or child nodes which facilitate the nesting of sub-graphs. Each component is assigned a set of methods and properties such as those outlined in Figure 3.7b and described below. The objective of the simulations in this research are to explore the relationships between Micro-Services (μSs) and their hosts in the context of evaluating the QoS approaches described in the previous section.

Simulation Core Components

The core elements of the simulation include: servers, virtual machines, and Micro-Services (μSs) which are listed detailed below and depicted in Figure 3.7b. As this research is not considering workflows it not necessary to model them or services. Instead a workload model can be used to represent the potential interference to any given μS of interest.

Servers are modelled as *Nodes* whose primary function is the scheduling of processes which is performed in a Round Robin fashion. Each server hosts one or more processes, which may either be μSs or virtual machines, and is comprised of resources including CPU and memory. The Cloud server models from [166–168] which were verified against those in CloudSIM by Calheiros et al. [171] and Moreno et al. [156] are adopted for this research. The simulation code listing can be found in Listing D.1 in C.

Virtual Machines operate in a similar fashion to servers with the exception that they are hosted by a server and utilise the *Child Node* API. The virtual machine model used for this research are those from Albatli et al. [169] which are representative of industrial virtualisation infrastructure. The simulation code listing can be found in Listing C.2 in C.

Micro-Services are represented as *Actions* which are hosted by either servers or virtual machines. Their primary function is the modelling of CPU and memory utilisation over time based on a specified resource pattern (Section 2.1.3). The simulation code listing can be found in Listing C.4 in C.

Interference Workload is also an *Action* hosted directly by a server that represents the interference experienced by the virtual machine or μS on the specified server. The simulation code listing can be found in Listing C.3 in C. The workload is specified as following a particular Cloud workload pattern as detailed in the previous chapter along with a mean utilisation.

The algorithms of the QoS approaches described earlier in this chapter are then encoded within the experiment scripts which are used to monitor the simulation. The next section outlines the design of experiments.

Design of Experiments

In Chapter 2, processes were described based on their: duration, length, disk, memory, and CPU usage and typical Cloud tasks were categorised into three types: short (0.7 million instructions (MI)), medium (16.6MI), and long (124MI) [156]. As with the server and virtual machine models the choice of these Cloud task models is due to their separate verification by Moreland et al. [72] and [166]. The third longer task type is not used in this research as it is not representative of μSs. Further each task is defined with an internal resource lifecycle relating to the acquisition and release of resources, specifically memory [93]. The combination of these patterns provides eight task configurations shown in Table 3.1 and Figure 3.8.

In order to model interference, the Cloud workload patterns discussed on Page 52 are used, and specifically three are adopted for the purposes of this research: static, periodic, and unpredictable. Combined this provides seventy-two configurations for each QoS approach. Figure 3.8 shows a summary of the experiment configurations where each of the interfering workloads is run at either a high, medium, or low level set at 95%, 90%, and 80% respectively.

The simulation results, shown in the next section, are compared by measuring the:

Prediction Accuracy using Mean Absolute Error (MAE) and Mean Percentage Error (MPE) as used by Zhu et al. [123] and Mean Percentage Waste (MPW). The MAE and MPE compare the measured R_i

Task	Cloud Type	Resource Acquisition	Resource Release
LA1	**T1**: CPU 0.6%, Memory 0.2%, Length 0.7MI	Lazy	Active
LA2	**T2**: CPU 2.9%, Memory 1.1%, Length 16.6MI	Lazy	Active
EA1	T1	Eager	Active
EA2	T2	Eager	Active
LN1	T1	Lazy	Non-Expiring
LN2	T2	Lazy	Non-Expiring
EN1	T1	Eager	Non-Expiring
EN2	T2	Eager	Non-Expiring

Table 3.1: Micro-Service (μS) process types based on observed Cloud tasks and resource lifecycle models.

against the predicted QoS value $\hat{R}_i$ whilst the MPW takes the average amount of time by which the prediction was greater than the measured R_i. In the following equations the calculation is constrained using the square Iverson Brackets for summation conditions [172]:

$$\begin{aligned} MAE &= \frac{\sum_i^N |\hat{R}_i - R_i|}{N} \\ MPE &= \frac{MAE}{\sum_i^N \hat{R}_i / N} \\ MPW &= \frac{\sum_i^N (\hat{R}_i - R_i)[\hat{R}_i > R_i]}{\sum_i^N \hat{R}_i [\hat{R}_i > R_i]} \end{aligned} \tag{3.5}$$

QoS Violation measuring the Absolute Violation Count (AVC) as well as the Mean Absolute Violation (MAV) and Mean Percentage Violation (MPV), constrained using Iverson Brackets:

$$\begin{aligned} AVC &= \sum_i [\hat{R}_i < R_i] \\ MAV &= \frac{\sum_i (\hat{R}_i - R_i)[\hat{R}_i < R_i]}{AVC} \\ MPV &= \frac{MAV}{\sum_i \hat{R}_i / N} \end{aligned} \tag{3.6}$$

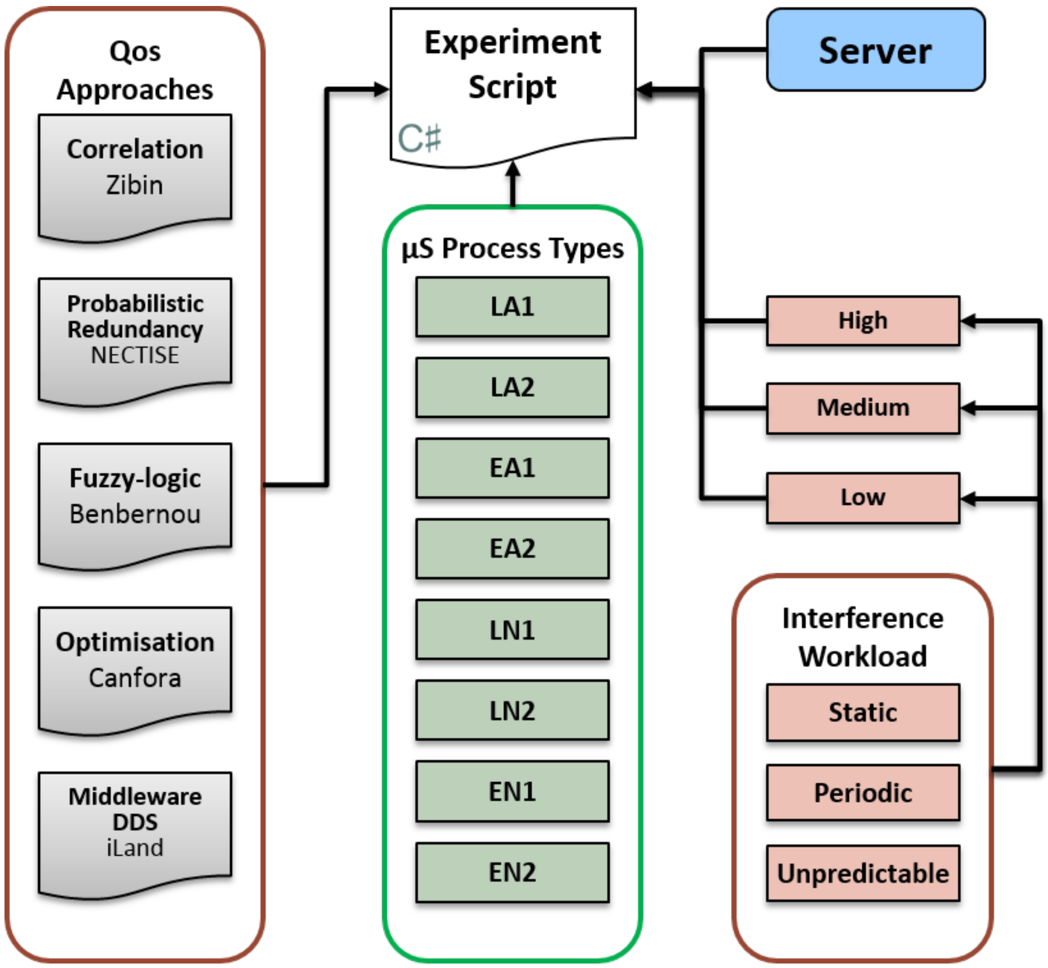


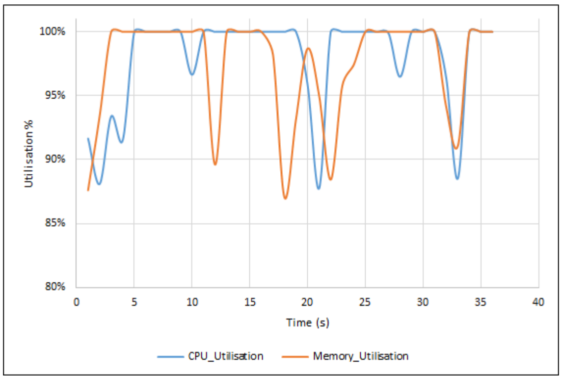
Figure 3.8: Overview of QoS experiment showing 6 core existing approaches, 12 μS types, and workload interference patterns.

3.3.2 Preliminary Simulation Analysis and Results

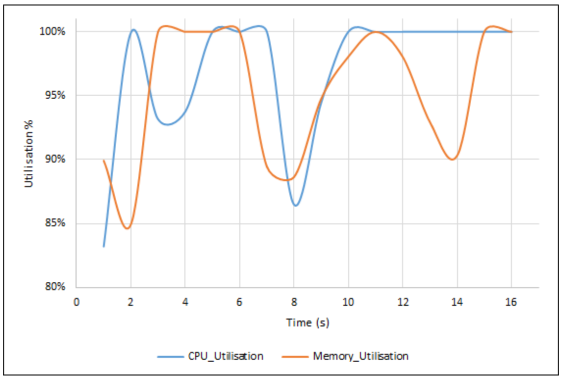
This section depicts the preliminary results of simulating each of the approaches detailed in the previous section in the context of the specified Micro-Services and interference workloads.

Figure 3.9 depicts 2 samples of server CPU and memory utilisation which are subject to a periodic interference workload. In this case the average interfering workload utilisation is 80% on top of which the individual Micro-Services are executed. The μS depicted in Figure 3.9c, hosted on the server from Figure 3.9a, was allocated on average 31% of the requested CPU time and 51% of the requested memory resulting in an average execution time 59% longer than its specified length of 16ms. Similarly the μS in Figure 3.9d on server Figure 3.9b took on average 42% longer than specified with 52% of requested resources being allocated on average.

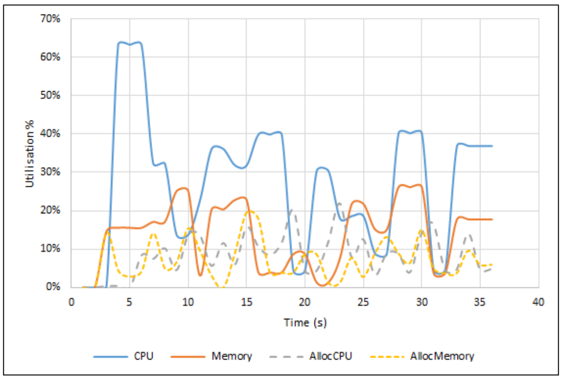
In terms of the μS resource utilisation there is an important observation to be made around



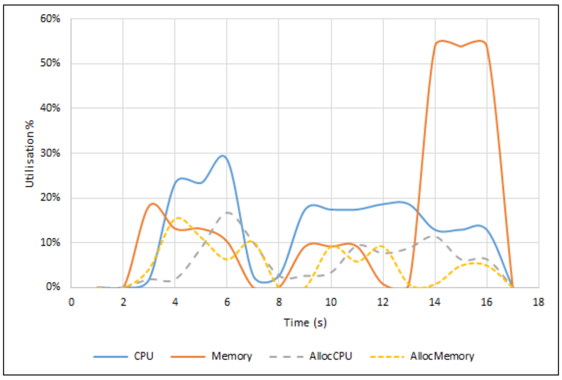
(a) A server instance hosting LA2



(b) A server instance hosting EN1



(c) Micro-Service LA2 requested and allocated resources



(d) Micro-Service EN1 requested and allocated resources

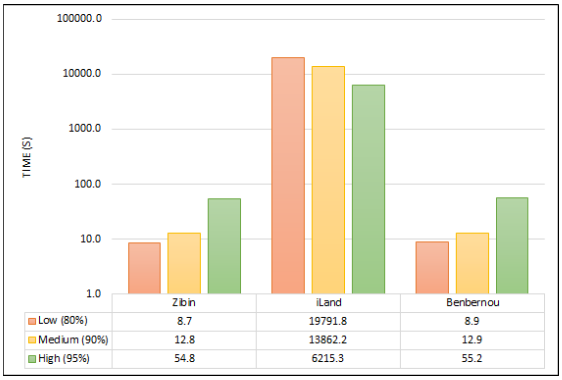
Figure 3.9: Server and Micro-Service utilisation with low periodic interference (80%) workloads

the shape of the graphs. In the Figure 3.9c μS example the lazy nature of the process acquiring resources and its active release causes a relatively unstable shape with numerous points at which it can blocked. This also means that it much harder to identify correlation between the pattern of requested resources and those allocated. In the case of Figure 3.9d the eager nature of the process results in a much more predictable pattern that can be seen in both the requested and allocated resources.

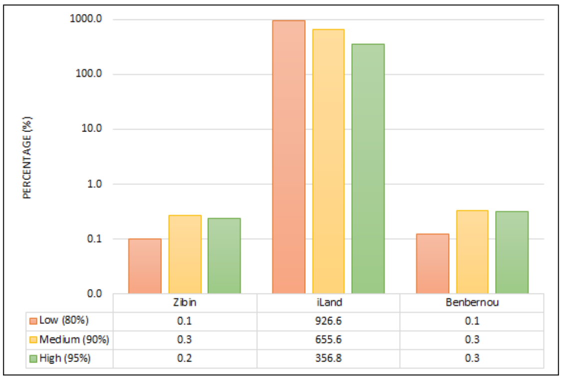
The remainder of this section considers each of the measures discussed in Section 3.3.1 relating to prediction accuracy and QoS violation.

Prediction Accuracy

Eqn. 3.5 defines the measures for Mean Absolute Error (MAE) and Mean Percentage Waste (MPW) by comparing the predicted QoS and actual observed response-time values. Figure 3.10



(a) Mean Absolute Error (MAE)



(b) Mean Percentage Waste (MPW)

Figure 3.10: Average accuracy of QoS across all μS types

depicts the average MAE and MPW of each of the approaches under low, medium, and high interference. Most notably the real-time iLand approach (described in Section Section 3.2.2) results in the largest error by a significant margin (a). It can also be seen that this is the only approach in which the wasted allocated time (b) is in excess of 100% of the actual execution time. The approaches by Zibin et al. (Section Section 3.2.1) and Benbernou et al. (Section Section 3.2.6) have average errors within 2% of each other, 25.4 and 25.7 respectively.

Further, with the exception of the iLand approach, the observed error increases along with the interference load. The 10% increase in interference from low to medium results in a 44% for Benbernou, and 47% for Zibin. The further 5% increase to a high interference workload results in a further 18%, 328% and 329% error increase. Additionally comparing between the μS lengths of the medium length Cloud Types (T2, see Table 3.1) reveals an exponentially increased error rate of 186% compare to T1 whilst T3 has a further increased error rate of 1897%.

Beyond the MAE the Figure 3.11 depicts the cumulative wasted time by each of the QoS approaches. Each of the approaches tends towards some plateau defined by the worst observed response-time. For the μSs which acquire resources eagerly the QoS predicted by Zibin et al. consistently wastes less time, 14% rather than 24%, than the approach by Benbernou. In the case of the remaining μSs both approaches waste between 26% and 28% of allocated computation time. However unlike the absolute error observed, the waste percentage decreases proportional to the μS's length for both the approach by Zibin et al. and Benbernou et al.. In the case of iLand the wasted time increases exponentially with the length of the μS.

So far the non-Real-Time approaches using correlation by Zibin et al. and fuzzy-logic by Benbernou et al. have demonstrated very similar results, whilst the real-time iLand method wastes

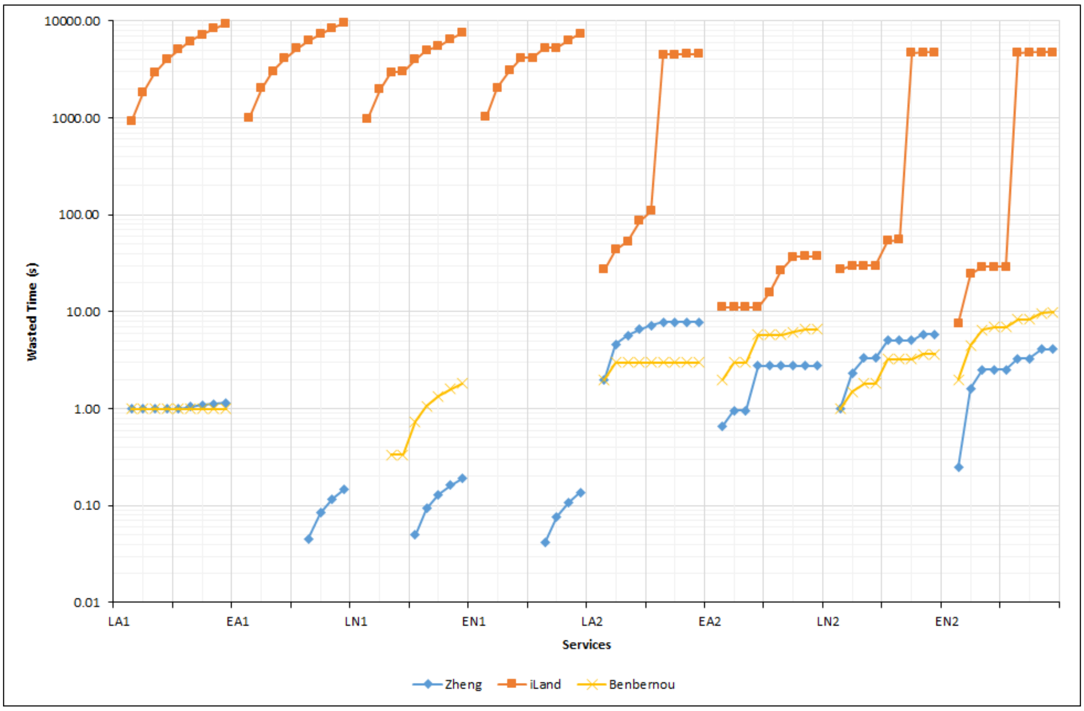
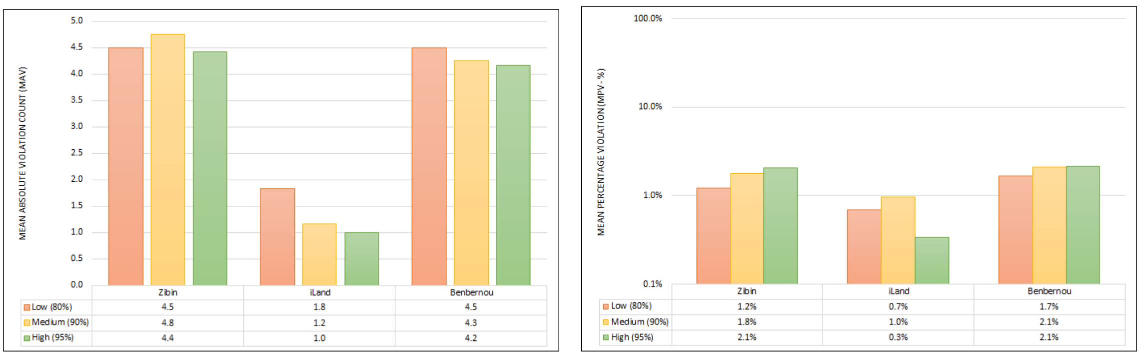


Figure 3.11: Cumulative wasted time due to over allocated QoS with 80% interference.

significant execution time.

QoS Violation

Eqn. 3.6 provides the calculation for the Absolute Violation Count (AVC) and the resulting Mean Absolute Violation (MAV). As shown in Figure 3.12, the iLand approach sees the fewest violations (13%), due to extreme cases, with response-times being only 0.7% longer than the specified QoS in those instances. The remaining approaches are again very similar on average with 1.7%



(a) Mean Absolute Violation (MAV) (b) Mean Percentage Violation (MPV)

Figure 3.12: QoS violations across all μS types

and 2% MPV for Zibin et al. and Benbernou et al. respectively.

3.4 Conclusions: Limitations of Current Techniques

This chapter has explored the existing approaches to RT-QoS. Simulation was then used to analyse in terms of QoS violations and QoS accuracy the effectiveness of a representative subset of those approaches:

Estévez-Ayres et al. [144] presents a real-time perspective on RT-QoS and over-estimates execution time to mitigate possible interference. As a result although this approach results in the lowest number of QoS violations (less than 20%) it also results in the most wasted computation time by three orders of magnitude.

Benbernou et al. [111] uses fuzzy-logic QoS predictions directly proportional to the observed Worst-Case Response Time. It demonstrated wasting less than 1% of allocated computation time whilst reducing QoS violations to less than 50% of μS invocations.

Zibin Zheng et al. [146] demonstrated very similar results to Benbernou et al. [111] with a very slight improvement with respect to both wasted computation time and QoS violations.

The simulation results firstly show that there is a trade-off between the accuracy, measured using MAE and MPW, and the number of QoS violations, measured using MAV. Those approaches with very few violations introduce large margins of error. Conversely those approaches with low error margins suffer from many more violations. Further the results shown in Figure 3.9 demonstrated the complexity of the problem of predicting QoS.

The results have also shown that the length of the μS can also impact the accuracy of the predictions in addition to the resource acquisition pattern which any given μS adopts.

3.4.1 The Research Challenge

The results presented in this chapter demonstrate that current approaches to predicting RT-QoS trade-off prediction accuracy against the number of observed QoS violations. As a result those

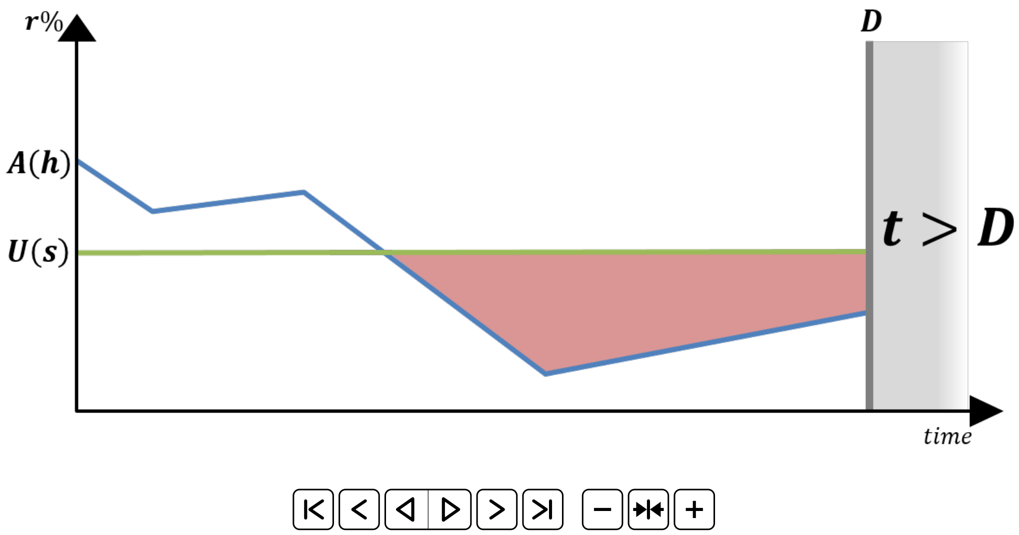


Figure 3.13: Example resource availability vs. required resources leading to a deadline miss

approaches with very few violations have QoS predictions that are orders of magnitude greater than the actual observed response-times. Additionally the simulation results have shown the complexity of understanding response-times of μSs running on servers with interfering workloads. The QoS approaches do not take into account issues of resource utilisation and do not explore any of the resource utilisation patterns, as shown in Figure 3.9. It is therefore anticipated that understanding the resource patterns will assist in forming more accurate predictions.

Hypothesis

It is anticipated that a better understanding of the relationship between resource utilisation and execution progress, based on actual observations, would facilitate the forming of more accurate RT-QoS predictions without compromising the number of QoS violations.

Such an approach must consider the level of interference experienced by a μS and aim to minimise both the Absolute Violation Count (AVC) and the Mean Percentage Waste (MPW). It would also be advantageous for such a method to provide online monitoring of μS execution progress so as to identify potential RT-QoS violations before they occur, which is not provided by the existing methodologies. In Figure 3.13 a visual representation of the problem is shown where the required resources are not available resulting in the deadline being missed.

The remainder of this thesis presents an approach that considers the relationship between the compute resources and μS execution performance. In the next chapter the mathematical formalisms for the approach are outlined. The theoretical potential of the proposed method is demonstrated using real-time schedulability theory in the context of an interfering workload. Then in Chapter 5 the simulations results are explored before in Chapter 6 the experimental results are discussed.

Chapter 4

n-D Framework for RT-SOA QoS

The previous chapters have introduced and reviewed the concepts of QoS prediction and looked at some of the limitations of existing techniques. Chapter 3 specifically details the limitations with regards to the MPV and the MPW. In the context of real-time systems the violation count must be minimised for both firm and soft deadlines and must be 0 where there are hard deadlines. The existing attempt to handle these by pessimistically over-allocating resources and time to a given μS results in significant wasted computation time.

This chapter presents the mathematical formalisms for predicting RT-QoS in the context of non-static resource-level interference.The framework can be used to estimate the response-time and remaining execution time of the specific μS given the current and historical n-dimensional state of the resources, such as CPU and memory. First the real-time premise for the framework is formalised to provide the foundation for the rest of the chapter. Then Subsequently the framework is developed and then evaluated against that premise.

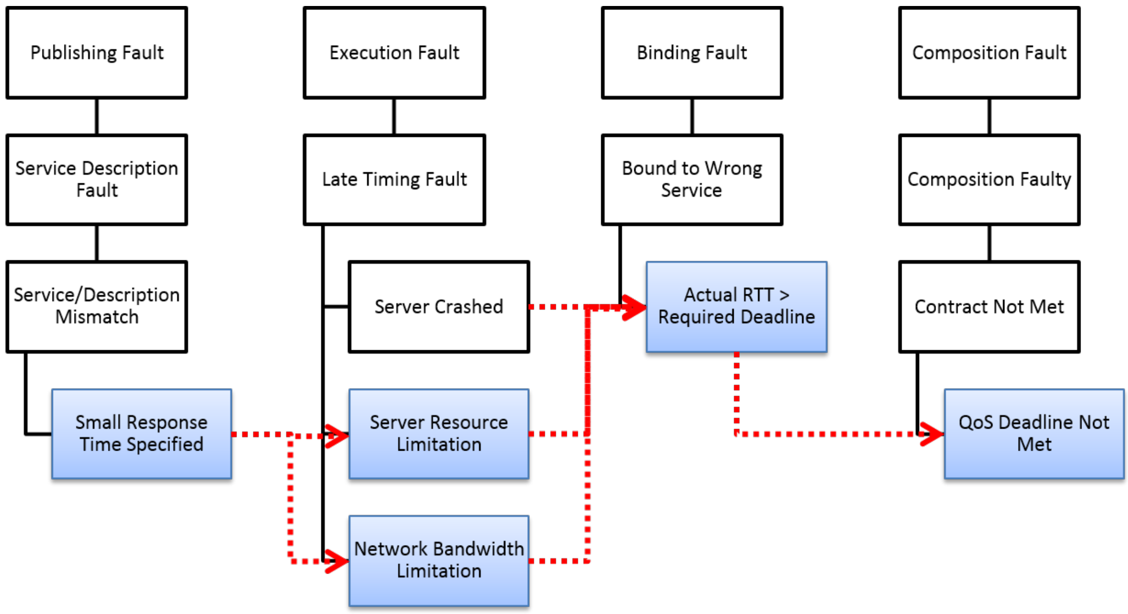


Figure 4.1: Extended SOA fault taxonomy from Bruning et al. [102]

4.1 Framework Premise

As discussed on Page 44 the central tenet of real-time systems is the ability to perform schedulability tests to indicate whether a given process will meet the required deadlines. Chapter 3 presented a hypothesis linking the QoS prediction to understanding resource utilisations. Therefore the SOA fault taxonomy presented in Chapter 2 Figure 2.14 can be extended with additional fault classes for the specific description fault whereby the specified response-time can not be delivered due to resource limitations within either the host server or across the network. As shown in Figure 4.1 this can therefore result in the QoS deadline not being adhered to [170].

In order to test this hypothesis and develop a framework to model this relationship a schedulability test must defined to indicate whether the scheduled processes are schedulable or not. A *sufficient* schedulability test indicates that the processes are definitely schedulable; whilst a *necessary* test can indicate whether they are definitely not schedulable.

Since this research is not concerned with workflow-level QoS whether the μSs are periodic or not can be disregarded by assuming that the deadline is equal to its period. Therefore the traditional utilisation schedulability test (Eqn. 2.2) can be rearranged as follows:

$$U = \frac{C_i + I_i}{D_i} \tag{4.1}$$

Where I_i represents the total utilisation by other processes interfering with μS i. However, unlike in traditional real-time systems these are not static values. The interference experienced by the μS will vary over its execution duration. Therefore the total availability of resources from when the μS is requested to its deadline must be greater than or equal to the resources required by that μS:

$$\forall r, \sum^{D}(1 - I_{i,r,t}) \geq U_{i,r} \tag{4.2}$$

Where the interference $I_{i,r}$ experienced by i is a percentage of resources at a given time t. Then over the computation time of i for all the resource types the total interference must leave sufficient resources for computation to complete. For example in the below figure the interference is not sufficient to cause the deadline to be missed:

t	1	2	3	4	$\sum^{D=4}(1 - I_r)$		
I_{r1}	10%	20%	15%	30%	3.25		$3.25 \geq 2.75$
I_{r2}	15%	25%	40%	20%	3		$3 \geq 2.05$
U_{r1}	60%	55%	80%	80%		2.75	
U_{r2}	50%	40%	55%	60%		2.05	

The remainder of this section outlines the notation that will be used in building the framework.

4.1.1 Notation

The notation used in this chapter follows the following format:

$$\Theta_t[\phi](s_n)^x\{r\}$$

Where:

Θ is the function or component of interest.

t is the time stamp.

ϕ are constraints being applied using Iverson brackets.

s is the μS being modelled. Although required for a full statement, some statements omit this in order to improve readability resulting in the form: $\Theta^{x}_{t,\{r,i\}}[\phi]$

n is the execution instance of the μS. If it is not present indicates that Θ is targeting the overall model covering all execution instances.

x is the specified named component of the model.

$\{r, i\}$ is the set of dimensions and coordinates within the model. For example if there are 2 resource dimensions the r values provide the 2D coordinates.

Additionally:

$\cdot$ represents a placeholder.

$\|\cdot\|_f$ represents some form of normalisation using function f.

$[\cdot]$ represents constraints or conditions being applied.

And finally below is a list of the symbols used in this chapter along with the page number of their formal definition:

$\mathbf{S}$	Set of all Micro-Services	85
$\mathcal{S}$	A Service in the registry	85
$\mathbf{s}$	Micro-Service	86
p	Execution Progress	86
f	Observation and monitoring frequency.	86
k	The number of observation points.	86
$\mathbf{RTT}$	Response-Time	86
h	Host	86
$\mathbb{H}$	Set of all Hosts	86
$\mathcal{R}$	Set of all Resource Types	86
r	A resource type with a capacity and measure of performance	87
d_r	Discrete set of resource values	87
α	The observed resource availability on the host	88
j	The model coordinate values for each resource dimension.	88
U	The utilisation model.	89
ω	An observation.	89

Ω	Set of all observations for a given Micro-Service instance.	89
F	Forecast of the resources required until execution completes.	90
$\mathcal{M}$	The model itself.	91
T	Set of historical response-times	91
I	Indicator function defining the density of the model at any given point.	91
$\mathcal{TTF}$	Time-to-Finish for Micro-Service execution.	91
$\mathcal{D}^{-1}$	Inverse distance measure between any two points in the model.	93
D	The Micro-Service or Service Deadline	94

The next section presents the development of the framework.

4.2 Initial Framework

The framework can be broken down into three distinct units defining the: relationship between μSs and their execution environment; the resource utilisation models; and response-time prediction. This section details each of these providing mathematical definitions for each element of the model.

4.2.1 Service and Environment Model

The first element of the framework is the definition of services and Micro-Services (μSs) as described in Section 2.1.2 on Page 23. Subsequently each μS executes in a host environment which must be modelled with respect to its resources.

Services and Micro-Service (μS)

As discussed in Section 2.1.2 services can be decomposed into Micro-Services (μSs) as their smallest practical functional elements. In that context each service in the registry, or set of all services, is comprised of a subset of μSs, from the set $\mathbf{S}$ of all μSs, and the interactions between them as shown in Figure 4.2. In this example service $\mathcal{S}^1$ can be decomposed into a mixture of other services $\mathcal{S}^2$ and $\mathcal{S}^3$ as well as μSs s^1, s^2, and s^3 in addition to the interactions between them e^1 to e^7. As shown in the case of $\mathcal{S}^2$ a service may iteratively be comprised of μSs or other services.

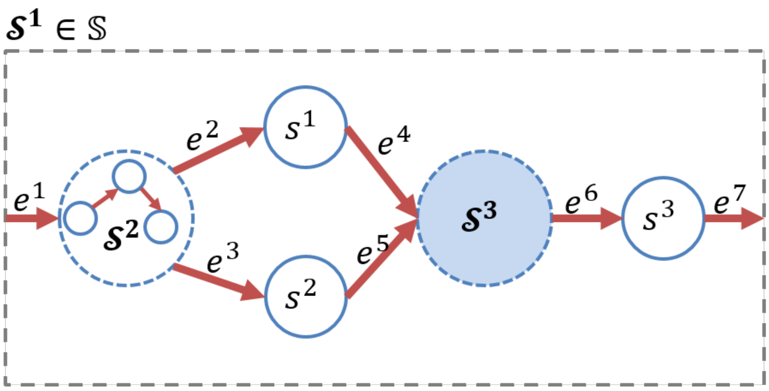


Figure 4.2: Decomposition of services into μSs

A μS $\mathbf{s}$ is defined as a functional element for which it is not practical to decompose into smaller components. It may execute one or more times, and any given execution instance is defined as: $\mathbf{s}_n$. The execution progress p of the μS can be monitored at a frequency f such that there will be an average of k_n observations given a response-time $\mathbf{RTT}$:

$$\begin{aligned} &\forall \mathbf{s} \in \mathbf{S}, \forall n, \exists f \in \mathbb{Z}^+ : \\ &\quad k(\mathbf{s}_n) = \frac{\mathbf{RTT}}{f} \\ &\quad k(\mathbf{s}) = \frac{\sum_i^n k(\mathbf{s}_i)}{n} : \\ &\quad\quad 0 \leq p(\mathbf{s}_n) \leq k(\mathbf{s}) \end{aligned} \tag{4.3}$$

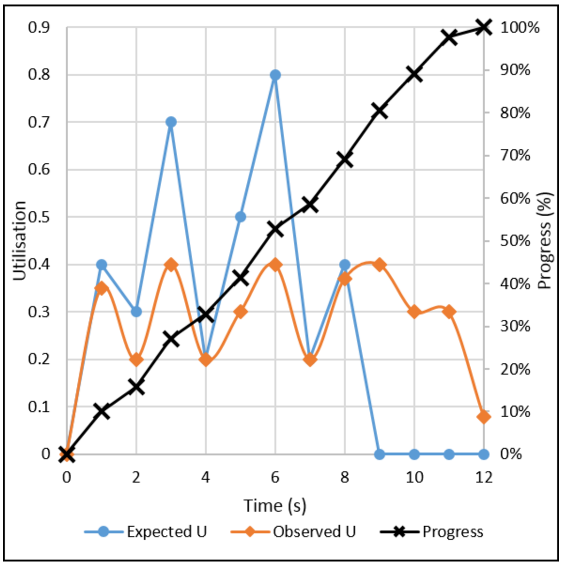
Which provides a measure of execution progress p as visually shown in Figure 4.3. $k(\mathbf{s}_n)$ provides for a particular execution instance the number of observation points that were collected over the response $\mathbf{RTT}$ with the specified frequency f . Averaging this over all n execution instances provides $k(\mathbf{s})$. The execution progress is then a value between 0 and $k(\mathbf{s})$. Figure 4.3b shows with $k(\mathbf{s}) = 9$ given a frequency of $f = 1s$. The actual progress of the $\mathbf{s}$ instance in question is slower than anticipated resulting in 12 observations: $k(\mathbf{s}_n) = 12$.

Host Environment and Resource Availability

Each execution instance $\mathbf{s}_n$ of a μS will execute on a host sever h , from the set of all host servers $\mathbb{H}$, which will have set of resources $\mathcal{R}$ which could include CPU and memory:

$$\begin{aligned} &\forall h \in \mathbb{H}, \forall r \in \mathcal{R} : \\ &\quad h(\mathbf{s}_n)_r = \|h(\mathbf{s}_n)_{r_{cap}} \cdot h(\mathbf{s}_n)_{r_{perf}}\|_{\underline{\%}} \end{aligned} \tag{4.4}$$

(a) Visual representation of progress



(b) Observed utilisation below expected resulting in execution slowdown with progress updating every second

Figure 4.3: Example execution progress

Such that the resource provision by any given host is normalised to a percentage against the smallest provision of that particular resource r. The definition of h_r takes into account both the capacity of the provided resource r_{cap}, such as memory or CPU capacity, as well as its performance r_{perf}, for example comparing a *2GHz* processor to *3GHz* processor. The normalisation adopted by $\|\cdot\|_{\underline{\%}}$ is:

$$\|x\|_{\underline{\%}} = \frac{x}{\min x \in X}$$

Normalising based on the smallest value of x from the set of all X. The result is a ratio of the particular x of interest divided by that smallest value. For example if the smallest value in X is 2 and the particular x of interest is 3 then the normalised value will be 1.5 which corresponds to 150%.

Further to constrain the possible values for r to a discrete set of isolated points it is discretised to d_r. Were d_r is a set of discrete possible values between 0 and 1:

$$\begin{aligned} &\forall r \in \mathcal{R}, \exists d_r \subset \mathbb{R}^+ : \\ &\qquad \forall x \in d_r, 0 \leq x \leq 1 \forall y \in \{d_r \setminus x\}, \exists \delta > 0 : \\ &\qquad\qquad dist(x,y) > \delta \end{aligned} \tag{4.5}$$

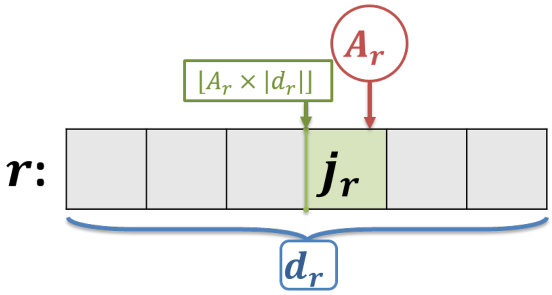


Figure 4.4: Framework multi-dimensional coordinate system indexed by j

Such that for every pair of elements (x, y) between 0 and 1 from the discrete space d_r there is positive distance between them unless they are the same point. For example if there are 5 values in the set d_r they could be $\{0, 0.25, 0.5, 0.75, 1\}$.

Therefore at any given time t the resource on h may be in partial or complete use by other interfering processes. As such a measure of resource availability α can be defined:

$$\forall r \in \mathcal{R}, \forall t, \alpha(h, \mathbf{s}_n) = 1 - \overbrace{(U(h(\mathbf{s}_n))_{r,t}}^{\text{Host utilisation}} - \overbrace{U(\mathbf{s}_n)_{r,t})}^{\mu\text{S utilisation}} \tag{4.6}$$

Where $U(h(\mathbf{s}_n))_r$ provides the sum of resource utilisation by all processes (including $\mathbf{s}_n$) hosted by h and $U(\mathbf{s}_n)_r$ the resources utilised by only μS at time t. This defines the available resources to $\mathbf{s}_n$ as being those either already being consumed by it or those not being consumed at all on h . The availability observations can then be combined to provide a total and average observed availability over the duration of a μS execution instance:

$$\begin{aligned} A_r^{\Sigma}(h, \mathbf{s}_n) &= \textstyle\sum_{p=0}^{k} \alpha(h, \mathbf{s}_n)_{r,p} \\ A(h, \mathbf{s}_n)_r &= \frac{A_r^{\Sigma}(h, \mathbf{s}_n)}{k} \end{aligned} \tag{4.7}$$

Finally these availability values can be converted into coordinate values j in the discrete space of d_r as shown in Figure 4.4:

$$j = \{r \in \mathcal{R} : \lfloor A(h, \mathbf{s}_n)_r \times |d_r| \rfloor\} \tag{4.8}$$

4.2.2 Resource Utilisation Model

In order to predict the response-time of a given μS it is necessary to model the relationship between performance and resource utilisation. Therefore depending on the resources made available to the μS there may be a slowdown or speed-up of performance. This section builds on the definitions for resource availability on a μS's host from the previous section and introduces the multi-dimensional framework for modelling the resource utilisation.

Utilisation Model

The model itself is a multi-dimensional space of discrete points which are indexed by j . j is defined by Eqn. 4.8 and provides the context or constraints under which the μS is executing:

$$\begin{aligned} &\mathbf{RTT}[A(h(\mathbf{s}_n)](\mathbf{s}_n): \\ &\qquad \{\forall r: A(h(\mathbf{s}_n)_r)\} \rightarrow j: \\ &\qquad\qquad \forall r, \exists U(\mathbf{s})_{j_r} \end{aligned} \tag{4.9}$$

Which specifies that the availability of resource, across all the resource types, generates a co-ordinate set j . For every possible j there exists a utilisation model U for each resource type.

Subsequently in order to populate U data must be collected from a running μS instance $\mathbf{s}_n$. These observations ω will occur at the specified frequency f , resulting in a set of observations Ω for that μS instance. The observations themselves must be either downsampled or interpolated $[\![\cdot]\!]_{ip}$ to provide k values to be used by the model. Each observation is also normalised $[\![\cdot]\!]_{0..1}$ as a percentage of the resources available on the largest host:

$$\{\forall r, \forall \omega \in \Omega(\mathbf{s}_n) : [\![\omega_r]\!]_{0..1}\} \xrightarrow{[\![\cdot]\!]_{ip}} \{u_r \in U(\mathbf{s})_j\} \tag{4.10}$$

The interpolated observations u_r are then used to populate the utilisation model U . Each point in the model as indexed by j and the execution progress p is a 5-tuple comprising of the mean, minimum, maximum, variance, and sum of squared differences for that configuration point over the history of execution instances:

$$\forall j, \forall x \in \{\mu, \wedge, \vee, \sigma^2, SQD\}, \forall p \leq k : \exists U^x_{j,p} \tag{4.11}$$

Each component within the 5-tuple is updated iteratively with each new execution instance n. The mean can therefore be calculated as the previous mean $U(\mathbf{s})^{\mu_{n-1}}_{j,r,p}$ in addition to the difference between interpolated observation $u(\mathbf{s}_n)_{r,p}$ and the previous mean divided by the number of instances in that particular configuration $|U(\mathbf{s})_{j,p}|$ which will always be less than n:

$$U(\mathbf{s})^{\mu_n}_{j,r,p} = U(\mathbf{s})^{\mu_{n-1}}_{j,r,p} + \frac{u(\mathbf{s}_n)_{r,p} - U(\mathbf{s})^{\mu_{n-1}}_{j,r,p}}{|U(\mathbf{s})_{j,p}|} \tag{4.12}$$

Then in order to calculate the variance the sum of squared differences SQD is derived from the mean:

$$\begin{aligned} U(\mathbf{s})^{SQD_n}_{j,r,p} &= U(\mathbf{s})^{SQD_{n-1}}_{j,r,p} + \Big(u(\mathbf{s}_n)_{r,p} - U(\mathbf{s})^{\mu_n}_{j,r,p}\Big)^2 \\ U(s)^{\sigma^2_n}_{j,r,p} &= \frac{U(\mathbf{s})^{SQD_n}_{j,r,p}}{n} \end{aligned} \tag{4.13}$$

In addition the observed minimum and maximum resource utilisations are calculated by comparing the new observation against historical runs:

$$\begin{aligned} U(s)^{\wedge_n}_{j,r,p} &= \min\Big(U(s)^{\wedge_{n-1}}_{j,r,p}, u(\mathbf{s}_n)_{r,p}\Big) \\ U(s)^{\vee_n}_{j,r,p} &= \max\Big(U(s)^{\vee_{n-1}}_{j,r,p}, u(\mathbf{s}_n)_{r,p}\Big) \end{aligned} \tag{4.14}$$

These can then be summarised to provide a forecast F of the remaining resources required for execution to complete from a given point p and the variance of utilisation over the execution duration:

$$\begin{aligned} &\forall r, \forall j, \forall p \leq k, \exists F(s) : \\ &\qquad F(s)^{x \in \{\mu, \wedge, \vee\}}_{j,r,p} = \textstyle\sum_{i=p}^{k} U(s)^{x}_{j,r,k-i} \\ &\qquad F(s)^{\sigma^2}_{j,r,p} = \frac{\sum_{i=p}^{k} U(s)^{\sigma^2}_{j,r,k-i}}{k-p} \end{aligned} \tag{4.15}$$

The forecast calculations shown in the above equations specifically calculate the summations in reverse order with suffix $k - i$ as this algorithmically reduces the computation needed from $\mathcal{O}(k^2)$ to a linear calculation of $\mathcal{O}(k)$. This forecasting method assumes that a given μS will have a relatively consistent resource utilisation pattern for a given resource availability configuration. The next section uses these utilisation models and forecasts to estimate response-times for μSs.

4.2.3 Predictive Model

The utilisation and availability observations allow for a model to be constructed that represents the μS's performance with respect to the resources with which it is provided. This model can be defined in two stages with respect to structure and response-time prediction.

Model Structure

The model $\mathcal{M}$ exists as an $|\mathcal{R}| + 1$ dimensional model which accounts for each of the resource types in addition to the dimension of time. $\mathcal{M}$ is therefore a $k \prod_{\forall r \in \mathcal{R}} d_r$ matrix. If $k = 10$, $d_{r1} = 3$, and $d_{r2} = 4$ the model would be a $10 \times 3 \times 4$ matrix. Each point in the matrix is itself a 6-tuple capturing all the elements of the utilisation, forecast, and predictive models:

$$\forall m \in \mathcal{M}, \mathcal{M}_j = m, m = \langle t^{\vee}, t^{\mu}, T_j, U_j, F_j, I_j \rangle \tag{4.16}$$

Where T stores all the recorded response-times observed by the μS: $T_j(\mathbf{s}) = \{T_j(\mathbf{s}), \mathbf{RTT}\}$ and U_j is the utilisation model discussed in the previous section. I acts as the indicator function, as used by Zibin Zheng et al. [146] as a significance weighting, and is defined as a measure of the density or number of execution instances that ran under a given availability constraint:

$$\forall j, I_j = |\mathcal{M}_j^T| \tag{4.17}$$

Response-Time Prediction

This model structure can then be used to predict the response-times of μSs as they are deployed based on the current availability of resources. The resource availability is mapped onto the coordinates j which is used to index the model. For the prediction the first two parts of the 6-tuple can be used to give either a nominal t^{μ} or pessimistic prediction $t^{\vee}$:

$$\mathbf{RTT}[A(h(\mathbf{s}_n))](\mathbf{s}_n) = \left\langle m^{t^{\mu}}, m^{t^{\vee}} \right\rangle \tag{4.18}$$

The nominal prediction $m^{t^{\mu}}$ can be calculated as an iterative mean of response-times using the same form as Eqn. 4.12. The pessimistic prediction provides the observed WCET and is therefore more appropriate for hard real-time systems.

Once execution has started the time-to-finish $\mathcal{TTF}$ for the μS instance can be estimated using the execution progress p :

$$\mathcal{TTF}[A(h(\mathbf{s}_n))](\mathbf{s}_n) = \left(1 - \frac{p}{k}\right) m^x \tag{4.19}$$

Where x is either of the response-time predictions $t^{\vee}$ or t^{μ}. p is then estimated by comparing the observed and expected resource utilisation at time t:

$$p(\mathbf{s}_n)_t = \max\left\{p(\mathbf{s}_n)_{t-1}, \min\left\{\forall r \in \mathcal{R} : \frac{1}{k}\left\lfloor\frac{k \cdot \sum_{x=0}^{t} [\![\Omega(\mathbf{s}_n)_{j,r,x}]\!]_{0..1}}{F(\mathbf{s})_{j,r}}\right\rfloor\right\}\right\} \tag{4.20}$$

Where the observed utilised resources Ω are totalled and compared against the forecast resources F . This comparison is against each resource type and the worst case provide the estimate of the execution progress. Therefore the estimation pessimistically chooses the worst-case scenario by choosing the resource dimension with minimum observed utilisation compared to the forecast utilisation. Within the min function k rounds those values to the nearest multiple of k . This assumes that the work done by the μS is non-decreasing as time progresses .

Initial and Sparse Model Prediction

The model so far facilitates the estimation of response-times and remaining computation time of μSs using previous μS execution data under the currently observed resource availabilities. There are however two situations where the current model is not sufficient:

1. **Initial-case** where the model is empty with no execution information about the μS.
2. **Sparse-case** where there is information about execution performance under a subset of resource availability configurations.

In the first instance the prediction must be based on WCETs provided by the μS developer. This however may be provided in three forms:

1. Response-time purely as a nominal value.
2. WCET with a utilisation model or resource requirement.
3. Response-time with or without resource utilisation information but alongside a host specification.

Where resource utilisation information is provided this can be used in conjunction with traditional real-time systems techniques to estimate the WCET:

$$\mathbf{RTT}[A(h(s_n))] = \left\langle \frac{U^{\mu}_{provided}}{A(h(s_n))}, \frac{U^{\vee}_{provided}}{A(h(s_n))} \right\rangle \tag{4.21}$$

However if resource utilisation data is not available it must be assumed for the worst-case instance that the μS utilised 100% of the host's resources for its execution duration. Additionally unless a host specification is provided it must be assumed that the response-time, and if provided utilisation information, is relative to the most powerful available host of the available hosts $\mathbb{H}$. If, however, a host specification is provided this can be used as a normalising factor for the provided information.

In the instance where the model is sparsely populated, where μS execution has occurred under other resource configurations than those currently observed, an estimate can be inferred from the existing data. Specifically by considering the distance from the current configuration j and existing data points within the model an inverse distance measure can be calculated $\mathcal{D}^{-1}$:

$$\mathcal{D}^{-1} = \frac{1}{dist(i,j)} \tag{4.22}$$

Where $dist(a, b)$ can use any distance measure such as Euclidean or Manhattan distances between points a and b within the model space specified by the dimensions of r and size d_r. This can be used in combination with the indicator function I to assign greater significance to those configurations i that are closest to j and have more historical data.

$$\mathbf{RTT}[A(h(\mathbf{s}_n))](\mathbf{s}_n) = \left\langle \frac{\sum_{i \neq j}\left(I(\mathbf{s})_j \cdot \mathcal{M}(\mathbf{s})_j^{\mu} \cdot \mathcal{D}^{-1}(i,j)\right)}{\sum_{i \neq j}(I(\mathbf{s})_j \cdot \mathcal{D}^{-1}(i,j))}, \frac{\sum_{i \neq j}\left(I(\mathbf{s})_j \cdot \mathcal{M}(\mathbf{s})_j^{\vee} \cdot \mathcal{D}^{-1}(i,j)\right)}{\sum_{i \neq j}(I(\mathbf{s})_j \cdot \mathcal{D}^{-1}(i,j))} \right\rangle \tag{4.23}$$

Complete Model

The resulting complete model can therefore be expressed as a conditional statement:

$$\mathbf{RTT}[A(h(\mathbf{s}_n))] = \begin{cases} \text{(Eqn. 4.21)} & \mathcal{M}^T \equiv \emptyset \\ \beta_1 \cdot \text{(Eqn. 4.23)} + \beta_2 \cdot \text{(Eqn. 4.21)} & \mathcal{M}_j^T \equiv \emptyset, \exists i : \mathcal{M}_i^T \neq \emptyset \\ \gamma_1 \cdot \text{(Eqn. 4.18)} + \gamma_2 \cdot \text{(Eqn. 4.23)} + \gamma_3 \cdot \text{(Eqn. 4.21)} & \mathcal{M}_j^T \neq \emptyset \end{cases} \tag{4.24}$$

Where the cases represent the initial and empty, sparsely-populated, and probabilistic models respectively. This statement allows for the three methods to be combined utilising the weight factors β and γ where $\beta_1 + \beta_2 = 1$ and $\gamma_1 + \gamma_2 + \gamma_3 = 1$. Overall this method provides the functionality shown in Figure 4.5 whereby the response-time is periodically estimated and can be

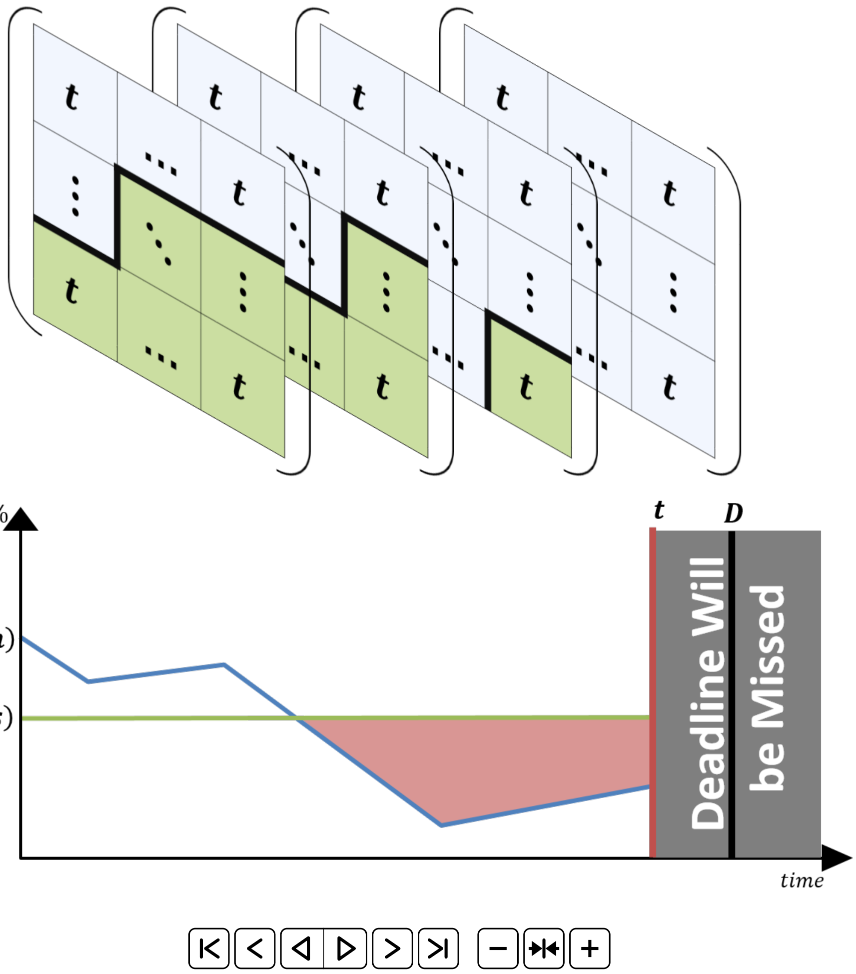


Figure 4.5: Example from Figure 3.13 where the deadline miss is identified by the framework prior to the event.

used to identify during execution whether a deadline **D** is likely to be missed given the current resource availability.

The next section demonstrates how this framework for predicting the response-time of a μS can be used and evaluates its effectiveness with respect to schedulability, scalability, and efficiency.

4.3 Theoretical Evaluation and Application

This section presents a detailed analysis of the multi-dimensional mathematical framework presented in the earlier parts of this chapter. First though the application of the framework, its workings in practice, and algorithmic efficiency will be shown before the effectiveness of the approach is considered. The presented framework will be analysed firstly with respect to real-time systems schedulability and its ability to provide deadline guarantees (Section Section 4.3.2).

4.3.1 Framework Application and Algorithmic Analysis

In order to use the framework presented in this chapter it must be first expressed algorithmically. In Algorithm 1 the core of the algorithm is presented and then is expanded in subsequent algorithms later in this section.

The core algorithm consists of two phases:

1. **Online monitoring** shown as line 1 to line 17 which summarises the online observation of resource utilisation and availability whilst updating the predicted time-to-finish and progress during μS execution.

2. **Model updating** on line 18 to line 22 which focusses on constructing and updating the predictive model with the recorded data.

Each of these phases are expanded into their constituent parts in the remainder of this section.

Online Monitoring

In the first phase, as the resource utilisation of the μS and on the server is observed, the forecast model can be used to predict the remaining $\mathcal{TTF}$.

As discussed on Page 86 the availability of resources to an executing μS may vary over its execution duration, as depicted in Figure 4.5. The proposed framework requires observation of the resource availability at a constant frequency f . These observations are used to calculate the model coordinates j , as described in Eqn. 4.8, for the current configuration as shown in Figure 4.4 and in Algorithm 3 on Page 215 in the appendices.

Then as described on Page 91 the execution progress p can be estimated as shown in Algorithm 4. As described on Page 92 the algorithm accounts for (see line 4 to line 7) the provision of either a utilisation value or expected response-time as part of the initial case. Given the estimation of p , the time-to-finish $\mathcal{TTF}$ can be predicted as shown in Algorithm 5 and described on Page 91.

Algorithm 3, Algorithm 4, and Algorithm 5 can then integrated into the online monitoring and prediction process shown in Algorithm 2. In terms of performance the first two algorithms operate linearly with respect to the number of resource dimensions: $\mathcal{O}(r)$ which can be assumed

Algorithm 1: Core Algorithm

```
begin Online Monitoring
    Invoke s on h
    Start Timer
    p = 0
    Ω = EmptySet
    while s running do
        if Timer.Elapsed ≥ f then
            begin in parallel
                ω = OBSERVE_UTILISATION( s ,h)
                a = OBSERVE_AVAILABILITY(h)
            end
            Ω .Add(ω)
                                                /* See Algorithm 2 */
            ( p , TTF ) = UPDATE_PREDICTION( s , Ω ,a)
        end
        RTT = Stop Timer
    end
end
begin Model Updating
    n++
                                                /* See Algorithm 7 */
    T = UPDATE_DATA_SETS(RTT, A)
                                                /* See Algorithm 6 */
    U, A = BUILD_MODELS(u, a)
end
```

to be relatively small and would remain fixed for a given system. The calculation of $\mathcal{TTF}$ in Algorithm 5 operates in terms of size of each resource dimension: $\mathcal{O}(2 \cdot |d_r|^r)$. The overall online monitoring algorithm is therefore $\mathcal{O}(2 \cdot r + 2 \cdot |d_r|^r)$ which given a constant d_r and set of resources results in a constant execution time for a given system implementation.

Model Updating

The second phase of the core algorithm focusses on updating the models after execution has finished. This phase, as can be seen in Algorithm 1, consists of two steps:

1. Updating the data sets
2. Rebuilding the utilisation and availability models

Algorithm 2: Online monitoring and prediction

Input: α as calculated in Eqn. 4.6
Input: Ω the set of resource utilisation observation
Input: k specified as a constant
Result: $\mathcal{TTF}$ the estimated time-to-finish
Result: p the estimated execution progress

Startup: p defined initially as 0

/* At least 1 of **RTT** or $U_{provided}$ must be supplied */
Optional: $U_{provided}$ the statically provided, otherwise $U_{provided} = \infty$
Optional: **RTT** statically provided, otherwise $RTT = \infty$
Optional: h provided as the benchmark, otherwise $h = \infty$

/* See Algorithm 3 */
j = CALCULATE_J(α)
/* See Algorithm 4 */
p = ESTIMATE_PROGRESS($j, p, \Omega, k, [U, \mathbf{RTT}, h]$)
/* See Algorithm 5 */
$\mathcal{TTF}$ = ESTIMATE_TTF($j, p, \Omega, [U, \mathbf{RTT}, h]$)

The first stage can be seen in Algorithm 6 where from line 1 to line 5. Then from line 6 to line 9 the indicator function I is increased appropriately.

Subsequently the observed utilisation data from the execution instance is used to rebuild the models that are used to estimate execution progress. In Algorithm 7 first the average availability is calculated as per Eqn. 4.7 then the resource models are updated. The utilisation model is updated first as described in Section Section 4.2.2 on Page 89 and then the forecast models calculated. In terms of performance Algorithm 7 (line 1 to line 6) runs in $\mathcal{O}(2 \cdot r)$, taking into account the complexity of Algorithm 3. The remainder then loops on p and resources with a time complexity of $\mathcal{O}(k \cdot r)$. This gives an overall complexity of $\mathcal{O}(2 \cdot r + k \cdot r)$. This can be simplified to $\mathcal{O}((2 + k)r) \approx \mathcal{O}(k \cdot r)$.

These two constituent algorithms can be combined together to form the second phase of Algorithm 1. This allows the core algorithm in total to operate with $\mathcal{O}(|d_r|^r + k \cdot r)$ efficiency. Therefore for a fixed system configuration, where both k and r are predefined, the process time complexity is a constant value and does not increase with respect to the number of μSs or number of execution instances. However in terms of space complexity as the number of μSs and execution instances increases the storage required will increase linearly $\mathcal{O}(s \cdot n)$.

Deadline Alerts

As depicted in Figure 4.5 as the resource availability varies over the execution duration the point of interest (defined by j) within the model changes and provides an updated prediction on response-time and time-to-finish $\mathcal{TTF}$. Therefore at the estimated progress point it is possible to show whether the predicted response-time will meet the deadline $\mathbf{D}$ or not. Additionally it will show whether, based on previous runs, it is possible for the μS instance to meet the deadline given an increase in resource availability. This is shown in Algorithm 8 which checks each alternative configuration i that is larger than j to see if the $\mathcal{TTF}$ is less than or equal to the deadline. This warning which would form part of the first phase can be run in $\mathcal{O}(|d_r|^r \cdot 2 \cdot |d_r|^r) \approx \mathcal{O}(|d_r|^{2 \cdot r})$. As each part of this is constant the algorithm continues to operate in constant time for a given resource configuration and fidelity.

Given this algorithmic overview of how the framework can be applied, the next section explore in detail how it can be used with real-time systems schedulability tests.

4.3.2 Proof: The Bounds of Schedulability

At the beginning of this chapter the utilisation-based schedulability test was redefined in terms of interference on each resource dimension (Eqn. 4.2) which directly correlates with the resource availability metric used in this framework (Eqn. 4.6). Utilisation-based analysis, as described on Page 44, provides a sufficient schedulability test such that if the conditions are met the μS will meet the deadline. Although it does not guarantee that the μS will miss the deadline if the conditions are not met it can be reasonably assumed that as the interference grows and the schedulability condition is violated to a greater degree the likelihood that the deadline may still be met will decrease.

In order to evaluate the framework the following scenarios must be considered with respect to the workload patterns described in Section 2.3.3:

Scenario 1 Static workload resulting in a continuous level of interference.

Scenario 2 Dynamic workload either random, once-in-a-lifetime, or periodic in nature.

Scenario 3 Continuously increasing resulting in greater levels of interference and therefore reduced resource availability.

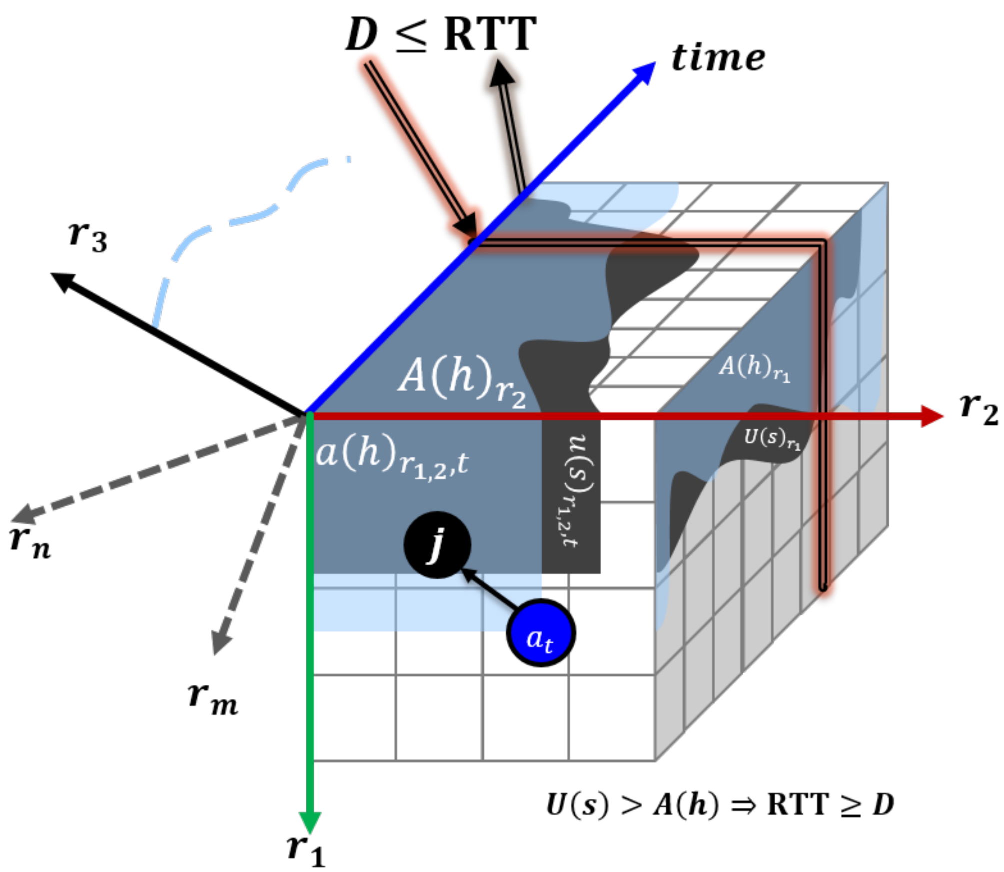


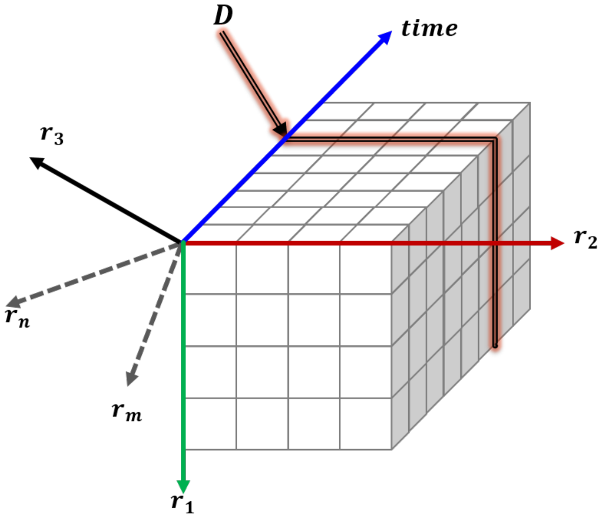
Figure 4.6: Visual proof without words of the proposed mathematical framework. For simplicity using single average (μ) values from 5-tuple representing U and $\mathbf{RTT}$.

Scenario 4 Continuously decreasing resulting in less interference and more resources available to the μS.

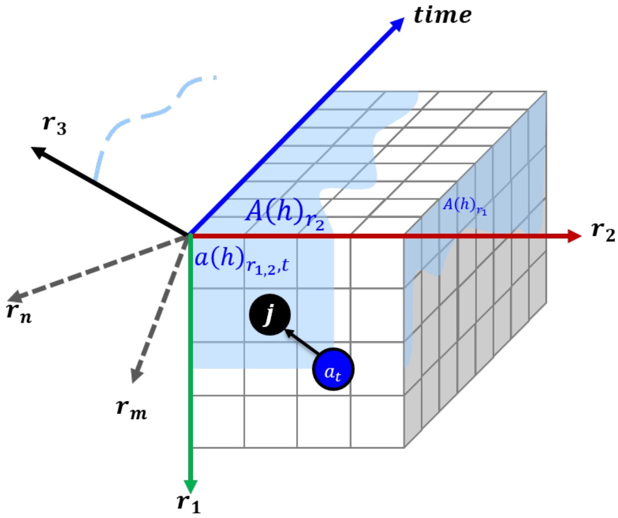
Using these scenarios the remainder of this section explores the presented framework in the context of the utilisation-based schedulability test in the form of visual and inductive proofs.

Visual Proof

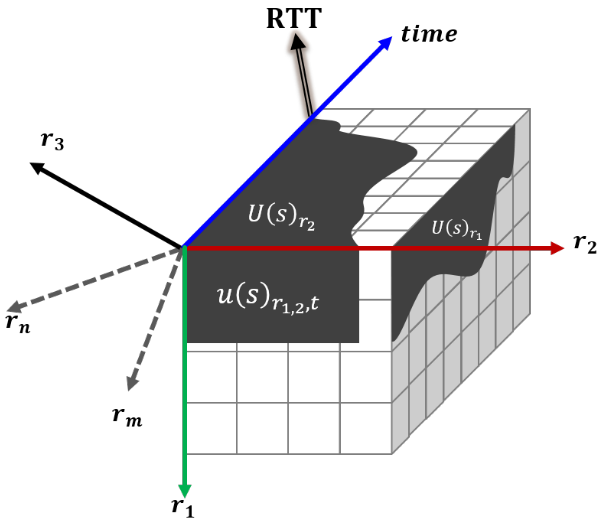
A visual proof, otherwise known as a *"proof without words"*, must demonstrate using a diagram the mathematical statement that the proposed framework satisfies the schedulability condition described earlier. Figure 4.6 depicts the framework in the context of *Scenario 2* above with a dynamic workload resulting in the assigned deadline potentially being missed. The nature of the sufficient schedulability condition dictates that it cannot guarantee the failure of the μS to meet the deadline as defined by the implication relation $\implies$.



(a) Mutli-dimensional space of the model with respect to $r \in \mathcal{R}$ and time with the μS's deadline D.



(b) Instantaneous resource availability α of h and total availability A.



(c) Required resources by the μS, u and U .

Figure 4.7: Construction of the visual proof.

The construction of the proof is shown in Figure 4.7 comprising of three stages:

Figure 4.7a Multi-dimensional space of the model with a deadline assigned to the μS.

Figure 4.7b The observed resource availability varying randomly over time and mapped onto the coordinate system value j .

Figure 4.7c The required resources by the μS which are greater than those available to it resulting in a response-time $\mathbf{RTT}$ greater than the specified deadline.

The following section details an inductive proof for each of the scenarios listed above.

Inductive Proof

In order to demonstrate that the proposed framework is theoretically effective in estimating the response-times of μSs and importantly predicting when and if a RT-QoS violation of a deadline being missed might occur this section details an inductive proof. The general proof is outlined before being applied to the individual scenarios detailed on Page 99.

Before outlining the proof itself there are four assumptions that must be considered:

Assumption 1 Section Section 4.2.2 in Eqn. 4.15 assumes that a given μS will have a consistent resource utilisation pattern for a given resource availability configuration. Therefore for a given configuration j the forecast utilisation F will be consistent with minimal variation.

Assumption 2 Section Section 4.2.3 on Page 90 in Eqn. 4.20 assumes that the work done by the μS is non-decreasing over time. This means that as the progress is recalculated the new progress value must be larger or equal to the last calculated progress p_{t-1}. This assumption holds true long as the μS cannot fail, restart, or resume execution from an earlier point. The framework is therefore ignoring faults and failures that are not related to the performance degradation due to interference but focussing only on those discussed in Section Section 4.1 Figure 4.1.

Assumption 3 Section Section 4.3.2 on Page 98 outlines the basic assumption that resource interference results in a slowdown of execution of the μS.

Assumption 4 According to traditional real-time systems execution time calculations the response-time of the μS will be approximately equal to:

$$max \left\{ \forall r \in \mathcal{R} : \frac{U(\mathbf{s})_r}{j} \right\} \tag{4.25}$$

Where in a single threaded and single core environment $U \equiv C$ which would be the number of CPU units required to complete (WCET). The framework therefore assumes that these real-time assumptions can be mapped onto a multi-threaded environment by representing the interference as a percentage.

Given the these assumptions using an inductive proof it can be shown that at a given point in time the set of identified points in the model stating that the μS will meet the deadline is *sound*. Additionally that set is the *complete* set of resource configurations for which it can be guaranteed that the μS will meet the deadline according to the *sufficient* utilisation-based schedulability test. It may not be the *complete* set based on an *exact* schedulability analysis.

The generic proof is therefore as follows:

Base Case The current resource availability $\alpha \mapsto j$ at time t is such that $\mathbf{RTT}_t[j] = D$.

Inductive Case If $i = j + x$ where $0 \leq x \leq |d_r|$, incremented in any resource dimension such that $\sum_r^{\mathcal{R}} i_r > \sum_r^{\mathcal{R}} j_r$. Which means that according to Assumption 4:

$$\begin{aligned}
\tfrac{U}{i} = \tfrac{U}{j+x} &\implies \mathbf{RTT}_t[i] = \tfrac{U}{i} \\
&\implies \mathbf{RTT}_t[i] = \tfrac{U}{j+x} \\
&\implies \mathbf{RTT}_t[i] = \tfrac{U}{jidx} - \tfrac{U \cdot x}{i} \\
&\implies \mathbf{RTT}_t[i] = \mathbf{RTT}_t[j] - \tfrac{U \cdot x}{i} \\
&\implies \mathbf{RTT}_t[i] \leq \mathbf{RTT}_t[j]
\end{aligned}$$

Therefore $\forall i \in \Gamma : i \geq j \implies \mathbf{RTT}_t[i] \leq \mathbf{RTT}_t[j]$

And $\forall i \in \Gamma : i \geq j \implies \mathbf{RTT}_t[i] \leq D$

Which shows that in the positive case where the μS has access to more resources, indicated by $i > j$, the response-time will reduce. This can then be applied to each of the scenarios detailed on Page 99 with static, dynamic, increasing, or decreasing interference over the execution duration of the μS:

Static takes the form of the generic proof where $\alpha_{t+1} = \alpha_t \implies j_{t+1} \equiv j_t$:

Base Case The response-time will be less than or equal to the deadline:

$$\frac{U}{j} \equiv \frac{F_{p=0}}{j} \implies \mathbf{RTT}_t[j] \leq D$$

Inductive Case Throughout execution the forecast time-to-finish $\mathcal{TTF}$ will be less than or equal to the deadline.

$$\frac{F_{p=1}}{j_{t+1}} \equiv \frac{F_{p=1}}{j_t} \implies \mathcal{TTF}_{t+1}[j_t] \equiv \mathbf{RTT}_t[j] - 1$$

Therefore throughout μS execution at all time points x the time-to-finish will correspond with the original estimated response-time:

$$\forall x \in \mathbb{Z}^+ : \tfrac{F_{p=x}}{j_{t+x}} = \mathcal{TTF}_{t+x}[j_{t+x}] = \mathbf{RTT}_t[j] - x$$
$$\implies \mathcal{TTF}_{t+x}[j_{t+x}] \leq D - x$$

Dynamic involves three unique cases whereby the μS either: (A) meets the deadline as expected; (B) fails to meet the deadline; (C) or is expected to fail to meet the deadline but performance improves allowing the deadline to be met. In each case α_{t+1} may or may not be equivalent to α_t but the average resource availability from $t = 0$, the point when the μS is started, to $t = D$ is defined by A according to (Eqn. 4.7). The actual observed response-time will be $\frac{U}{A}$ as defined by (item 4.25). The three cases are therefore outlined below:

A Is the case where although the interference experienced by the μS is dynamic, at every point during execution the resources required by the μS F in order to complete by the deadline D

are sufficiently provided:

$$\begin{aligned}\frac{F_{p=0}}{j_{t=0}} = \mathcal{TTF}_t[j] \leq \mathbf{D} : & \forall x, 0 \leq x \leq D : \\ & \textstyle\sum_{t=0}^{x} \omega_t \geq \frac{x}{\mathbf{D}} \cdot F_{p=0} \\ & \implies \textstyle\sum_{t=0}^{x} \omega_t \geq F_{p=0} - F_{p=\frac{x}{D}} \\ & \implies x + \frac{F_{p=\frac{x}{D}}}{j_{t=x}} \leq D \\ & \implies \frac{F_{p=0}}{A^{\Sigma}} \leq 1 \\ & \implies \mathbf{RTT}[A^{\Sigma}] \leq D\end{aligned}$$

B Is the instance where unlike in (A) the total resource availability for $0 \leq t \leq D$ is less than that required for the μS to meet the deadline. At the beginning though the $\mathbf{RTT}_t[j] \leq D$ but later interference will cause the failure:

$$\begin{aligned}\exists x, 0 < x < D : & \omega_{t=x} < u_{p=\frac{x}{D}} \\ & \bigwedge \textstyle\sum_{t=x+f}^{D} j_t \leq F_{p=\frac{x+f}{D}}[j_{t=x}] \\ & \implies \mathcal{TTF}_{t=x}[j_{t=x}] > D \\ & \bigwedge \mathbf{RTT}[A^{\Sigma}] \geq D\end{aligned}$$

Where even one instance of adequate resources not being available may result in the deadline being missed. Which is identified at time x with $\mathcal{TTF}_{t=x} > D$. In the above equation the $\wedge$ denotes the logical "And".

Additionally in some cases the framework may be able to show that it is not possible for the deadline to be met, even if adequate resources were to become available at a later point, such that $\nexists i : \mathcal{TTF}[i] \leq D$.

C Removes the constraint from (B) such that availability of resources after x is greater than those originally required and is sufficient for a new forecast under the new constraints of $j_{t=x}$. As in (B) there will be a point x such that the forecast time-to-finish indicates that the deadline will be missed:

$$\exists x, 0 < x < D : \omega_{t=x} < u_{p=\frac{x}{D}} \implies \mathcal{TTF}_{t=x}[j_{t=x}] > D$$

However, in this case there will be a point y such that the available resources are more than anticipated and sufficient for the deadline to be met:

$$\begin{aligned}\exists y, x < y < D : \omega_{t_y} > u_{p=\frac{x}{D}} \\ \bigwedge \frac{F_{p=\frac{y}{D}}[j_{t=y}]}{j_{t=y}} \leq D \\ \bigwedge \textstyle\sum_{t=y+f}^{D} j_t \geq F_{p=\frac{y+f}{D}}[j_{t=y}] \\ \implies \mathcal{TTF}_{t=y}[j_{t=y}] \leq D \\ \implies \mathbf{RTT}[A^{\Sigma}] \leq D\end{aligned}$$

Increasing is a specific case of the dynamic case (B) where interference is increasing such that $\forall x, j_{t+x} < j_t$. This itself has two cases where (A) the deadline may still be met because the increase is not sufficient to delay the μS or (B) the deadline is missed.

A If the resource availability is continuously increasing such that the deadline can still be met, the following must apply:

$$\mathbf{RTT}[j_{t=0}] < D \bigwedge \mathbf{RTT}[A^{\Sigma}] \leq D$$

B Where the increase results in the deadline being missed the framework will identify at time x, in the same manner as previously, that the predicted time-to-finish is greater than the deadline.

Furthermore there will be a point $x \leq y < D$ such that there is no configuration i for which the deadline could be met.

Decreasing is a basic case such that $\forall x, j_{t+x} > j_t$. This implies that the $\forall x, \mathcal{TTF}_{t+x} < \mathcal{TTF}_{t+x-f}$ and that the final observed response-time will be less than deadline: $\mathbf{RTT}[A^{\Sigma}] < D$.

Note however, that in all scenarios, but this one in particular, there is never a situation where meeting the deadline can be guaranteed without knowledge of the interference to come: $\exists i : \mathcal{TTF}_t[i] > D$.

The framework has so far been been theoretically applied to various scenarios of resource interference resulting in the periodic updating of the predicted time-to-finish and where appropriate identifying the potential missing of a deadline as well as critically identifying where the deadline can no longer be met regardless of any future resource availability.

4.4 Summary

This chapter has presented and analysed the mathematical framework for modelling μS response-times based on resource utilisation and subsequent availability on specific hosts. The methods for estimating the execution progress and time-to-finish have also been presented both mathematically and algorithmically. The algorithms presented have been shown to be computable in constant time as a function of the number of resource types r , the fidelity of modelling those resources d_r , and the fidelity by which execution progress is measured k : $\mathcal{O}(|d_r|^r \cdot k)$. In terms of storage space the framework scales linearly with respect to the number of μSs and the number of execution instances: $\mathcal{O}(s \cdot n)$.

The next chapter will apply the algorithms presented here in the simulations configured in Chapter 3.

Chapter 5

Simulation Evaluation of RT-SOA QoS

This chapter builds on the simulation used in Chapter 3 to evaluate the existing QoS approaches, and the mathematics and algorithms from the Chapter 4 with further more extensive simulations analysing the performance of the proposed framework for $n-$dimensional prediction of RT-QoS.

First an overview of the simulation design is presented, building on Chapter 3. Then the simulation results are presented in four stages:

1. In Section 5.2 an overview of the actual observed response-times compared with the allocated QoS is presented.

2. Section 5.3 focusses on the RT-QoS violations, both their frequency using the AVC and the amount by which the deadlines are violated with the MPV.

3. Section 5.4 considers the wasted computational and resource time due to overallocation of time for service execution measured using the MPW.

4. And Section 5.5 combines the measure of overallocation and RT-QoS violation to consider the trade-off between them.

Finally a summary of the results of the framework is presented in Section 5.6. Each stage of the analysis considers the effectiveness of the proposed framework against the measures described in Chapter 3.

5.1 Overview & Simulation Design

In Chapter 3 Section 3.3 a detailed description of the simulation of QoS approaches was presented. In this chapter the same simulation methods, using the same configuration for servers, virtual machines, workload interference, and μSs is used to evaluate the proposed algorithmic framework from Chapter 4. This includes using the same simulations models that are detailed in Appendix C. In this section a summary of the simulation configuration is presented with respect specifically to the workload patterns, μS types, and measures and metrics being used to evaluate the methods as described in Figure 3.8.

5.1.1 Workload Patterns and Micro-Service Types

In order to simulate the interference experienced by a given μS a set of Cloud workload patterns are used (see Page 52). Specifically in this chapter the focus is on periodic and continuously changing or random workloads which may result in inadequate resource availability (whereas in Chapter 3 only a periodic workload was evaluated). The simulated interference models assume that the interfering workload has a mean resource utilisation which is classified as either high, medium, or low (80%, 90%, and 95% as described on Page 72). Although physical servers could often have workloads significantly lower than those being used in these simulations, the workload must be high enough to interfere in an observable fashion.

The evaluations in the following sections focus on the performance of the approaches under the influence of each of each these workload types.

The μS models are the same as those adopted in Chapter 3 imitating Cloud task with respect to their expected CPU and memory utilisation as well as their duration. The Cloud task types are extended with resource patterns to provide a set of eight μSs.

5.1.2 Measures and Metrics

In order to evaluate the effectiveness of the QoS approaches the metrics from Chapter 3 will again be adopted here:

- **QoS violation** as defined by Eqn. 3.6 with respect to frequency and degree:
  - Absolute Violation Count (AVC) measuring the cumulative total number of QoS violations for a given μS.
  - Mean Absolute Violation (MAV) measuring the level of violation in terms of seconds between the predicted QoS and the observed response-time.
  - Mean Percentage Violation (MPV) measuring the violation as a percentage of the response-time.
- **Prediction accuracy** as defined in Eqn. 3.5 in terms of error and overallocation:
  - Mean Absolute Error (MAE) measuring the error in terms of seconds between the predicted QoS and the observed response-time. For each QoS violation the MAE is equivalent to the MAV.
  - Mean Percentage Error (MPE) showing the error as a percentage of the response-time.
  - Mean Percentage Waste (MPW) measuring the overallocation as a percentage of the response-time.

The next section will explore the prediction accuracy in terms of absolute error and the following sections will in turn evaluate the effectiveness of the approach according to QoS violations and overallocation.

5.2 Observed Response-Time vs. Predicted RT-QoS

As described previously, the accuracy of the predictions with respect to the actual observed response-time can be measured using the Mean Absolute Error (MAE) shown in Eqn. 3.5. In this section therefore the accuracy of the μS execution instances are analysed under periodic, unpredictable, and continuously increasing workloads.

5.2.1 Interference: Periodic

Considering first the periodic workload the observed response-times and the corresponding predicted RT-QoS is shown in Figure 5.1. The shorter μSs have an average response-time of 14.6s with a standard deviation of 1.9s whilst the longer μSs average response-time is 33.4s with a

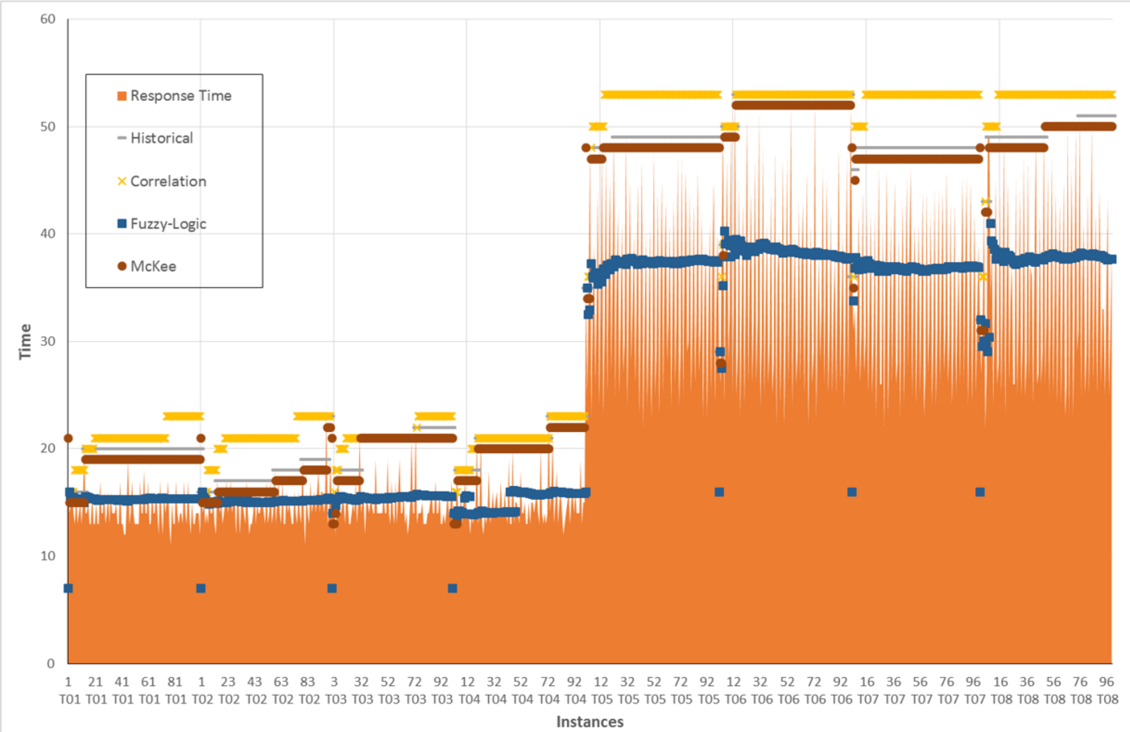


Figure 5.1: Response-Time vs. worst-case QoS with periodic interference

ID	RTT	Historical	Correlation	Real-Time	Fuzzy-Logic	McKee
T01	14.3	39%	49%	2339%	13%	32%
T02	14.7	23%	46%	2535%	11%	17%
T03	14.7	41%	47%	1937%	12%	39%
T04	14.8	44%	45%	2577%	12%	38%
Average 1-4		**37%**	**47%**	**2347%**	**12%**	**32%**
T05	33.9	52%	64%	14494%	27%	49%
T06	34.5	62%	62%	14621%	30%	60%
T07	33.5	50%	66%	14936%	26%	48%
T08	34.0	54%	63%	14209%	29%	52%
Average 5-8		**54%**	**64%**	**14565%**	**28%**	**52%**
Average		**46%**	**55%**	**8456%**	**20%**	**42%**

Table 5.1: Average response-times and QoS MPE by μS type and QoS approach with periodic interference

ID	RTT	Historical	Correlation	Real-Time	Fuzzy-Logic	McKee
T01	15.6	31%	51%	653%	8%	26%
T02	15.6	51%	52%	481%	8%	46%
T03	15.4	28%	52%	603%	9%	22%
T04	15.4	27%	53%	686%	9%	22%
Average 1-4		**34%**	**52%**	**606%**	**9%**	**29%**
T05	15.4	43%	59%	12967%	23%	41%
T06	36.9	57%	58%	11806%	25%	55%
T07	36.4	46%	58%	12385%	23%	44%
T08	36.5	53%	59%	12610%	23%	51%
Average 5-8		**50%**	**58%**	**12442%**	**23%**	**48%**
Average		**42%**	**55%**	**6524%**	**16%**	**38%**

Table 5.2: Average response-times and QoS prediction MPE by μS type and QoS approach with continuously changing interference

standard deviation of 8.4s. As should be anticipated the longer processes have a greater exposure to interference which increases non-linearly. Shown in the graph are the worst-case RT-QoS predictions for the historical, correlation based by Zheng et al. [147], fuzzy-logic by Benbernou et al. [111], as well as the proposed approach. The predictions using the real-time iLand method by García-valls et al. [142] are not shown on the graph as they overcompensate on average by 8456%. Most notably the fuzzy-logic approach predicts a lower response-time than either of the other approaches.

The summarising results are shown in Table 5.1 where the prediction accuracy for short μSs is 36% better than for the longer μSs across all the approaches (83% if the iLand predictions are included). The fuzzy-logic approach demonstrates a more accurate prediction with a 20% MPE. The proposed approach comes in 2^{nd} place with a 42% error which is an improvement on both the historical and correlation based methods.

5.2.2 Interference: Continuously Changing

In the context of a randomly/continuously changing workload the response-times may vary from periodic interference. Table 5.2 shows the average response-times for the μSs under the random workload as well as the MPE for each approach. As with the periodic workload the MPE of the fuzzy-logic approach by Benbernou et al. [111] is the least at 16% and the proposed approach comes in 2^{nd} place at 38% outperforming each of the other techniques.

5.2.3 Summary

In summary this section has looked at the QoS accuracy of the proposed approach in terms of the Mean Percentage Error (MPE) under both a periodic and continuously changing workload. In both cases the most accurate method was the fuzzy-logic approach with an average of 18% MPE across all μS execution instances. The proposed technique produced an MPE of 40%, which is more an accurate than the historical, correlation, or real-time techniques with MPEs of 44%, 55%, and 7490% respectively.

The fuzzy-logic technique demonstrates greater prediction accuracy as it uses the average response-time observed for any given fuzzy symbol, whereas the proposed approach utilises the observed Worst-Case Execution Time (WCET). As will be shown in the subsequent sections the use of WCET facilitates the proposed method to reduce the number of RT-QoS violations by a significant degree compared to the fuzzy-logic technique.

5.3 RT-QoS Violations

In this section the level of RT-QoS violation by each of the approaches is discussed. This will focus firstly on the AVC followed by the MPV from Eqn. 3.6. As in the previous section the interference workload patterns will be first considered individually before the approaches are analysed with respect to their overall RT-QoS violations.

5.3.1 Interference: Periodic

Under the influence of a periodic interfering workload the AVC is shown in Figure 5.2. Clearly the fuzzy-logic (Benbernou) and iLand approaches perform significantly worse than the others with an average of 30 and 12 violations per μS type across all execution instances. This is compared to an average of 3.2 violations across the other approaches.

Figure 5.3 depicts the average AVC for short and medium length μSs for each approach as well as the overall average violation count. The correlation and historical based methods depict the least number of violations with 18 and 21 total violations respectively. The proposed methodology demonstrated a total of 38 violations such that 4.75% of μS instances resulted in a RT-QoS violation.

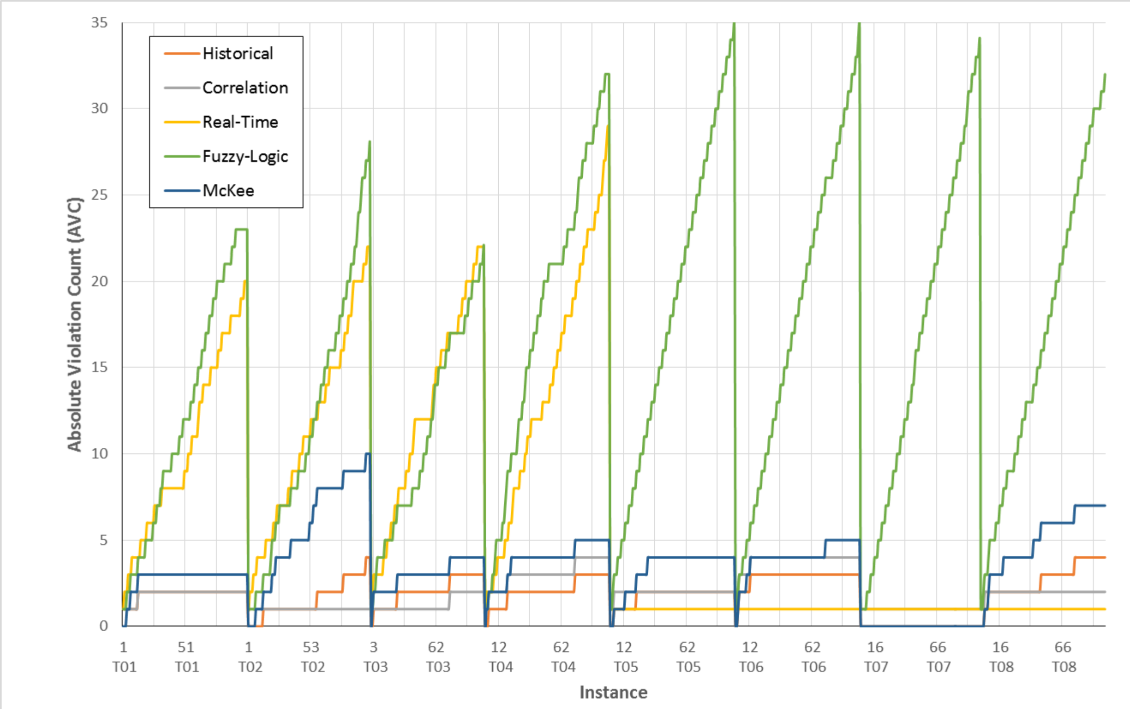
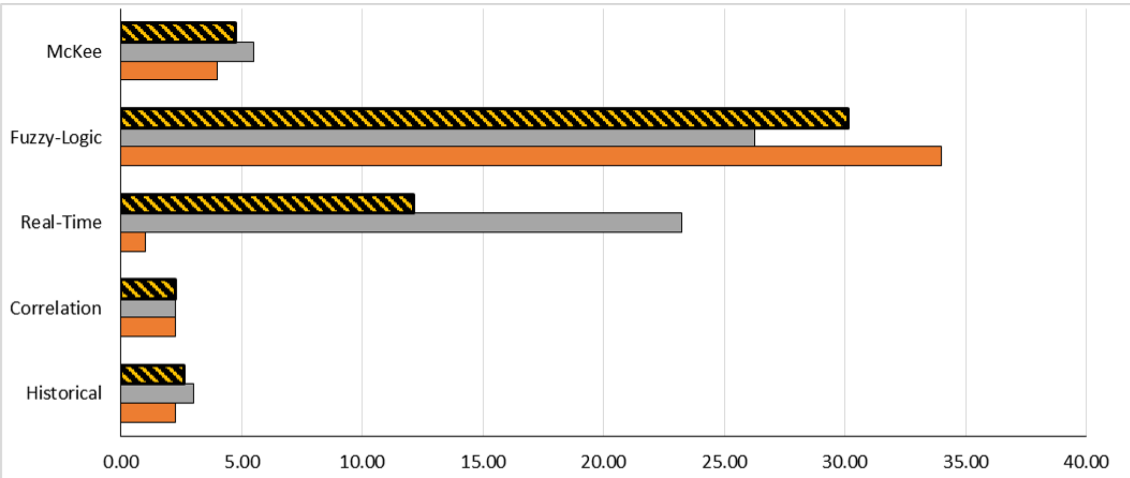


Figure 5.2: AVC across all execution instances with periodic workload



	Historical	Correlation	Real-Time	Fuzzy-Logic	McKee
Average	2.63	2.25	12.13	30.13	4.75
T1-4	3.00	2.25	23.25	26.25	5.50
T5-8	2.25	2.25	1.00	34.00	4.00

Figure 5.3: Average AVC for μS clusters by short and medium length

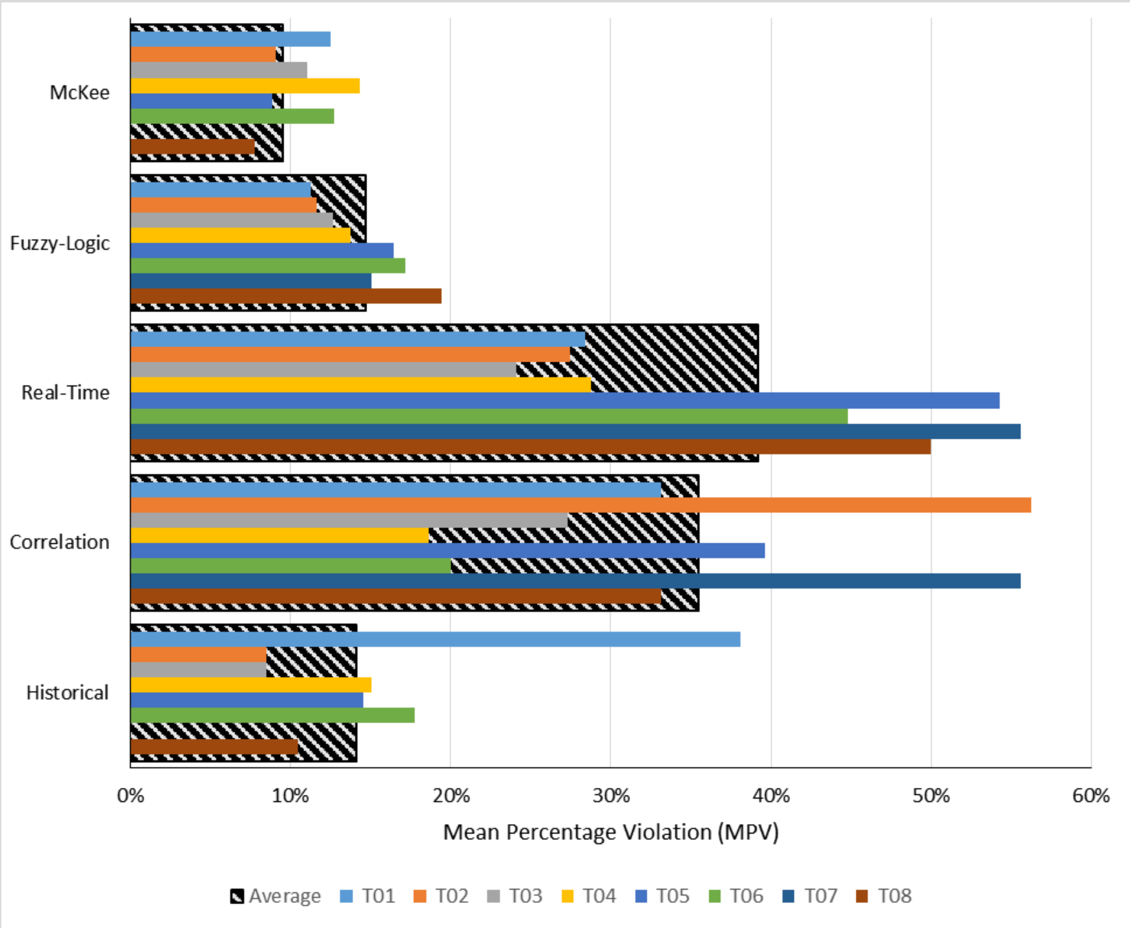


Figure 5.4: MPV across all approaches with periodic interference

Although the proposed method results in more violations than both the correlation and historical techniques the percentage by which the RT-QoS is violated is less. Figure 5.4 shows the MPV for each approach by μS and averaged across all μS types. One of the worst performing approaches is the correlation based method with an average violation of 35% compared to the proposed method with an average MPV of only 10%. The other approaches return greater MPVs with fuzzy-logic at 15%, iLand at 39%, and the historical method 14%.

5.3.2 Interference: Continuously Changing

RT-QoS violations for the various approaches under a random or continuously changing interfering workload are shown in Table 5.3a in terms of the MPV and Table 5.3b depicts the AVC. In the first instance the proposed and historical approaches both have the smallest MPV of 11% whilst the other approaches are between 12% and 46%. In terms of the violation count the fuzzy-logic

ID	RTT	Historical	Correlation	Real-Time	Fuzzy-Logic	McKee
T01	15.6	10%	53%	44%	7%	10%
T02	15.6	20%	29%	38%	8%	19%
T03	15.4	7%	31%	42%	9%	8%
T04	15.4	10%	31%	42%	8%	9%
Average 1-4		**12.1%**	**36.2%**	**41.6%**	**7.9%**	**11.6%**
T05	15.4	10%	38%	50%	17%	8%
T06	36.9	13%	23%	56%	17%	15%
T07	36.4	7%	38%	56%	17%	9%
T08	36.5	8%	28%	38%	16%	8%
Average 5-8		**9.7%**	**31.9%**	**49.8%**	**16.8%**	**10.1%**
Average		**10.9%**	**34.0%**	**45.7%**	**12.4%**	**10.8%**

(a) Mean Percentage Violation (MPV)

ID	RTT	Historical	Correlation	Real-Time	Fuzzy-Logic	McKee
T01	15.6	3	1	59	46	9
T02	15.6	2	3	57	44	3
T03	15.4	3	2	52	38	7
T04	15.4	2	2	50	41	4
Average 1-4		**2.5**	**2**	**54.5**	**42.25**	**5.75**
T05	15.4	4	2	1	37	7
T06	36.9	3	4	1	37	3
T07	36.4	5	2	1	36	5
T08	36.5	6	2	2	38	8
Average 5-8		**4.5**	**2.5**	**1.25**	**37**	**5.75**
Average		**3.5**	**2.25**	**27.875**	**39.625**	**5.75**

(b) Absolute Violation Count (AVC)

Table 5.3: QoS prediction by μS type and QoS approach with random interference

approach clearly under-performs with 40% of μS instances violating the QoS (there were 100 execution instances per μS). The proposed approach comes in 3^{rd} with 6% of instances violating the RT-QoS whilst the historical and correlation based methods come in with 5% and 3% respectively.

5.3.3 Summary

This section has discussed RT-QoS violations of each of the predictions by the various approaches, using the Absolute Violation Count (AVC) and Mean Percentage Violation (MPV). The correlation based method resulted in the least violations with only 2% of μS instances violating the RT-QoS. This is closely followed by the historical method and the proposed approach with 3% and 5% respectively. However the violations in the proposed approach are only 11% (MPV) compared to 13% and 35% respectively.

Further the fuzzy-logic technique, which in the previous section was the most accurate in terms of MPE, observes 35% (30% more than the proposed method) of execution instances violated the RT-QoS by an average of 14%. Also the traditional real-time approach by Garcia-Valls et al. [137] observed 20% of execution instances violating the QoS as it is not designed to handle the dynamic workload which the μSs experienced.

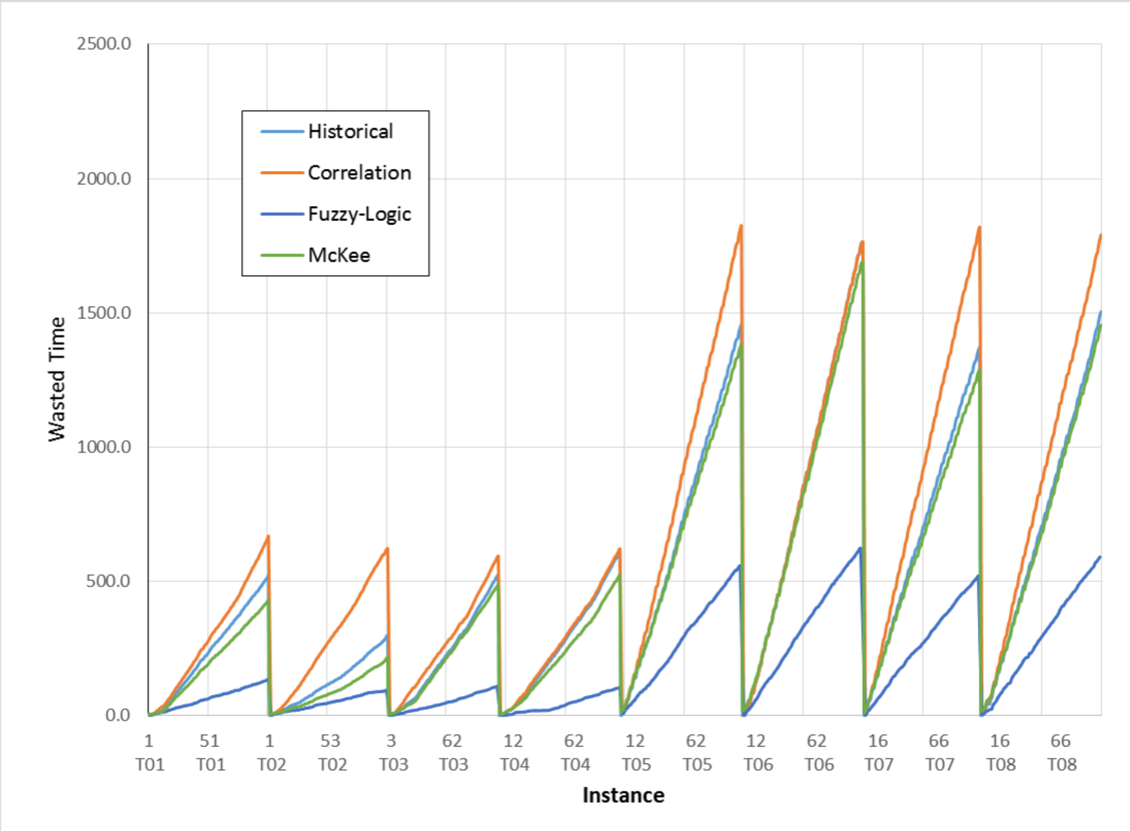
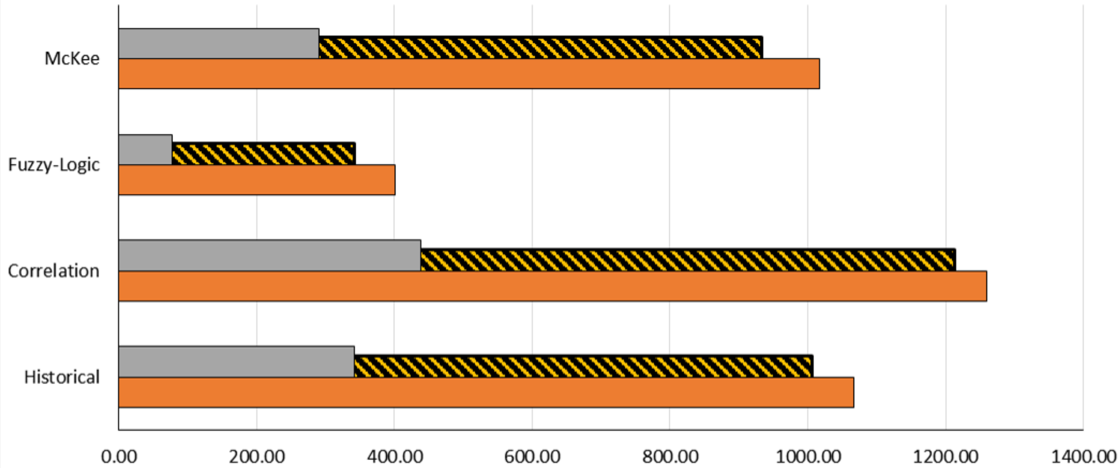


Figure 5.5: Cumulative wasted execution time due to overallocation

5.4 QoS Accuracy

In this section the wasted execution time due inaccurate predictions is evaluated. Particularly of interest are those situations where the predicted RT-QoS is not violated and prediction is overtly pessimistic. This results in allowing μSs significantly more execution time than actually required. In turn this means that a μS might not be selected for a given workflow or task as the QoS estimates that it will take too long to run. In this section the term *"Overallocation"* will be used to discuss prediction inaccuracy when the prediction is larger than the observed response-time.

This section will focus firstly on the cumulative wasted time over the execution instances followed by the Mean Percentage Waste (MPW) from Eqn. 3.5. The evaluation will first consider the results under a periodic workload before looking at continuously changing workloads.



	Historical	Correlation	Fuzzy-Logic	McKee
Average	1006.00	1212.63	342.00	933.50
T1-4	489.00	626.25	110.88	414.50
T5-8	1523.00	1799.00	573.12	1452.50

Figure 5.6: Average cumulative total wasted execution time for each μS cluster

ID	RTT	Historical	Correlation	Real-Time	Fuzzy-Logic	McKee
T01	14.3	31%	38%	3503%	12%	27%
T02	14.7	18%	37%	2702%	10%	14%
T03	14.7	30%	38%	1853%	10%	32%
T04	14.8	34%	37%	3124%	9%	32%
Average 1-4		**28.2%**	**37.2%**	**2795.5%**	**10.1%**	**26.2%**
T05	33.9	49%	59%	14258%	33%	50%
T06	34.5	57%	59%	13959%	35%	60%
T07	33.5	47%	62%	14755%	30%	46%
T08	34.0	48%	57%	13640%	31%	52%
Average 5-8		**50.4%**	**59.5%**	**14153.2%**	**32.2%**	**52.0%**
Average		**39.3%**	**48.4%**	**8474.3%**	**21.2%**	**39.1%**

Table 5.4: Mean Percentage Waste (MPW) across all μS execution instances with periodic interference

ID	RTT	Historical	Correlation	Real-Time	Fuzzy-Logic	McKee
T01	15.6	22%	39%	940%	8%	23%
T02	15.6	43%	45%	1077%	9%	41%
T03	15.4	23%	43%	862%	9%	20%
T04	15.4	21%	42%	946%	10%	19%
Average 1-4		**27.2%**	**42.1%**	**956.0%**	**8.8%**	**25.4%**
T05	15.4	42%	54%	12760%	26%	44%
T06	36.9	54%	56%	10578%	30%	53%
T07	36.4	44%	53%	12076%	27%	43%
T08	36.5	48%	50%	12358%	25%	49%
Average 5-8		**46.8%**	**53.3%**	**11942.8%**	**27.1%**	**47.1%**
Average		**37.0%**	**47.7%**	**6449.4%**	**17.9%**	**36.3%**

Table 5.5: QoS prediction MPW by μS type and QoS approach with random interference

5.4.1 Interference: Periodic

Considering first the performance under a periodic interfering workload. Figure 5.5 depicts the cumulative overallocation over the execution instances of each of the μSs. The overallocation by the iLand approach is not shown as it is over $100\times$ greater than the others, 248208 time units in comparison to 873 on average by the other approaches. This can also be seen in Figure 5.6 (except the real-time approach due to the scale) where the fuzzy-logic approach by Benbernou et al. [111] wastes the least time, followed by the proposed approach and then by the remaining approaches.

Table 5.4 details the wastage relative to the execution duration. The iLand method overallocates by nearly 8500% whilst each of the other approaches on average overallocate by less than 50%. The proposed method performs noticeably better than the correlation-based approach by Zibin Zheng et al. [146]. However it only marginally improves on the historical based method and wastes more time than the fuzzy-logic approach. For each of the approaches the level of overallocation increases with the length of the μS. The proposed method demonstrates the smallest increase with an increase factor of 2 compares with a factor 3 for the fuzzy-logic approach.

5.4.2 Interference: Continuously Changing

Under a random interfering workload the wasted execution time for each each QoS technique is presented in Table 5.5 in terms of MPW. The traditional real-time technique wastes on average 6449% of execution time relative to the observed response-time of the μSs. The fuzzy-logic technique by Benbernou et al. [111] wastes the least at only 18% whilst the proposed method wastes 36% an improvement over the historical, correlation based, and real-time methods.

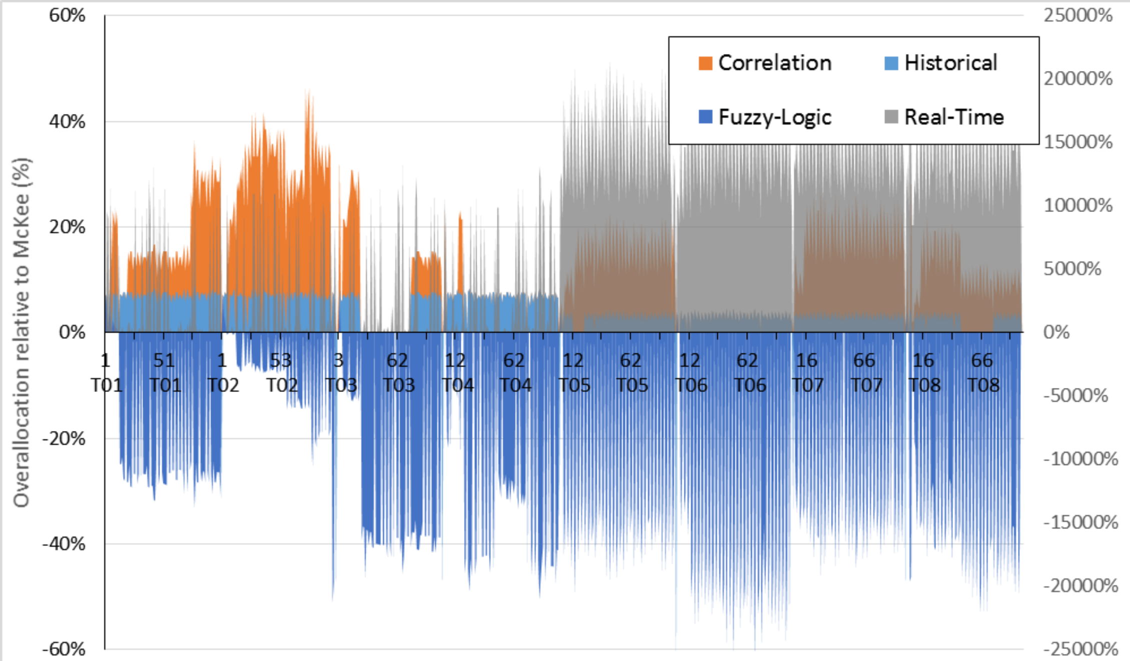


Figure 5.7: Percentage waste difference between existing QoS approaches and proposed method with respect to response-time. The real-time approach is shown on the secondary (right) axis with a different scale.

5.4.3 Summary

This section has explored the wasted execution time by each of the QoS approaches due to overallocation in terms of the MPW. The iLand technique by Garcia-Valls et al. [137] overallocates by an average of 7461% with respect to the actual execution times. The proposed technique wastes an average of 38% compared with 38% and 48% for the historical and correlation based methods respectively. The fuzzy-logic technique having a higher level of accuracy, as discussed in Section 5.2, wastes the least time with an average MPW of 20%.

5.5 Trade-off: RT-QoS Violation vs. Overallocation

The previous two sections have considered independently the RT-QoS violations and overallocation due to predictions by the various approaches. In this section the trade-off between QoS violation and overallocation is considered. As before the periodic workload will be considered first before the other workload patterns.

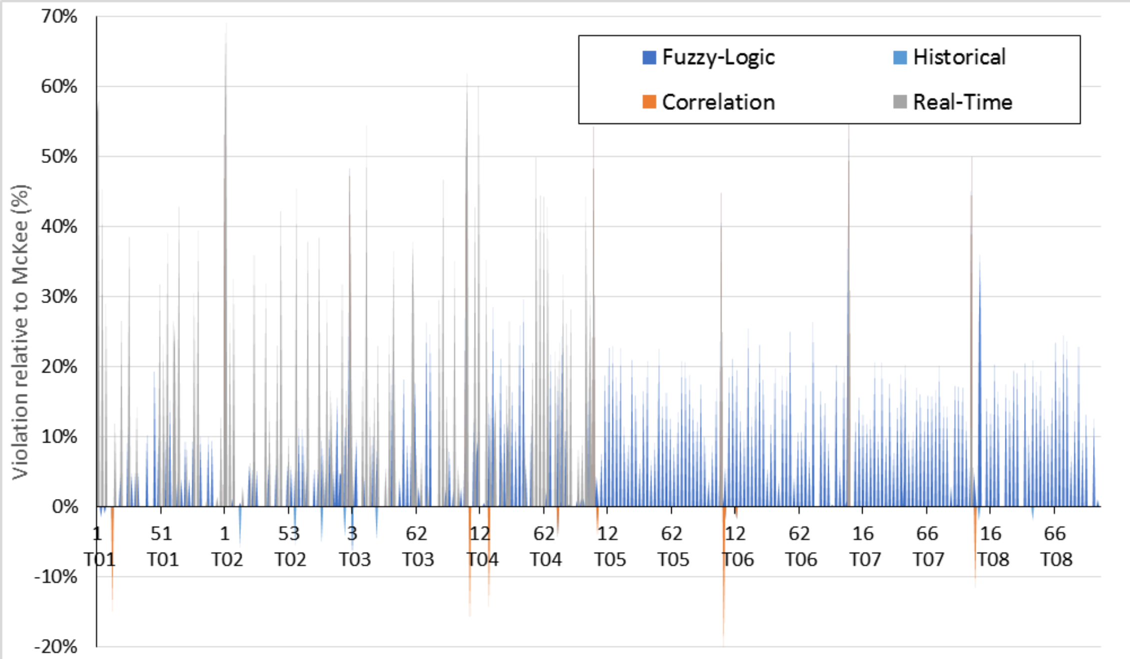


Figure 5.8: Percentage violation difference between existing QoS approaches and proposed method with respect to response-time

5.5.1 Interference: Periodic

Figure 5.7 and Figure 5.8 depict the overallocation of execution time and RT-QoS violation with respect to the observed response-time of the μSs. The figures show the difference between between the existing approaches and the proposed method. With respect to overallocation clearly the proposed technique performs better than all the approaches, other than the fuzzy-logic technique. This is due to the similarity between the approaches, with the proposed method using aspects of both the historical and correlation-based methods and introducing more accuracy with real-time utilisation information. Although the fuzzy-logic technique wastes less execution time, as shown in Figure 5.8 is results in significantly more violations and violations by a greater degree.

The Waste-Violation bar chart in Figure 5.9 and Table 5.6 detail the average percentage wastage and violation for each of the methods. The fuzzy-logic approach demonstrates an overallocation of 30% less than the proposed method, with respect to the observed response-times. The historical technique wastes 4% more, correlation 14% more, and the real-time iLand method 9667% more than the proposed approach. The real-time method performs the worst across both aspects with a MPV of 28% more than the proposed technique.

The trade-off between violation and wastage can be considered by combining the mean per-

		Historical	Correlation	Real-Time	Fuzzy-Logic
Waste	*Count*	657	718	604	-502
	Mean	3.98%	13.92%	9667.19%	-32.10%
	Std.Dev	0.055	0.100	70.310	0.142
	Var	0.003	0.010	4943.445	0.020
Violation	*Count*	-16	-1	97	232
	Mean	-0.51%	18.15%	27.97%	13.40%
	Std.Dev	0.128	0.310	0.171	0.105
	Var	0.017	0.096	0.029	0.011
MP Trade-off	*Hard-RT*	-0.5%	18.1%	28.0%	13.4%
	Firm-RT	1.0%	16.7%	3241.0%	-1.8%
	Soft-RT	2.5%	15.3%	6454.1%	-16.9%
	Not RT	4.0%	13.9%	9667.2%	-32.1%
Count Trade-off	*Hard-RT*	-16	-1	97	232
	Firm-RT	208	239	266	-13
	Soft-RT	433	478	435	-257
	Not RT	657	718	604	-502

Table 5.6: Difference in QoS violation and wasted execution time between existing approaches and proposed method: $Approach - Proposed$

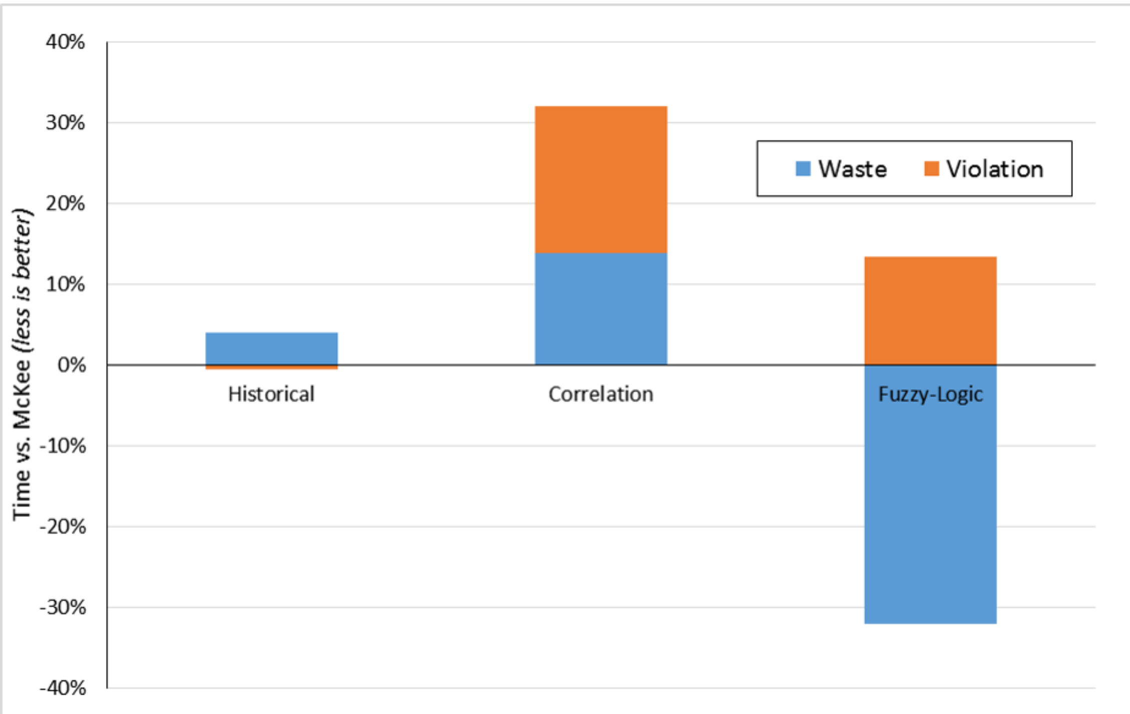


Figure 5.9: Aggregate difference between QoS approaches and proposed method for MPW and MPV with periodic interference. Real-time approach not shown due to axis scale.

		Historical	Correlation	Real-Time	Fuzzy-Logic
Waste	*Count*	727	770	501	-452
	Mean	4.10%	17.36%	8988.89%	-33.51%
	Std.Dev	0.048	0.125	73.489	0.149
	Var	0.002	0.016	5400.647	0.022
Violation	*Count*	-27	-2	219	297
	Mean	-3.14%	21.38%	41.33%	10.64%
	Std.Dev	0.017	0.293	0.186	0.103
	Var	0.000	0.086	0.034	0.011
MP Trade-off	*Hard-RT*	-3.1%	21.4%	41.3%	10.6%
	Firm-RT	-0.7%	20.0%	3023.8%	-4.1%
	Soft-RT	1.7%	18.7%	6006.4%	-18.8%
	Not RT	4.1%	17.4%	8988.9%	-33.5%
Count Trade-off	*Hard-RT*	-27	-2	219	297
	Firm-RT	224	255	313	47
	Soft-RT	476	513	407	-202
	Not RT	727	770	501	-452

Table 5.7: Difference in QoS violation and wasted execution time between existing approaches and proposed method: $Approach - Proposed$ with continuously changing interference

centages weighted according to the focus of the system. In the case of a hard real-time system the only consideration is RT-QoS violation and as shown in Table 5.6 the proposed approach outperforms each of the techniques by between 13 and 28%, other than the historical technique which is within 1% of the proposed technique. The firm and soft approaches assign $\frac{2}{3}$ and $\frac{1}{3}$ respectively to RT-QoS violation and the remaining to wastage. In both of these instances the proposed technique outperforms each of the existing approaches, other than fuzzy-logic.

The final section of the table depicts the trade-off with respect to the number of μS execution instances. In terms of hard real-time QoS compliance the proposed method outperforms the iLand and fuzzy-logic techniques with 97 and 232 less violations respectively. Overall the correlation-based technique by Zheng et al. has 1 less violation than the proposed approach and the historical method 16 less, which is equivalent to 2% of all μS execution instances.

5.5.2 Interference: Continuously Changing

As with the periodic interfering workloads the observed performance of the QoS techniques is a trade-off between violation and overallocation. Table 5.7 details this trade-off with respect to how each of the methods performs in comparison to the proposed method. As detailed in the previous

		Historical	Correlation	Real-Time	Fuzzy-Logic
Waste	*Count*	1384	1488	1105	-954
	Mean	4.04%	15.65%	9324.83%	-32.81%
Violat	*Count*	-43	-3	316	529
	Mean	-1.84%	19.78%	34.71%	12.01%
MP Trade-off	*Hard-RT*	-1.8%	19.8%	34.7%	12.0%
	Firm-RT	0.1%	18.4%	3131.4%	-2.9%
	Soft-RT	2.1%	17.0%	6228.1%	-17.9%
	Not RT	4.0%	15.7%	9324.8%	-32.8%
Count Trade-off	*Hard-RT*	-43	-3	316	529
	Firm-RT	433	494	579	35
	Soft-RT	908	991	842	-460
	Not RT	1384	1488	1105	-954

Table 5.8: Combined difference in QoS violation and waste between existing and proposed approaches

sections the proposed method wastes less execution time than the historical, correlation based, and real-time iLand techniques, but more than the fuzzy-logic method. In terms of RT-QoS violation the fuzzy-logic approach resulted in 297 more violations than the proposed method.

The trade-off as described in the previous section considers both the mean percentage violation and overallocation as well as the number of execution instances that are effected. From a hard real-time perspective the proposed technique outperforms each of the existing methods, excluding historical, by an average of 24% MPV. The historical technique demonstrates a 3% MPV better than the proposed method. Moving towards less strict real-time conditions, from firm through soft towards non-real-time systems, the benefit of the proposed approach against the historical, correlation based, and real-time techniques increases whilst the fuzzy-logic approach also improves.

5.5.3 Summary

This section has presented a trade-off of RT-QoS violations against overallocation of execution time by RT-QoS prediction methods in terms of the difference between them and the proposed method. Table 5.6 and Table 5.7 detailed the trade-off in terms of mean percentage and the number of μS execution instances affected. Under the influence of both periodic and continuously changing workloads the proposed method wasted less execution time than the historical, correlation based, and real-time methods, but more than the fuzzy-logic approach. The proposed method also resulted in more QoS violations compared to the historical and correlation based techniques.

The proposed method performs the best in comparison to the existing techniques under firm real-time conditions, wasting less execution time and violating RT-QoS by a reduced percentage across an average of 193 μS instances which is equivalent to 12% of all μS execution instances (see Table 5.8).

5.6 Summary

This chapter has presented simulation analysis of the proposed RT-QoS prediction approach, from the previous chapter. The simulation has evaluated the accuracy of the predictions under non-static interfering workloads, in terms of periodic and continuously changing cloud workloads. The performance of the proposed technique has been evaluated against the existing techniques, discussed in Chapter 3.

The Mean Percentage Error (MPE) was used to evaluate the accuracy of the predictions in Section 5.2. The proposed approach demonstrates an accuracy over the historical, correlation based [146], and the real-time [59] approaches with an MPE of 42% compared with 46%, 55%, and 8456% respectively. This accuracy was considered specifically in terms of overallocation or Mean Percentage Waste (MPW) in Section 5.4.

Although the fuzzy-logic approach by Benbernou et al. [111] was more accurate than the proposed technique as it used average response-times rather than the Worst-Case Execution Times (WCETs) (20% MPE) it terms of RT-QoS violation it was significantly worse with 30% more μS execution instances violating the RT-QoS. The RT-QoS violations observed by the proposed method were the least severe with Mean Percentage Violations (MPVs) of only 11% across 5% of execution instances (Section 5.3).

Finally in Section 5.5 the trade-off between overallocation and RT-QoS violation was considered under for systems with hard, firm, soft, and no real-time constraints. The proposed approach demonstrated the largest improvement against the existing techniques under firm real-time conditions, with an improvement against each of the techniques. The next chapter will explore the configurations of the proposed approach and validate it in a series of use cases.

Chapter 6

Experimental Validation of RT-SOA QoS

6.1 Overview

In this chapter the proposed framework from Chapter 4 which was evaluated using simulation in the last chapter is explored in the context of several real scenarios. The first case study explores its application within the Cloud domain, specifically considering data processing and workload scheduling challenges (Section 6.2.1). Then a generic Cloud is considered and the RT-QoS framework is proposed to sit within the generic architecture as part of the resource abstraction layer. Using the Function-as-a-Service (FaaS) paradigm real experiments are then conducted in Section 6.2.3.

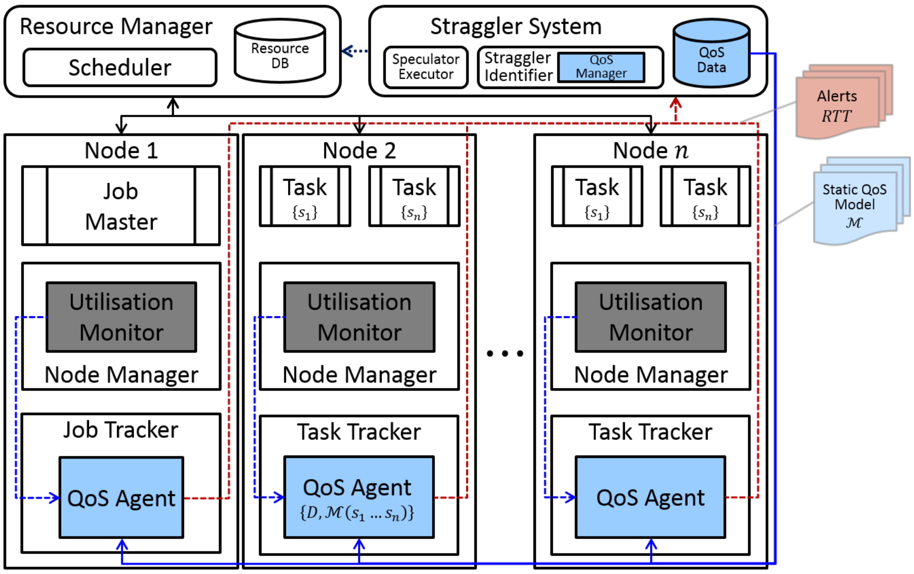


Figure 6.1: Generic MapReduce deployment architecture with straggler and QoS management

6.2 Case Study: Cloud

The domain of Cloud computing is one of the primary locations for which general QoS approaches have been designed. SLAs are used to define the expected performance of cloud hosted services. This section briefly looks at a series of applications for predictive QoS approaches in the cloud domain before presenting in Section 6.2.2 a generic Cloud-based system architecture for the proposed RT-QoS framework.

6.2.1 Cloud Applications

Three of the major uses for Cloud computing are data processing of large, or big, data; scheduling of workloads across Cloud server infrastructure; and managing computation on systems that are connected to the cloud but located on the Edge or are Internet of Things (IoT) elements.

Data Processing

In terms of data processing in the cloud this primarily refers to the use of techniques such as MapReduce originally developed by Google and part of Hadoop [126]. This technology was designed to split a *job* into a set of smaller *tasks*, or μSs, each of which operated in parallel on different segments of the data input. In relation to QoS one of the major challenges with

MapReduce is the Longtail problem [157] caused by one or more tasks running slower than the rest and therefore causing the entire job to be delayed. This problem is currently addressed using speculation techniques to predict a straggler task and identify host servers which have a degrading performance [173; 168].

Execution progress of a task, or μS, in MapReduce is modelled as one of three states: *idle*, *in-progress*, or *complete*. However, the framework proposed in this research could provide an estimation of the progress as a percentage accurate to $\frac{100}{k}$.

Figure 6.1 depicts the general deployment architecture of MapReduce systems, such as Hadoop. The proposed RT-QoS framework requires agents operating on each node monitoring the task execution performance and the resource utilisation by the task and on the node itself. Each agent receives a static QoS model $\mathcal{M}$ with the summary of the data, in terms of response-time and resource utilisation at each progress point p . The agents alert the RT-QoS manager when the predicted response-time $\mathbf{RTT}$ is greater than the deadline.

Workload Scheduling

Also in the context of Cloud computing, as well as more generally, scheduling theory as discussed on Page 45 does not traditionally allow for unknown workloads and utilisation patterns. Scheduling of tasks traditionally assumes block-shaped tasks that can be scheduled based on their height and length referring to their resource utilisation and execution time respectively. There is a need to bridge the gap between scheduling theory and system practice in a non-deterministic world. The work by Primas et al. [174] introduces the concept of resource-boxing as an offline analysis technique to convert resource utilisations by tasks into boxes that can then be scheduled using traditional theoretical methods.

The proposed RT-QoS framework in this research lends itself to this boxing technique as an online *boxing* mechanism by breaking tasks into k individual boxes. Additionally the proposed framework captures the concepts of *slowdown* and *stretch* due to resource contention making the boxing technique useful for using traditional scheduling algorithms in real-world domains with imperfect knowledge of the execution environment. Additionally in combination with the previous section the QoS agents would be used to alert the resource manager and scheduler directly at either a predefined frequency or at scheduling events.

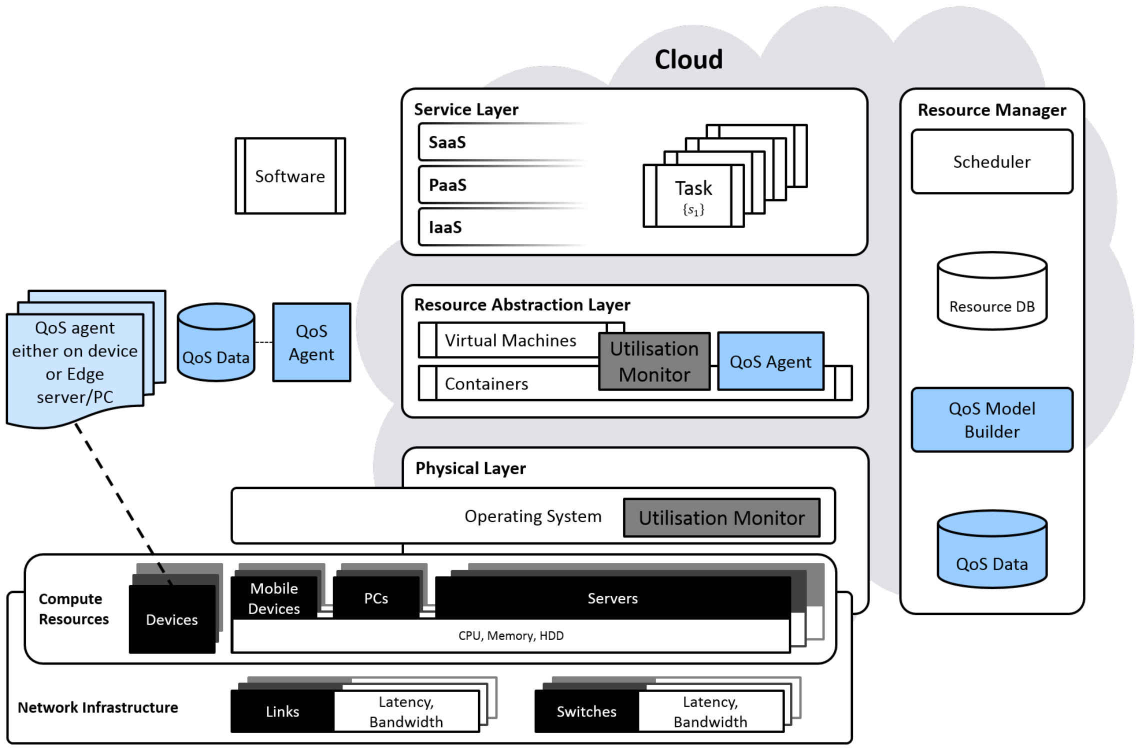


Figure 6.2: Generic Cloud architecture with QoS agents

6.2.2 Generic System Architecture for Cloud

At a slightly more abstract level, away from any individual application, the proposed RT-QoS framework can be introduced in general Cloud environments. This section summarises the traditional architecture of a Cloud system as defined by NIST [175] in the context of the deployment of the framework within that architecture, as shown in Figure 6.2.

Service layers

The uppermost layers within a Cloud computing architecture are the service layers comprising of Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS), and Infrastructure as a Service (IaaS). The software layer may comprise of both applications, functions, and data as services (FaaS, DaaS). Functions specifically refer to μSs but applications themselves may also be decomposed into μSs as discussed on Page 85. The proposed framework monitors the service layers and the individual model elements of the framework map to individual services within these layers.

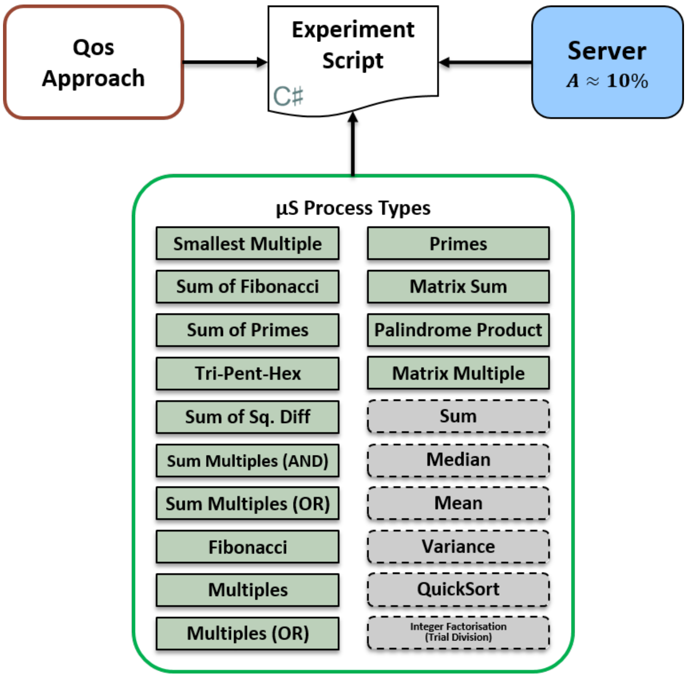


Figure 6.3: Outline of general μS QoS experiments with 20 μS types. (*14 Euler Problems and 6 general functions shown with rounded dashed boxes*)

Resource Abstraction

The resource abstraction layer provides the containers and virtual machines typically used to host the service layers within the Cloud. Here, as shown in Figure 6.2, QoS agents can be deployed along with utilisation monitors within each container or VM.

Physical Computation

Below the various levels of abstraction lies the underlying operating systems which must also provide utilisation monitors for the RT-QoS framework. Beyond the Cloud environment any software may also be monitored using the framework with local QoS agents. Further afield towards the domain of IoT with devices the QoS can either be monitored with a local agent on the device, or remotely over the network. Although the latter may result in the model having less resource dimensions or less fidelity due to remote access permissions but would account for network latencies.

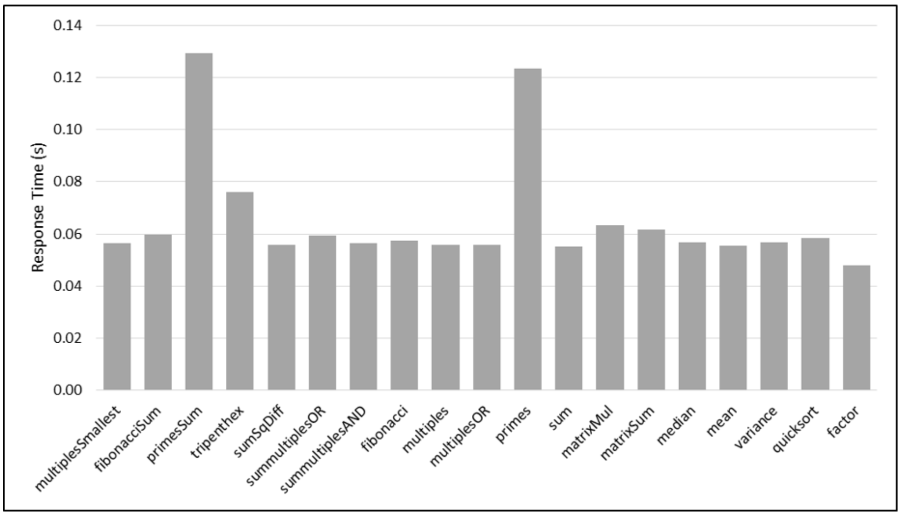
6.2.3 Experimental Results

The remainder of this section focusses on evaluating the RT-QoS framework with functions as μSs. Specifically the following set of μSs (shown in Figure 6.3), most of which are solution to problems from from *Project Euler*[a], are used and the corresponding code listings can be found in Appendix D:

1. **Smallest Multiple** *"What is the smallest positive number that is evenly divisible by all of the numbers from 1 to 20?"*
2. **Sum of Even Fibonacci** *"By considering the terms in the Fibonacci sequence whose values do not exceed four million, find the sum of the even-valued terms."*
3. **Sum of Primes** *"Find the sum of all the primes below two million."*
4. **Tri-Pent-Hex** *"Find the next triangle number that is also pentagonal and hexagonal."*
5. **Sum of Squared Differences** *"Find the difference between the sum of the squares of the first one hundred natural numbers and the square of the sum."*
6. **Sum of Both Multiples** *"Find the sum of all the multiples of 3 and 5 below 1000."*
7. **Sum of Multiples** *"Find the sum of all the multiples of 3 or 5 below 1000."*
8. **Fibonacci Sequence** *"Calculate all the Fibonacci below 4 million."*
9. **Multiples of Both** *"Find all the multiples of both 3 and 5 below 1000."*
10. **Multiples of Either** *"Find all the multiples of either 3 or 5 below 1000."*
11. **Primes** *"Find all the prime numbers below 10 thousand."*
12. **Matrix Sum** *"Find the maximum sum of matrix elements with each element being the only one in his row and column."*
13. **Palindrome Product** *"Find the largest palindrome made from the product of two numbers."*
14. **Matrix Multiple** *"Calculate the multiple of two matrices."*

In addition to the following μSs:

[a] https://projecteuler.net/



Name	RTT
multiplesSmallest	0.06
fibonacciSum	0.06
primesSum	0.13
tripenthex	0.08
sumSqDiff	0.06
summultiplesOR	0.06
summultiplesAND	0.06
fibonacci	0.06
multiples	0.06
multiplesOR	0.06
primes	0.12
sum	0.05
palindrome*	15.08
matrixMul	0.06
matrixSum	0.06
median	0.06
mean	0.06
variance	0.06
quicksort	0.06
factor	0.05
Average	**0.82**

Figure 6.4: Average response-times of μS instances. Palindrome μS not shown as it's an order of magnitude slower.

1. **Sum** Calculates the sum of the provided set of numbers.

2. **Median** Finds the median value of the provided set of numbers.

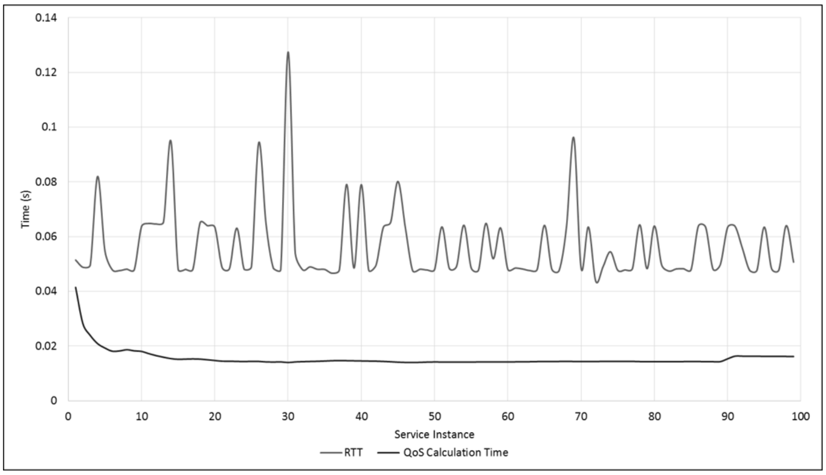
3. **Mean** Calculates the mean value of the provided set of numbers.

4. **Variance** Calculates the variance across the provided set of numbers.

5. **QuickSort** Sorts the provided set of values using the Quick Sort algorithm.

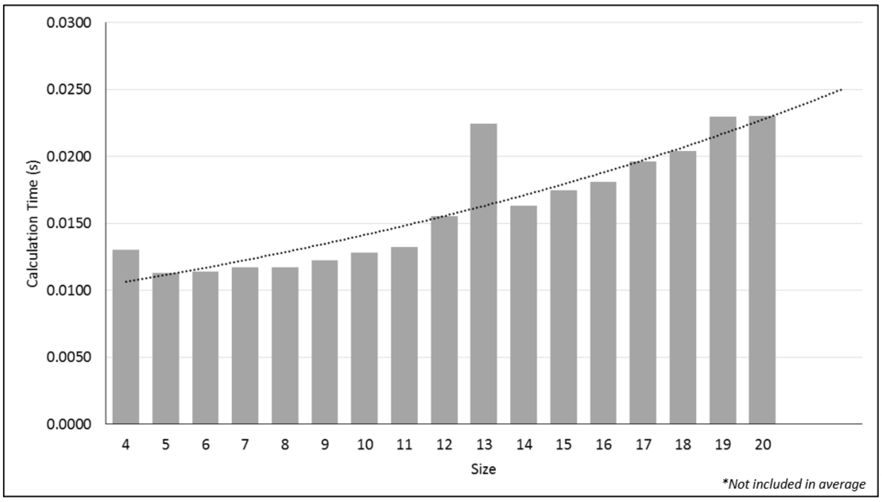
6. **Integer Factorisation** Finds the integer factors of the supplied digit using the trial division method.

Across 100 instances of each μS the average response-time, under an average interfering workload of 90%, was 0.8s as shown in Figure 6.4. If the *palindrome* μS is ignored as it is significantly slower, the average response-time is 65ms with an average standard deviation across execution instances of 26ms. This represents a standard deviation of 40% compared with the 20% observed in the previous chapter's simulations. Looking specifically at *Palindromes*, *Prime Sum*, and *Primes* μSs the standard deviation percentages are 1%, 20%, and 10% respectively providing no indicative pattern or obvious relationship between μS length and the variance observed in response-time due to interference.



Name	Average-Calculation Time (s)
multiplesSmallest	0.016
fibonacciSum	0.013
primesSum	0.009
tripenthex	0.010
sumSqDiff	0.014
summultiplesOR	0.015
summultiplesAND	0.015
fibonacci	0.015
multiples	0.015
multiplesOR	0.014
primes	0.010
sum	0.014
palindrome*	0.006
matrixMul	0.012
matrixSum	0.014
median	0.013
mean	0.014
variance	0.013
quicksort	0.013
factor	0.014
Average	**0.013**

Figure 6.5: QoS calculation time of the new **QoS!** (**QoS!**) framework for the *Smallest Multiple* μS instances and average across all μS types.



Name	Average # Observations
multiplesSmallest	3.27
fibonacciSum	4.06
primesSum	11.30
tripenthex	6.12
sumSqDiff	3.62
summultiplesOR	3.65
summultiplesAND	3.46
fibonacci	3.60
multiples	3.32
multiplesOR	3.49
primes	9.36
sum	3.53
palindrome*	1358.84
matrixMul	4.36
matrixSum	3.90
median	3.74
mean	3.61
variance	3.67
quicksort	3.96
factor	3.19
Average	**4.484**

Figure 6.6: Increasing QoS calculation time of the new framework as resource fidelity is increased and average number of observations per execution instance.

Performance

As outlined on Page 86 the QoS is monitored with a given frequency f to get a minimum number of k observations. In order to do so the execution time of the QoS calculation must be sufficiently small and scale appropriately with execution instances. As shown in Figure 6.5 the calculation time remains relatively constant across all the μS types at 13ms. The graph shows the nature of the calculation time over 100 execution instances of the *Smallest Multiple* μS with a model initialisation phase with the calculations taking significantly longer during the first 10 instances.

The calculation time is exponentially dependent on the size of the resource dimensions d_r and number of resource dimensions $|\mathcal{R}|$, Figure 6.6 therefore shows the increase in fidelity, i.e. the increase in $|d_r|$, and the corresponding increase in calculation time from 13ms up to 23ms as $|d_r|$ is increased from 4 to 20. As can be seen in Figure 6.6 the QoS calculation time begins to rise rapidly beyond $|d_r| = 9$ and as is outlined in Table 6.1 this configuration not only identifies the most violations but also identifies them earlier than most other configurations. In this particular case $|\mathcal{R}| = 2$ capturing CPU and memory but other dimensions could include required network bandwidth, storage and other I/O.

As $|d_r|$ was increased, the number of QoS observations decreased from an average of 5 per μS instance to 3 (ignoring the Palindrome μS). This is in line with the algorithmic complexity discussed in Chapter 4 on Page 94. As long as the approach provides at least 3 observations per μS execution it is an improvement on the methods used in monitoring Map Reduce tasks. Wider afield finding an ideal target number of observations (i.e. target k) would be domain dependent and could factor in the process resource life-cycle discussed on Page 30.

These performance results correspond with the algorithmic complexity of $\mathcal{O}(|d_r|^{2r})$ detailed on Page 97. In comparison to the techniques outlined in Chapter 3 the performance of the proposed framework is runs in relative constant time compared to those previous approaches which slowdown linearly or quadratically with the number of services and service execution instances.

Accuracy

The accuracy of the predictions as measured in the previous chapters using the Mean Percentage Error (MPE), Mean Percentage Violation (MPV), and Mean Percentage Waste (MPW) metrics. Figure 6.7 depicts these across all the μS types and their execution instances. The total error can be split into the two categories for overallocation and QoS violation which are loosely logarithmic

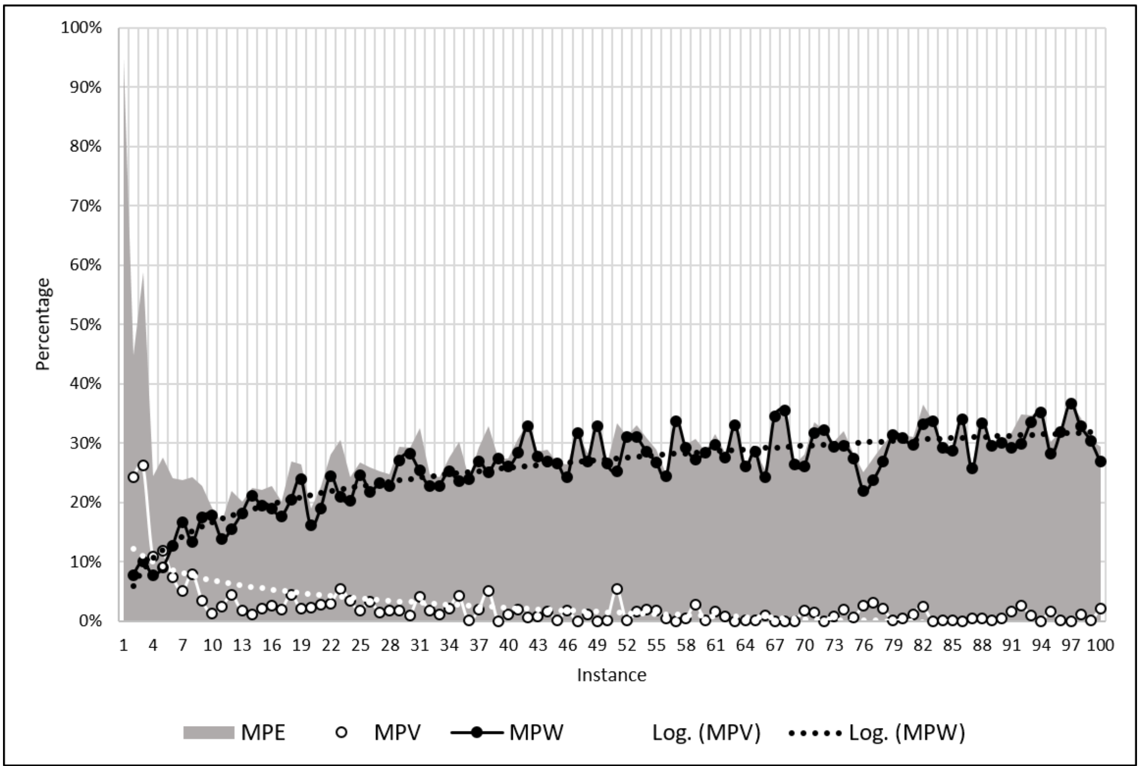


Figure 6.7: MPE, MPV, and MPW across μS instances

tending towards an average overallocation of below 35% and a near zero violation percentage of 2% which represents missing the deadline by just over 1ms. The AVC reduces logarithmically from an average across all μS types of 0.4 violations during the first 10 instances to 0.2 for the next 10 instances and down to an average of 0.1 violations by instance 100.

The MPE across all the μS types was 30% with a standard deviation of 12%. The corresponding MPV was 17% (compared to the 11% demonstrated by the simulations in the previous chapter) and the MPW was 30%. If outlying μS response-times are ignored, as indicated using either Grubbs or the Generalized ESD Test [176], the average wasted allocated time is 56ms (single outlier) or 55ms (2 or 3 outliers) equating to between 43% and 47% of allocated time.

As the size of d_r is increased there is marginal increase observed in the MPW as can be seen in Figure 6.8. The affect of the increasing the fidelity of the model will however be further explored in the rest of this section.

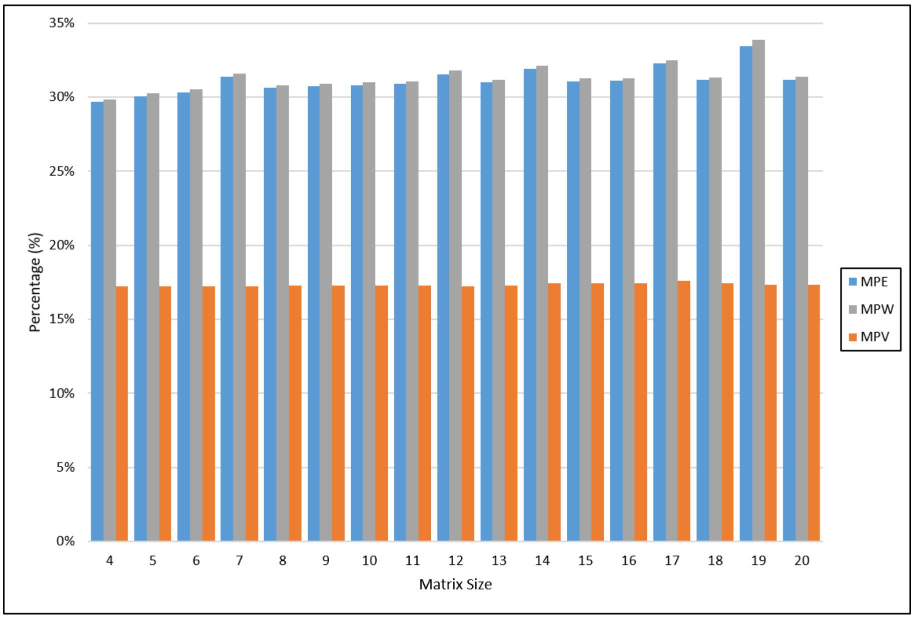


Figure 6.8: Prediction accuracy against framework matrix size defined by $|d_r| = 4 \rightarrow 20$

Alerts

The increased fidelity, achieved through increasing the size of d_r, allows the framework to provide more alerts as directed by Algorithm 8. Table 6.1 details for each matrix size correlating to d_r the average alert time as a percentage of execution progress across all μS instances and the coverage as a percentage of violations covered by those alerts increasing from 71% up to between 87% and 94%. The best performance was observed where $|d_r| = 9$ such that each bin accounts for approximately 11% of resource availability. This configuration demonstrated the best coverage, at 94%, and some of the earliest alert times, on average within the first 8ms of execution (or 2s for the palindrome μS). The choice of $|d_r|$ will depend on the level of granularity that is possible to be monitored, and should trade-off the speed of the prediction against the required accuracy. From an accuracy perspective the selection of $|d_r|$ will be an optimisation for which there could be several local optima.

WCET vs. Average RTT

The approach so far has used the pessimistic or worst-case analysis provided by the framework. As directed by Eqn. 4.18 the framework also provides estimation based on the average response-times. The corresponding MPE is up 6% to 36% and the MPV is up 4.7% to 22%. That increase in violations is seen alongside an increase in the AVC to 89% of instances. Across those instances

Matrix Size	4	5	6	7	8	9	10	11	12
Alert Time	14.0%	14.0%	14.6%	20.4%	15.1%	13.7%	14.7%	14.0%	14.0%
Coverage	71.4%	71.4%	80.0%	86.7%	91.7%	93.5%	86.6%	85.8%	85.8%

Matrix Size	13	14	15	16	17	18	19	20
Alert Time	14.0%	13.5%	14.0%	13.5%	13.6%	13.9%	14.3%	14.3%
Coverage	85.9%	86.7%	85.8%	86.9%	87.2%	85.5%	87.7%	87.6%

Table 6.1: Alert warning time and coverage of violations which represents the percentage of violations that were identified before they occurred.

which didn't violate QoS the MPW was 58%.

The measures for violation and overallocation do not capture the overall ratio of QoS violation to overallocation. Table 6.2 therefore shows weighted combinations across all the μS types and execution instances. The WCET approach demonstrates an average overallocation of 12% whilst the average approach demonstrates an average violation of 8%.

Depending on the nature of the application it may be appropriate to use either of the approaches provided by the framework or a combination of both. For non-real-time or soft real-time system the average method is likely to be sufficient. This would result in a small percentage of deadlines being missed by 23%, equivalent to 15ms, but making better use of the available computational resources. In the context of hard deadlines the WCET approach must be used and depending on the criticality of the application an additional margin or error could be included such as an additional 50% of the estimated length of the process.

6.2.4 Cloud Summary

This section has looked specifically at the application of the proposed framework from this research to predicting the QoS of μSs in the Cloud. A set of 20 μSs were used that each provided a different mathematical function. Given these and an average interfering workload of 90% the experiments in this section looked at the execution performance of the framework itself followed by its accuracy.

As outlined on page Page 133 the QoS calculation time must be sufficiently small relative to the response-time of the μS. In the examples shown the calculation time averages between 13ms and 23ms when modelling the resource utilisation in blocks of between 25% and 5%. Therefore to have 2 or more observation points during the μS execution the response-times would have to be above 26ms in the first instance or 46ms in the more detailed case in order to provide an alert

MicroService	Av	WCET	|Diff|
multiplesSmallest	-14.3%	18.5%	**4.2%**
fibonacciSum	-9.8%	11.6%	**1.8%**
primesSum	-4.9%	16.5%	**11.6%**
tripenthex	-8.4%	10.6%	**2.2%**
sumSqDiff	-11.8%	16.9%	**5.0%**
summultiplesOR	-12.3%	6.9%	**-5.4%**
summultiplesAND	-11.4%	23.9%	**12.5%**
fibonacci	-11.6%	7.3%	**-4.3%**
multiples	-13.0%	8.1%	**-4.8%**
multiplesOR	-10.0%	5.9%	**-4.1%**
primes	-4.8%	6.1%	**1.3%**
sum	-11.5%	12.6%	**1.0%**
palindrome	31.2%	31.2%	**0.0%**
matrixMul	-8.5%	12.8%	**4.3%**
matrixSum	-11.7%	5.8%	**-5.9%**
median	-11.3%	9.6%	**-1.6%**
mean	-11.7%	6.9%	**-4.7%**
variance	-9.8%	11.9%	**2.1%**
quicksort	-6.1%	15.7%	**9.5%**
factor	-11.9%	2.7%	**-9.2%**
Average	**-8.2%**	**12.1%**	**0.8%**

Table 6.2: Comparison of accuracy (as a weighted average between MPV and MPW) between using WCET and average response-times

during execution. The framework provided alerts across 94% of instances, when configured with $|d_r| = 9$, which resulted in violations with an alert being fired on average within the first 14% of the execution time.

The framework has been evaluated in terms of the Mean Percentage Violation (MPV) and overallocation (MPW). In the first instance QoS was violated across 14% of instances with an average violation of 17%. The remaining execution instances were overallocated by an average of 30%. Finally the analysis looked at the use of the average and Worst-Case Execution Times (WCETs) for QoS allocation. In the average case the framework underallocated by 8% whilst the worst-case approach overallocated by 12% on average across all instances. The results are representative of the simulation results in Chapter 5 where the MPW was 38%, compared with 30% in the experiments, and the MPV was 11%, compared against the 17% observed in this section.

6.3 Summary

This chapter has presented a series of case studies applying the framework developed earlier in Chapter 4. Case studies are considered in the domain of Cloud computing. In the first two cases the software and systems architectures required to facilitate the RT-QoS framework in a non-intrusive manner, i.e. not requiring change to the underlying infrastructure, are presented. Also experimental results from the domain of Cloud computing are shown, demonstrating that the proposed RT-QoS framework can be practically applied in this domain.

The following and final chapter of this thesis provides a summary of the work that has been presented and outlines key areas that require further research.

Chapter 7

Conclusions and Future Work

In this chapter the work presented throughout this thesis is summarised. The major contributions of the research are outlined and an evaluation of the research in terms of the objectives from Chapter 1 is presented in Section 7.3. Then a discussion is presented of some of the future directions that can be explored as part of this work.

7.1 Summary

The work in this thesis is focussed on exploring Quality of Service (QoS) for Real-Time Service-Oriented Architectures (RT-SOAs). The research is centered on providing a mechanism to capture the relationship between computational resources and the execution performance of Micro-Services (μSs). The developed framework is used to predict the response-times of μSs executing in environments with interfering workloads. A comparison is also made against existing approaches to RT-QoS and the tradeoffs between the techniques are explored.

Chapter 2 presents the background concepts underpinning this research. The core concepts of service orientation are presented such as loose coupling and modularity which form the basis of Service-Oriented Architectures (SOAs). Then the core concepts of systems their execution environment and the respective components are introduced. These ideas are then mapped onto SOAs and the taxonomy of SOA faults is extended with a focus on timing faults.

These concepts provide the basis for the remainder of the chapter which explores in detail the concepts related to RT-SOAs. First the theoretical concepts of schedulability from traditional real-time systems is explained with an introduction to notation and a definition of deadlines. Then the concept of resource adequacy is explored to help understand the challenges caused by resource interference. Given an understanding of real-time systems schedulability with concepts such as Worst-Case Execution Time (WCET) and execution slowdown the chapter focusses on real-time service QoS. The various QoS parameters that are used by different approaches are explored and mapped back to the concepts of resource adequacy with Cloud resource interference patterns. Finally some of the high level challenges in RT-QoS research are discussed under the categories of workflow management and QoS monitoring and prediction.

Given the background concepts of RT-SOAs and RT-QoSs Chapter 3 presents a detailed study of existing techniques. First a review of 80 QoS approaches is presented where the approaches are categorised into seven groups. Then from each category the most significant contributions are identified to be explored in more detail. Those approaches are studied in terms of their effectiveness of providing QoS for real-time systems. The remainder of the chapter then focusses on experimentally evaluating, using simulation, four of those approaches in the context of interfering workloads. The metrics of accuracy MAE, MPE, and MPW that are used for evaluation of approaches throughout this thesis are presented. Also those metrics relating to QoS violations including AVC, MAV, and MPV are outlined. The chapter concludes presenting the results of the simulation identifying the limitations of the existing work in handling service execution in dynamic environments.

In Chapter 4 the identified limitations in the existing work are used to form the basis of a new framework for modelling the QoS of μSs. From a systems modelling perspective the framework clearly distinguishes between μSs, services, the host machines, and the interfering workloads. A model is then presented capturing the resource requirements over the execution duration of a μS instance and this is used to formulate a predictive framework. The framework is used to estimate execution progress and the remaining execution time, or time-to-finish, of an instance.

Then in Section 4.3 the mathematics are outlined algorithmically and shown to scale linearly in terms of storage space and number of μSs. The entire method presented in the chapter is based on the schedulability concepts presented in Chapter 2. This allows the remainder of the chapter to focus on proving the ability of the framework to identify the bounds and solution space of schedulability. This is first shown visually before being evaluated inductively against schedulability tests with static, dynamic, continuously increasing, and continuously decreasing interfering workloads.

Chapter 5 takes the framework from the previous chapter and evaluates it using simulation. The framework is tested against two dynamic workload patterns and at each stage evaluated against the existing techniques described in Chapter 3. First the raw accuracy of the predictions were considered demonstrating an improvement against three of the four existing approaches, being beaten only by the fuzzy-logic approach. The focus then turns to understanding the trade-off between overallocation and QoS violation exploring each aspect individually before combining them in Section 5.5. The proposed framework is shown to waste at least 4% less execution time than the historical, probabilistic, correlation, and real-time middleware based approaches; whilst the fuzzy-logic technique remains more accurate wasting less than the proposed method. However when combined with QoS violations the fuzzy-logic technique results in 33% more violations than the proposed method. Finally the trade-off between QoS violation and overallocation is considered in terms of hard, firm, and soft deadlines with the proposed framework outperforming each of the existing approaches in the context of firm deadlines.

Finally Chapter 6 presents an overall assessment of the proposed framework for use in various domains. First an experimental evaluation using a set of twenty numerical functions as μSs is presented for the domain of Cloud computing. The framework demonstrated an average overallocation of 12% when predicting QoS based on WCETs and an underallocation of 8% when using the average response-times. The framework's application in that domain is also presented in terms of data processing applications and workload scheduling.

7.2 Research Contributions

The main contributions within this thesis can be summarised as:

- *An analysis and classification of existing QoS techniques.* This looked at eighty existing approaches, classified and then analysed the most significant contributions from each category. The seven identified categories were: correlation, optimisation, containment, middleware, fuzzy-logic, cost, and tolerance. The most relevant four categories (correlation, middleware, fuzzy-logic, and tolerance) were chosen and experimentally evaluated using simulation for their capability in handling non-static interfering workloads.

- *A mathematical n-dimensional framework capturing the relationship between μS execution performance and the execution environment.* This research presented a detailed framework explicitly modelling the difference between μSs and services and their interconnectivity. Then the relationship between μSs and their host server's resources was explored in terms of resource utilisation and availability. The framework was then proved to facilitate the use of real-time schedulability techniques.

- *Extensive simulation analysing the effectiveness of the QoS framework under various interfering Cloud workloads.* The mathematical framework was implemented and evaluated against periodic and random interfering workloads of various degrees. In order to assess the proposed mechanism measures of accuracy, overallocation, and QoS violation were used and the trade-off between overallocation and violation was studied.

- *A series of case studies and a prototype system.* These case studies were used to explore the application of the proposed QoS framework specifically in the domain of Cloud computing. The prototype system was used to evaluate proposed approach in a real system.

The contributions in this thesis bring together the worlds of traditional real-time systems, Cloud computing, and service-orientation with the fundamental requirement of understanding the relationship between the environment and the services' execution performance. The classification of QoS techniques has shown that there are various approaches that can be followed that are appropriate for different domains. Further it has been shown that there is a trade-off to be considered between accuracy of QoS and the level of violation that is deemed acceptable.

7.3 Overall Research Evaluation

In Chapter 1 Section 1.3 the research objectives of this thesis were discussed. The success of this thesis in achieving these objectives is listed below:

i. *To provide an in-depth analysis and classification of existing techniques for service QoS prediction.* This thesis has reviewed, in Chapter 3, in detail eighty existing QoS approaches. These have been classified into seven categories with some sub-groups within a couple of the individual categories. Each class has then be evaluated both theoretically and experimentally to identify the benefits and limitations to using each approach for RT-QoS prediction.

ii. *To provide a theoretical mechanism for efficient and accurate prediction of QoS.* In Chapter 4 this thesis has presented a new framework that mathematically defines the relationship between μS execution performance and the host environment's resources. This model has been applied to prediction response-times for QoS definitions that are conditional based on resource availability.

iii. *To provide efficient scalable algorithms for the prediction and management of QoS.* The mathematical framework from Chapter 3 was implemented algorithmically and demonstrated to be linearly scalable with respect to both the number of services and the number of execution instances. The approach is less efficient in terms of the number of resource dimensions, which is however anticipated to be remain considerably small and will remain constant for any given system configuration.

iv. *To provide an empirical evaluation of the proposed techniques.* Chapter 5 and Chapter 6 of this thesis evaluated the effectiveness of the presented framework using both simulation and experimentation. The simulation in Chapter 5 considered the effectiveness of the approach under various interfering Cloud workloads and compared this against existing techniques. Chapter 6 presented an experimental evaluation in the context of Cloud computing and also at a high level in terms of human task performance.

In summary it can be seen that all four major research objectives have been successfully completed. Finally the hypothesis outlined at the end of Chapter 3:

> **Hypothesis**
>
> *It is anticipated that a better understanding of the relationship between resource utilisation and execution progress, based on actual observations, would facilitate the forming of more accurate RT-QoS predictions without compromising the number of QoS violations.*

Has been shown to be valid with the proposed framework using a mapping between resource utilisation and performance to provide a more accurate QoS prediction than previous techniques.

7.4 Future Work

There are several future directions with which the work in this thesis could be enhanced. There are also future research areas which build upon the foundations of this research. Some of these opportunities are highlighted below.

7.4.1 Real-Time Quality of Service

Using the classification of QoS methods as the basis for future work there are clear opportunities within individual categories to improve the accuracy and safety of QoS definitions. Three major areas are:

1. *The use of machine learning for providing QoS methods.* This research has not taken a machine learning or optimisation approach which with sufficient scale could provide some novel techniques. Specifically the use of neural networks by the likes of Luo et al. [177] when combined with a sufficiently large data set of service executions could result in the production of a manifold function combining each of the resource dimensions with the complexities of utilisation patterns.

2. *The combination of various techniques as a mixed method.* Across the wide range of s-cenarios and system types different approaches will be more suited than others. There is therefore remains the question to identify which approaches are most suited to which scenarios. These factors may include the real-time nature of the system, varying from hard to soft. Alternatively in the Cloud domain this could be understanding be level at which the system of interest sits, for example a SaaS system will likely have less information regarding resource availability than a PaaS or IaaS system. A further perspective could be the transition over time between techniques, for example transitioning from the framework proposed in this thesis towards a machine learning approach as the number of services and execution instances becomes sufficiently large.

3. *The exploration of workflow-level QoS.* Workflows have been briefly mentioned in this thesis in Chapter 2 and introduce several challenges with regards to QoS. As alluded to in 2.1.3 two of the most challenging patterns are transient triggers and arbitrary cycles, the latter being a specific case of the well known halting problem. These patterns, and others, require

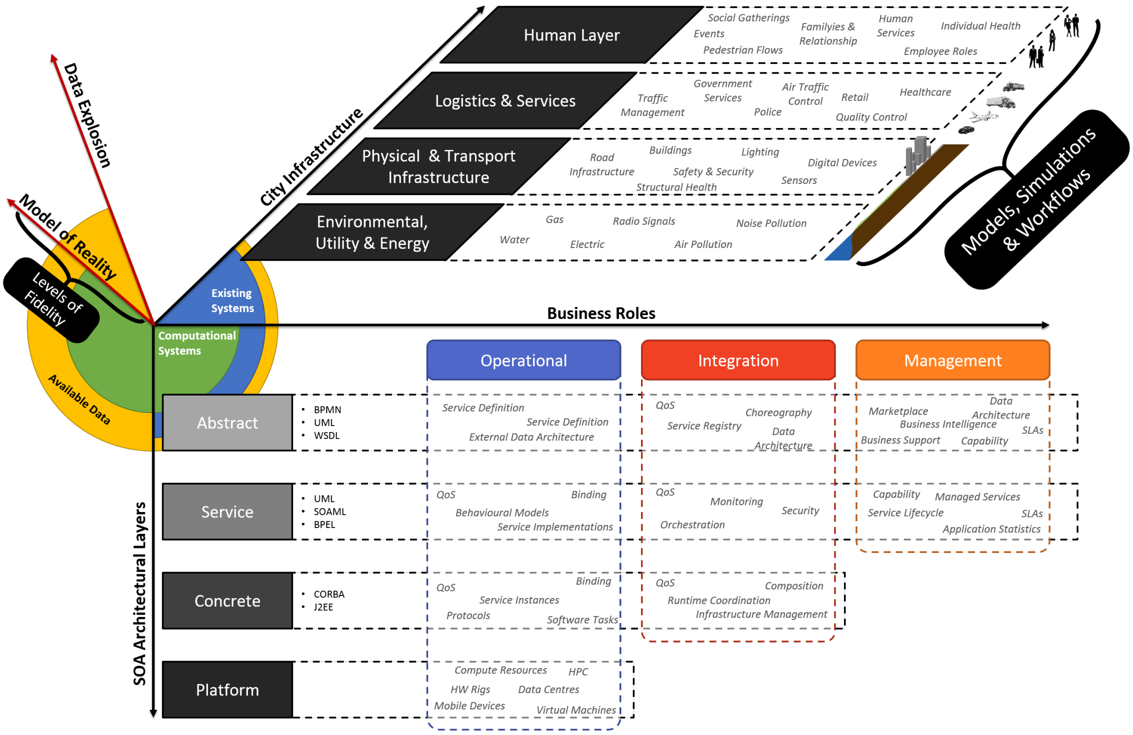


Figure 7.1: Multi-dimensionality of Internet of Simulation (IoS)

the combination QoS predictions for individual services into a system level prediction. It is likely that research in this area will need to consider specific domains and understand the constraints that may be applied before looking at the general case. For example this thesis's n-dimensional framework could be extended to capture the input, parameter, and resulting changes in output values as posed in [65].

7.4.2 Internet of Simulation

The Internet of Simulation (IoS) provides a completely new domain of research with a wide range of challenges to be addressed [178; 179]. Most interestingly will be the integration of research from the domains of IoT, Cloud computing, Edge computing, as well as from the non-computing domains looking at manufacturing, business, and social situations. Two key areas building directly on this thesis are:

- *Expanding the n-dimensional QoS framework.* The multi-dimensional nature of the QoS framework presented in this thesis provides the basis for exploring the multi-dimensional nature of the reality of IoS. As simulations are integrated the QoS challenges become

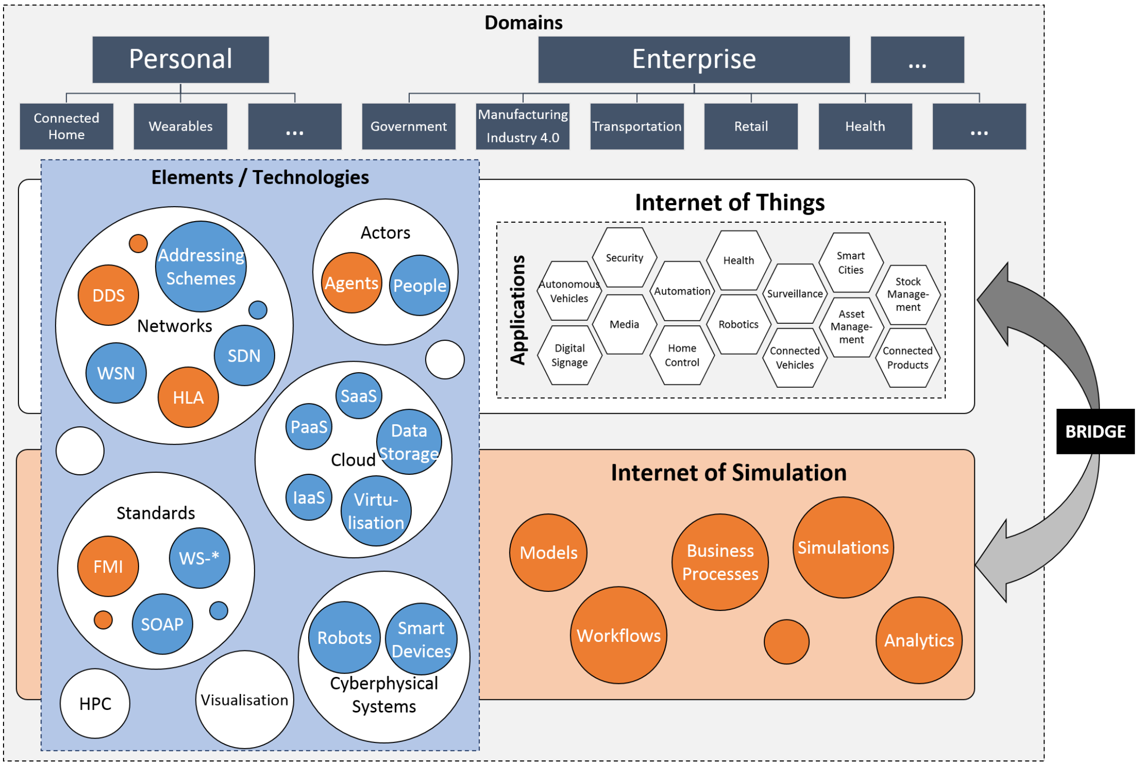


Figure 7.2: IoS an extension of IoT, domains applications, elements, and technologies

more complex as they must facilitate temporal integration and synchronisation in order to maintain simulation accuracy. This area of QoS for Simulation-as-a-Service (SIMaaS) and also Workflow-as-a-Service (WFaaS) is specific domain of the workflow QoS challenge described previously.

Moving then away from QoS the multi-dimensional nature of the *model of reality*, as described by Clement et al. [180] and shown in Figure 7.1, also opens an interesting area of research.

- *Real-time bridge with IoT.* As shown in Figure 7.2 in order to connect the virtual world of IoS to the real world of IoT there must be some real-time bridge. Developing this capability requires understanding of the networking and infrastructure requirements in addition to the QoS issues mentioned previously. This area also opens up an interesting dialogue on the semantics and standards for interoperability, such as High Level Architecture (HLA), that will be required to facilitate data and control exchange in the smart cities of the future.

IoS derives from the evolving need of global industry, in particular automate, aerospace, and defence for virtualisation of the engineering and manufacturing processes [133]. Specifically IoS

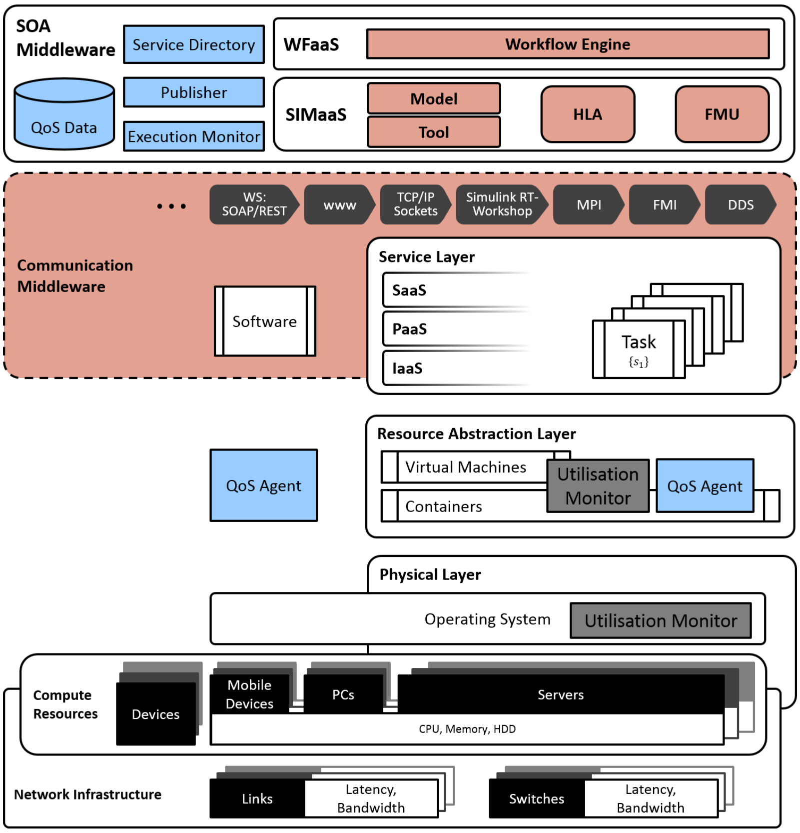


Figure 7.3: Generic architecture for Internet of Simulation

is focussed on enabling: knowledge sharing, evolving fidelity and agile engineering, complex integration, supply chain integration, massive-scalability, simulation as a utility, and integration with IoT.

The core concepts of IoS revolve around Simulation-as-a-Service (SIMaaS) and Workflow-as-a-Service (WFaaS). The former builds on the SaaS paradigm but introduces the time and clock management as well as having to handle causality. The latter facilitates the construction of SOA workflows consisting of simulations rather than processes. Furthermore the WFaaS must allow a recursive relationship allowing individual workflows to be nested within others as if they were individual simulation services. IoS can be explicitly defined as:

- *A specialism of the Internet of Things comprised of interconnected virtual system components, agents, or virtual environments defined by cross-domain collections of network-enabled, variable fidelity and heterogeneous models and simulations.*

- *Through composing multiple virtual entities by defining their interactivity a system simula-*

tion can be constructed and distributed.

- *The simulated things contained in the IoS can be connected to the IoT via a Real-Time Bridge.*

From an architectural perspective the core challenge in IoS is the extension of generic Cloud architectures to support the SIMaaS and WFaaS paradigms for simulation integration in a usable and efficient manner. This could build on the basic form shown in Figure 7.3 which extends the generic Cloud architecture that was depicted in Chapter 6 in Figure 6.2. Most notably any implementation of IoS will have to take account of both heterogeneous infrastructure including HPC systems, IoT devices and other in-the-loop simulators. To do so standards compliance with the likes of Functional Mock-up Interface (FMI) for in-memory communication [181] and IEEE HLA will be essential [182]. In order to guarantee simulation performance and accuracy the RT-QoS framework of this research will have to be adapted to predict the execution time between individual simulation time steps rather than end-to-end execution times.

Bibliography

[1] M.P. Papazoglou. Service-oriented computing: concepts, characteristics and directions. In *Proceedings of the 7th International Conference on Properties and Applications of Dielectric Materials (Cat. No.03CH37417)*, pages 3–12. IEEE Comput. Soc, 2003. ISBN 0-7695-1999-7. doi: 10.1109/WISE.2003.1254461. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1254461.

[2] Algirdas Avizienis, J.-C. Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, jan 2004. ISSN 1545-5971. doi: 10.1109/TDSC.2004.2. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1335465.

[3] W3C. Web Services Glossary, 2004. URL https://www.w3.org/TR/ws-gloss/{#}webservice.

[4] InnoQ. Web Services Standards Overview, 2007.

[5] Gerardo Pardo-castellote. OMG Data-Distribution Service (DDS): Architectural Overview. Technical report, Real-Time Innovations, Inc. (RTI), 2005.

[6] Zibin Zheng, Yilei Zhang, and Michael R Lyu. Investigating QoS of Real-World Web Services. *IEEE Transactions on Services Computing*, 7(1):32–39, jan 2014. ISSN 1939-1374. doi: 10.1109/TSC.2012.34. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6357180.

[7] Boris Lublisnky. Defining SOA as an architectural style, jan 2007. URL http://www.ibm.com/developerworks/architecture/library/ar-soastyle/.

[8] Michael P. Papazoglou, Paolo Traverso, Schahram Dustdar, and Frank Leymann. Service-Oriented Computing: State of the Art and Research Challenges. *Computer*, 40(11):38–45, nov 2007. ISSN 0018-9162. doi: 10.1109/MC.2007.400. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4385255.

[9] Benjamin Mueller, Goetz Viering, Frederik Ahlemann, and Gerold Riempp. Towards Unverstanding the Sources of the Economic Potential of Service-Orientated Architecture: Findings from the Automotive and Banking Industry. In *ECIS*, pages 1608–1619, 2007.

[10] Nicolas Gold, Andrew Mohan, Claire Knight, and Malcolm Munro. Understanding service-orientated software. *IEEE Software*, 44(October):71–77, oct 2008. URL `http://dro.dur.ac.uk/621/1/621.pdf`.

[11] Len Bass, Paul Clements, Rick Kaxman, and Rick Kazman. *Software Architecture in Practice, Second Edition*. Addison-Wesley, second edi edition, 2006. ISBN 0321154959.

[12] Jorge L. Sanz, Valeria Becker, Juan Cappi, Ankur Chandra, Joseph Kramer, Kelly Lyman, Nitin Nayak, Pablo Pesce, Ignacio Terrizzano, and John Vergo. Business Services and Business Componentization: New Gaps between Business and IT. In *IEEE International Conference on Service-Oriented Computing and Applications (SOCA '07)*, pages 271–278. IEEE, jun 2007. ISBN 0-7695-2861-9. doi: 10.1109/SOCA.2007.11. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4273436`.

[13] M. Turner, D. Budgen, and P. Brereton. Turning software into a service. *Computer*, 36(10):38–44, oct 2003. ISSN 0018-9162. doi: 10.1109/MC.2003.1236470. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1236470`.

[14] Enrique Castro-Leon, Jackson He, and Mark Chang. Scaling Down SOA to Small Businesses. In *IEEE International Conference on Service-Oriented Computing and Applications (SOCA '07)*, pages 99–106. IEEE, jun 2007. ISBN 0-7695-2861-9. doi: 10.1109/SOCA.2007.37. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4273415`.

[15] David Booth, Hugo Haas, F McCabe, E Newcomer, M Champion, C Ferris, and D Orchard. Web Services Architecture. *Group*, 22(February):19–26, 2004. ISSN 13583948. doi: 10.1023/B:BTTJ.0000015492.03732.a6. URL `http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/{%}5Cnhttps://www.w3.org/TR/ws-arch/{#}discovery`.

[16] Le Sun, Hai Dong, and Jamshaid Ashraf. Survey of Service Description Languages and Their Issues in Cloud Computing. *2012 Eighth International Conference on Semantics, Knowledge and Grids*, pages 128–135, oct 2012. doi: 10.1109/SKG.2012.49. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6391820`.

[17] Ian J. Taylor and Andrew Harrison. *From P2P and Grids to Services on the Web*. Computer Communications and Networks. Springer London, London, 2 edition, 2009. ISBN 978-1-84800-122-0. doi: 10.1007/978-1-84800-123-7. URL `http://link.springer.com/10.1007/978-1-84800-123-7`.

[18] Haithem Mezni, Walid Chainbi, and Khaled Ghedira. An autonomic registry-based SOA model. *2011 IEEE International Conference on Service-Oriented Computing and Applications (SOCA)*, pages 1–4, dec 2011. doi: 10.1109/SOCA.2011.6166265. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6166265`.

[19] Shahram Dustdar and Mike P. Papazoglou. Services and Service Composition An Introduction (Services und Service Komposition Eine Einführung). *Information Technology*, 2:86–92, feb 2008. ISSN 1611-2776. doi: 10.1524/itit.2008.0468. URL `http://www.degruyter.com/doi/10.1524/itit.2008.0468`.

[20] Michael P. Papazoglou, Paolo Traverso, Schahram Dustdar, and Frank Leymann. Service-Oriented Computing: State of the Art and Research Challenges. *Computer*, 40(11):38–45, nov 2007. ISSN 0018-9162. doi: 10.1109/MC.2007.400. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4385255http://ieeexplore.ieee.org/document/4385255/`.

[21] Paul Gustavson, Tram Chase, Larry Root, and Karl Crosson. Moving Towards a Service-Oriented Architecture (SOA) for Distributed Component Simulation Environments. In *Proceedings of the 2005 Spring Simulation Interoperability Workshop, San Diego*, 2005.

[22] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. 1994.

[23] Paul R. Allen and Sam Higgins. *Service Orientation: Winning Strategies and Best Practices*. 2006.

[24] David Webster, Lu Liu, Duncan Russell, Colin Venters, Zongyang Luo, and Jie Xu. Migrating Legacy Assets through SOA to Realize Network Enabled Capability. In *New Directions in Web Data Management 1*, pages 311–346. 2011. doi: 10.1007/978-3-642-17551-0_11. URL http://link.springer.com/10.1007/978-3-642-17551-0{_}11.

[25] Don Dox, John DeVadoss, Kris Dorrocks, Mark Baciak, Atanu Bannerjee, Shy Cohen, William Oellermann, Brenton Webster, Dave Green, John Evdemon, Roger Wolter, Kirk Haselden, Simon Guest, Kim Cameron, and Fred Chong. *Service Oriented Architecture (SOA) in the Real World*. Microsoft, 2007. URL http://www.microsoft.com/en-us/download/details.aspx?id=16187.

[26] James A. Rowson and Alberto Sangiovanni-Vincentelli. Interface-based design. In *Proceedings of the 34th annual conference on Design automation conference - DAC '97*, pages 178–183, New York, New York, USA, 1997. ACM Press. ISBN 0897919203. doi: 10.1145/266021.266060. URL http://portal.acm.org/citation.cfm?doid=266021.266060.

[27] Theo Haerder and Andreas Reuter. Principles of transaction-oriented database recovery. *ACM Computing Surveys*, 15(4):287–317, dec 1983. ISSN 03600300. doi: 10.1145/289.291. URL http://portal.acm.org/citation.cfm?doid=289.291.

[28] Dan Pritchett. BASE: AN ACID ALTERNATIVE. *Queue*, 6(3):48–55, may 2008. ISSN 15427730. doi: 10.1145/1394127.1394128. URL http://portal.acm.org/citation.cfm?doid=1394127.1394128.

[29] W T Tsai, Hessam S Sarjoughian, Wu Li, and Xin Sun. Timing specification and analysis for service-oriented simulation. *Proceedings of the 2009 Spring Simulation Multiconference*, pages 51:1—-51:9, 2009. URL http://dl.acm.org/citation.cfm?id=1639809.1639862.

[30] Pierre Châtel, Jacques Malenfant, and Isis Truck. QoS-based Late-Binding of Service Invocations in Adaptive Business Processes. In *2010 IEEE International Conference on Web Services*, pages 227–234. IEEE, jul 2010. ISBN 978-1-4244-8146-0. doi: 10.1109/ICWS.

2010.74. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5552782.

[31] W3C. Web Services Description Language (WSDL) 1.1, 2001. URL http://www.w3.org/TR/wsdl.

[32] Nebil Ben Mabrouk, Sandrine Beauche, Elena Kuznetsova, Nikolaos Georgantas, and Valérie Issarny. QoS-Aware Service Composition in Dynamic Service Oriented Environments. In *Lecture Notes in Computer Sicence*, volume 5896, pages 123–142. 2009. ISBN 9783642104459. doi: 10.1007/978-3-642-10445-9_7. URL http://link.springer.com/10.1007/978-3-642-10445-9{_}7.

[33] Zibin Zheng and Michael R. Lyu. A Distributed Replication Strategy Evaluation and Selection Framework for Fault Tolerant Web Services. In *2008 IEEE International Conference on Web Services*, pages 145–152. IEEE, sep 2008. doi: 10.1109/ICWS.2008.42. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4670170.

[34] Duncan Russell, Nik Looker, Lu Liu, and Jie Xu. Service-Oriented Integration of Systems for Military Capability. In *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, pages 33–41. IEEE, may 2008. ISBN 978-0-7695-3132-8. doi: 10.1109/ISORC.2008.45. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4519558.

[35] Anthony Sargeant, Paul Townend, Jie Xu, and Karim Djemame. Evaluating the Dependability of Dynamic Binding in Web Services. *2012 IEEE 14th International Symposium on High-Assurance Systems Engineering*, pages 139–146, oct 2012. doi: 10.1109/HASE.2012.28. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6375608.

[36] M. Tian, A. Gramm, H. Ritter, and J. Schiller. Efficient Selection and Monitoring of QoS-Aware Web Services with the WS-QoS Framework. In *IEEE/WIC/ACM International Conference on Web Intelligence (WI'04)*, pages 152–158. IEEE, 2004. ISBN 0-7695-2100-

2. doi: 10.1109/WI.2004.10084. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1410797.

[37] W3C and Innoq. Web Services Standards, 2007. URL http://www.w3.org/2002/ws/{#}documents.

[38] Thomas Erl, Anish Karmarkar, Priscilla Walmsley, Hugo Haas, Umit Yalcinalp, Canyang Kevin Liu, David Orchard, Andre Tost, and James Pasley. *Web Service Contract Design and Versioning for SOA*. Prentice Hall, 2008. ISBN 9780136135173.

[39] Mike P. Papazoglou. The Challenges of Service Evolution. In *Advanced Information Systems Engineering*, volume 26 Suppl 4, pages 1–15. Springer Berlin Heidelberg, Berlin, Heidelberg, 1997. doi: 10.1007/978-3-540-69534-9_1. URL http://www.ncbi.nlm.nih.gov/pubmed/9506431http://link.springer.com/10.1007/978-3-540-69534-9{_}1.

[40] Vasilios Andrikopoulos, Salima Benbernou, and Mike P. Papazoglou. On the Evolution of Services. *IEEE Transactions on Software Engineering*, 38(3):609–628, may 2012. ISSN 0098-5589. doi: 10.1109/TSE.2011.22. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5728828.

[41] David Webster, Paul Townend, and Jie Xu. Interface Refactoring in Performance-Constrained Web Services. *2012 IEEE 15th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*, pages 111–118, apr 2012. doi: 10.1109/ISORC.2012.23. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6195868.

[42] Martin Fowler and Kent Beck. *Refactoring: improving the design of existing code*. Addison-Wesley Professional, 1999.

[43] Fred B Schneider. *Trust in Cyberspace*. National Academies Press, 1999. ISBN 0309065585. URL http://lib.leeds.ac.uk/record=b1941478.

[44] Liang Zhengping, Liu Xiaoli, Wu Guoqing, Yang Min, and Zhang Fan. A formal Framework for Trust management of Service-oriented Systems. In *IEEE International Conference on Service-Oriented Computing and Applications (SOCA '07)*, pages 241–248. IEEE, jun

2007. ISBN 0-7695-2861-9. doi: 10.1109/SOCA.2007.3. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4273432.

[45] Pankesh Patel, Ah Ranabahu, and Ap Sheth. Service level agreement in cloud computing. 2009. URL http://corescholar.libraries.wright.edu/knoesis/78/.

[46] Gaurav Jain, Deepali Singh, and Shekhar Verma. Service level agreements in IP networks. *Information Management & Computer Security*, 10(4):171–177, oct 2002. ISSN 0968-5227. doi: 10.1108/09685220210436967. URL http://www.emeraldinsight.com/doi/abs/10.1108/09685220210436967.

[47] Heiko Ludwig, Alexander Keller, Asit Dan, and R. King. A service level agreement language for dynamic electronic services. In *Proceedings Fourth IEEE International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems (WECWIS 2002)*, pages 25–32. IEEE Comput. Soc, 2003. ISBN 0-7695-1567-3. doi: 10.1109/WECWIS.2002.1021238. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.69.7687{&}rep=rep1{&}type=pdfhttp://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1021238.

[48] Open Grid Forum. Web services agreement specification (ws-agreement), 2007. URL https://www.ogf.org/documents/GFD.107.pdf.

[49] Sukhpal Singh and Inderveer Chana. QoS-Aware Autonomic Resource Management in Cloud Computing. *ACM Computing Surveys*, 48(3):1–46, dec 2015. ISSN 03600300. doi: 10.1145/2843889. URL http://dl.acm.org/citation.cfm?doid=2856149.2843889.

[50] Markus C Huebscher and Julie a McCann. A survey of autonomic computingdegrees, models, and applications. *ACM Computing Surveys*, 40(3):1–28, aug 2008. ISSN 03600300. doi: 10.1145/1380584.1380585. URL http://portal.acm.org/citation.cfm?doid=1380584.1380585.

[51] Nicola Muscettola, P.Pandurang Nayak, Barney Pell, and Brian C. Williams. Remote Agent: to boldly go where no AI system has gone before. *Artificial Intelligence*, 103(1-2):

5–47, aug 1998. ISSN 00043702. doi: 10.1016/S0004-3702(98)00068-X. URL http://linkinghub.elsevier.com/retrieve/pii/S000437029800068X.

[52] Paul Horn. Autonomic computing: IBM's Perspective on the State of Information Technology. 2001.

[53] J.O. Kephart and D.M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, jan 2003. ISSN 0018-9162. doi: 10.1109/MC.2003.1160055. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1160055.

[54] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 25 (6):599–616, jun 2009. ISSN 0167739X. doi: 10.1016/j.future.2008.12.001. URL http://portal.acm.org/citation.cfm?id=1528937.1529211http://linkinghub.elsevier.com/retrieve/pii/S0167739X08001957.

[55] Jokha Al-Kalbani, Naoufel Kraiem, Zuhoor Al-Khanjari, and Yassine Jamoussi. Towards a Comprehensive View of Web Services QoS Models. *International Journal of Multimedia and Ubiquitous Engineering*, 10(10):259–272, oct 2015. ISSN 19750080. doi: 10.14257/ijmue.2015.10.10.26. URL http://www.sersc.org/journals/IJMUE/vol10{_}no10{_}2015/26.pdf.

[56] Lu Liu, Jie Xu, Duncan Russell, John K. Davies, David Webster, Zongyang Luo, and Colin Venters. Dynamic Service Integration for Reliable and Sustainable Capability Provision. *International Journal of Systems Science*, 2010.

[57] Liming Chen and Algirdas Avizienis. N-version programming : a fault-tolerance approach to reliability. *FTCS-8*, 3:3–9, 1978.

[58] P.E. Ammann and J.C. Knight. Data diversity: an approach to software fault tolerance. *IEEE Transactions on Computers*, 37(4):418–425, apr 1988. ISSN 00189340. doi: 10.1109/12.2185. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=2185.

[59] Marisol García-valls, Pablo Basanta-val, Marga Marcos, and Elisabet Estévez. A bi-dimensional QoS model for SOA and real-time middleware. *International Journal of Computer Science and Engineering*, 2013.

[60] David W McKee, David Webster, and Jie Xu. Enabling Decision Support for the Delivery of Real-Time Services. In *2015 IEEE 15th International Symposium on High-Assurance Systems Engineering*. IEEE, jan 2015. ISBN 978-1-4799-3466-9. doi: 10.1109/HASE.2015.18.

[61] OMG. The OMG and Service Oriented Architecture. Technical report.

[62] Scott Simmons. BPM Voices : Evaluating BPM applications : BPM design reviews and Rubik ' s Cubes. Technical report, 2013.

[63] T. Cucinotta, A. Mancina, G.F. Anastasi, G. Lipari, L. Mangeruca, R. Checcozzo, and F. Rusina. A Real-Time Service-Oriented Architecture for Industrial Automation. *IEEE Transactions on Industrial Informatics*, 5(3):267–277, aug 2009. ISSN 1551-3203. doi: 10.1109/TII.2009.2027013. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5173504http://ieeexplore.ieee.org/document/5173504/`.

[64] Guang Zhou, Yansheng Zhang, Farokh Bastani, and I-Ling Yen. Service-Oriented Robotic Swarm Systems: Model and Structuring Algorithms. In *2012 IEEE 15th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*, pages 95–102. IEEE, apr 2012. ISBN 978-0-7695-4643-8. doi: 10.1109/ISORC.2012.21. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6195866http://ieeexplore.ieee.org/document/6195866/`.

[65] David W McKee, David Webster, Jie Xu, and David Battersby. DIVIDER: Modelling and Evaluating Real-Time Service-Oriented Cyberphysical Co-Simulations. In *2015 IEEE 18th International Symposium on Real-Time Distributed Computing*, pages 272–275. IEEE, apr 2015. ISBN 978-1-4799-8781-8. doi: 10.1109/ISORC.2015.30. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7153816`.

[66] J Huang, F Bastani, I.-L. Yen, J Dong, W Zhang, F.-J. Wang, and H.-J. Hsu. Extending service model to build an effective service composition framework for cyber-physical systems. In *2009 IEEE International Conference on Service-Oriented Computing and Applications (SOCA)*, volume 00, pages 1–8. IEEE, dec 2009. ISBN 978-1-4244-5300-9. doi: 10.1109/SOCA.2009.5410453. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5410453.

[67] M. García Valls and P. Basanta Val. A real-time perspective of service composition: Key concepts and some contributions. *Journal of Systems Architecture*, 59(10):1414–1423, 2013. ISSN 13837621. doi: 10.1016/j.sysarc.2013.06.008. URL http://dx.doi.org/10.1016/j.sysarc.2013.06.008.

[68] Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch Lafuente, Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin, and Larisa Safina. Microservices: yesterday, today, and tomorrow. jun 2016. URL http://arxiv.org/abs/1606.04036.

[69] Martin Fowler and James Lewis. Microservices a definition of this new architectural term. URL http://martinfowler.com/articles/microservices.html.

[70] Iria Estévez-Ayres, Pablo Basanta-Val, and Marisol García-Valls. Composing and scheduling service-oriented applications in time-triggered distributed real-time Java environments. *Concurrency and Computation: Practice and Experience*, 26(1):152–193, jan 2014. ISSN 15320626. doi: 10.1002/cpe.2958. URL http://doi.wiley.com/10.1002/cpe.2958.

[71] Arnon Rotem-Gal-Oz. Services, Microservices, Nanoservices oh my!, 2016. URL http://arnon.me/2014/03/services-microservices-nanoservices/.

[72] James D Moreland, Shahram Sarkani, and Thomas Mazzuchi. Service-Oriented Architecture (SOA) Instantiation within a Hard Real-Time , Deterministic Combat System Environment. *Systems Engineering*, 17:1–14, 2013. doi: 10.1002/sys.

[73] Ismael Solis Moreno, Peter Garraghan, Paul Townend, and Jie Xu. Analysis, Modeling and Simulation of Workload Patterns in a Large-Scale Utility Cloud. *IEEE Transactions on Cloud Computing*, 2(2):208–221, apr 2014. ISSN 2168-7161. doi: 10.1109/TCC.2014.

2314661. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6782445.

[74] Chris Peltz. Web services orchestration and choreography. *Computer*, 36(10):46–52, oct 2003. ISSN 0018-9162. doi: 10.1109/MC.2003.1236471. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1236471.

[75] OMG. Business Process Model and Notation (BPMN). Technical Report January, 2011.

[76] Rachid Hamadi and Boualem Benatallah. A Petri net-based model for web service composition. *Journal of Shanghai University (English Edition)*, 17:191–200, aug 2003. ISSN 1007-6417.

[77] Bartosz Kiepuszewski, Arthur H.M. ter Hofstede, and Wil M.P. van der Aalst. Fundamentals of Control Flow in Workflows.

[78] Bertram Ludäscher, Ilkay Altintas, Chad Berkley, Dan Higgins, Efrat Jaeger, Matthew Jones, Edward a. Lee, Jing Tao, and Yang Zhao. Scientific workflow management and the Kepler system. *Concurrency and Computation: Practice and Experience*, 18(10):1039–1065, aug 2006. ISSN 1532-0626. doi: 10.1002/cpe.994. URL http://doi.wiley.com/10.1002/cpe.994.

[79] Andreas Schonberger and Guido Wirtz. Sequential Composition of Multi-Party Choreographies. In *2010 IEEE International Conference on Service-Oriented Computing and Applications (SOCA)*, 2010.

[80] S. de Gyves Avila and Karim Djemame. Fuzzy Logic Based QoS Optimization Mechanism for Service Composition. In *2013 IEEE Seventh International Symposium on Service-Oriented System Engineering*, pages 182–191. IEEE, mar 2013. ISBN 978-0-7695-4944-6. doi: 10.1109/SOSE.2013.28. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6525521.

[81] Guobing Zou, Qiang Lu, Yixin Chen, Ruoyun Huang, You Xu, and Yang Xiang. QoS-Aware Dynamic Composition of Web Services Using Numerical Temporal Planning. *IEEE Transactions on Services Computing*, 7(1):18–31, jan 2014. ISSN 1939-1374. doi: 10.1109/TSC.2012.27. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6331478.

[82] Antonio Brogi and Razvan Popescu. Automated Generation of BPEL Adapters. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 4294 LNCS, pages 27–39. 2006. ISBN 3540681477. doi: 10.1007/11948148_3. URL http://link.springer.com/10.1007/11948148{_}3.

[83] W.M.P van der Aalst, Arthur H M Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14(3):5–51, 2003. doi: 10.1023/A:1022883727209.

[84] Nick Russell, Arthur H M Hofstede, W.M.P van der Aalst, and N. Mulyar. WORKFLOW CONTROL-FLOW PATTERNS A Revised View. Technical report, 2006. URL www.workflowpatterns.com.

[85] Nick Russell, Wil M P Van Der Aalst, Arthur H M Hofstede, and David Edmond. Workflow Resource Patterns : Identification , Representation and Tool Support. .

[86] Nick Russell, Arthur H M Hofstede, David Edmond, and Wil M P Van Der Aalst. Workflow Data Patterns : Identification , Representation and Tool Support. pages 353–368, 2005.

[87] Wil M P Van Der Aalst, Arjan J Mooij, and Christian Stahl. Service Interaction : Patterns , Formalization , and Analysis.

[88] Nick Russell, Wil Van Der Aalst, and Arthur Hofstede. Workflow Exception Patterns. pages 288–302, 2006.

[89] Nick Russell, Wil M P Van Der Aalst, and Arthur H M Hofstede. Exception Handling Patterns in Process-Aware Information Systems. .

[90] W.M.P. van der Aalst and a.H.M. ter Hofstede. YAWL: yet another workflow language. *Information Systems*, 30(4):245–275, jun 2005. ISSN 03064379. doi: 10.1016/j.is.2004.02.002. URL http://linkinghub.elsevier.com/retrieve/pii/S0306437904000304.

[91] Martin Davis. *The undecidable: Basic papers on undecidable propositions, unsolvable problems and computable functions*. 1965.

[92] NA Nataliya Mulyar. *Patterns for process-aware information systems: an approach based on colored Petri nets*. PhD thesis, Technische Universiteit Eindhoven, 2009.

[93] Michael Kircher and Prashant Jain. Patterns for resource management. *Pattern-oriented software architecture*, 3, 2004.

[94] K. Birman and T. Joseph. Exploiting virtual synchrony in distributed systems. *ACM SIGOPS Operating Systems Review*, 21(5):123–138, nov 1987. ISSN 01635980. doi: 10.1145/37499.37515. URL `http://portal.acm.org/citation.cfm?doid=37499.37515`.

[95] Imed Abbassi, Mohamed Graiet, Souha Boubaker, Mourad Kmimech, and Nejib Ben Hadj-Alouane. A Formal Approach for Verifying QoS Variability in Web Services Composition Using EVENT-B. In *2015 IEEE International Conference on Web Services*, pages 519–526. IEEE, jun 2015. ISBN 978-1-4673-7272-5. doi: 10.1109/ICWS.2015.75. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7195610`.

[96] OED Online. *Oxford English Dictionary*. Oxford University Press, 2017.

[97] INCOSE. *Systems Engineering Handbook*. 3.1 edition, 2007.

[98] Algirdas Avizienis, Jean-Claude Laprie, and Brian Randell. Fundamental Concepts of Dependability. Technical Report 010028, University of Newcastle; UCLA; LAAS, 2001.

[99] Jean-Claude Laprie. *Dependability : basic concepts and terminology in English, French, German, Italian, and Japanese*. 1992. ISBN 3211822968.

[100] Jean-Claude Laprie. Dependable computing and fault-tolerance. *Digest of Papers FTCS-15*, 2(11), 1985.

[101] Brian Randell. System structure for software fault tolerance. *IEEE Transactions on Software Engineering*, 1(2), 1975.

[102] Stefan Bruning, Stephan Weissleder, and Miroslaw Malek. A Fault Taxonomy for Service-Oriented Architecture. In *10th IEEE High Assurance Systems Engineering Symposium (HASE'07)*, pages 367–368. IEEE, nov 2007. ISBN 0-7695-3043-5. doi: 10.1109/HASE.

2007.46. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4404761.

[103] Hermann Kopetz. Real-Time Scheduling. In *Real-Time Systems: Design Principles for Distributed Embedded Applications*, pages 239–258. Springer, 2nd edition, 2011.

[104] Alan Burns and Andy Wellings. *Scheduling*. 2001.

[105] Raimund Kirner. A Uniform Model for Tolerance-Based Real-Time Computing. In *2014 IEEE 17th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*, pages 9–16. IEEE, jun 2014. ISBN 978-1-4799-4430-9. doi: 10.1109/ISORC.2014.8. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6899125.

[106] Jinkyu Lee, Insik Shin, and Arvind Easwaran. Convex optimization framework for intermediate deadline assignment in soft and hard real-time distributed systems. *Journal of Systems and Software*, 85(10):2331–2339, oct 2012. ISSN 01641212. doi: 10.1016/j.jss.2012.04.050. URL http://linkinghub.elsevier.com/retrieve/pii/S0164121212001100.

[107] D. Sow and A. Eleftheriadis. Representing information with computational resource bounds. In *Conference Record of Thirty-Second Asilomar Conference on Signals, Systems and Computers (Cat. No.98CH36284)*, volume 1, pages 452–456. IEEE. ISBN 0-7803-5148-7. doi: 10.1109/ACSSC.1998.750904. URL http://ieeexplore.ieee.org/document/750904/.

[108] Gregory J. Chaitin. On the Length of Programs for Computing Finite Binary Sequences. *Journal of the ACM*, 13(4):547–569, oct 1966. ISSN 00045411. doi: 10.1145/321356.321363. URL http://portal.acm.org/citation.cfm?doid=321356.321363.

[109] G Arun Kumar, Snehanshu Saha, Aravind Sundaresan, and Bidisha Goswami. A QoS aware Novel Probabilistic strategy for Dynamic Resource Allocation. *Distributed, Parallel, and Cluster Computing*, pages 1–10, mar 2015. URL http://arxiv.org/abs/1503.07038.

[110] K. Dragoon and V. Dvortsov. Z-Method for Power System Resource Adequacy Applications. *IEEE Transactions on Power Systems*, 21(2):982–988, may 2006. ISSN 0885-8950. doi: 10.1109/TPWRS.2006.873417. URL http://ieeexplore.ieee.org/document/1626406/.

[111] Salima Benbernou, Allel Hadjali, Naouel Karam, and Mourad Ouziri. Managing QoS Acceptability for Service Selection : A Probabilistic Description Logics Based Approach. In *Proceedings of the 28th International Workshop on Description Logics*, volume 1350, 2015.

[112] Chia-Ming Wu, Ruay-Shiung Chang, and Hsin-Yu Chan. A green energy-efficient scheduling algorithm using the DVFS technique for cloud datacenters. *Future Generation Computer Systems*, 37:141–147, jul 2014. ISSN 0167739X. doi: 10.1016/j.future.2013.06.009. URL http://dx.doi.org/10.1016/j.future.2013.06.009http://linkinghub.elsevier.com/retrieve/pii/S0167739X13001234.

[113] Anton Beloglazov and Rajkumar Buyya. Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in Cloud data centers. *Concurrency and Computation: Practice and Experience*, 24(13):1397–1420, sep 2012. ISSN 15320626. doi: 10.1002/cpe.1867. URL http://doi.wiley.com/10.1002/cpe.1867.

[114] Lei Shi, John Furlong, and Runxin Wang. Empirical evaluation of vector bin packing algorithms for energy efficient data centers. In *2013 IEEE Symposium on Computers and Communications (ISCC)*, pages 000009–000015. IEEE, jul 2013. ISBN 978-1-4799-3755-4. doi: 10.1109/ISCC.2013.6754915. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6754915http://ieeexplore.ieee.org/document/6754915/.

[115] Mohammed Rashid Chowdhury, Mohammad Raihan Mahmud, and Rashedur M. Rahman. Study and performance analysis of various VM placement strategies. In *2015 IEEE/ACIS 16th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, pages 1–6. IEEE, jun 2015. ISBN 978-1-4799-8676-7. doi: 10.1109/SNPD.2015.7176234. URL http://ieeexplore.ieee.org/document/7176234/.

[116] a Horri and Gh Dastghaibyfard. A Novel Cost Based Model for Energy Consumption in Cloud Computing. *The Scientific World Journal*, 2015(Article ID 724524):1–10, 2015. ISSN 2356-6140. doi: 10.1155/2015/724524. URL http://www.hindawi.com/journals/tswj/2015/724524/.

[117] Brian Kell and Willem-Jan van Hoeve. An MDD Approach to Multidimensional Bin Packing. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 7874 LNCS, pages 128–143. 2013. ISBN 9783642381706. doi: 10.1007/978-3-642-38171-3_9. URL http://link.springer.com/10.1007/978-3-642-38171-3{_}9.

[118] Sebastian Berndt, Klaus Jansen, and Kim-Manuel Klein. Fully Dynamic Bin Packing Revisited. pages 1–41, nov 2014. URL http://arxiv.org/abs/1411.0960.

[119] Brian Kroth. CS787 Project: Bin Packing: A Survey and its Applications to Job Assignment and Machine Allocation. 2013.

[120] C. L. Liu and James W. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *Journal of the ACM*, 20(1):46–61, jan 1973. ISSN 00045411. doi: 10.1145/321738.321743. URL http://portal.acm.org/citation.cfm?doid=321738.321743.

[121] J. Xu and D.L. Parnas. Scheduling processes with release times, deadlines, precedence and exclusion relations. *IEEE Transactions on Software Engineering*, 16(3):360–369, mar 1990. ISSN 00985589. doi: 10.1109/32.48943. URL http://ieeexplore.ieee.org/document/48943/.

[122] Maciej Drozdowski. *Scheduling for Parallel Processing*, volume 1 of *Computer Communications and Networks*. Springer London, London, 2009. ISBN 978-1-84882-309-9. doi: 10.1007/978-1-84882-310-5. URL http://link.springer.com/10.1007/978-1-84882-310-5.

[123] Jieming Zhu, Pinjia He, Zibin Zheng, and Michael R Lyu. Towards Online, Accurate, and Scalable QoS Prediction for Runtime Service Adaptation. In *2014 IEEE 34th International Conference on Distributed Computing Systems*, pages 318–327. IEEE, jun 2014. ISBN 978-

1-4799-5169-7. doi: 10.1109/ICDCS.2014.40. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6888908.

[124] Dalaijargal Purevsuren, Saif Ur Rehman, Gang Cui, Nwe Nwe Htay Win, and Bao Jian-Min. Cone Dominance-Based Interactive Evolutionary Multiobjective Algorithm for QoS-Driven Service Selection Problem. In *2014 IEEE 17th International Conference on Computational Science and Engineering*, pages 940–945. IEEE, dec 2014. ISBN 978-1-4799-7981-3. doi: 10.1109/CSE.2014.189. URL http://ieeexplore.ieee.org/document/7023699/.

[125] Klaithem Al Nuaimi, Nader Mohamed, Mariam Al Nuaimi, and Jameela Al-Jaroodi. A Survey of Load Balancing in Cloud Computing: Challenges and Algorithms. *2012 Second Symposium on Network Cloud Computing and Applications*, pages 137–142, dec 2012. doi: 10.1109/NCCA.2012.29. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6472470.

[126] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Communications of the ACM*, 51(1):107, 2008. ISSN 00010782. doi: 10.1145/1327452.1327492. URL http://dl.acm.org/citation.cfm?id=1327452.1327492{%}5Cnhttp://portal.acm.org/citation.cfm?doid=1327452.1327492.

[127] Hai Zhong, Kun Tao, and Xuejie Zhang. An approach to optimized resource scheduling algorithm for open-source cloud systems. In *ChinaGrid Conference (ChinaGrid), 2010 Fifth Annual*, pages 124–129. IEEE, 2010.

[128] Shuo Liu, Gang Quan, and Shangping Ren. On-line scheduling of real-time services for cloud computing. In *Services (SERVICES-1), 2010 6th World Congress on*, pages 459–464. IEEE, 2010.

[129] W.t. Tsai, Yann-hang Lee, Zhibin Cao, Yinong Chen, and Bingnan Xiao. RTSOA: Real-Time Service-Oriented Architecture. In *IEEE International Symposium on Service-Oriented System Engineering (SOSE'06)*, pages 49–56. Ieee, oct 2006. ISBN 0-7695-2726-4. doi: 10.1109/SOSE.2006.27. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4027117.

[130] Dionysios Efstathiou, Peter McBurney, Steffen Zschaler, and Johann Bourcier. Flexible QoS-aware service composition in highly heterogeneous and dynamic service-based systems. In *2013 IEEE 9th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, volume 2013, pages 592–599. IEEE, oct 2013. ISBN 978-1-4799-0428-0. doi: 10.1109/WiMOB.2013.6673418. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6673418.

[131] Duncan Russell, Lu Liu, Zongyang Luo, Colin Venters, David Webster, and Jie Xu. Realizing Network Enabled Capability Through Dependable Dynamic Systems Integration. In *2010 10th IEEE International Conference on Computer and Information Technology*, number Cit, pages 1269–1274. IEEE, jun 2010. ISBN 978-1-4244-7547-6. doi: 10.1109/CIT.2010.229. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5577875.

[132] Lu Liu, Duncan Russell, David Webster, Zongyang Luo, Colin Venters, Jie Xu, and John K. Davies. Delivering Sustainable Capability on Evolutionary Service-oriented Architecture. *2009 IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*, pages 12–19, mar 2009. doi: 10.1109/ISORC.2009.9. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5232004.

[133] David McKee, David Webster, Paul Townend, Jie Xu, and Davd Battersby. Towards a Virtual Integration Design and Analysis Enviroment for Automotive Engineering. In *2014 IEEE 17th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*, pages 413–419. IEEE, jun 2014. ISBN 978-1-4799-4430-9. doi: 10.1109/ISORC.2014.38. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6899178http://ieeexplore.ieee.org/document/6899178/.

[134] Markus Fiedler, Tobias Hossfeld, and Phuoc Tran-Gia. A generic quantitative relationship between quality of experience and quality of service. *IEEE Network*, 24 (2):36–41, mar 2010. ISSN 0890-8044. doi: 10.1109/MNET.2010.5430142. URL

http://ieeexplore.ieee.org/stamp/stamp.jsp?tp={&}arnumber=5430142http://ieeexplore.ieee.org/document/5430142/.

[135] Iria Estévez-Ayres, Marisol García-Valls, Pablo Basanta-Val, and Jorge Díez-Sánchez. A hybrid approach for selecting service-based real-time composition algorithms in heterogeneous environments. *Concurrency and Computation: Practice and Experience*, 23 (15):1816–1851, oct 2011. ISSN 15320626. doi: 10.1002/cpe.1766. URL http://doi.wiley.com/10.1002/cpe.1766.

[136] Iria Estévez-Ayres, Marisol García-Valls, and Pablo Basanta-Val. Enabling WCET-based composition of service-based real-time applications. *ACM SIGBED Review*, 2(3):25–29, jul 2005. ISSN 15513688. doi: 10.1145/1121802.1121808. URL http://portal.acm.org/citation.cfm?doid=1121802.1121808.

[137] M. Garcia-Valls, R. Fern'ndez-Castro, I. Estevez-Ayres, P. Basanta-Val, and I. Rodriguez-Lopez. A Bounded-time Service Composition Algorithm for Distributed Real-time Systems. *2012 IEEE 14th International Conference on High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems*, pages 1413–1420, jun 2012. doi: 10.1109/HPCC.2012.207. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6332343.

[138] Iria Estévez-Ayres, Pablo Basanta-val, Marisol García-valls, Jesús A Fisteus, and Luís Almeida. QoS-Aware Real-Time Composition Algorithms for Service-Based Applications. *IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS*, 5(3):278–288, 2009.

[139] Marisol García-Valls, Patricia Uriol-Resuela, Felipe Ibáñez-Vázquez, and Pablo Basanta-Val. Low complexity reconfiguration for real-time data-intensive service-oriented applications. *Future Generation Computer Systems*, 37:191–200, 2014. ISSN 0167739X. doi: 10.1016/j.future.2013.10.019. URL http://dx.doi.org/10.1016/j.future.2013.10.019.

[140] Rosbel Serrano-Torres, Marisol García-Valls, and Pablo Basanta-Val. Virtualizing DDS middleware: Performance challenges and measurements. *IEEE International Conference*

on Industrial Informatics (INDIN), pages 71–76, 2013. ISSN 19354576. doi: 10.1109/INDIN.2013.6622860.

[141] Marisol García Valls and Pablo Basanta Val. Comparative analysis of two different middleware approaches for reconfiguration of distributed real-time systems. *Journal of Systems Architecture*, 60:221–233, 2014. ISSN 13837621. doi: 10.1016/j.sysarc.2013.08.010.

[142] Marisol García-valls, Pablo Basanta-val, and Iria Estévez-Ayres. Supporting Service Composition and Real-Time Execution through Characterisation of QoS Properties. In *SEAMS '11 Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 110–117, 2011. ISBN 9781450305754.

[143] Marisol Garcia Valls, Iago Rodríguez Lopez, and Laura Fernández Villar. iLAND: An Enhanced Middleware for Real-Time Reconfiguration of Service Oriented Distributed Real-Time Systems. *IEEE Transactions on Industrial Informatics*, 9(1):228–236, feb 2013. ISSN 1551-3203. doi: 10.1109/TII.2012.2198662. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6198329`.

[144] Iria Estévez-Ayres, Luís Almeida, Marisol García-Valls, and Pablo Basanta-Val. An Architecture to Support Dynamic Service Composition in Distributed Real-Time Systems. *IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, 10, 2007.

[145] Erol ahin. Swarm Robotics: From Sources of Inspiration to Domains of Application. volume 8, pages 10–20. 2005. doi: 10.1007/978-3-540-30552-1_2. URL `http://link.springer.com/10.1007/978-3-540-30552-1{_}2`.

[146] Zibin Zheng, Hao Ma, Michael R. Lyu, and Irwin King. QoS-Aware Web Service Recommendation by Collaborative Filtering. *IEEE Transactions on Services Computing*, 4(2):140–152, apr 2011. ISSN 1939-1374. doi: 10.1109/TSC.2010.52. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5674010`.

[147] Zibin Zheng, Hao Ma, Michael R. Lyu, and Irwin King. Collaborative Web Service QoS Prediction via Neighborhood Integrated Matrix Factorization. *IEEE Transactions on Services Computing*, 6(3):289–299, jul 2013. ISSN 1939-1374. doi: 10.1109/TSC.2011.

59. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6122009.

[148] Xinyu Wang, Jianke Zhu, Zibin Zheng, Wenjie Song, Yuanhong Shen, and Michael R Lyu. A Spatial-Temporal QoS Prediction Approach for Time-aware Web Service Recommendation. *ACM Transactions on the Web*, 10(1):1–25, feb 2016. ISSN 15591131. doi: 10.1145/2801164. URL http://dl.acm.org/citation.cfm?doid=2870642.2801164.

[149] Stan Schneider. What Is Real-Time SOA? *Real-Time Innovations*, (June), 2010.

[150] Héctor Pérez and J. Javier Gutiérrez. Modeling the QoS parameters of DDS for event-driven real-time applications. *Journal of Systems and Software*, 104:126–140, jun 2015. ISSN 01641212. doi: 10.1016/j.jss.2015.03.008. URL http://linkinghub.elsevier.com/retrieve/pii/S016412121500059X.

[151] Abdelzahir Abdelmaboud, Dayang N.A. Jawawi, Imran Ghani, Abubakar Elsafi, and Barbara Kitchenham. Quality of service approaches in cloud computing: A systematic mapping study. *Journal of Systems and Software*, 101:159–179, 2015. ISSN 01641212. doi: 10.1016/j.jss.2014.12.015. URL http://linkinghub.elsevier.com/retrieve/pii/S0164121214002830.

[152] Hong-linh Truong, Robert Samborski, and Thomas Fahringer. Towards a Framework for Monitoring and Analyzing QoS Metrics of Grid Services. In *2006 Second IEEE International Conference on e-Science and Grid Computing (e-Science'06)*, pages 65–65. IEEE, dec 2006. ISBN 0-7695-2734-5. doi: 10.1109/E-SCIENCE.2006.261149. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4031038http://ieeexplore.ieee.org/document/4031038/.

[153] Hei-Chia Wang, Wei-Pin Chiu, and Suei-Chih Wu. QoS-driven selection of web service considering group preference. *Computer Networks*, 93:111–124, dec 2015. ISSN 13891286. doi: 10.1016/j.comnet.2015.10.014. URL http://dx.doi.org/10.1016/j.comnet.2015.10.014http://linkinghub.elsevier.com/retrieve/pii/S1389128615003874.

[154] Hejiao Huang and Yun Peng. Throughput Maximization with Traffic Profile in Wireless Mesh Network. In *Computing and Combinatorics*, pages 531–540. Springer Berlin Heidelberg, Berlin, Heidelberg. doi: 10.1007/978-3-540-69733-6_52. URL http://link.springer.com/10.1007/978-3-540-69733-6{_}52.

[155] Sukhpal Singh and Inderveer Chana. Resource provisioning and scheduling in clouds: QoS perspective. *The Journal of Supercomputing*, 72(3):926–960, mar 2016. ISSN 0920-8542. doi: 10.1007/s11227-016-1626-x. URL http://link.springer.com/10.1007/s11227-016-1626-x.

[156] Ismael Solis Moreno, Peter Garraghan, Paul Townend, and Jie Xu. An Approach for Characterizing Workloads in Google Cloud to Derive Realistic Resource Utilization Models. *2013 IEEE Seventh International Symposium on Service-Oriented System Engineering*, pages 49–60, mar 2013. doi: 10.1109/SOSE.2013.24. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6525503.

[157] Jeffrey Dean and Luiz André Barroso. The tail at scale. *Communications of the ACM*, 56 (2):74, feb 2013. ISSN 00010782. doi: 10.1145/2408776.2408794. URL http://dl.acm.org/citation.cfm?doid=2408776.2408794.

[158] Christoph Fehling, Frank Leymann, Ralph Retter, Walter Schupeck, and Peter Arbitter. *Cloud Computing Patterns*. Springer Vienna, Vienna, 2014. ISBN 978-3-7091-1567-1. doi: 10.1007/978-3-7091-1568-8. URL http://link.springer.com/10.1007/978-3-7091-1568-8http://www.cloudcomputingpatterns.org/.

[159] Joost Bosman, Hans van den Berg, and Rob van der Mei. Real-Time QoS Control for Service Orchestration. In *2015 27th International Teletraffic Congress*, pages 152–158. IEEE, sep 2015. ISBN 978-1-4673-8422-3. doi: 10.1109/ITC.2015.25. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7277438.

[160] Gerardo Canfora, Massimiliano Di Penta, Raffaele Esposito, and Maria Luisa Villani. An approach for QoS-aware service composition based on genetic algorithms. In *Proceedings of the 2005 conference on Genetic and evolutionary computation - GECCO '05*, pages 1069–1075, New York, New York, USA, 2005. ACM Press. ISBN 1595930108. doi: 10.

1145/1068009.1068189. URL http://portal.acm.org/citation.cfm?doid=1068009.1068189.

[161] Rajinder Sandhu and Sandeep K Sood. Scheduling of big data applications on distributed cloud based on QoS parameters. *Cluster Computing*, 18(2):817–828, jun 2015. ISSN 1386-7857. doi: 10.1007/s10586-014-0416-6. URL http://link.springer.com/10.1007/s10586-014-0416-6.

[162] Kwei-Jay Lin, Mark Panahi, Yue Zhang, Jing Zhang, and Soo-Ho Chang. Building Accountability Middleware to Support Dependable SOA. *IEEE Internet Computing*, 13(2):16–25, mar 2009. ISSN 1089-7801. doi: 10.1109/MIC.2009.28. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4797932http://ieeexplore.ieee.org/document/4797932/.

[163] Tommaso Cucinotta, Kleopatra Konstanteli, and Theodora Varvarigou. Advance reservations for distributed real-time workflows with probabilistic service guarantees. In *2009 IEEE International Conference on Service-Oriented Computing and Applications (SOCA)*, volume 00, pages 1–8. IEEE, dec 2009. ISBN 978-1-4244-5300-9. doi: 10.1109/SOCA.2009.5410268. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5410268.

[164] Tript Kaur, Damandeep Kaur, and Ashish Aggarwal. Cost model for software as a service. In *2014 5th International Conference - Confluence The Next Generation Information Technology Summit (Confluence)*, pages 736–741. IEEE, sep 2014. ISBN 978-1-4799-4236-7. doi: 10.1109/CONFLUENCE.2014.6949281. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6949281http://ieeexplore.ieee.org/document/6949281/.

[165] Ismail Assayad. Joint SW/HW Modelling and Design Exploration Using P-Ware. In *2008 32nd Annual IEEE International Computer Software and Applications Conference*, pages 1341–1346. IEEE, 2008. ISBN 978-0-7695-3262-2. doi: 10.1109/COMPSAC.2008.22. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4591777http://ieeexplore.ieee.org/document/4591777/.

[166] Peter Garraghan, David McKee, Xue Ouyang, David Webster, and Jie Xu. SEED: A

Scalable Approach for Cyber-Physical System Simulation. *IEEE Transactions on Services Computing*, 9(2):199–212, mar 2016. ISSN 1939-1374. doi: 10.1109/TSC.2015.2491287. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7298453http://ieeexplore.ieee.org/document/7298453/.

[167] Peter Garraghan, Xue Ouyang, Renyu Yang, David McKee, and Jie Xu. Straggler Root-Cause and Impact Analysis for Massive-scale Virtualized Cloud Datacenters. *IEEE Transactions on Services Computing*, 1374(c):1–1, 2016. ISSN 1939-1374. doi: 10.1109/TSC.2016.2611578. URL http://ieeexplore.ieee.org/document/7572191/.

[168] Xue Ouyang, Peter Garraghan, David McKee, Paul Townend, and Jie Xu. Straggler Detection in Parallel Computing Systems through Dynamic Threshold Calculation. In *2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA)*, pages 414–421. IEEE, mar 2016. ISBN 978-1-5090-1858-1. doi: 10.1109/AINA.2016.84. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7474119http://ieeexplore.ieee.org/document/7474119/.

[169] A Albatli, D McKee, P Townend, L Lau, and J Xu. Prov-te: A provenance-driven diagnostic framework for task eviction in data centers, 2017.

[170] David W McKee, Stephen Clement, Jie Xu, and David Battersby. n-Dimensional Qos Framework for Real-Time Service-Oriented Architectures. *IEEE International Symposium on Real-time Data Processing for Cloud Computing*, 2017.

[171] Rodrigo N Calheiros, Rajiv Ranjan, Anton Beloglazov, César A. F. De Rose, and Rajkumar Buyya. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, 41(1):23–50, jan 2011. ISSN 00380644. doi: 10.1002/spe.995. URL http://doi.wiley.com/10.1002/spe.995.

[172] Donald E Knuth. Two notes on notation. *American Mathematical Monthly*, 99(5):403–422, 1992. ISSN 00029890. doi: 10.2307/2325085. URL http://arxiv.org/abs/math/9205211.

[173] Xue Ouyang, Peter Garraghan, Changjian Wang, Paul Townend, and Jie Xu. An Approach for Modeling and Ranking Node-Level Stragglers in Cloud Datacenters. In *2016 IEEE International Conference on Services Computing (SCC)*, pages 673–680. IEEE, jun 2016. ISBN 978-1-5090-2628-9. doi: 10.1109/SCC.2016.93. URL http://ieeexplore.ieee.org/document/7557513/.

[174] Bernhard Primas, Peter Garraghan, Karim Djemame, and Natasha Shakhlevich. Resource boxing: Converting realistic cloud task utilization patterns for theoretical scheduling. In *Proceedings of the 9th International Conference on Utility and Cloud Computing - UCC '16*, pages 138–147, New York, New York, USA, 2016. ACM Press. ISBN 9781450346160. doi: 10.1145/2996890.2996897. URL http://dl.acm.org/citation.cfm?doid=2996890.2996897.

[175] NIST. NIST Cloud Computing Reference Architecture. Technical report, 2011. URL http://collaborate.nist.gov/twiki-cloud-computing/pub/CloudComputing/Meeting12AReferenceArchitectureMarch282011/NIST{_}CCRATWG{_}029.pdf.

[176] Bernard Rosner. Percentage points for a generalized esd many-outlier procedure. *Technometrics*, 25(2):165–172, 1983.

[177] Xiong Luo, Hao Luo, and Xiaohui Chang. Online Optimization of Collaborative Web Service QoS Prediction Based on Approximate Dynamic Programming. *International Journal of Distributed Sensor Networks*, 2015:1–9, 2015. ISSN 1550-1329. doi: 10.1155/2015/452492. URL http://www.hindawi.com/journals/ijdsn/2015/452492/.

[178] DW Mckee, SJ Clement, X Ouyang, J Xu, R Romano, and J Davies. The Internet of Simulation, a Specialisation of the Internet of Things with Simulation and Workflow as a Service (SIM/WFaaS). 2017. URL http://eprints.whiterose.ac.uk/111418/.

[179] DW Mckee, SJ Clement, J Almutairi, and J Xu. Massive-Scale Automation in Cyber-Physical Systems: Vision & Challenges. 2017. URL http://eprints.whiterose.ac.uk/112214/.

[180] SJ Clement, DW Mckee, and J Xu. Service-Oriented Reference Architecture for Smart Cities. 2017. URL http://eprints.whiterose.ac.uk/113342/.

[181] T Blochwitz, M Otter, M Arnold, C Bausch, C Clauß, H Elmqvist, A Junghanns, J Mauss, M Monteiro, T Neidhold, D Neumerkel, H Olsson, J Peetz, and S Wolf. The Functional Mockup Interface for Tool independent Exchange of Simulation Models The Functional Mock-Up Interface. In *Modelica'2011 Conference*, pages 105–114, 2011.

[182] Wei Xiong and Wei-Tek Tsai. HLA-Based SaaS-Oriented Simulation Frameworks. *2014 IEEE 8th International Symposium on Service Oriented System Engineering*, pages 376–383, apr 2014. doi: 10.1109/SOSE.2014.74. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6830933`.

[183] Yan Hu, Qimin Peng, Xiaohui Hu, and Rong Yang. Web Service Recommendation Based on Time Series Forecasting and Collaborative Filtering. In *2015 IEEE International Conference on Web Services*, pages 233–240. IEEE, jun 2015. ISBN 978-1-4673-7272-5. doi: 10.1109/ICWS.2015.40. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7195574`.

[184] Balika. J. Chelliah and K Vivekanandan. Aggregated Quantified Response Time Matrix Formulation (ARMF) - A New Quality of Service Paradigm Technique. *Journal of Software*, 10(9):1070–1078, 2015. ISSN 1796217X. doi: 10.17706/jsw.10.9.1070-1078. URL `http://www.jsoftware.us/index.php?m=content{&}c=index{&}a=show{&}catid=156{&}id=2524`.

[185] Raed Karim, Chen Ding, Ali Miri, and Md Shahinur Rahman. End-to-End QoS Prediction Model of Vertically Composed Cloud Services via Tensor Factorization. In *2015 International Conference on Cloud and Autonomic Computing*, pages 158–168. IEEE, sep 2015. ISBN 978-1-4673-9566-3. doi: 10.1109/ICCAC.2015.29. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7312150http://ieeexplore.ieee.org/document/7312150/`.

[186] Sayali Narkhede and Praveenkumar Keskar. Web service Recommendation by combining QOS and user comments. *International Journal on Recent and Innovation Trends in Computing and Communication*, 3(8):5215–5219, 2015.

[187] Wen'an Tan, Le'er Li, and Yong Sun. Performance Prediction and Analysis of Quality of Services for Cross-Organizational Workflows. In *2014 IEEE 11th Internation-*

al Conference on e-Business Engineering, pages 145–150. IEEE, nov 2014. ISBN 978-1-4799-6563-2. doi: 10.1109/ICEBE.2014.34. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6982072http://ieeexplore.ieee.org/document/6982072/.

[188] Wei Lo, Jianwei Yin, Ying Li, and Zhaohui Wu. Efficient web service QoS prediction using local neighborhood matrix factorization. *Engineering Applications of Artificial Intelligence*, 38(0):14–23, feb 2015. ISSN 09521976. doi: 10.1016/j.engappai.2014.10.010. URL http://www.sciencedirect.com/science/article/pii/S0952197614002504http://linkinghub.elsevier.com/retrieve/pii/S0952197614002504.

[189] Yao Zhao, Qi Pi, Chengduo Luo, and Danfeng Yan. CAPred: A Prediction Model for Timely QoS. In *2015 IEEE International Conference on Web Services*, pages 599–606. IEEE, jun 2015. ISBN 978-1-4673-7272-5. doi: 10.1109/ICWS.2015.85. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7195620.

[190] Xiong Luo, Ji Liu, Dandan Zhang, and Xiaohui Chang. A large-scale web QoS prediction scheme for the Industrial Internet of Things based on a kernel machine learning algorithm. *Computer Networks*, 101:81–89, jun 2016. ISSN 13891286. doi: 10.1016/j.comnet.2016.01.004. URL http://linkinghub.elsevier.com/retrieve/pii/S1389128616000189.

[191] N Arulanand and P M Ananth. Optimized Weight Vector for QoS Aware Web Service Selection Algorithm Using Particle Swarm Optimization. *International Journal of Computer, Electrical, Automation, Control and Information Engineering*, 9(4):11–16, 2015.

[192] Valeria Cardellini, Emiliano Casalicchio, Vincenzo Grassi, Stefano Iannucci, Francesco Lo Presti, and Raffaela Mirandola. MOSES: A Framework for QoS Driven Runtime Adaptation of Service-Oriented Systems. *IEEE Transactions on Software Engineering*, 38(5):1138–1159, sep 2012. ISSN 0098-5589. doi: 10.1109/TSE.2011.68. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5963694http://ieeexplore.ieee.org/document/5963694/.

[193] S Canale, F. Delli Priscoli, S. Monaco, L Palagi, and V Suraci. A reinforcement learning approach for QoS/QoE model identification. In *2015 34th Chinese Control Conference (CCC)*, pages 2019–2023. IEEE, jul 2015. ISBN 978-9-8815-6389-7. doi: 10.1109/ChiCC.2015.7259941. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7259941.

[194] Jinkyu Lee and Insik Shin. Intermediate Deadline Assignment for Distributed Real-Time Systems : Utility Maximization and Challenges. In *Feedback Computing*, 2012.

[195] Sen Luo, Bin Xu, and Yixin Yan. An accumulated-QoS-first search approach for semantic web service composition. In *2010 IEEE International Conference on Service-Oriented Computing and Applications (SOCA)*, pages 1–4. IEEE, dec 2010. ISBN 978-1-4244-9802-4. doi: 10.1109/SOCA.2010.5707191. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5707191http://ieeexplore.ieee.org/document/5707191/.

[196] Fateh Seghir and Abdellah Khababa. A hybrid approach using genetic and fruit fly optimization algorithms for QoS-aware cloud service composition. *Journal of Intelligent Manufacturing*, apr 2016. ISSN 0956-5515. doi: 10.1007/s10845-016-1215-0. URL http://link.springer.com/10.1007/s10845-016-1215-0.

[197] Yang Syu, Yong-Yi Fanjiang, Jong-Yih Kuo, and Shang-Pin Ma. Applying Genetic Programming for Time-Aware Dynamic QoS Prediction. In *2015 IEEE International Conference on Mobile Services*, volume 32, pages 217–224. IEEE, jun 2015. ISBN 978-1-4673-7284-8. doi: 10.1109/MobServ.2015.39. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7057607http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7226693http://ieeexplore.ieee.org/document/7226693/.

[198] Yang Syu, Yong-Yi Fanjiang, Jong-Yih Kuo, and Jui-Lung Su. Quality of Service timeseries forecasting for Web Services: A machine learning, Genetic Programming-based approach. In *2016 Annual Conference on Information Science and Systems (CISS)*, pages 343–348. IEEE, mar 2016. ISBN 978-1-4673-9457-4. doi: 10.1109/CISS.2016.7460526. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7460526http://ieeexplore.ieee.org/document/7460526/.

[199] Tao Yu and Kwei-jay Lin. Service Selection Algorithms for Composing Complex Services with Multiple QoS Constraints. In *Service-Oriented Computing - ICSOC 2005*, volume 3826, pages 130–143. 2005. doi: 10.1007/11596141_11. URL http://link.springer.com/10.1007/11596141{_}11.

[200] Tao Yu, Yue Zhang, and Kwei-Jay Lin. Efficient algorithms for Web services selection with end-to-end QoS constraints. *ACM Transactions on the Web*, 1(1):6–es, may 2007. ISSN 15591131. doi: 10.1145/1232722.1232728. URL http://portal.acm.org/citation.cfm?doid=1232722.1232728.

[201] Xin Yu, Thomas Weise, Ke Tang, and Steffen Bleul. QoS-aware semantic web service composition for SOAs. In *2010 IEEE International Conference on Service-Oriented Computing and Applications (SOCA)*, pages 1–4. IEEE, dec 2010. ISBN 978-1-4244-9802-4. doi: 10.1109/SOCA.2010.5707192. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5707192http://ieeexplore.ieee.org/document/5707192/.

[202] Luca Abeni and Tommaso Cucinotta. Efficient virtualisation of real-time activities. *2011 IEEE International Conference on Service-Oriented Computing and Applications (SOCA)*, pages 1–4, dec 2011. doi: 10.1109/SOCA.2011.6166248. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6166248.

[203] Danilo Ardagna, Raffaela Mirandola, Marco Trubian, and Li Zhang. Run-time resource management in SOA virtualized environments. *Proceedings of the 1st international workshop on Quality of service-oriented software systems - QUASOSS '09*, page 39, 2009. doi: 10.1145/1596473.1596484. URL http://portal.acm.org/citation.cfm?doid=1596473.1596484.

[204] Tommaso Cucinotta, Gaetano Anastasi, and Luca Abeni. Respecting Temporal Constraints in Virtualised Services. In *2009 33rd Annual IEEE International Computer Software and Applications Conference*, pages 73–78. IEEE, 2009. ISBN 978-0-7695-3726-9. doi: 10.1109/COMPSAC.2009.118. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5254145http://ieeexplore.ieee.org/document/5254145/.

[205] Tommaso Cucinotta and Gaetano F Anastasi. A heuristic for optimum allocation of real-time service workflows. In *2011 IEEE International Conference on Service-Oriented Computing and Applications (SOCA)*, pages 1–4. IEEE, dec 2011. ISBN 978-1-4673-0319-4. doi: 10.1109/SOCA.2011.6166216. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6166216http://ieeexplore.ieee.org/document/6166216/.

[206] Gaetano F. Anastasi, Tommaso Cucinotta, Giuseppe Lipari, and Marisol Garcia-Valls. A QoS registry for adaptive real-time service-oriented applications. In *2011 IEEE International Conference on Service-Oriented Computing and Applications (SOCA)*, pages 1–8. IEEE, dec 2011. ISBN 978-1-4673-0319-4. doi: 10.1109/SOCA.2011.6166243. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6166243.

[207] Y. Feng, W. Zhijian, and H. Qian. A novel QoS-aware mechanism for provisioning of virtual machine resource in cloud. *Journal of Algorithms & Computational Technology*, 0(0):1–7, may 2016. ISSN 1748-3018. doi: 10.1177/1748301816649077. URL http://act.sagepub.com/lookup/doi/10.1177/1748301816649077.

[208] Mark Panahi, Weiran Nie, and Kwei-Jay Lin. The Design and Implementation of Service Reservations in Real-Time SOA. In *2009 IEEE International Conference on e-Business Engineering*, pages 129–136. IEEE, 2009. ISBN 978-0-7695-3842-6. doi: 10.1109/ICEBE.2009.26. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5342122http://ieeexplore.ieee.org/document/5342122/.

[209] Mark Panahi, Weiran Nie, and Kwei-Jay Lin. A Framework for Real-Time Service-Oriented Architecture. In *2009 IEEE Conference on Commerce and Enterprise Computing*, pages 460–467. Ieee, jul 2009. ISBN 978-0-7695-3755-9. doi: 10.1109/CEC.2009.78. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5210760.

[210] Jing Zhang, Yi-chin Chang, and Kwei-jay Lin. A dependency matrix based framework for QoS diagnosis in SOA. In *2009 IEEE International Conference on Service-Oriented*

Computing and Applications (SOCA), volume 00, pages 1–8. IEEE, dec 2009. ISBN 978-1-4244-5300-9. doi: 10.1109/SOCA.2009.5410261. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5410261.

[211] Jing Zhang, Xiaoqi Zhang, and Kwei-Jay Lin. An efficient Bayesian diagnosis for QoS management in service-oriented architecture. *2011 IEEE International Conference on Service-Oriented Computing and Applications (SOCA)*, pages 1–8, dec 2011. doi: 10.1109/SOCA.2011.6166214. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6166214.

[212] Weiran Nie, Sen Zhou, and Kwei-jay Lin. Real-time service process scheduling with intermediate deadline overrun management. In *2012 Fifth IEEE International Conference on Service-Oriented Computing and Applications (SOCA)*, pages 1–8. IEEE, dec 2012. ISBN 978-1-4673-4775-4. doi: 10.1109/SOCA.2012.6449453. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6449453.

[213] Ridha Benosman, Yves Albrieux, and Kamel Barkaoui. Performance evaluation of a massively parallel ESB-oriented architecture. *2012 Fifth IEEE International Conference on Service-Oriented Computing and Applications (SOCA)*, pages 1–4, dec 2012. doi: 10.1109/SOCA.2012.6449435. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6449435.

[214] Shaowei Liu, Kaijun Ren, Kefeng Deng, and Junqiang Song. A dynamic resource allocation and task scheduling strategy with uncertain task runtime on IaaS clouds. In *2016 Sixth International Conference on Information Science and Technology (ICIST)*, pages 174–180. IEEE, may 2016. ISBN 978-1-5090-1224-4. doi: 10.1109/ICIST.2016.7483406. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7483406http://ieeexplore.ieee.org/document/7483406/.

[215] Andres Quiroz, Hyunjoo Kim, Manish Parashar, Nathan Gnanasambandam, and Naveen Sharma. Towards autonomic workload provisioning for enterprise Grids and clouds. In *2009 10th IEEE/ACM International Conference on Grid Computing*, pages 50–57. IEEE, oct 2009. ISBN 978-1-4244-5148-7. doi: 10.1109/GRID.2009.5353066. URL http://ieeexplore.ieee.org/document/5353066/.

[216] Georgios L. Stavrinides and Helen D. Karatza. A Cost-Effective and QoS-Aware Approach to Scheduling Real-Time Workflow Applications in PaaS and SaaS Clouds. In *2015 3rd International Conference on Future Internet of Things and Cloud*, pages 231–239. IEEE, aug 2015. ISBN 978-1-4673-8103-1. doi: 10.1109/FiCloud.2015.93. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7300823.

[217] Michael Boniface, Bassem Nasser, Juri Papay, Stephen C. Phillips, Arturo Servin, Xiaoyu Yang, Zlatko Zlatev, Spyridon V. Gogouvitis, Gregory Katsaros, Kleopatra Konstanteli, George Kousiouris, Andreas Menychtas, and Dimosthenis Kyriazis. Platform-as-a-Service Architecture for Real-Time Quality of Service Management in Clouds. *2010 Fifth International Conference on Internet and Web Applications and Services*, pages 155–160, 2010. doi: 10.1109/ICIW.2010.91. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5476775.

[218] Zhenghua Fu, Chai Wah Wu, Jun-Jang Jeng, and Hui Lei. PACTS: A Service Oriented Architecture for Real-Time Peer-Assisted Content Delivery Service. In *2008 32nd Annual IEEE International Computer Software and Applications Conference*, pages 1205–1210. IEEE, 2008. ISBN 978-0-7695-3262-2. doi: 10.1109/COMPSAC.2008.153. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4591749http://ieeexplore.ieee.org/document/4591749/.

[219] Luis Garces-Erice. Building an Enterprise Service Bus for Real-Time SOA: A Messaging Middleware Stack. In *2009 33rd Annual IEEE International Computer Software and Applications Conference*, pages 79–84. Ieee, 2009. ISBN 978-0-7695-3726-9. doi: 10.1109/COMPSAC.2009.119. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5254146.

[220] Steffen Pr, Guido Moritz, Elmar Zeeb, Ralf Salomon, Frank Golatowski, and Dirk Timmermann. Applicability of Web Service Technologies to Reach Real Time Capabilities. In *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, pages 229–233. IEEE, may 2008. ISBN 978-0-7695-3132-8. doi: 10.1109/ISORC.2008.

30. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4519582http://ieeexplore.ieee.org/document/4519582/.

[221] Abdel-Rahman Al-Ghuwairi, Mohammad N. Khalaf, Laith Al-Yasen, Zaher Salah, Ayoub Alsarhan, and Aladdin Hussein Baarah. A Dynamic Model for automatic Updating cloud computing SLA (DSLA). In *Proceedings of the International Conference on Internet of things and Cloud Computing - ICC '16*, pages 1–7, New York, New York, USA, 2016. ACM Press. ISBN 9781450340632. doi: 10.1145/2896387.2896442. URL http://dl.acm.org/citation.cfm?doid=2896387.2896442.

[222] Xiaohua Guo, Jing Jiang, and Xuefei Li. A New Method to QoS Global Optimal Service Selection Driven by Credit. In *Proceedings of the 2015 International Conference on Materials Engineering and Information Technology Applications*, number Meita, pages 785–792, Paris, France, 2015. Atlantis Press. ISBN 978-94-6252-103-2. doi: 10.2991/meita-15.2015.144. URL http://www.atlantis-press.com/php/paper-details.php?id=25838570.

[223] Tian Huat Tan, Manman Chen, Jun Sun, Yang Liu, Étienne André, Yinxing Xue, and Jin Song Dong. Optimizing selection of competing services with probabilistic hierarchical refinement. In *Proceedings of the 38th International Conference on Software Engineering - ICSE '16*, pages 85–95, New York, New York, USA, 2016. ACM Press. ISBN 9781450339001. doi: 10.1145/2884781.2884861. URL http://dl.acm.org/citation.cfm?doid=2884781.2884861.

[224] Yutu Liu, Anne H Ngu, and Liang Z Zeng. QoS computation and policing in dynamic web service selection. In *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters - WWW Alt. '04*, page 66, New York, New York, USA, 2004. ACM Press. ISBN 1581139128. doi: 10.1145/1013367.1013379. URL http://doi.acm.org/10.1145/1013367.1013379http://portal.acm.org/citation.cfm?doid=1013367.1013379.

[225] Nguyen Cao Hong Ngoc, Donghui Lin, Takao Nakaguchi, and Toru Ishida. QoS-Aware Service Composition in Mobile Environments. In *2014 IEEE 7th International Conference on Service-Oriented Computing and Applications*, pages 97–104. IEEE, nov 2014.

ISBN 978-1-4799-6833-6. doi: 10.1109/SOCA.2014.51. URL http://ieeexplore.ieee.org/document/6978596/.

[226] Divya Sachan, Saurabh Kumar Dixit, and Sandeep Kumar. QOS Aware Formalized Model for Semantic Web Service Selection. *International journal of Web & Semantic Technology*, 5(4):83–100, oct 2014. ISSN 09762280. doi: 10.5121/ijwest.2014.5406. URL http://airccse.org/journal/ijwest/papers/5414ijwest06.pdf.

[227] Radu Calinescu, Lars Grunske, Marta Kwiatkowska, Raffaela Mirandola, and Giordano Tamburrelli. Dynamic QoS Management and Optimization in Service-Based Systems. *IEEE Transactions on Software Engineering*, 37(3):387–409, may 2011. ISSN 0098-5589. doi: 10.1109/TSE.2010.92. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5611553.

[228] Tong Gao, Hachem Moussa, I-Ling Yen, Farokh Bastani, and Jun-Jang Jeng. Service Composition for Real-Time Assurance. *2008 32nd Annual IEEE International Computer Software and Applications Conference*, pages 1174–1179, 2008. doi: 10.1109/COMPSAC.2008.183. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4591744.

[229] Hiranya Jayathilaka, Chandra Krintz, and Rich Wolski. Response time service level agreements for cloud-hosted web applications. In *Proceedings of the Sixth ACM Symposium on Cloud Computing - SoCC '15*, pages 315–328, New York, New York, USA, 2015. ACM Press. ISBN 9781450336512. doi: 10.1145/2806777.2806842. URL http://dl.acm.org/citation.cfm?doid=2806777.2806842.

[230] Mingkun Yang, Qimin Peng, and Xiaohui Hu. Estimating the dynamic performance of composed services: a probability theory based approach. In *Proceedings of the 1st International Workshop on Crowd-based Software Development Methods and Technologies - CrowdSoft 2014*, pages 61–66, New York, New York, USA, 2014. ACM Press. ISBN 9781450332248. doi: 10.1145/2666539.2666576. URL http://dl.acm.org/citation.cfm?doid=2666539.2666576.

[231] Zhu Yong, Li Wei, Luo Junzhou, and Zheng Xiao. A novel two-phase approach for QoS-aware service composition based on history records. In *2012 Fifth IEEE Inter-*

national Conference on Service-Oriented Computing and Applications (SOCA), pages 1–8. IEEE, dec 2012. ISBN 978-1-4673-4775-4. doi: 10.1109/SOCA.2012.6449451. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6449451http://ieeexplore.ieee.org/document/6449451/`.

[232] Saeed Zareian, Rodrigo Veleda, Marin Litoiu, Mark Shtern, Hamoun Ghanbari, and Manish Garg. K-Feed - A Data-Oriented Approach to Application Performance Management in Cloud. In *2015 IEEE 8th International Conference on Cloud Computing*, pages 1045–1048. IEEE, jun 2015. ISBN 978-1-4673-7287-9. doi: 10.1109/CLOUD.2015.148. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7214159http://ieeexplore.ieee.org/document/7214159/`.

[233] Pengcheng Zhang, Yuan Zhuang, Hareton Leung, Wei Song, and Yu Zhou. A Novel QoS Monitoring Approach Sensitive to Environmental Factors. In *2015 IEEE International Conference on Web Services*, pages 145–152. IEEE, jun 2015. ISBN 978-1-4673-7272-5. doi: 10.1109/ICWS.2015.29. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7195563`.

[234] Huiyuan Zheng, Jian Yang, and Weiliang Zhao. QoS probability distribution estimation for web services and service compositions. In *2010 IEEE International Conference on Service-Oriented Computing and Applications (SOCA)*, pages 1–8. IEEE, dec 2010. ISBN 978-1-4244-9802-4. doi: 10.1109/SOCA.2010.5707144. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5707144http://ieeexplore.ieee.org/document/5707144/`.

[235] Feng Gao, Muhammad Intizar Ali, Edward Curry, and Alessandra Mileo. QoS-aware adaptation for complex event service. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing - SAC '16*, pages 1597–1604, New York, New York, USA, 2016. ACM Press. ISBN 9781450337397. doi: 10.1145/2851613.2851806. URL `http://dl.acm.org/citation.cfm?doid=2851613.2851806`.

[236] Anthony Sargeant, Paul Townend, Jie Xu, and Karim Djemame. Evaluating the Dependability of Dynamic Binding in Web Services. In *2012 IEEE 14th International Symposium on High-Assurance Systems Engineering*, pages 139–146. IEEE, oct 2012. ISBN 978-1-

4673-4742-6. doi: 10.1109/HASE.2012.28. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6375608.

[237] Nik Looker. *Dependability analysis of web services*. PhD thesis, Durham University, 2006. URL http://etheses.dur.ac.uk/2888/.

[238] Khalid Alhamazani, Rajiv Ranjan, Prem Prakash Jayaraman, Karan Mitra, Fethi Rabhi, Dimitrios Georgakopoulos, and Lizhe Wang. Cross-Layer Multi-Cloud Real-Time Application QoS Monitoring and Benchmarking As-a-Service Framework. *IEEE Transactions on Cloud Computing*, pages 1–1, 2015. ISSN 2168-7161. doi: 10.1109/TCC.2015.2441715. URL http://ieeexplore.ieee.org/document/7126933/.

[239] Paul Buck and Qi Shi. Service Oriented Testing of Web Services. *Journal of Computer Sciences and Applications*, 3(3):21–26, 2015. doi: 10.12691/jcsa-3-3A-3.

[240] Giuseppe Cicotti, Salvatore D'Antonio, Rosario Cristaldi, and Antonio Sergio. How to Monitor QoS in Cloud Infrastructures: The QoSMONaaS Approach. In *Studies in Computational Intelligence*, volume 446, pages 253–262. 2013. ISBN 9783642325236. doi: 10.1007/978-3-642-32524-3_32. URL http://link.springer.com/10.1007/978-3-642-32524-3{_}32.

[241] Giuseppe Cicotti, Luigi Coppolino, Salvatore D Antonio, and Luigi Romano. Runtime Model Checking for SLA Compliance Monitoring and QoS Prediction. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, 6(2):4–20, 2015.

[242] Carolyn McGregor and J. Mikael Eklund. Real-Time Service-Oriented Architectures to Support Remote Critical Care: Trends and Challenges. In *2008 32nd Annual IEEE International Computer Software and Applications Conference*, pages 1199–1204. IEEE, 2008. ISBN 978-0-7695-3262-2. doi: 10.1109/COMPSAC.2008.175. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4591748http://ieeexplore.ieee.org/document/4591748/.

[243] Marcelo Silva, Fernando Lins, and Erica Sousa. Towards a Methodology for Performance Measurement of Service-Based Systems. *International Journal of Scientific & Engineering Research*, 6(4):131–137, 2015.

[244] Viktoriya Degeler, Ilce Georgievski, Alexander Lazovik, and Marco Aiello. Concept mapping for faster QoS-aware web service composition. *2010 IEEE International Conference on Service-Oriented Computing and Applications (SOCA)*, (2):1–4, dec 2010. doi: 10.1109/SOCA.2010.5707193. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5707193`.

[245] Jianxun Deng, Lihong Jiang, and Hongming Ca. Complement Service Composition through Domain Template and Requirement Context. In *2014 IEEE 11th International Conference on e-Business Engineering*, pages 320–325. IEEE, nov 2014. ISBN 978-1-4799-6563-2. doi: 10.1109/ICEBE.2014.62. URL `http://ieeexplore.ieee.org/document/6982100/`.

[246] Azlan Ismail, Jun Yan, and Jun Shen. Towards dynamic formation of temporal constraints for the service level agreements negotiation. In *2009 IEEE International Conference on Service-Oriented Computing and Applications (SOCA)*, volume 00, pages 1–8. IEEE, dec 2009. ISBN 978-1-4244-5300-9. doi: 10.1109/SOCA.2009.5410465. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5410465http://ieeexplore.ieee.org/document/5410465/`.

[247] Wei Li and William Guo. QoS prediction for dynamic reconfiguration of component based software systems. *Journal of Systems and Software*, 102:12–34, apr 2015. ISSN 01641212. doi: 10.1016/j.jss.2014.12.001. URL `http://dx.doi.org/10.1016/j.jss.2014.12.001http://linkinghub.elsevier.com/retrieve/pii/S0164121214002787`.

[248] Junio C. Lima, Ricardo C. A. Rocha, and Fabio M. Costa. An Approach for QoS-aware Selection of Shared Services for Multiple Service Choreographies. In *2016 IEEE Symposium on Service-Oriented System Engineering (SOSE)*, pages 221–230. IEEE, mar 2016. ISBN 978-1-5090-2253-3. doi: 10.1109/SOSE.2016.62. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7473029http://ieeexplore.ieee.org/document/7473029/`.

[249] F. J. Monaco, M. Nery, and M. M. L. Peixoto. An orthogonal real-time scheduling architecture for responsiveness QoS requirements in SOA environments. In *Proceedings of the 2009 ACM symposium on Applied Computing - SAC '09*, page 1990, New York, New York,

USA, 2009. ACM Press. ISBN 9781605581668. doi: 10.1145/1529282.1529724. URL http://portal.acm.org/citation.cfm?doid=1529282.1529724.

[250] Yu Sun, Jules White, Sean Eade, and Douglas C. Schmidt. ROAR: A QoS-oriented modeling framework for automated cloud resource allocation and optimization. *Journal of Systems and Software*, 116:146–161, jun 2016. ISSN 01641212. doi: 10.1016/j.jss.2015.08.006. URL http://linkinghub.elsevier.com/retrieve/pii/S0164121215001715.

[251] Le-hung Vu, Manfred Hauswirth, and Karl Aberer. QoS-Based Service Selection and Ranking with Trust and Reputation Management. Number 507483, pages 466–483. 2005. doi: 10.1007/11575771_30. URL http://link.springer.com/10.1007/11575771{_}30.

[252] Bin Xu and Yixin Yan. An efficient QoS-driven service compositon approach for large-scale service oriented systems. In *2009 IEEE International Conference on Service-Oriented Computing and Applications (SOCA)*, volume 00, pages 1–8. IEEE, dec 2009. ISBN 978-1-4244-5300-9. doi: 10.1109/SOCA.2009.5410471. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5410471http://ieeexplore.ieee.org/document/5410471/.

[253] Panfeng Xue, I-ling Yen, and Kendra M Cooper. QoS-driven dynamic adaptation in media intensive systems. In *2011 IEEE International Conference on Service-Oriented Computing and Applications (SOCA)*, pages 1–8. IEEE, dec 2011. ISBN 978-1-4673-0319-4. doi: 10.1109/SOCA.2011.6166242. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6166242http://ieeexplore.ieee.org/document/6166242/.

Appendices

Appendix A

List of Existing Real-Time Service-Oriented Architecture (RT-SOA) QoS Approaches

Table A.1: Summary of correlation-based approaches

Authors	**Summary**	**Features**
Hu, Peng, Hu, and Yang (2015) [183]	A matrix based approach identifying the relationship between services and users over time.	- Similar to Zibin Zheng et al. [146] - Users vs service over time - Service selection - Not Real-Time - Not Resource Aware
J. Chelliah and Vivekanandan (2015) [184]	A matrix based approach that aggregates QoE and predicts new QoS using the PCC.	- Similar to Zibin Zheng et al. [146] - Service Availability - Not Real-Time - Not Resource Aware
Karim, Ding, Miri, and Rahman (2015) [185]	Presents an approach that integrates QoS across different layers of the Cloud infrastructure to estimate the end-to-end response-time of a service.	- Similar to Zibin Zheng et al. [146] - Layers for IaaS & SaaS - Not Resource Aware - Not Real-Time

continued ...

. . . Table A.1 continued

Authors	Summary	Features
Narkhede and Keskar (2015) [186]	Presents a recommendation scheme for web services based on QoS and QoE data.	- Similar to Zibin Zheng et al. [146] - Service selection - Not Real-Time - Not Resource Aware
Sandhu and Sood (2015) [161]	A QoS aware scheduling approach for big data in the cloud.	- Probabilistic - Resource-Aware - Uses PCC, similar to [146] - Cost aware - Not Real-Time
Tan, Li, and Sun (2014) [187]	Analyses the correlation between service level and workflow QoS.	- Cost aware - Uses Neural Networks - Particle Swarm optimisation - Uses PCC similar to [146] - Not Real-Time - Not Resource Aware

continued . . .

. . . Table A.1 continued

Authors	**Summary**	**Features**
Wang, Zhu, Zheng, Song, Shen, and Lyu (2016) [148]	Provides an analysis of the problem of static QoS definitions and proposes a method based on past user experience to predict future QoS. Their research presents an example of services who response-times vary over a period of time.	- Service selection for workflows - User-based QoE - Least squares regression - Similar to [146] - Not Real-Time - Not Resource-Aware
Zibin Zheng, Hao Ma, Lyu, and King (2011) [146; 188; 147; 123; 6; 148; 189; 177; 190]	Presents an approach using Pearson's Correlation Coefficient (PCC) to compare the QoE of different users using PCC and Top-K factorisation. This method forms the foundation of many of the other approaches mentioned here.	- Uses PCC - Not Real-Time - Not Resource Aware

Table A.2: Summary of optimisation-based approaches

Authors	Summary	Features
Arulanand and Ananth (2015) [191]	Cyberphysical Systems in the context of SWARM particle optimisation where QoS captures the dependability attributes.	- Service selection for workflows - Uses optimisation - Not Resource-Aware
Cardellini, Casalicchio, Grassi, Iannucci, Presti, and Mirandola (2012) [192]	Presents the MOSES system which aims to optimise service selection for workflows given predefined QoS.	- Workflow re-composition - Cost [164] & reliablity - Not real-Time - Not Resource Aware
Canale, Delli Priscoli, Monaco, Palagi, and Suraci (2015) [193]	Aims to balance the fairness of the system by contrasting Quality of Service (QoS) and Quality of Experience (QoE).	- Reinforcement learning - Infrastructure agnostic - Service centric - Not Real-Time - Not Resource-Aware

continued ...

...Table A.2 continued

Authors	Summary	Features
Canfora, Di Penta, Esposito, and Villani (2005) [160]	An approach using GAs to perform constrained service composition.	- Workflow composition - Based on GAs - Based on cost [164], availability & reliability - Not Real-Time - Not Resource-Aware
Kumar, Saha, Sundaresan, and Goswami (2015) [109]	Utilises a bidding competition to allocate resources to services within a workflow.	- Uses Multi-Linear regression - Resource-Aware - Assumes full system control - Not Real-Time
Lee, Shin, and Easwaran (2012) [106; 194]	An optimisation framework to assign intermediate deadlines to tasks to maximise system utility under timing constraints.	- Traditional Schedulability - Real-Time - Resource Aware - Assumes full system control
Luo, Xu, and Yan (2010) [195]	A search mechanism for service selection accounting for QoS.	- Service selection - Not Real-Time - Not Resource Aware

continued ...

...Table A.2 continued

Authors	**Summary**	**Features**
Purevsuren, Rehman, Cui, Win, and Jian-Min (2014) [124]	A service selection approach using multi-objective optimisation	- Service selection for workflows - Optimisation approach - Considers cost and availability - Not Real-time - Not resource aware
Seghir and Khababa (2016) [196]	Adopts genetic algorithms for workflow composition and presents QoS calculations for various workflow patterns.	- Uses GA optimisation - Service selection - Cost aware - Not resource aware - Not real-time
Syu, Fanjiang, Kuo, and Ma (2015) [197; 198]	A genetic programming (GP) approach to predicting QoS.	- GP & Neural Networks - Not Real-Time - Not Resource-Aware

continued ...

...Table A.2 continued

Authors	**Summary**	**Features**
Yu and Lin (2005) [199; 200]	Broker-based method for QoS aware service composition.	- Optimisation approach - Parameter agnostic - Potentially resource-aware, cost is suggested - Service selection - Not Real-Time
Yu, Weise, Tang, and Bleul (2010) [201]	Throughput maximisation approach for workflow composition	- Domain aware using ontologies - Not Real-Time - Not Resource-Aware
Zhou, Zhang, Bastani, and Yen (2012) [64]	Application of SOA to robotic swarms whose functionality is decided using workflow composition	- Service selection for workflows - Real-Time - Not Resource-Aware

Table A.3: Summary of container-based approaches

Authors	Summary	Features
Abeni and Cucinotta (2011) [202]	Provide an approach defining a *"supply bound function"* to define a limit on the minimum amount of execution time a virtual machine will be allocated when hosting periodic real-time services.	- Resource-Aware - Real-Time - Service is black box - Use containers to control resource allocation - Assumes full OS control
Ardagna, Mirandola, Trubian, and Zhang (2009) [203]	Uses virtualisation to manage resource allocation to services.	- Resource-Aware - Control resources using containers - Assume linear function between resources and response-time
Cucinotta, Mancina, Anastasi, Lipari, Mangeruca, Checcozzo, and Rusina (2009) [63; 163; 204–206]	Presents a series of work extending network middleware to handle soft real-time services in a cyberphysical system before considering formal modelling of CPU intensive service computation for scheduling services within workflows on virtual machines.	- Energy Resource Aware - IaaS - Constraint checking - Container based - Not Real-Time - Assumes RTOS

continued . . .

...Table A.3 continued

Authors	Summary	Features
Feng, Zhijian, and Qian (2016) [207]	Using the attributes of dependability to form QoS to manage the assignment of virtual machines in cloud computing environments.	- Resource-aware - Control resources using containers - Full control of cloud infrastructure - Not Real-Time
Lin, Panahi, Zhang, Zhang, and Chang (2009) [162; 208–213]	Presents the RT-Llama approach for monitoring and guaranteeing service performance using virtual CPU reservations & intermediate deadlines.	- CPU Aware - Real-Time - Container based - Assumes full system control - Scheduling Micro-Services (μSs) of a service - Assumes sequential execution order
Liu, Ren, Deng, and Song (2016) [214]	A method for allocation of resources to services by managing the virtual machines.	- Real-Time, assuming ideal conditions - Resource-Aware - Container based - Assumes full system control

continued ...

...Table A.3 continued

Authors	Summary	Features
Quiroz, Kim, Parashar, Gnanasambandam, and Sharma (2009) [215]	Allows over-provisioning of resources based on the assumption that many services do not utilise all their requested resources.	- Uses VM provisioning - Resource-Aware - Not Real-time - Assumes full system control
Stavrinides and Karatza (2015) [216]	Looks at minimising workflow execution times across users whilst adhering to deadlines with a system wide QoS as well as service level QoS.	- Real-Time - Uses virtual machine selection, i.e. containers - Not resource aware

Table A.4: Summary of middleware-based approaches

Authors	Summary	Features
Boniface, Nasser, Papay, Phillips, Servin, Yang, Zlatev, Gogouvitis, Katsaros, Konstanteli, Kousiouris, Menychtas, and Kyriazis (2010) [217]	A PaaS approach to RT-SOA for image processing.	- Uses neural-networks - No implementation detail - Requires platform control

continued ...

. . . Table A.4 continued

Authors	Summary	Features
Estévez-Ayres, Almeida, García-Valls, and Basanta-Val (2007) [144; 70; 138; 136; 135; 67; 141; 143; 137; 142; 59; 139; 140; 138]	Defines an extensive, and formally defined, model that utilises schedulability tests to inform workflow compositions as part of the i-Land project.	- "Soft" Real-Time centric - Traditional schedulability - Workflow composition - Resource-Aware - Assume full system control - Semi-adaptive - Uses DDS for communication backbone
Fu, Wu, Jeng, and Lei (2008) [218]	Proposes PACTS as a service-oriented approach to managing video streaming whilst guaranteeing frame rates by mapping user QoE to frame rates and server bandwidths.	- Real-Time - Full system control - Resource-aware (Network bandwidth) - Uses DDS
Garces-Erice (2009) [219]	A middleware approach extending DDS with a scheduling layer to manage service deadlines.	- Real-Time - Full system control - Utilise DDS - Not resource-aware

continued . . .

. . . Table A.4 continued

Authors	**Summary**	**Features**
Moreland, Sarkani, and Mazzuchi (2013) [72]	A DDS approach to RT-SOA in military C2 systems.	- Real-Time - Resource Aware - Similar to Tsai et al. [129] - Assumes full system control - Not full SOA, e.g. no loose-coupling
Pérez and Gutiérrez (2015) [150]	Use DDS to support a Real-Time Service-Oriented Architecture (RT-SOA) within a controlled automotive environment with a sequential workflow.	- Based on DDS, similar to Tsai et al. [129] - Real-Time - Not Resource-Aware
Pr, Moritz, Zeeb, Salomon, Golatowski, and Timmermann (2008) [220]	An approach adapting SOAP to manage real-time network communication for SOA applied to robot control systems.	- Real-Time - Resource-Aware, network and CPU - Requires control of network protocols
Schneider (2010) [149]	Utilisation of DDS for a real-time enterprise service bus for RT-SOA.	- DDS approach - Resource Aware - Real-Time - Assumes full network control

continued . . .

...Table A.4 continued

Authors	Summary	Features
Tsai, Lee, Cao, Chen, and Xiao (2006) [129]	Describes an RT-SOA based on DDS and identifies many of the core requirements for RT-SOAs over and above SOAs.	- Uses DDS - Real-Time - Not Resource-Aware

Table A.5: Summary of fuzzy-logic based approaches

Authors	Summary	Features
Benbernou, Hadjali, Karam, and Ouziri (2015) [111]	Adopts a fuzzy logic approach to categorise response-times as either good, bad, or medium subject in the context of memory utilisation.	- Probabilistic - Fuzzy Logic - Not Resource-Aware - Service selection for workflows - Not Real-Time

continued ...

...Table A.5 continued

Authors	Summary	Features
de Gyves Avila and Djemame (2013) [80]	Fuzzy Logic based mechanism for adapting workflows based on historical performance data.	- Fuzzy-logic - Service selection for workflow composition - Self adaptation - Energy & cost aware [164] - Not Real-Time - Not resource-aware

Table A.6: Summary of cost-based approaches

Authors	Summary	Features
Abbassi, Graiet, Boubaker, Kmimech, and Hadj-Alouane (2015) [95]	Provides a formal notation using Event-B to describe the constraints on web services and therefore compute the response-time.	- Extensible mathematics - Considers availability and cost [164] - Not Real-Time - Not Resource Aware - Are not aware of existing literature - No demonstrable results - Service is white box

continued ...

... Table A.6 continued

Authors	Summary	Features
Al-Ghuwairi, Khalaf, Al-Yasen, Salah, Alsarhan, and Baarah (2016) [221]	Provides an online update to a SLA which measures the average monthly up-time percentage.	- Measures service availability - User - provider based - Cost aware [164] - Not Resource-Aware - Not Real-Time
Guo, Jiang, and Li (2015) [222]	A method for selecting services based on a *"credit score"* partially aimed at managing CPU utilisation.	- Workflow QoS - Availability & cost [164] - Partially CPU aware - Not Real-Time
Tan, Chen, Sun, Liu, André, Xue, and Dong (2016) [223]	Presents a methodology for selecting services, based on a probabilistic ranking for a workflow and calculating the overall workflow QoS.	- Cost & Availability aware - Service selection for workflows - Not Resource-Aware - Not Real-Time
Kaur, Kaur, and Aggarwal (2014) [164]	Detailed cost modelling of Software-as-a-Service.	- Cost modelling - Not Real-time - Not resource-aware

continued ...

. . . Table A.6 continued

Authors	Summary	Features
Liu, Ngu, and Zeng (2004) [224]	A QoS registry approach using cost and reputation to select services.	- Service ranking for selection - Execution cost [164] - Not Real-Time - Not Resource-Aware
Ngoc, Lin, Nakaguchi, and Ishida (2014) [225]	A method for handling dynamic mobile environments where execution duration varies by observing the availability of the individual Micro-Service.	- Micro-Service selection for services and workflows - Models cost [164] - Not Real-Time - Not Resource Aware
Sachan, Kumar Dixit, and Kumar (2014) [226]	Utilise several parameters in their QoS definition and rank the web services based on them allowing the selection of the most appropriate service for the user.	- Service selection for workflows - Static cost aware - Static resource aware - Not real-time

continued . . .

. . . Table A.6 continued

Authors	Summary	Features
Singh and Chana (2016) [155]	Looks at resource allocation in the Cloud to manage processor utilisation as well as energy consumption.	- Resource Aware, CPU, memory, storage, and networks - Energy & cost - Assumes full system control - Not Real-Time
Wang, Chiu, and Wu (2015) [153]	Classifies users into stakeholder groups based on QoS preference and model the workflow level QoS.	- Service selection for workflows - Cost aware - Not Real-Time - Not Resource-Aware

Table A.7: Summary of probabilistic approaches

Authors	Summary	Features
Calinescu, Grunske, Kwiatkowska, Mirandola, and Tamburrelli (2011) [227]	Workflow composition using QoS on performance and reliability using probabilistic calculus.	- Probabilistic - Adaptive - Workflow management - Service selection for workflows - Agnostic about the infrastructure - Not Real-Time - Not Resource Aware
Gao, Moussa, Yen, Bastani, and Jeng (2008) [228]	Extend WSDL with timing information pertaining to response-time requirements and the guaranteed response-time of a web service.	- Real-Time - History based prediction - Uses geodesic distance & network latency - Not resource-aware
Jayathilaka, Krintz, and Wolski (2015) [229]	Presents a predictive method for web services using a binomial distribution where the model gets updated based on repeated SLA violations.	- Static code analysis - Periodically monitor response-times - History based - Not Real-Time - Not Resource-Aware

continued …

...Table A.7 continued

Authors	Summary	Features
Yang, Peng, and Hu (2014) [230]	Assuming sequential workflows this work presents methods for calculating the likely response-time of a workflow.	- Probabilistic - Reputation/trust based - Assumes sequential workflow execution - Not Real-Time - Not Resource-Aware
Zhu Yong, Li Wei, Luo Junzhou, and Zheng Xiao (2012) [231]	Historical data analysis method for service s-election for workflow composition	- Historical data based - Not Real-Time - Not Resource-Aware
Zareian, Veleda, Litoiu, Shtern, Ghanbari, and Garg (2015) [232]	A data mining approach to managing service performance in the cloud.	- CPU resource aware - Data mining approach - Not Real-Time
Zhang, Zhuang, Leung, Song, and Zhou (2015) [233]	A probabilistic method that takes into accoun-t the geographical location of users, servers, and the network performance between them.	- Historical data - Not Real-time - Not Resource-Aware

continued ...

... Table A.7 continued

Authors	Summary	Features
Zheng, Yang, and Zhao (2010) [234]	Probability estimation method for service selection.	- Service composition - Not Real-Time - Not Resource-Aware
Zou, Lu, Chen, Huang, Xu, and Xiang (2014) [81]	A workflow planning approach to ensure workflow QoS fromt he service QoS.	- Service selection for workflows - Probabilistic - Not Real-Time - Not Resource-Aware

Table A.8: Example probabilistic-driven redundancy

Authors	Summary	Features
Gao, Ali, Curry, and Mileo (2016) [235]	An approach for managing processing streams by adapting redundancy levels based on QoS	- Service selection for workflows - Not resource-aware - Not Real-time

continued ...

. . . Table A.8 continued

Authors	Summary	Features
Russell, Liu, Luo, Venters, Webster, and Xu (2010) [131; 24; 35; 132; 34; 236; 56; 41; 237]	In the NECTISE project utilises service redundancy to increase the level of provided QoS	- Probabilistic - Service selection for workflows - Not Real-Time - Not Resource-Aware

Table A.9: Summary of monitoring approaches

Authors	Summary	Features
Alhamazani, Ranjan, Jayaraman, Mitra, Rabhi, Georgakopoulos, and Wang (2015) [238]	Introduces a benchmarking and monitoring agent to analyse QoS taking into account network bandwidth, CPU load, and response-times.	- Models communication overhead - Network bandwidth - Infrastructure agnostic - Workload aware - Not Real-Time - Not Resource-Aware

continued . . .

...Table A.9 continued

Authors	**Summary**	**Features**
Bosman, van den Berg, and van der Mei (2015) [159]	Periodically probe services to monitor their response-times.	- Periodic probing - Service selection for workflows - System agnostic - Not Resource-Aware - Not Real-Time
Buck and Shi (2015) [239]	Presents a method for testing the response-times of web services, in the context of flight monitoring.	- Probabilistic using historical data - Offline approach
Cicotti, D'Antonio, Cristaldi, and Sergio (2013) [240; 241]	Provides a monitoring system, QoSMONaaS, to check service compliance with advertised QoS and is demonstrated in the context of energy monitoring.	- Resource-Aware, power consumption - Periodic modelling checking - Not Real-Time
McGregor and Eklund (2008) [242]	RT-SOA for remote healthcare monitoring.	- Not Real-Time - Not Resource Aware
Silva, Lins, and Sousa (2015) [243]	A methodology for evaluating web service performance using experimentation.	- Not Real-Time - Not Resource-Aware

Table A.10: Summary list of QoS approaches

Authors	Summary	Features
Assayad (2008) [165]	Presents P-Ware an embedded systems approach to modelling the QoS of media processing software running directly on hardware.	- Real-Time - Not Resource-Aware - HW solution
Degeler, Georgievski, Lazovik, and Aiello (2010) [244]	A search-based solution for finding the best set of services for a workflow's response-time.	- Service selection for workflows - Not Resource-aware - Not Adaptative
Deng, Jiang, and Ca (2014) [245]	An approach overlaying workflow composition with domain understanding.	- Workflow composition - Not Resource-aware - Not real-time - Requires knowledge of service's function
Efstathiou, McBurney, Zschaler, and Bourcier (2013) [130]	A SOA decision support system with QoS constraints on response-time and battery power.	- Workflow composition - Real-time constraints - Energy modelling - Not resource-aware

continued . . .

…Table A.10 continued

Authors	Summary	Features
Ismail, Yan, and Shen (2009) [246]	Observes the resource utilisation per timeslot	- Resource utilisation based - Service selection - Not Real-Time - Assume full system control
Li and Guo (2015) [247]	An approach controlling CPU utilisation to dynamically reconfigure workflows.	- Real-Time - Resource-Aware, CPU core level - Workflow reconfiguration - Assumes full system control
Lima, Rocha, and Costa (2016) [248]	Presents an approach to share service instances across workflows to reduce system workload.	- Workflow choreography - Not Real-Time - Not Resource Aware
Ben Mabrouk, Beauche, Kuznetsova, Georgantas, and Issarny (2009) [32]	An approach to adhere to workflow QoS rather than individual service QoS.	- Service selection for workflows - Not Real-Time - Not Resource Aware

continued …

. . . Table A.10 continued

Authors	Summary	Features
Monaco, Nery, and Peixoto (2009) [249]	A load balancing scheduling algorithm for Real-Time Service-Oriented Architectures	- Real-Time - Resource Aware - Assumes full system control
Sun, White, Eade, and Schmidt (2016) [250]	Present ROAR a framework for managing resource allocation to services in Clouds.	- Resource-Aware, CPU, memory, network, & disk - Not Real-Time - Assumes full system control
Vu, Hauswirth, and Aberer (2005) [251]	A trust-based method for ranking services for selection.	- Service selection for workflows - Trust modelling - Not Real-Time - Not Resource-Aware
Xu and Yan (2009) [252]	QoS driven workflow composition.	- Full service implementation aware - Service selection - Not Real-Time - Not Resource-Aware

continued . . .

...Table A.10 continued

Authors	Summary	Features
Xue, Yen, and Cooper (2011) [253]	QoS adaptation for managing media stream frame rates by reducing frame resolutions.	- Real-Time - Network bandwidth aware - Data-context aware

Appendix B

Algorithms

B.1 Online Monitoring Algorithms

As described on Page 95

Algorithm 3: Calculating the Availability Coordinates

```
Input: α as calculated in Eqn. 4.6
Result: j the model coordinates
Result: A_sum the total resource availability, see Eqn. 4.7
foreach r ∈ R do
    begin Update Availability and calculate j
        A_sum[r] = A_sum[r] + a[r]
        j[r] = CEIL(a[r]d[r])
    end
end
```

As used on Page 95

Algorithm 4: Estimating Execution Progress

Input: j as calcualted in Algorithm 3

Input: p_{t-1} the previously estimated progress

Input: Ω the set of resource utilisation observation

Input: k specified as a constant

Input: I the density model

Input: F model

Result: p the estimated execution progress

/* At least 1 of **RTT** or $U_{provided}$ must be supplied */

Optional: $U_{provided}$ the statically provided, otherwise $U_{provided} = \infty$

Optional: **RTT** statically provided, otherwise $RTT = \infty$

Optional: h provided as the benchmark, otherwise $h = \infty$

begin Estimate progress
- $p_{temp} = \infty$
- **foreach** *resource r in* $\mathcal{R}$ **do**
  - **if** $I.Sum == 0$ **then** Initial Case
    - $h = MIN(h, \mathbb{H}.Max)$
    - $F[j, r] = MIN(U_{provided}, h \times \mathbf{RTT})$
  - **end**
  - $temp = FLOOR((k \times \Omega[r].Sum) \div F[j, r]) \div k$
  - $p_{temp} = MIN(p_{temp}, temp)$
- **end**
- $p = MAX(p, p_{temp})$

end

As used on Page 95

Algorithm 5: Estimating the time-to-finish

Input: j as calcualted in Algorithm 3

Input: p the estimated progress

Input: Ω the set of all resource utilisation observation

/* At least 1 of **RTT** or $U_{provided}$ must be supplied */

Optional: $U_{provided}$ the statically provided, otherwise $U_{provided} = \infty$

Optional: **RTT** statically provided, otherwise $RTT = \infty$

Optional: h provided as the benchmark, otherwise $h = \infty$

if $I_j > 0$ **then** Standard Model
  $RTT = M[j]$
end
else if $I.Sum == 0$ **then** Initial Case
  $h = MIN(h, \mathbb{H}.Max)$
  $U = MIN(U_{provided}, h \times RTT)$
  $RTT = U \div j$
end
else Sparse Model
  begin Calculate $\mathcal{D}^{-1}$ from j of all points in the matrix
    foreach i **do**
      $D[i] = 1 \div ABS(i - j)$
    end
  end
  begin Calculate **RTT**
    $num = 0$
    $denom = 0$
    foreach $i \neq j$ **do**
      $num+ = I[j] \times M[i] \times D[i]$
      $denom+ = I[i] \times D[i]$
    end
    $RTT = num \div denom$
  end
end
$\mathcal{TTF} = (p \div k) \times$ **RTT**

B.2 Model Updating Algorithms

As described on Page 97

Algorithm 6: Update response-time data sets

Input: j as calcualted in Algorithm 3

Input: The recorded response-time $\mathbf{RTT}$

Input: $\mathcal{M}$ the response-time model

Result: I updated density matrix

Result: T data set of response-times

Result: $\mathcal{M}$ updated response-time model

begin Historical record
- $T[j] = T[j].ADD(RTT)$
- $maxT[j] = MAX(maxT[j], RTT)$
- $totalT[j] = T[j].Sum$

end

begin Indicator function
- $I[j] + +$
- $sumI = I.Sum$

end

begin Response-Time model
- $M[j] = \{totaltT[j] \div I[j], maxT[j]\}$

end

As used on Page 97

Algorithm 7: Building the utilisation and availability models

Input: some input

begin Average j
  foreach $r \in \mathcal{R}$ **do**
    $A[n, r] = A_{sum}[r] \div (k + 1)$
  end
  `/* See Algorithm 3 */`
  j_temp$+=$ CALCULATE_J($A[n]$)
end
for $(p = k, p \geq 0, p - -)$ **do**
  foreach $r \in \mathcal{R}$ **do**
    begin Updating Utilisation Model
      $\text{sum}_u[p, r, j] += u[p, r]$
      $\text{min}_u[p, r, j] = \text{min}(u[p, r], \text{min}_u[p, r, j])$
      $\text{max}_u[p, r, j] = \text{min}(u[p, r], \text{max}_u[p, r, j])$
      $\text{delta} = u[p, r] - \text{mu}_u[p, r, j]$
      $\text{mu}_u[p, r, j] = \text{sum}_u[p, r, j] \div (n + 1)$
      $\text{M2}_u[p, r, j] = \text{M2}_u[p, r, j] + \text{pow}(u[p, r] - mu_u[p, r], 2)$
      if $n \geq 2$ **then**
        $\text{var}_u[p, r, j] = M2_u[p, r, j] \div n$
      end
    end
    begin Update Forecast Model
      $F_{mean}[p, r, j] = F_{mean}[p + 1, r, j] + \text{mu}_u[p, r, j]$
      $F_{min}[p, r, j] = F_{min}[p + 1, r, j] + \text{min}_u[p, r, j]$
      $F_{max}[p, r, j] = F_{max}[p + 1, r, j] + \text{max}_u[p, r, j]$
      $F_{varSum}[p, r, j] = F_{varSum}[p, r, j] + \text{var}_u[p, r, j]$
      $F_{var}[p, r, j] = F_{varSum}[p + 1, r, j] \div (kp + 1, j)$
    end
  end
end

B.3 Deadline Miss Alert

As described on Page 98

Algorithm 8: Deadline miss alert

Input: $\mathcal{TTF}$ as predicted by Algorithm 2

Input: j availability configuration

Input: p estimated progress

Output: ALERT(_) warning of the potential deadline miss

begin Initial deadline check

if $\mathcal{TTF} > \boldsymbol{D}$ **then**

begin Re-configuration Check

j_{target} = NULL

foreach $i >= j$ **do**

/* See Algorithm 5 */

$\mathcal{TTF}_{temp}$ = ESTIMATE_TTF($i, p, \Omega, [U, \mathbf{RTT}, h]$)

if $\mathcal{TTF}_{temp} <= \boldsymbol{D}$ **then**

$j_{target} = i$

BREAK LOOP

end

end

end

if $j_{target} = NULL$ **then**

ALERT(NULL)

end

else Report the required configuration

ALERT(j_{target})

end

end

end

Appendix C

Code Listings for QoS Approach Simulation

This appendix contains the code listings for the simulations from Chapters 3 and 5.

C.1 Server Code Listing

Listing C.1: Code listing for the Server model from the simulations

```
public class Server : Node
{
  #region Properties

  double _CPU_Capacity = 1;

  [IsAccess]
  public double CPU_Capacity
  {
```

```
        get { return _CPU_Capacity; }
        set { _CPU_Capacity = value; }
    }

    double _Memory_Capacity = 1;

    [IsAccess]
    public double Memory_Capacity
    {
        get { return _Memory_Capacity; }
        set { _Memory_Capacity = value; }
    }

    double _CPU_Utilisation;

    [IsAccess]
    public double CPU_Utilisation
    {
        get { return _CPU_Utilisation; }
        set { _CPU_Utilisation = value; }
    }

    double _Memory_Utilisation;

    [IsAccess]
    public double Memory_Utilisation
    {
        get { return _Memory_Utilisation; }
        set { _Memory_Utilisation = value; }
    }

    public InterferenceTask Interference
    {
        get
        {
            if (this.Functions.Any(f => f.GetType() == typeof(
```

```
InterferenceTask)))
        return this.Functions.First(f => f.GetType() == typeof(
InterferenceTask)) as InterferenceTask;
      return null;
    }
  }

  public IFunction Task
  {
    get
    {
      if (this.Functions.Any(f => f.GetType() == typeof(MicroService)))
        return this.Functions.Last(f => f.GetType() == typeof(
MicroService)) as MicroService;
      return null;
    }
  }

  #endregion

  public Server() : base() { }

  public override void Main()
  {
    while (ALIVE)
    {
      if (Task != null)
      {
        bool taskFinished = false;

        while (!taskFinished && ALIVE && Task != null)
        {
          #region running

          if (Interference != null && Task != null)
          {
```

```
            switch (Interference.Status)
            {
              case ActionStatus.Finished:
              case ActionStatus.Failed:
              case ActionStatus.Killed:
                Interference.Start(); //Restart
                break;
              case ActionStatus.NotStarted:
              case ActionStatus.Paused:
                Interference.Start();
                break;
              default:
                break;
            }

            switch (Task.Status)
            {
              case ActionStatus.Finished:
              case ActionStatus.Failed:
              case ActionStatus.Killed:
                taskFinished = true;
                break;
              case ActionStatus.NotStarted:
              case ActionStatus.Paused:
                Task.Start();
                break;
              default:
                break;
            }

            //Schedule: allocate CPU & Memory
            Interference.AllocCPU = Math.Min(Interference.CPU, CPU_Capacity);
            CPU_Utilisation = Interference.AllocCPU;
            Interference.AllocMemory = Math.Min(Interference.Memory, Memory_Capacity);
```

```
                Memory_Utilisation = Interference.AllocMemory;

                double A_cpu = CPU_Capacity - CPU_Utilisation;
                double A_mem = Memory_Capacity - Memory_Utilisation;

                if (this.Task.GetType() == typeof(VirtualMachine))
                {
                  ((VirtualMachine)this.Task).AllocCPU = Math.Min(A_cpu, ((
    VirtualMachine)this.Task).CPU);
                  ((VirtualMachine)this.Task).AllocMemory = Math.Min(A_mem,
     ((VirtualMachine)this.Task).Memory);

                  CPU_Utilisation += ((VirtualMachine)this.Task).AllocCPU;
                  Memory_Utilisation += ((VirtualMachine)this.Task).
    AllocMemory;
                }
                else
                {
                  ((Task)this.Task).AllocCPU = Math.Min(A_cpu, ((Task)this.
    Task).CPU);
                  ((Task)this.Task).AllocMemory = Math.Min(A_mem, ((Task)
    this.Task).Memory);

                  CPU_Utilisation += ((Task)this.Task).AllocCPU;
                  Memory_Utilisation += ((Task)this.Task).AllocMemory;
                }
              }
              #endregion
            }

            this.Wait();
            this.Wait();

            if (Task != null)
            {
              if (Task.Status != ActionStatus.Finished)
```

```
                Task.Kill();
              else
                Functions.Remove(Task);
            }

            this.Wait();
            this.Wait();
          }
      }
  }
}
```

C.2 Virtual Machine Listing

Listing C.2: Code listing for the Virtual Machine model from the simulations

```
public class VirtualMachine : ChildNode
{
  #region Properties

  double _CPU;

  [IsAccess]
  public double CPU
  {
    get { return _CPU; }
    set { _CPU = value; }
  }

  double _Memory;

  [IsAccess]
  public double Memory
  {
    get { return _Memory; }
    set { _Memory = value; }
```

```
}

double _AllocCPU = 0;

[IsAccess]
public double AllocCPU
{
  get { return _AllocCPU; }
  set { _AllocCPU = value; }
}

double _AllocMemory = 0;

[IsAccess]
public double AllocMemory
{
  get { return _AllocMemory; }
  set { _AllocMemory = value; }
}

double _CPU_Capacity;

[IsAccess]
public double CPU_Capacity
{
  get { return _CPU_Capacity; }
  set { _CPU_Capacity = value; }
}

double _Memory_Capacity;

[IsAccess]
public double Memory_Capacity
{
  get { return _Memory_Capacity; }
  set { _Memory_Capacity = value; }
```

```
}

double _CPU_Utilisation;

[IsAccess]
public double CPU_Utilisation
{
  get { return _CPU_Utilisation; }
  set { _CPU_Utilisation = value; }
}

double _Memory_Utilisation;

[IsAccess]
public double Memory_Utilisation
{
  get { return _Memory_Utilisation; }
  set { _Memory_Utilisation = value; }
}

public MicroService Task
{
  get
  {
    return this.Functions[0] as MicroService;
  }
  set
  {
    this.Functions = new List<IFunction>() { value };
  }
}

private MathNet.Numerics.Distributions.Normal overhead;

#endregion

```

```
public VirtualMachine()
  : base()
{
  overhead = new MathNet.Numerics.Distributions.Normal(.05, .01);
}

public override void Main()
{
  bool taskFinished = false;

  while (ALIVE && !taskFinished)
  {
    if (Task != null)
      switch (Task.Status)
      {
        case ActionStatus.Finished:
        case ActionStatus.Failed:
        case ActionStatus.Killed:
          taskFinished = true;
          break;
        case ActionStatus.NotStarted:
        case ActionStatus.Paused:
          Task.Start();
          break;
        default:
          break;
      }

    double A_cpu = CPU_Capacity * (AllocCPU / CPU) - overhead.Sample();
    double A_mem = Memory_Capacity * (AllocMemory / Memory) - overhead.Sample();

    Task.AllocCPU = Math.Min(A_cpu, Task.CPU);
    Task.AllocMemory = Math.Min(A_mem, Task.Memory);

```

```
      CPU_Utilisation += Task.AllocCPU;
      Memory_Utilisation += Task.AllocMemory;
    }
  }
}
```

C.3 Interference Workload Listing

Listing C.3: Code listing for the Interference Workload model from the simulations

```
public abstract class Task : Function
{
  double _CPU;

  [IsAccess]
  public double CPU
  {
    get { return _CPU; }
    set { _CPU = value; }
  }

  double _Memory;

  [IsAccess]
  public double Memory
  {
    get { return _Memory; }
    set { _Memory = value; }
  }

  double _AllocCPU = 0;

  [IsAccess]
  public double AllocCPU
  {
    get { return _AllocCPU; }
```

```
    set { _AllocCPU = value; }
  }

  double _AllocMemory = 0;

  [IsAccess]
  public double AllocMemory
  {
    get { return _AllocMemory; }
    set { _AllocMemory = value; }
  }

  double _CPU_Mean;

  [IsAccess]
  public double CPU_Mean
  {
    get { return _CPU_Mean; }
    set { _CPU_Mean = value; }
  }

  double _Memory_Mean;

  [IsAccess]
  public double Memory_Mean
  {
    get { return _Memory_Mean; }
    set { _Memory_Mean = value; }
  }


  protected MathNet.Numerics.Distributions.Normal cpuVariance,
    memoryVariance;
}

public class InterferenceTask : Task
```

```
{
  #region Properties

  WorkloadPattern _WorkloadType;

  public WorkloadPattern WorkloadType
  {
    get { return _WorkloadType; }
    set { _WorkloadType = value; }
  }

  MathNet.Numerics.Random.MersenneTwister rand;

  #endregion

  public InterferenceTask()
    : base()
  {
    cpuVariance = new MathNet.Numerics.Distributions.Normal(0, .05);
    memoryVariance = new MathNet.Numerics.Distributions.Normal(0, .05);
    rand = new MathNet.Numerics.Random.MersenneTwister(true);
  }

  public override void Main()
  {
    CPU = CPU_Mean;
    Memory = Memory_Mean;

    bool? once = null;
    double rad = 0;

    while (ALIVE)
    {
      switch (WorkloadType)
      {
        case WorkloadPattern.Static:
```

```
          CPU = CPU_Mean + cpuVariance.Sample();
          Memory = Memory_Mean + memoryVariance.Sample();
          break;
        case WorkloadPattern.Unpredictable:
          CPU = rand.NextDouble();
          Memory = rand.NextDouble();
          break;
        case WorkloadPattern.OnceInALifetime:
          if (once == true)
          {
            if (rand.NextDouble() > .8)
            {
              once = false;
              CPU = CPU_Mean + cpuVariance.Sample();
              Memory = Memory_Mean + memoryVariance.Sample();
              WorkloadType = WorkloadPattern.Static;
            }
          }
          else if (once == null)
          {
            if (rand.NextDouble() > .8)
            {
              once = true;
              CPU = Math.Min(1, 0.98 + cpuVariance.Sample());
              Memory = Math.Min(1, 0.98 + memoryVariance.Sample());
            }
          }
          break;
        case WorkloadPattern.Periodic:
          CPU = Math.Min(1, (MathNet.Numerics.Trig.Sin(rad) / 2) + cpuVariance.Sample() + CPU_Mean);
          Memory = Math.Min(1, (MathNet.Numerics.Trig.Sin(rad) / 2) + memoryVariance.Sample() + Memory_Mean);
          rad = rad + rand.NextDouble();
          break;
        case WorkloadPattern.ContIncreasing:
```

```
                    CPU = CPU + Math.Abs(cpuVariance.Sample());
                    Memory = Memory + Math.Abs(memoryVariance.Sample());
                    break;
                case WorkloadPattern.ContDecreasing:
                    CPU = CPU - Math.Abs(cpuVariance.Sample());
                    Memory = Memory - Math.Abs(memoryVariance.Sample());
                    break;
            }
        }
    }
}

public enum WorkloadPattern
{
    Static=0,
    Periodic=1,
    OnceInALifetime=2,
    Unpredictable=3,
    ContIncreasing=4,
    ContDecreasing=5
}
```

C.4 Micro-Service Listing

Listing C.4: Code listing for the Micro-Service model from the simulations

```
public class MicroService : Task
{
    #region Properties

    string _Name;

    [IsAccess]
    public string Name
    {
        get { return _Name; }
```

```
    set { _Name = value; }
  }

  private List<double> _CPU_Utilisation, _Memory_Utilisation;

  public List<double> CPU_Utilisation
  {
    get { return _CPU_Utilisation; }
    set { _CPU_Utilisation = value; }
  }

  public List<double> Memory_Utilisation
  {
    get { return _Memory_Utilisation; }
    set { _Memory_Utilisation = value; }
  }

  double _CPU_Utilisation_Total = 0, _Memory_Utilisation_Total = 0;

  [IsAccess]
  public double CPU_Utilisation_Total
  {
    get { return _CPU_Utilisation_Total; }
    set { _CPU_Utilisation_Total = value; }
  }

  [IsAccess]
  public double Memory_Utilisation_Total
  {
    get { return _Memory_Utilisation_Total; }
    set { _Memory_Utilisation_Total = value; }
  }

  double _Progress;

  [IsAccess]
```

```
public double Progress
{
  get { return _Progress; }
  set { _Progress = value; }
}

int _Length;

[IsAccess]
public int Length
{
  get { return _Length; }
  set { _Length = value; }
}

ResourceAcquistion _AcquisitionType;

public ResourceAcquistion AcquisitionType
{
  get { return _AcquisitionType; }
  set { _AcquisitionType = value; }
}
ResourceRelease _ReleaseType;

public ResourceRelease ReleaseType
{
  get { return _ReleaseType; }
  set { _ReleaseType = value; }
}

TaskType _TaskType;

public TaskType TaskType
{
  get { return _TaskType; }
  set { _TaskType = value; }
```

```
}

double _StartTime;

[IsAccess]
public double StartTime
{
  get { return _StartTime; }
  set { _StartTime = value; }
}

double _ResponseTime;

[IsAccess]
public double ResponseTime
{
  get { return _ResponseTime; }
  set { _ResponseTime = value; }
}

#endregion

public MicroService()
  : base()
{
  cpuVariance = new MathNet.Numerics.Distributions.Normal(0, .02);
  memoryVariance = new MathNet.Numerics.Distributions.Normal(0, .02);

  CPU_Utilisation = new List<double>();
  Memory_Utilisation = new List<double>();
}

public override void Main(string[] args)
{
  _StartTime = this.Time;

```

```
int midx = args.ToList().IndexOf("MEM") + 1;
CPU_Utilisation = args.ToList().GetRange(1, Length).Select(s => double.Parse(s)).ToList<double>();
Memory_Utilisation = args.ToList().GetRange(midx, Length).Select(s => double.Parse(s)).ToList<double>();

switch (AcquisitionType)
{
  case ResourceAcquistion.Eager:
    CPU = CPU_Utilisation.Max();
    Memory = Memory_Utilisation.Max();
    break;
  case ResourceAcquistion.Lazy:
    CPU = CPU_Utilisation[0];
    Memory = Memory_Utilisation[0];
    break;
}

int count = 0;
int stepCount = 0;

while (count < Length)
{
  this.Wait();
  try
  {
    if (AcquisitionType == ResourceAcquistion.Lazy)
    {
      if (CPU_Utilisation[count] > CPU || ReleaseType == ResourceRelease.Active)
        CPU = CPU_Utilisation[count];
      if (Memory_Utilisation[count] > Memory || ReleaseType == ResourceRelease.Active)
        Memory = Memory_Utilisation[count];
    }
    else
```

```
        {
          if (ReleaseType == ResourceRelease.Active)
            CPU = CPU_Utilisation[count];
          if (ReleaseType == ResourceRelease.Active)
            Memory = Memory_Utilisation[count];
        }

        CPU_Utilisation_Total += AllocCPU;
        Memory_Utilisation_Total += AllocMemory;

        stepCount++;
        if (stepCount >= Math.Abs(MathNet.Numerics.SpecialFunctions.
    Logit(
            Math.Min(1, Math.Max(0, (1 - ((AllocCPU / CPU) / 2))))
          )))
        {
          count++;
          stepCount = 0;
        }
      }
      catch (Exception err)
      {
        Console.WriteLine(err.Message);
      }
      Progress = (double)count / (double)Length;
      if (Progress >= 1)
      {
        this.Status = ActionStatus.Finished;
        ResponseTime = this.Time - _StartTime;
      }
    }

    ResponseTime = this.Time - _StartTime;
    Progress = 1;

    this.Wait();
```

```
        this.Wait();
    }
}

public enum ResourceAcquistion
{
    Lazy=0,
    Eager=1
}
public enum ResourceRelease
{
    Active=0,
    NonReleasing=1
}
public enum TaskType
{
    Small=0,
    Medium=1,
    Large=2
}
```

Appendix D

Code Listings for QoS Approach Experiments

This appendix contains the code listings for the Micro-Services from Chapter 6.

Listing D.1: Code listing Micro-Services

```
#region multiples

static List<ulong> Multiples(ulong a,ulong max)
{
    List<ulong> mults = new List<ulong>();
    for (ulong i = 1; i < max; i++)
    {
        if (a % i == 0)
            mults.Add(i);
    }

    return mults;
}
```

```

static List<ulong> MultiplesOR(ulong a, ulong b, ulong max)
{
    List<ulong> mults = Multiples(a, max);
    List<ulong> mults2 = Multiples(b, max);
    foreach (ulong i in mults2)
        if (!mults.Contains(i))
            mults.Add(i);

    return mults;
}

static List<ulong> MultiplesOR(List<ulong> a, ulong max)
{
    List<List<ulong>>multss = new List<List<ulong>>();
    foreach (ulong b in a)
    {
        List<ulong> mult = Multiples(b, max);
        multss.Add(mult);
    }

    List<ulong> mults = new List<ulong>();
    foreach (List<ulong> ls in multss)
        foreach (ulong l in ls)
            if (!mults.Contains(l))
                mults.Add(l);

    return mults;
}

static List<ulong> MultiplesAND(ulong a, ulong b, ulong max)
{
    List<ulong> mults = new List<ulong>();
    List<ulong> mults1 = Multiples(a, max);
    List<ulong> mults2 = Multiples(b, max);
    foreach (ulong i in mults2)
```

```
            if (mults1.Contains(i))
                mults.Add(i);

        return mults;
}

static List<ulong> MultiplesAND(List<ulong> a, ulong max)
{
    List<List<ulong>> multss = new List<List<ulong>>();
    foreach (ulong b in a)
    {
        List<ulong> mult = Multiples(b, max);
        multss.Add(mult);
    }

    List<ulong> mults = new List<ulong>();
    int i = 0;
    foreach (List<ulong> ls in multss)
    {
        List<List<ulong>> temp = multss.ToList();
        temp.RemoveAt(i);

        foreach (ulong l in ls)
        {
            bool found = true;

            foreach (List<ulong> tmp in temp)
            {
                if (!tmp.Contains(l))
                    found = false;

                if (!found)
                    break;
            }
            if (found)
                mults.Add(l);
```

```
        }
        i++;
    }

    return mults;
}

static ulong SumMultiplesAND(ulong a, ulong b, ulong max)
{
    ulong sum = 0;

    List<ulong> mults = MultiplesAND(a, b, max);
    foreach (ulong i in mults)
        sum += i;

    return sum;
}

static ulong SumMultiplesOR(ulong a, ulong b, ulong max)
{
    ulong sum = 0;

    List<ulong> mults = MultiplesOR(a, b, max);
    foreach (ulong i in mults)
        sum += i;

    return sum;
}

static int SmallestMultiple(List<int> ns)
{
    int res = 0;

    bool found = false;

    while (!found)
```

```
    {
        res++;
        found = true;
        foreach (int n in ns)
        {
            if (res % n != 0)
                found = false;
            if (!found)
                break;
        }
    }

    return res;
}

#endregion

#region fibonacci

static List<ulong> Fibonacci(ulong max)
{
    List<ulong> fibs = new List<ulong>();
    fibs.Add(1);
    ulong i = 1;
    ulong prev = 1;
    while (i < max)
    {
        i = i + prev;
        prev = i;
        fibs.Add(i);
    }

    return fibs;
}

static ulong SumFibonacci(ulong max)
```

```
{
    List<ulong> fibs = Fibonacci(max);
    ulong sum = 0;
    foreach (ulong i in fibs)
        sum += i;

    return sum;
}

static ulong SumEvenFibonacci(ulong max)
{
    List<ulong> fibs = Fibonacci(max);
    ulong sum = 0;
    foreach (ulong i in fibs)
        if (i % 2 == 0)
            sum += i;

    return sum;
}

#endregion

#region primes

static List<ulong> Primes(ulong max)
{
    List<ulong> primes = new List<ulong>();
    for (ulong i = 2; i <= max; i++)
    {
        bool isprime = true;
        for (ulong j = 2; j < i; j++)
        {
            if (i % j == 0)
                isprime = false;
            if (!isprime)
                break;
```

```
            }
            if (isprime)
            {
                primes.Add(i);
            }
        }

        return primes;
}

static ulong SumPrimes(ulong max)
{
        ulong sum = 0;

        List<ulong> primes = Primes(max);
        foreach (ulong i in primes)
            sum += i;

        return sum;
}

#endregion

#region factorial

static ulong Factorial(ulong a)
{
        List<ulong> facts = new List<ulong>();
        for (ulong i = 1; i < a; i++)
            facts.Add(i);
        ulong sum = a;
        foreach (ulong i in facts)
            sum *= i;

        return sum;
}
```

```
static ulong FactorialSum(ulong a)
{
    List<ulong> facts = new List<ulong>();
    for (ulong i = 1; i < a; i++)
        facts.Add(i);
    ulong sum = a;

    foreach (ulong i in facts)
        sum += i;

    return sum;
}
#endregion

#region series

static List<int> SeriesProduct(List<int> series, int adjSize)
{
    //SortedList<int, int> products = new SortedList<int, int>(series.
   Count);
    int biggest = 0;
    int index = 0;
    for (int i = 0; i < (series.Count - adjSize); i++)
    {
        List<int> subSeries = series.GetRange(i, adjSize);
        int sum = 1;
        foreach (int j in subSeries)
            sum *= j;

        if (sum > biggest)
        {
            biggest = sum;
            index = i;
        }
    }
```

```

    List<int> digits = series.GetRange(index, adjSize);

    return digits;
}
#endregion

#region triangel/pentagonal/hexagonal

static List<ulong> TriPentHex(ulong max)
{
    List<ulong> tris = new List<ulong>();
    List<ulong> pents = new List<ulong>();
    List<ulong> hexs = new List<ulong>();

    for (ulong i = 1; i <= max; i++)
    {
        ulong tri = (i * (i + 1)) / 2;
        if (tri % 1 == 0)
            tris.Add(tri);

        ulong pent = (i * (3 * i - 1)) / 2;
        if (pent % 1 == 0)
            pents.Add(pent);

        ulong hex = (i * (2 * i - 1));
        if (hex % 1 == 0)
            hexs.Add(hex);
    }

    List<ulong> all = new List<ulong>();
    foreach (ulong j in tris)
    {
        if (pents.Contains(j) && hexs.Contains(j))
            all.Add(j);
    }
```

```

    return all;
}

#endregion

#region selfPowers

static ulong SelfPower(int n)
{
    ulong pow = 0;
    for (int i = 1; i <= n; i++)
    {
        pow += (ulong)Math.Pow(i, i);
    }
    return pow;
}

#endregion

#region sum

static ulong Sum(List<int> ns)
{
    ulong sum = 0;
    foreach (int n in ns)
        sum += (ulong)n;
    return sum;
}

#endregion

#region sum square difference

static ulong SumSquareDiff(int n)
{
```

```
    ulong sumsq = 0;
    for (int i = 1; i <= n; i++)
        sumsq += (ulong)Math.Pow(i, 2);

    ulong sqsum = 0;
    for (int i = 1; i <= n; i++)
        sqsum += (ulong)i;
    sqsum = sqsum * sqsum;

    ulong diff = sqsum - sumsq;

    return diff;
}

#endregion

#region palindromes

static int PalindromeProduct(int digits)
{
    int res = 0;

    int max = (int)Math.Pow(10, digits) - 1;
    int min = (int)Math.Pow(10, digits - 1);
    for (int a = max; a >= min; a--)
    {
        for (int b = max; b >= min; b--)
        {
            Console.WriteLine("{0} x {1}", a.ToString(), b.ToString());
            int tmp = a * b;
            string tp = tmp.ToString();
            int n = tp.Length;
            bool palin = true;
            for (int i = 0; i < n / 2; i++)
            {
                if (tp[i] != tp[n - 1 - i])
```

```
                    palin = false;
                if (!palin)
                    break;
            }
            if (palin && tmp > res)
                res = tmp;
        }
    }

        return res;
}

#endregion
```