IMPROVING THE CURRENCY OF INFORMATION IN LARGE-SCALE GRID INFORMATION SYSTEMS

A THESIS SUBMITTED TO THE UNIVERSITY OF MANCHESTER
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
IN THE FACULTY OF ENGINEERING AND PHYSICAL SCIENCES

2013

By
Laurence Howard Field
School of Computer Science

Contents

Word Count: 45206

List of Tables

List of Figures

Abstract

Grid computing aims to seamlessly deliver computing power as a resource similar to how electrical power is delivered over the electrical power grid. This thesis studies the scalability of Grid information systems, which enable the discovery of resources in a Grid computing infrastructure and provide further information about their structure and state. It is motivated by the observation that existing approaches for query processing in a Grid environment do not take into consideration the dynamic nature of the information. The concept of static and dynamic information has been widely used in Grid computing to describe the information about resources even though a detailed definition does not exist and its use may be misleading. This work claims that all information is dynamic and that it is the expected frequency of change which defines the relative dynamic nature of information types. As Grid infrastructures increase in size, the volume of information describing all the resources in the infrastructure increases and hence the number of changes due to its dynamic nature will also increase. Since not all information changes at the same frequency, one can use a measurement that defines the freshness of information to optimise a Grid information system. Ultimately, the freshness is maximized if each change is propagated through the system as soon as it occurs at the resource, and hence the time taken to propagate this information, the latency, is minimized. This thesis proposes a novel Grid information system architecture that reduces the latency for dynamic information by sending each change as it occurs. The proposed architecture not only improves the system's ability to manage dynamic information, but in addition improves the efficiency of the system by reducing both the network throughput and operational overhead. The proposed architecture is evaluated by considering the real use cases from the Worldwide LHC Computing Grid, and is compared against the performance results of the system currently used in production today. The proposed architecture is shown to significantly improve the information freshness for dynamic information.

Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright

Acknowledgements

First and foremost, I would like to express my gratitude to my supervisor Dr Rizos Sakellariou for his guidance and support. This thesis would not have been possible without his suggestions, advice and insight. In addition, I would particularly like to thank my co-supervisor and mentor Dr Markus Schulz who encouraged me to embark upon this journey. I am also grateful to Dr Ian Bird for the opportunity to work at CERN on the WLCG project and to Dr Steve Fisher for introducing me to distributed computing, Grid computing and Grid information systems. I would also like to thank Dr Stuart Paterson, Dr Oliver Keeble and Catharine Noble for their comments and suggestions.

Large-scale collaborative endeavours such as the WLCG are not possible without contributions from many exceptional individuals. I would like to take this opportunity to thank all those in the Grid community who I have the fortune to work alongside for the stimulating discussions and proving such a rewarding working environment.

Finally, I would like to thank my friends and family for their understanding while focussing with this thesis. In particular, my parents for supporting a childhood interest in computing. My son Jules, who arrived midway through this thesis, and most importantly my partner Myriam Zitterbart for her love, support and understanding.

Chapter 1

Introduction

1.1 Grid Computing

Grid computing aims to *seamlessly* deliver computing power as a resource similar to how electrical power is delivered over the electrical power grid [2]. This goal is realised for geographically-dispersed computing resources that span multiple organisations through the provision of a Grid infrastructure encompassing software, services, policies and operations. A distinctive characteristic of Grid computing is that of the *virtual organisation* [3] which is used to support multi-institutional scientific collaboration and is formed around the policies that govern the sharing of resources. Members of a virtual organisation can share resources in a controlled manner in such a way that any member may contribute to the collaboration in order to achieve a shared goal. The resources themselves are not limited to computational resources and can include almost any resource that can be shared over the Internet.

The Grid computing paradigm evolved from the field of metacomputing research and has been the beneficiary of significant research funding [4]. There are diverse opinions on the exact definition of Grid computing [5], which are explored in Section 2.1. This thesis focusing on the Grid computing for the large-scale scientific challenges, such as the simulation, storage, processing and analysis of the data generated by the Large Hadron Collider (LHC) [6], where the unprecedented computational capacity could only be delivered through the sharing of distributed resources. The nature of scientific collaboration where scientists are geographically distributed around the globe and the resources at their disposal are provided at their institution forms the backdrop to this scenario. Successive Grid computing projects [7, 8, 9, 10, 11, 12, 13] have endeavoured to provide a common platform that simplifies resource-sharing for multiple, disparate virtual organisations through the deployment of a Grid computing infrastructure.

The vision was that Grid computing would achieve a similar success in terms of its transformation effect as the Web or the electric power grid itself [2]. The reality is that Grid computing has seen only limited adoption outside the scientific domain [14]. However, Grid infrastructures such as the Worldwide LHC Computing Grid (WLCG)[9] exist, that support user communities who depend on their continued existence and operation. A key challenge is ensuring that these infrastructures can continue to scale for the increasing numbers of Grid resources and for those users who depend on them. This thesis focuses specifically on Grid information systems, which is the component of Grid computing infrastructures that enables the resources to be discovered and used. A brief overview of Grid information systems is provided in Section 1.2. The main challenges for Grid information systems and the motivation for this work are presented in Section 1.3 along with the work's aims and contributions in Section 1.4. Section 1.5 concludes this chapter with an overview of the structure of this dissertation.

1.2 Grid Information Systems

Grid information systems enable the discovery of resources in a Grid infrastructure and provide further information about their structure and state. This information is a precursor for enabling other Grid functions, such as job scheduling and data management, which utilise multiple cooperating Grid services. To facilitate interoperability between both Grid resources and the Grid computing infrastructures themselves, the information provided conforms to an information model and as such is restricted to a finite set. The information is obtained from the information system by submitting a query to a Grid information service, which returns the result. As the resources are distributed, the Grid information system is the infrastructure that enables the query to be resolved. The information sources are the resources themselves and hence Grid information is as distributed and shows the same behaviour as the underlying resources. When a new resource joins the infrastructure, the Grid information system expands to accommodate the new resource and the resulting additional information. Any changes that occur in the underlying resources are reflected in the Grid information system. A mechanism to remove the information once the resource has been decommissioned is used to prevent stale information accumulating in the system. Particular attention is given to ensure that Grid information systems continue to function and provide the correct response in the face of an unreliable infrastructure. The Grid information system should also be unaffected by the query load that is generated by users of the Grid infrastructure or by other services which may also initiate queries.

The specific feature of Grid information systems is that they have to integrate information from a diverse set of resources distributed among multiple administrative domains. The fundamental problem is to resolve queries that may need to consider many information sources and hence this presents a scalability challenge with respect to the number of resources that are connected via a global infrastructure spanning the Internet. However in addition, Grid information systems also have to consider the mechanisms used to obtain the information from a diverse set of resources and an information model that describes these. As a result, Grid information system architectures have been defined along with various implementations in an attempt to realise this goal.

Grid information systems are related to Grid monitoring systems with respect to the dissemination of state information about Grid resources. The prerequisite of a Grid monitoring system is a Grid information system which is required for the discovery of Grid resources. While there are existing solutions for monitoring at the site-level, such a Ganglia [15] and Nagios [16], at the global-level Grid monitoring is an area that would benefit from further research. The advancements relating the the handling of highly-dynamic state information on a global scale are of mutual benefit.

1.3 The Problem

This section presents the main challenges for Grid information systems, the specific limitations that motivate this work, and outlines the thesis proposal.

1.3.1 Challenges for Grid information Systems

For more than a decade various approaches for Grid information systems have been proposed and evaluated. The first main challenge for Grid informations systems is with the information itself. In order for information from a diverse set of resources to be used, it must be understood. A number of initiatives have attempted to define information models for Grid computing. The result of interoperation activities between

Grid infrastructures required further efforts to define a unified model. The second challenge is to ensure that all resources are instrumented to provide valid information in accordance to the information model. The third challenge is to evaluate queries that may need to consider all information from the resources themselves. The difficulties with this are directly related to the scale and complexity of the Grid infrastructure itself. An infrastructure encompassing a few identical resources in similar administrative domains, for example academic institutes in the same country, cannot be compared to a global infrastructure encompassing of thousands of diverse resources that has to consider national boundaries. Resource heterogeneity presents the specific challenges of obtaining and understanding the information about those resources. The policies imposed by autonomous administrative domains can lead to resource heterogeneity and present additional barriers for obtaining information. Of the above challenges, this thesis focuses on the scalability limitations of Grid information systems.

1.3.2 Scalability Challenges

The WLCG is the largest Grid computing infrastructure in existence today [17] and has been built to support hundreds of multi-disciplinary virtual organisations. As such, the issue of scale is paramount to its success. As the Grid information system reflects the infrastructure itself, it is particularly sensitive to scalability issues. There are three fundamental parameters that contribute to the scalability of Grid information systems; the number of resources, the complexity in terms of the distribution and the number of queries. As the infrastructure grows, the Grid information system will need to handle more information from the resources, more administrative domains and more queries. Therefore the limitations of the existing approaches need to be understood and, if necessary, new, more scalable approaches found. As the primary function of the Grid information system is to respond to queries, a single query should be unaffected if the infrastructure grows. Although the cost, such as network throughput of the system, of providing the response is important for operators of the Grid information system, it is irrelevant from a quality of service perspective. A method is therefore required to measure the key metrics of a query so that the scalability of a Grid information system can be evaluated. The performance of queries to information services is typically measured in terms of how quickly queries are answered, the query response time, and how accurately the query results compared to the actual properties of the resources they describe i.e., the information freshness.

Large-scale information services, such Web search engines, have faced similar scalability challenges; however, any techniques or solutions need to be evaluated for their applicability to the Grid environment. Although in general information services can resolve many queries against large volumes of data, and provide up-to-date query results, their applicability to the Grid environment needs to be evaluated. An in-depth understanding of the Grid computing problem is therefore required to identify the concepts that are unique and ascertain if they pose a barrier for existing solutions. Furthermore, an understanding of the limitations of Grid information systems may be resolved by considering other information systems that have proven themselves in different domains.

1.3.3 Outline of Proposal

This thesis investigates the scalability limitations of existing Grid information systems and proposes an alternative approach which improves on the current state-of-the-art. An attempt is made to place this work and hence Grid information systems in context to other domains of computer science. The novelty of Grid information systems is the information model that describes the underlying Grid computing infrastructure. The main challenge is to resolve queries across a global infrastructure encompassing of thousands of information sources that describe diverse resources. This represents a use case for distributed query processing, where the information model, global nature and number of information sources defines a specific scenario. The key novelty of this thesis is the observation that the currency of information in the Grid information system is a key metric and the adoption of a method, that was original presented in the context of Web search, to provide this measurement. This insight is used to design alternative approach that offers improved freshness for information about resources that changes frequently and hence scalability is improved with respect to the number of resources that can be supported for the same level of freshness. In general this work aims to further knowledge on Grid information systems.

1.4 Aims and Contributions

The aim of this work is to ensure Grid information systems are able to meet the current and future scalability requirements for Grid infrastructures. Specifically, this thesis contributes:

- An investigation into the nature of Grid information to identify, group and measure the changes that occur. These measurements are used throughout the thesis as reference metrics for Grid information.

- A method that takes into account these measurements to provide a quality metric for Grid information. This quality metric is used to provide a measurement for the freshnesses of information in a Grid information system and hence the accuracy of the returned result for a query.

- A benchmarking methodology that includes this quality metric for Grid information and a demonstration of its use for evaluating Grid information systems.

- The identification of the main architectural approaches for Grid information systems along with an evaluation via simulation that uses the reference metrics for Grid information.

- A novel Grid information system architecture that addresses the limitations of existing approaches, along with a prototype, which is evaluated using the benchmarking method.

1.5 Thesis Structure

The remainder of the thesis is structured as follows.

Chapter 2 provides a brief overview of Grid computing and highlights its unique concepts. The problem of query processing in a Grid environment is discussed and implementations of Grid information systems that have had production exposure are introduced. The evolution of the Grid information model is presented, which mirrors the evolution of Grid computing itself.

Chapter 3 investigates the nature of Grid information and identifies the changes that occur along with measurements of their frequency. A concept from the domain of Web search, used to measure the freshness of information, is evaluated using these metrics for its applicability to Grid information. The work in this chapter was published in the proceedings of the 11th IEEE/ACM International Conference on Grid Computing [18].

Chapter 4 presents a benchmarking methodology for Grid information systems and demonstrates its use for evaluating Grid information systems. The work in this chapter was published in the proceedings of the 17th International Euro-Par Conference [19].

Chapter 5 describes the main architectural approaches for processing queries in a Grid environment and evaluates them via simulation using the reference metrics for Grid information.

Chapter 6 presents a novel architecture for Grid information systems, along with a concrete implementation and evaluates it using the Grid information system benchmarking method.

Chapter 7 reviews the thesis contributions, and discusses potential future work.

Chapter 2

Background

This chapter presents an overview of Grid computing, Grid information systems and the Grid infrastructure, on which this thesis is based. Due to the diverse opinions on Grid computing, no definitive definition exists [5]. However, there is a common understanding of the vision for Grid computing that has transcended the technological changes and advances over the last few years. Therefore, in order to fully understand the Grid computing paradigm, it is necessary to understand its origins and how it has evolved. Hence, the original motivation for Grid computing is explored in order to understand that vision and its evolution is followed to provide a perspective on the future direction. The evolution of the Grid information model is also presented, which mirrors the evolution of the Grid computing paradigm itself. The problem of processing queries in a Grid environment is discussed and implementations of Grid information systems that have had production exposure are introduced.

Specifically, this chapter makes the following contributions:

- An overview of Grid computing along with a perspective on its future direction.
- An overview of existing Grid information models and their evolution.
- A discussion on query processing in a Grid environment.
- A survey of Grid information system implementations that have had production exposure.

The chapter is structured as follows; an overview of Grid computing, charting its evolution from the original vision, is given in Section 2.1 and introduces the Grid infrastructure on which this work is based; Section 2.2 provides an overview of existing Grid information models; query processing in a Grid environment is discussed in Section 2.3 and Grid information system implementations that have had production exposure are introduced in Section 2.4.

2.1 The Evolution of Grid Computing

The vision of Grid computing actualised at a workshop called *Building a Computational Grid* which was held at Argonne National Laboratory in September 1997 [20]. The primary goal was to solicit contributions and reviews for a book titled *The Grid: Blueprint for a New Computing Infrastructure* [21]. Pre-dating this, the term *metacomputing* was used to describe efforts that transparently connected heterogeneous computing resources via a networked environment in such a way that they could be used as easily as a single computer [22].

2.1.1 Metacomputing

During 1995, the Information Wide Area Year (I-WAY) experiment attempted to link 17 supercomputer centres to create an experimental environment for building distributed virtual reality applications [23], and was the first nationwide (U.S.) infrastructure to support collaborative computational science projects on a such a scale. A parallel goal for the I-WAY experiment was to uncover issues related to the use of distributed supercomputing centres over existing Asynchronous Transfer Mode (ATM) networks. ATM networks were chosen as at the time they provided higher bandwidth than the Internet and hence were able to handle audio, video and data more efficiently. A major challenge for the I-WAY experiment was to provide a uniform software environment across the geographically distributed and diverse supercomputing centres [23]. A software infrastructure, called I-Soft, was created to provide services including scheduling, authentication and auditing, parallel programming support, process creation and communication along with a distributed file system. Through this experience four main application classes were identified; desktop supercomputing, smart instruments, collaborative environments and distributed supercomputing.

In the influential paper *Globus: a Metacomputing Infrastructure Toolkit* [24], some general observations on the characteristics of metacomputing were made. A focus on scalability would be required to manage much larger collections of computational resources than the I-WAY experience (at the time it was the largest metacomputing experiment). Metacomputing applications may be required to execute in a wide range of environments as resource heterogeneity occurs at multiple levels, ranging from physical devices, through system software, to scheduling and usage policies. As resources are typically shared, the behaviour and performance can vary over time. Computational resources in multiple administrative domains may use different authentication mechanisms, authorisation schemes and access policies.

> "Fundamental to all of these issues is the need for mechanisms that allow applications to obtain real-time information about system structure and state, use that information to make configuration decisions, and be notified when information changes." [24]

The Globus Metacomputing Infrastructure Toolkit [24] aimed to address these issues by providing low-level mechanisms that could be used to implement higher-level services for a metacomputing infrastructure. It was composed of a set of modules where each module defined an interface that could be used by a higher-level service to invoke that module's function. These modules provided the following functionalities; resource location and allocation, communications, authentication, process creation and data access. The Metacomputing Directory Service (MDS) was introduced as a mechanism to access information about the underlying networked supercomputing system in a metacomputing environment. The core interfaces and services provided by the Globus Toolkit were intended to be used to construct higher-level policy components that could serve as middleware for application-level interfaces. The Globus components were first deployed during the I-WAY experiment as part of the I-Soft software environment [25].

2.1.2 The Advent of Grid Computing

Following on from the success of the I-WAY, in March 1997 the National Science Foundation (NSF) Partnerships for Advanced Computational Infrastructure program assembled teams of computer and computational science researchers to deploy metacomputing systems and tools across the U.S. [2]. This visionary prototype for the 21st century's distributed computing infrastructure was called the National Technology Grid. It was believed that just as the electrical power grid transformed the U.S., and indeed the world, during the past century, the Internet and the Web would provide a similar transformation. However, while rapid transformations were occurring in industry, they observed that there remained significant technical challenges for science and engineering.

A high-level view of the expected purpose, shape, and architecture was provided in [26]. Up until that point metacomputing experiments required heroic [1] efforts to utilise complex systems at a low level. The vision was to enable routine application support to geographically-separated people for the collaborative use of TeraFLOP computers and Petabyte storage systems which were connected by Gigabit networks. A Grid is the hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities [26]. Of these, they assert that it is the combination of dependability, consistency, and pervasiveness that will cause computational Grids to have a transforming effect [26]. Five classes of Grid applications were defined; distributed supercomputing, high-throughput computing, on-demand computing, data-intensive computing and collaborative computing. A key aspect was seen as the sharing of resources and it was unclear if the technical and political costs of sharing resources would outweigh the benefits, especially when crossing institutional boundaries. It was argued that resources would only be shared if there is a strong incentive, in terms of cost, scarcity or collaboration, and hence Grids would be tailored to support specific user communities [26].

The main barrier to achieving this vision was that existing systems lacked the features necessary for integrating across multiple organisations via the Internet. Three barriers were cited [26]; lack of centralised control, geographical distribution and international issues due to different policy environments. As a solution, a three-tier architecture was suggested [26] where a middle-tier mediates between sophisticated back-end services and a potentially simple front-end. The first step was the consideration of the security aspects and in particular, the authentication process to establish the identity of the user. Within an organisation it is reasonable to assume that every user has been pre-configured on every resource for which they have access. With a large number of potential users and resources in a Grid infrastructure, this assumption becomes unmanageable and is exacerbated by the dynamic and transient nature of both the users and the resources themselves. Therefore a more dynamic approach to authentication and access control was required. A solution was proposed based on the *delegation of trust* concept whereby Organisation A trusts a user because that user is trusted by Organisation B, with which Organisation A has a formal relationship [26].

Building upon this concept, a security architecture for computational Grids was

[1]The term heroic was used in [26] to describe the time investment that individual programmers were willing to contribute. While such extraordinary resource contributions may be required or available for pioneering endeavours, it is typically absent for subsequent reproduction and hence the approach may not be sustainable.

defined [27]. A security policy for Grid infrastructures was proposed that aimed to address the requirements for single sign-on and interoperability with local policies in a dynamic environment spanning multiple administrative domains. A distinctive characteristic of Grid computing is that a scientist can be a member of a multi-institutional scientific collaboration; a *virtual organisation*. In addition, the resources may require different authentication and authorisation mechanisms and policies, which will have a limited ability to change. As such, an individual user may be assigned a different account at the different institutions for enabling accounting and access control. They may have a regular account, a dynamically-assigned guest account or simply an account created for the collaboration. The key aspect of a Grid environment is that it consists of multiple trust domains, however individual operations are confined to a single trust domain and are subject to local security policy. A mapping from global to local subjects exists and operations between different trust domains requires mutual authentication. As a result, an authenticated global subject mapped into a local subject is governed by local authorisation decisions for access control.

2.1.3 The Age of Grid Computing

The concepts of Grid computing were further refined in the seminal paper "*The Anatomy of the Grid: Enabling Scalable Virtual Organisations*" [3], which was published in August 2001. Its aim was to clarify the nature of Grid computing and to establish a standard vocabulary along with an overall architectural framework. It states that the Grid concept is defined by the real and specific problem of "*coordinated resource-sharing and problem-solving in dynamic, multi-institutional virtual organisations*" [3]. Virtual organisations enable the sharing of resources in a controlled manner whereby any member of that virtual organisation may contribute to the collaboration in order to achieve a shared goal. The sharing of resources is controlled by policies defining what is shared, who is allowed to share, and the conditions under which sharing occurs. A virtual organisation is formed around the policies that govern their sharing of resources and may vary in their composition and motivation. Pre-existing distributed computing technologies at the time did not provide mechanisms for expressing the policies required for controlling the sharing relationships needed to coordinate the use of resources at multiple sites, establishing the identity of a consumer or resource (authentication), or for determining whether an operation is consistent with the applicable sharing relationships (authorisation).

The Grid architecture as defined was a protocol architecture, with protocols providing the basic mechanisms by which virtual organisation members and resources negotiate, establish, manage, and exploit sharing relationships. As the interaction is governed by protocols, local control is preserved and different implementations can interoperate. It specified various layers, following the principles of an hourglass model whereby;

> "The narrow neck of the hourglass defines a small set of core abstractions and protocols onto which many different high-level behaviours can be mapped (the top of the hourglass), and which themselves can be mapped onto many different underlying technologies (the base of the hourglass)" [28].

The base-level (fabric) provided interfaces for local control. The fabric-level implements the local, resource-specific operations as a result of higher-level sharing operations. Connectivity and Resource provided the neck. Two primary classes of resource layer protocols were defined; information protocols to obtain information about the structure and state of a resource and management protocols used to negotiate access to a shared resource according to a policy. In addition four specific resource protocols were defined [3];

- Grid Resource Information Protocol (GRIP).
- Grid Resource Registration Protocol (GRRP).
- Grid Resource Access and Management (GRAM).
- An extended version of the File Transfer Protocol (GridFTP).

The connectivity layer consisted of two parts; Internet protocols, enabling the exchange of data between the fabric layer and the resource, and protocols for authentication, secure communication, and authorisation. The implementation of the connectivity built upon on the public-key based Grid Security Infrastructure (GSI) protocols which itself built upon and extended the Transport Layer Security (TLS) protocols. Located slightly higher in the neck is the collective layer that enabled the coordination of multiple resources, building upon the connectivity and resource protocols. Here a number of services can be found that support functionalities such as resource discovery, monitoring, data replication, workload management, global policy enforcement and accounting. In addition, the high-level behaviours in the hourglass model represent the wide variety of specific applications that are used by the virtual organisation.

A significant shift took place in 2002 with the introduction of the Grid service [29]. Previously, a service was defined solely by the protocol used and the behaviours that it implements [3]. By leveraging concepts from the Web services community, the Open Grid Services Architecture (OGSA) [30] was defined. Within this architecture all Grid services are treated as network-enabled entities that provide some capability through the exchange of messages. This service-oriented view enables interoperability of those entities, regardless of the implementation, via the definition of common (standard) interfaces. The OGSA defines a Grid service as a Web service that provides a set of well-defined interfaces which also address the authorisation and policy issues that are crucial for resource sharing. How the capabilities are provided is hidden via the Grid service and from the user's perspective, details are visible only to the extent that they affect the quality of service delivered. A major driving factor for this evolution was to try and increase the adoption of Grid technology by industry. It was considered that business-to-business collaboration such as multi-organisation supply chain management, could be viewed as a virtual organisation. The integration of distributed systems among different organisations for the purpose of a shared goal has similarities to the scientific collaborations that originally motivated Grid computing. Although it is clear that Grid computing has shifted towards a Service-Orientated Approach, the specific OGSA interfaces have seen only limited adoption and many non-OGSA interfaces are used in production infrastructures today.

2.1.4 The WLCG Infrastructure

Over the past few years, the potential of Grid computing has seen many large national and international projects [7, 8, 9, 10, 11, 12, 13] provide significant investment for the deployment of Grid infrastructures [14]. The largest of these projects is the Worldwide LHC Computing Grid (WLCG) [31] that provides the simulation, storage, processing and analysis environment for the High Energy Physics community to handle the tens of Petabytes of data generated each year by the Large Hadron Collider (LHC) based at CERN. During the initial planning phase of LHC it became evident that due to the significant investments required in a power and cooling infrastructure, it would not be possible to place all the computing and storage resources at CERN or any other participating facility. Moreover, the CERN member states encouraged the sociological advantages linked with national provision whereby investments made locally would benefit institutes by providing training for students and the ability to leverage the resources for other local uses [9]. In addition, the resulting computing environment must provide a community of more than 10,000 physicists around the world [32] with access to the data and the ability to analyse it.

At the International Conference on Computing in High-Energy and Nuclear Physics in February 2000, the emerging field of Grid computing was discussed as it contained two fundamental concepts that matched the requirements; the integration of distributed computing resources and the provision of authentication and authorisation mechanisms to enable users to access resources residing in different administrative domains [9]. This realisation launched a number of projects in Europe [7, 33] and the United States [34] to deploy prototype testbeds in order to evaluate the ideas provided by the Globus Toolkit implementation against the needs of the LHC computing environment.

In Europe, the flagship project was the EU-Datagrid [35] which ran for three years starting in January 2001. In addition to evaluating the needs for the High Energy Physics community the project also considered the requirements for biology and earth observation. The project attempted to build upon the Globus Toolkit to provide higher-level services for workload management, data management, monitoring and information systems, fabric management, data storage and network monitoring. A large-scale testbed comprising of 21 sites from 9 countries was deployed as part of a two-phased development and evaluation program in order to understand if this could be integrated into a production-quality infrastructure.

Although many Grid projects pioneered the use of Grid technology for specific communities or regions, building a world-wide production Grid for use by thousands of physicists presented a challenge on a scale not previously attempted and required a significant effort to deploy the Grid infrastructure and build the supporting operational structures. The High Energy Physics collaborations themselves also needed to gain experience of such a novel computing environment. To address this, the LHC Computing Grid (LCG) project was conceived in 2001 to address these issues in two phases. Phase one (from 2002 to 2005) focused on developing prototype Grid components and understanding how to integrate these into the production services of participating computer centres. Phase two (from 2005 to 2008) saw the commissioning of the distributed computing environment in preparation for the start-up of the LHC. During this second phase the Worldwide LHC Computing Grid (WLCG) collaboration was created to oversee the long-term management of the infrastructure and to ensure that sufficient resources and support were provided from the participating computing centres. One major success of the WLCG endeavour has been the establishment of the networks of trust that provide the authentication of users through the issuance of certificates and membership in virtual organisations. This has been achieved through the creation of the International Grid Trust Federation (IGTF) [36], a body that ensures compliance with the common policies and guidelines between its participating members (the Policy Management Authorities). A world-wide Grid infrastructure spanning many countries, such as the WLCG, cannot exist without having comprehensive policies in place to ensure that the requirements of privacy and security are addressed and is something that is not address by Grid technology itself. The policies required for WLCG were developed over many years, in consultation with other Grid projects, to ensure that the same policies could be used everywhere to the greatest extent possible [9].

In February 2013, the LHC completed its inaugural three-year run and entered into a long shutdown period [37] to undergo maintenance and consolidation work, enabling it to run at a higher energy when it is restarted. At that time, the WLCG provided 320 thousand logical CPUs (cores) distributed between 295 computing clusters and processed 39 million jobs in the final month of the run. Throughout the shutdown, the WLCG continues to processes and analyse the data generated by the first run. To store that data the WLCG provides 245PB of online storage and 178PB of nearline (tape) storage, distributed between 248 storage systems.

2.1.5 Assessment

Even though WLCG is a success [38, 39], it seems that in general Grid computing has so far not met the original expectations. The original vision of Grid computing was to seamlessly deliver coordinated resource-sharing across multi-institutional virtual organisations. Unfortunately, the transparent usage of resources has not been achieved. Grids are complex, require expert knowledge of the underlying system and often result in a high operational overhead [14]. Hence, the effort required from both the virtual organisation and end-user to exploit the distributed resources remains higher than using a single computer. In the case of WLCG, the end-user is shielded by the individual experiment frameworks [40] and hence transparent access has been achieved. However, these frameworks are specific to each virtual organisation, were provided by the virtual organisation and require a team to ensure smooth operations. If the expectations for seamless Grid computing have not been realised, this suggests that the link between a generic infrastructure and specific requirements for each virtual organisation has been overlooked.

In essence, Grid infrastructures have not reached the same level of maturity as the electrical power grid with respect to the ease of use. In addition, the various efforts spawned to overcome the various issues have resulted in multiple, non-interoperable software stacks and infrastructures [41]. The very fact that initiatives exist to try and overcome this interoperability issue suggests that the original goal of Grid computing has been missed. While the need for interoperability may be understandable for different implementations, it is the original Grid protocols themselves that have seen significant divergence. Today, there exists isolated Grid islands that are based around a specific implementation and are tailored to their particular user communities. Only the High Energy Physics community has demonstrated the ability to benefit from Grid interoperability and deployed a large-scale infrastructure on a truly global scale. A mentioned previously in Section 2.1.2, it was unclear if the technical and political costs of sharing resources would outweigh the benefits, especially when crossing institutional boundaries. The needs of the LHC provided the necessary incentive, in terms of cost, scarcity and collaboration, for a global community who were already familiar with the concept of sharing resources in order to achieve a common goal.

Grid computing as a paradigm of sharing is still an aspiration for many scientific collaborators, however others have suggested moving to a more economic model. The Grid service, along with the concept of a service provider, is interesting from the perspective of the emerging domain of e-utilities, service providers who offer on-demand resource delivery though metered usage and subscription services. Although attempts have been made to increase the adoption of Grid technologies within industry, no commercial Grid computing providers have emerged [42]. Fundamentally, within industry a sharing concept involving multiple administrative domains has not emerged. A company represents one administrative domain whereby a hierarchical management structure defines and enforces the policies. Any business-to-business activity, such as the utilisation of spare capacity, would more naturally be offered as a service on a commercial basis. The resulting rise of cloud computing and other as-a-Service (Infrastructure, Platform and Service) approaches are seen as major challenges to Grid computing [42].

2.1.6 The Emergence of Cloud Computing

The cloud computing concept provides some utility to a user but hides the specific implementation via an abstract interface. Although such services are typically implemented using distributed computing methods, possibly spanning multiple data centres, the user is not exposed to these details and only experiences the quality of service provided. Neither the abstraction of a complex service to a simple interface nor the concept of a cloud are recent developments, and as-a-service providers were already in existence before the term was applied. One aspect of cloud computing is the application of a successful business model whereby a customer pays the provider on a consumption basis as a metered service. The cloud paradigm enables users to gain on-demand access to complex yet scalable services with nothing more than a simple Web browser and a credit card. This represents a step towards the original vision of Grid computing and places as-a-service providers as analogues to the utility companies in the electric power grid model. However, the concept of the virtual organisation as a solution for resource sharing is missing as there is a direct relationship between the user and the resource provider. The role of the virtual organisation serves to separate the user from the resource provider whereby the virtual organisation has a relationship with the resource provider and the user with the virtual organisation. What is required is a way to provide the user with seamless access to cloud resources without the user needing a direct relationship with the resource provider.

However, the current ecosystem is composed of multiple independent providers serving their own user communities. Cloud providers may have their own dedicated data centres or even transparently make use of another IaaS or PaaS providers. In any case, the resources are centrally controlled and belong to the same organisation. The use of such a black-box approach poses many questions of trust and policy for both academic and industrial users. In addition, cloud computing in general has adopted Web service APIs and the absence of specific standards in this area has resulting in proprietary interfaces and presents an interoperability barrier (vendor lock-in) for using multiple providers. The Amazon Elastic Compute Cloud (EC2) [43] is a prime example of a cloud computing provider. As an e-commerce company and online retailer, it needs to ensure that enough resources are available to handle peaks in demand that occur at specific times of the day or days in the year. What started as an internal consolidation process [43] developed into a product whereby the spare capacity could be outsourced to customers. The success of EC2 has resulted in the Amazon Web Services (AWS) being seen as the de facto standard in this domain and efforts are on-going

[44] to define real standards with the goal of achieving cloud interoperability.

Economics is also a driving factor for cloud computing. The leading companies providing cloud computing have invested billions of dollars in large data centres and benefit from economies of scale. Their drive towards efficiency along with competition between providers will result, especially for IaaS, in a low-margin utility business. If interoperability barriers between like-providers are overcome, the concepts of supply and demand of resources may be lead to a truly market-based approach. This may shift the focus from the service to a more abstract commodity-based view of computing.

In order to achieve this, switching between providers would need to be transparent, that is the cost of doing so must be low so that fluctuations in the market price can trigger a response. The infrastructure required to delivery this scenario would be highly dependent on information and hence an information system would play a key role. As such, much of the work in this thesis would also be relevant for a cloud-based service delivery model.

For users with either low-scale requirements (much less than the scale of the provider) or temporary high-usage of resources, cloud computing should offer a cost-effective solution over providing computing resources themselves. However, for those users whose demands are on the scale of cloud providers both in terms of time and resources, it may still be more cost-effective to provide bespoke, community-provided solutions. In either case, the rise of as-a-service utility providers is a significant step towards realizing the original electric power grid analogy.

2.1.7 Utility Computing

We are at the point where as-a-service utility providers are isolated and serve specific user communities. Users may wish to make use of multiple services from different providers and will face barriers including different administrative domains, heterogeneous services, technology, and interfaces to name just a few. What is still required are the mechanisms and motivation to establish an infrastructure that provides seamless interoperability between the cloud computing providers and enables users to view them as a single holistic resource.

This brings us back to the original vision of Grid computing, and indeed meta-computing, as an infrastructure to enable the seamless use of services across multiple administrative domains. Grid computing has proposed, implemented and gained experience of potential solutions to the problems of utilising services across different administrative domains, with a focus on collaboration between users. It is not that cloud computing is the successor to the Grid computing or a competitor but rather just part of a longer-term evolution. As-a-service providers only offer the computing fabric layer and hence only form part of the required solution. The combination of cloud computing providers delivering metered services and Grid infrastructures may finally realise the original vision.

2.2 Evolution of the Grid Information Model

A Grid information model describes the real entities and the relationships between those entities in a Grid infrastructure along with their semantics. The realisation into a concrete data model defines the syntax by which this information can be exchanged. This data model enables queries to be efficiently executed and ensures that there is agreement on the meaning with the information sources. The diversity of Grid information systems and information models presents a barrier for interoperability, as even if the information can be exchanged, its meaning may be ambiguous. The harmonisation of Grid information models is a principal aspect for Grid interoperation activities that try to bridge differences and enable virtual organisations to access resources independent of the Grid infrastructure affiliation. The information model is therefore the fundamental building block of not only the Grid information system, but of the Grid infrastructure itself.

2.2.1 MDS

The MDS information model, as shown in Figure 2.1, describes the physical and logical components of a compute resource as a set of hierarchical of elements. There are four fundamental elements; MdsHost, MdsDevice, MdsService and MdsSoftware. The most important of these elements is MdsHost which represents a networked computer and its various service access points e.g. a workstation, SMP parallel system, or the front-end server for a distributed parallel machine. MdsDevice, MdsService and MdsSoftware are used to represent local user-visible physical or logical devices, network-accessible services or an instance of installed software respectively. A set of specific attribute types can be used to add information about the particular instances of hosts, devices, services and software. Grouping objects can be used as collection points for the specific instances and provide summary information where applicable.

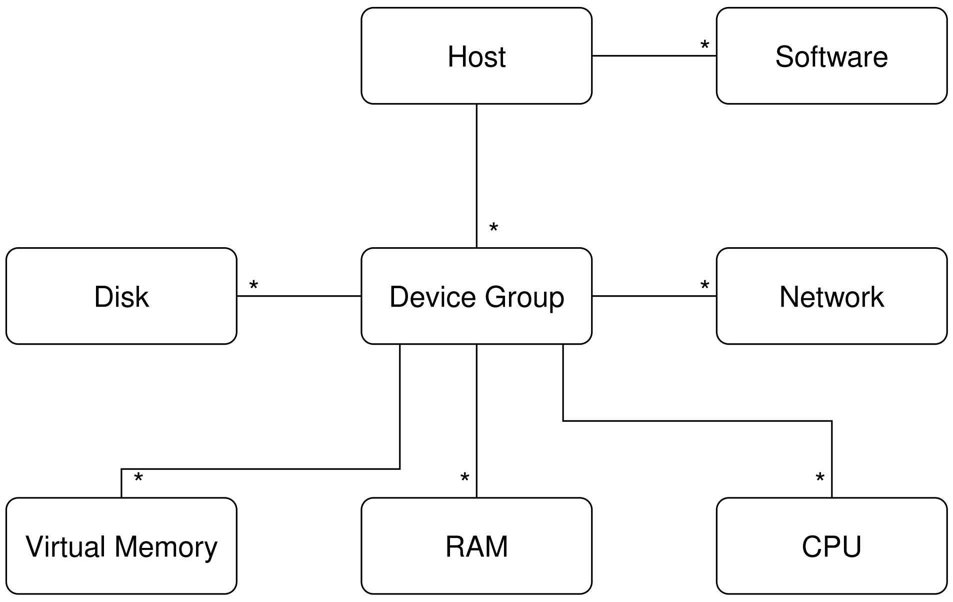

Figure 2.1: The MDS information model

2.2.2 EDG

The EDG project discovered that the MDS information model did not describe Grid entities in sufficient detail for use with higher-level services. The computing resources were managed by batch systems, and for higher-level scheduling it was important to know the state of the batch system rather than the state of the individual hosts in the cluster [7]. Similarly for data management, a description of the storage system was required. As such, each area defined its own information model to describe the entities required to enable the higher-level functionalities. For workload management, this was realized as a single entity, the ComputingElement. This entity described the Globus GRAM endpoint, the batch system, the state of the batch system and a simple description of the computing resource, which consisted of a homogeneous cluster. The storage system was defined with the StorageElement entity which described the endpoint, and the protocols supported by the storage system were defined by the StorageElementProtocol entity. Another entity defined was the CloseStorageElement which described the mount point of the local storage system.

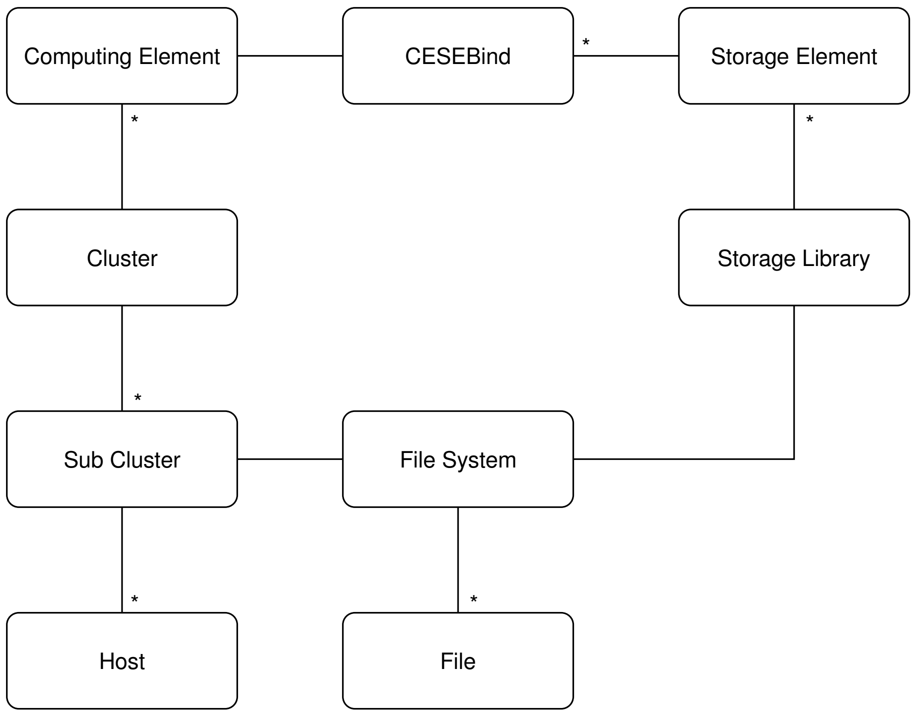

Figure 2.2: The GLUE v1.1 information model

2.2.3 GLUE 1.1

The Grid Laboratory Uniform Environment (GLUE) resulted from a collaborative effort between the DataTAG [33] and US-iVDGL [34], Globus [24], and EDG projects [7]. The aim was to define an information model (and LDAP data model) for a uniform representation of Grid resources, as shown in Figure 2.2. Information about Grid entities is encoded in terms of a hierarchy of named objects comprising attribute-value pairs that describe properties of the supported entities. At the top of this hierarchical structure is the ComputingElement which represents the entry point to a queue in a batch system. This entity has a number of associated entities that have a one-to-one relationship with the ComputingElement. These associated entities are used to describe the State, Policy and further information about the ComputingElement including authorisation rules. Below the Computing Element in the hierarchy are the Cluster and SubCluster entities. The SubCluster entity is used to describe the hardware and software properties of a homogeneous group of machines and the Cluster entity is used as an aggregation entity. The storage model was based around the concept of a StorageElement which provides access to Storage Libraries where files can be stored and retrieved. The Storage Element has associated entities representing the state and access policy for the service. The Storage Library describes the physical properties of the storage system as well as the file system used. A Storage Area represents the logical space which is associated to policy, state and access control entities. The storage spaces are aggregated to the Storage Element entity. The relationship between the Computing Element and the Storage Element is described by the Computing Element Storage Element Bind entity. This describes the mount point used in the cluster for the Storage Element and is associated to both the Computing Element and Storage Element. The iVDGL and DataTAG collaborations developed information providers to provide information in compliance with this model and they were installed on both the European and the U.S. resources. This represented a significant step towards the interoperability of transatlantic testbeds through the use of a common information model.

2.2.4 NorduGrid

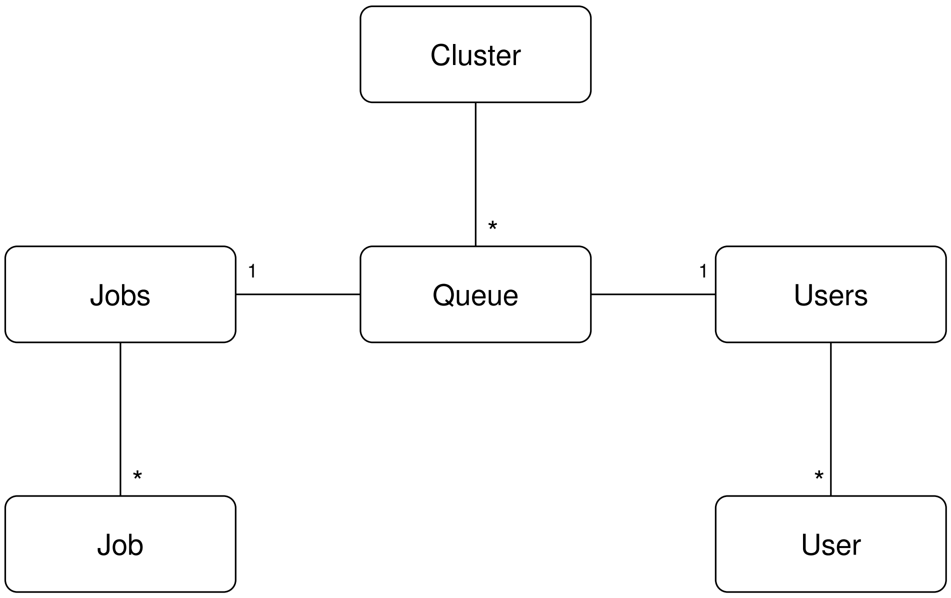

Figure 2.3: The ARC information model

The NorduGrid collaboration also found that the MDS model did not describe Grid entities in sufficient detail so the NorduGrid information model, as shown in Figure 2.3, was developed. The main entities described in this model are computing resource, Grid jobs, Grid users and storage resources. The computing resource is described using two primary entities, the nordugrid-cluster and nordugrid-queue. The nordugrid-cluster entity describes the hardware, software and middleware properties of a cluster in addition to its location and ownership. The batch system used to manage the resource as well as dynamic state information, such as number of queued/total jobs, is also part of this description. The nordugrid-queue entity represents either a traditional batch queue of a Local Resource Management System (LRMS) or an entire LRMS when the LRMS does not support conventional queues. Besides the usual queue-specific information, queue-level node attributes are also introduced to describe the hardware/software characteristics of computing nodes assigned to a certain queue. The concept of a computing queue plays a central role in NorduGrid as they are the job submission targets for the brokering process. Within the information model, every Grid user authorized to use a resource is described by a nordugrid-authuser entry. These entries are used to present the current state of the computing resource from the specific viewpoint of the user such as the number of running and waiting jobs. A Grid job entity is created for each job which is submitted to the computing resource. This entity describes various metadata and monitoring information about the job. The number of jobs does not cause a scalability problem as in the NorduGrid information system, information is not aggregated. Storage resources are described using the nordugrid-se entity. A storage resource consists of a physical data source (the storage space itself) plus the protocols, policies, services and interfaces which make the storage space available to the clients.

As the NorduGrid infrastructure provided resources to the WLCG, it was necessary to ensure interoperability between the NorduGrid information model and the GLUE information model, on which WLCG was based. To overcome the differences a translation approach [45] was chosen. This required translating the information from the GLUE information model to the NorduGrid information model and vice-versa. Once this translation was understood, translating information providers were developed to implement this approach. These would query the corresponding information system components in the other infrastructure and translate the information into the native model so that it could be added to the native information system. It was discovered that a major issue of this approach is with concepts that do not exist in one of the models. In such cases it is not possible to provide a translation approach, which is a limitation if that information is required. The result is a key-hole approach where only the minimum common sub-set of the two models, and hence related functionality, can be used.

2.2.5 GLUE 1.2 & 1.3

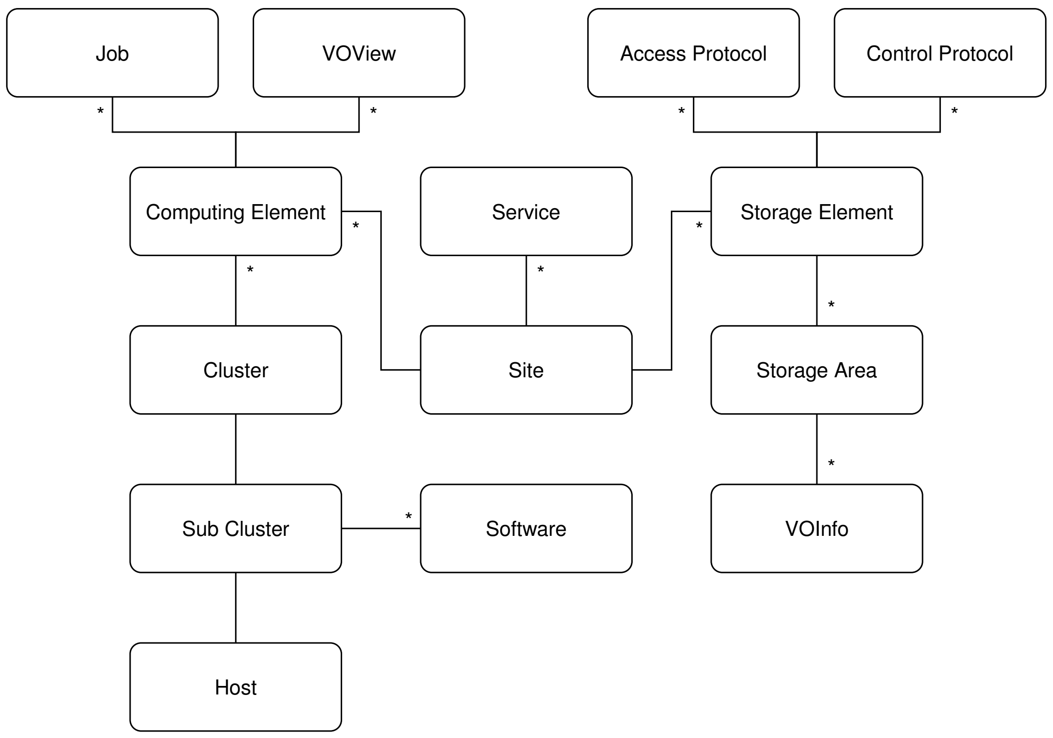

Figure 2.4: The GLUE v1.3 information model

The Open Science Grid [10] is a U.S. Grid computing infrastructure that supports scientific computing via an open collaboration of science researchers, software developers and resource providers. The OSG consortium builds and operates the OSG infrastructure, bringing together resources and researchers from universities and national laboratories. The capabilities and schedule of development are driven by U.S. participants that will use the LHC. As such, it became necessary to ensure that OSG could interoperate with WLCG so that the virtual organisations could seamlessly use both Grid infrastructures. Although WLCG and OSG produced software releases for their respective infrastructures that were based on components from the Virtual Data Toolkit (VDT) [46] (which included the Globus Toolkit), only a subset were deployed on both infrastructures and in addition, WLCG substantially augmented VDT with many extra components and services. Even if the same components are deployed, variations in versions, configuration and deployment models can occur.

With respect to the information system, both infrastructures used the same interface, LDAP, however there were significant differences in the information model.

WLCG was using GLUE with a number of extensions and OSG was using MDS, GLUE and their own Grid3 information model [47]. The Grid3 information model provided three additional entities; site, monitoring (status per virtual organisation) and location (installed software). The original aim of GLUE was to create a common information model to facilitate interoperability between E.U. and U.S. Grid projects for WLCG. As such the OSG pledged to switch to the GLUE information model on the condition that a minor revision was undertaken to address their additional (Grid3) use cases. A proposal for version 1.2 of the GLUE information was discussed which in addition addressed a number of minor problems with the original version. Soon after version 1.2 of the information model was defined, updated information providers were created, included in the respective software releases and rolled out across both infrastructures. This work successfully achieved interoperability between the two infrastructures and again demonstrated how to achieve interoperability by moving to an agreed common information model. A year later work began on version 1.3, shown in Figure 2.4, with the main focus of adding support for the Storage Resource Manager [48] in time for the start-up of the LHC.

2.2.6 The Grid Interoperability Experience

Over recent years a number of bilateral Grid interoperation activities [45, 41] emerged that attempted to bridge differences between Grid infrastructures with the aim of enabling virtual organisations to access resources independently of their Grid project affiliation. Without such efforts the virtual organisations are artificially limited by the boundaries created by the different Grid infrastructures or have to separate their entire work-flow by region. Following an ad-hoc meeting at Supercomputing 2005 in Seattle between representatives of different Grid infrastructures, the Grid Interoperability Now Community Group (GIN CG) was established in the Open Grid Forum that aimed to build upon these bilateral activities to create a more uniform Grid landscape [41]. The GIN activities focused on the four corner stones of Grid interoperability: Security, Workload Management, Data Management and Information Systems. For the information system, an interoperability matrix, Table 2.1, was created that showed amongst others, the participating infrastructures, the query protocol, the information model and the data model, in order to understand the complexity of the problem.

An attempt was made [1] to translate information from all Grids into a common form. As the GLUE information model was defined to facilitate interoperability, it was chosen as the information model and LDIF was used as the data model as it was the

Infrastructure	Query Protocol	Information Model	Data Model	Sites
EGEE [8]	LDAP	GLUE 1.3	LDIF	195
OSG [10]	LDAP	GLUE 1.3	LDIF	18
NorduGrid [13]	LDAP	NordGrid	LDIF	19
Naregi [49]	OGSA-DIA	GLUE 1.3	XML	5
TeraGrid [11]	WSRF	GLUE 1.3	XML	13
Pragma [50]	HTTP	GLUE 1.3	LDIF	29
DEISA [12]	WSRF	GLUE 1.3	XML	12
NGS [51]	LDAP	MDS	LDIF	7
APAC [52]	WSRF	GLUE 1.3	XML	4

Table 2.1: Interoperability matrix used for the Supercomputing 2006 demo [1]

most widely used. Translators were created that extracted information from the native information system, translated the query response and the result was used to populate a BDII. This information was used to create a demonstration [53] for Supercomputing 2006, which showed the location of all the sites from the Grid infrastructures that were participating in the GIN activity and to which Grid infrastructure they belonged. This demonstration was repeated at Supercomputing 2007 and showed how the Grid had evolved during the year. It also showed that a few more infrastructures had adopted the GLUE information model based on the success of the 2006 demonstration.

This experience highlighted a number of major obstacles with such an approach. Firstly, different naming and semantics of attributes resulted in the need for heuristics to populate certain attributes and the assumptions used may only be accurate by coincidence. Secondly, as some information is just not available, it is difficult to provide values for missing information. This can sometimes be worked around by either hard-coded values in the translator or using default values, but in many cases this is not possible. Poorly translated or missing attributes can be problematic for applications that rely on those attributes for successful operation. Even if the same information model is used, these problems are not entirely avoided as different projects may interpret the model slightly differently and may decide not to use some of the optional attributes.

2.2.7 GLUE 2.0

The GIN activity highlighted the importance of a common information model for Grid computing. While it is possible to do limited translations, if the fundamentals of the

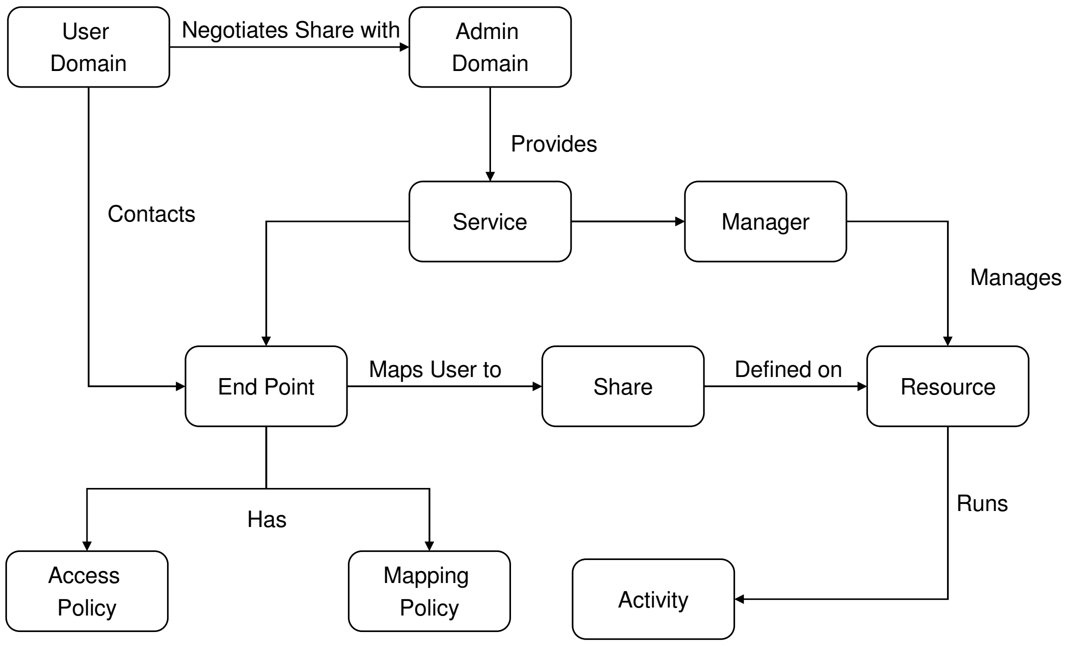

Figure 2.5: The GLUE v2.0 information model

information models differ or the information required for a particular use case is not present, it is impossible to provide a translation. If interoperability is to be achieved there needs to be agreement on the fundamental model of Grid computing, and the information required for the cross-infrastructure use cases needs to be identified. The survey of Grid information systems from the GIN CG showed that the majority of Grid sites had adopted the GLUE 1.3 information model [41]. While the NorduGrid information model had a very mature description of a Computing Element, it lacked a detailed storage model. With the agreement of all participants in the GIN CG, the GLUE Working Group was created within the Open Grid Forum to oversee a major revision of the GLUE information model that would consolidate the NorduGrid and GLUE 1.3 information models into a community-agreed standard. As the endeavour built upon existing information models, it benefited from several years of experience in the context of production Grid infrastructures.

The modelling activity concentrated on capturing the abstract representation of entities, their properties, operations and relationships in an implementation-independent information model. Before in-depth work could be undertaken on information modelling, a number of fundamental Grid concepts needed to be agreed. The result is the GLUE Main Entities model as shown in Figure 2.5.

At the core of this model is the concept of a Service and as such this demonstrates that a Grid has a Service-Orientated Architecture. A Service enables a User from a User Domain to run an Activity via an Endpoint on a Share of a Resource in accordance to an Access Policy. A Service is therefore a container which is used to describe a collocation of sub-components that are required to fulfil a particular function. Services are related to an Admin Domain which provides and manages the Service. This concept is important for infrastructures as there is a Service Level Agreement between the User Domain and the Admin Domain which places requirements on the quality of service provision. An Activity is a unit of work managed by a service and can have relationships to other activities being managed by different Services.

For many use cases it is necessary to define a more detailed information model about specific Services. The two main Services that require more detailed information are the Computing Service and the Storage Service. The Computing Service is a specialisation of a service for creating, monitoring and controlling computational activities more commonly known as jobs. This results in the following specialised entities for the Computing Service: Computing Manger, Computing Resource, Computing Share, Execution Environment, Application Environment and Job. A Job is an Activity that runs on a Computing Resource. The Computing Resource is a grouping concept for a set of different types of Execution Environments where the aggregation is defined by the common management scope e.g. a batch system. The Execution Environment provides a description of the hardware and software characteristics available to a Job. An Execution Environment may also contain one or more Application Environments. The Computing Share is a utilisation target for a set of Computing Resources defined by Mapping and Access Polices and characterized by the Status of the Computing Share. The Storage Service is a specialisation of a service for storing and retrieving file based data. This results in the following specialised entities for the Storage Service: Storage Manger, Storage Resource, Storage Share, Storage Service Capacity and Storage Share Capacity. The Storage Resource is a homogeneous storage device providing a storage capacity, managed by a local Storage Manager. The Storage Share is a utilisation target for a set of Storage Resources defined by Mapping and Access Polices and characterized by the Status of the Storage Share. The Storage Service Capacity is a description of the size and usage of a homogeneous storage extent; the storage extent is aggregated at the storage service level by the type of Storage Resource.

2.2.8 Other Models

The Common Information Model (CIM) [54], defined by the Distributed Management Task Force (DMTF) [55], describes managed elements in an IT environment. It is an open standard that aims to provide a consistent view of those elements independent of their manufacturer or provider in order for management software to work across different implementations of those entities. While it describes in detail the components of a system, it does not provide the high-level abstract descriptions of services that are required for Grid computing.

Classified Advertisements (ClassAds) [56] is the model used within the Condor system [56] for match-matching. A ClassAd is a form of attribute/expression property list where in the simplest case the expressions are simple constants (integer, floating point, or string) In Condor they are used for describing both the computational jobs and the resources on which they run. The ClassAds are *matched* using the match-maker to find suitable resources. This abstract framework can be used for anything that can be described by a list of attributes and expressions. However, there must be agreement on the attributes name in order for the matchmaking to succeed. Within this framework, attributes names can differ depending on the entity that produced the ClassAd. Condor defines eight types of ClassAds, along with their attributes and represents a defacto standard. These ClassAds are focused on the job scheduling use case (Job, Machine, Schedular etc.) and hence do not describe the full-set of high-level abstractions required for Grid Computing. The GLUE information model represents an open standard for attribute naming and can be described using the ClassAds framework.

2.3 Processing Queries in a Grid Environment

As stated in the previous section, one fundamental aspect of Grid computing is the need to obtain information about the structure and state [24] of the underlying networked system in a computing infrastructure. Challenges for such global information systems were outlined in an earlier paper [57] which argued that scalability, autonomy, and the (un)availability of information sources would make brute force selection techniques, exhaustive searches, and global synchronisation (consistency) impossible. A common query [57] made to such a system is the global selection query which needs to consider all information sources. Although contacting all information sources may be possible for a single query, due to the long latencies and required throughput, it was thought impossible for a large number of queries. Caching could improve the response time for queries but the questions are what to cache, where to cache, and how to cache? It was also pointed out that unavailability is also an issue as scale grows. The chance of failure in at least one component increases dramatically which means that at least some of information sources will be unavailable if there are many. Automatic techniques for identifying and removing information sources that never answer or have unreliable data may be required. Finally, if each organisation is autonomous, this could lead to the issue of non-uniformity within the system. The difficulties of maintaining consistency in the presence of scale, autonomy and unavailability need to be closely examined. They suggest that research should identify methods for delivering more parallelism in global query processing, possibly through a tree of query processing engines whose sole role is the forwarding of sub-queries to individual resources and the coalescing of results.

Today's Grid infrastructures are composed of thousands of Grid services that are widely distributed geographically. Information describing each Grid service is provided by the service itself and hence the Grid service is the primary information source. The information provided conforms to an information model and as such is restricted to a finite set. Queries need to be resolved that may consider all information sources, the global section query, in order to enable efficient work flows that may utilise multiple cooperating services.

To resolve a query over all information sources (data retrieval), it is first necessary to discover those information sources (information retrieval). Information retrieval [58] is concerned with retrieving information about a subject. Early examples of such systems focused on indexing and searching for relevant documents in a collection. More recently the introduction of Web search engines has driven the latest research in this area. Queries are typically expressed in natural language and documents or pages are ranked according to relevance. Data retrieval provides a solution, if possible, based on the assumption that the information is or is not available in a database [58]. It is not concerned with the origin of the information, only how the database is updated (e.g. transactions). The specific domain of distributed query processing is of interest as it deals with resolving queries over distributed information sources. Queries may be issued in a declarative query language such as SQL and executed over data (e.g. database). In both information retrieval and data retrieval the goal is to execute the respective queries as efficiently as possible in order to minimize the query response time.

In terms of data structure, Grid information is structured as it conforms to an information model and this enables a declarative query language to be used. In this respect Grid information systems share much in common with distributed query processing. However, due to the issue of global synchronisation, consistency requirements need to be relaxed and in addition, as information sources can be unavailable, the results maybe incomplete. Grid information systems therefore also share much in common with information retrieval systems. In a typical distributed database system it is the different tables (object types) that are distributed or tuples (object instances) are logically grouped. Grid information systems are extremely distributed, i.e. each tuple (object instance) is found at a different location. As there are many information sources and queries, scalability is a major concern. While information retrieval systems have proven to work with millions of sources and millions of clients [59], so far distributed query processing has not been attempted on such scale [58].

As Grid information systems share much in common with both information retrieval and distributed query processing, knowledge from these domains is highly relevant. However, while concepts from distributed query processing maybe useful and the goal is similar, to execute queries as efficiently as possible, within Grid information systems the focus is on efficiently executing many queries, from many clients for many information sources rather than optimizing an individual query. A typical architecture for query processing is shown in Figure 2.6. The query processor receives a query as input, translates it, creates an optimal query execution plan, and a query execution engine executes that plan in order to obtain the results of the query [58].

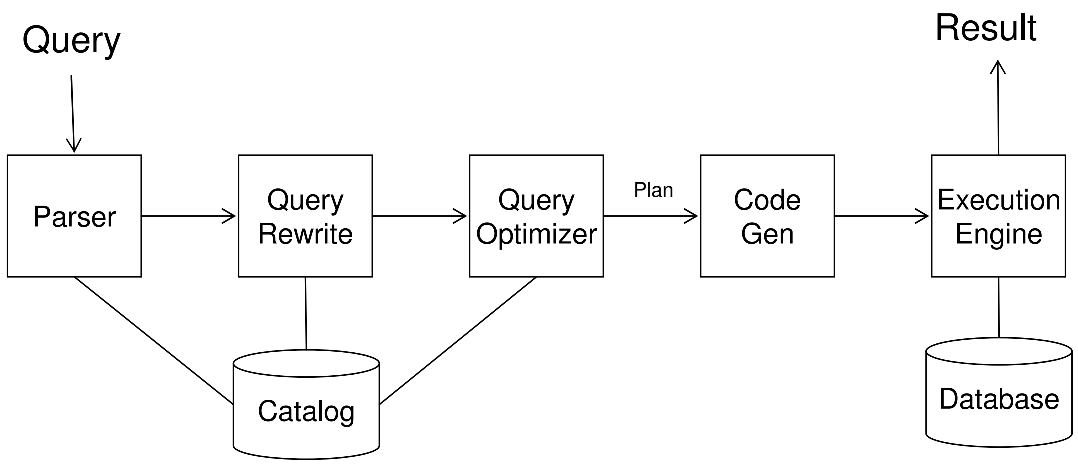

Figure 2.6: A typical architecture for query processing

This architecture can be used for any kind of database system and can also be applied to processing queries in a Grid environment. In this architecture a catalogue stores all the information needed in order to parse, rewrite, and optimize a query [58]. In addition to the definition of the database, the catalogue also provides information about how the database has been partitioned and physical information such as the location of copies of tables, information about indices, and statistics that are used to estimate the cost of a plan [58]. For a distributed database system, the questions of where to store and whether to replicate or cache the catalogue arise [58]. In environments where the catalogue can become very large (hundreds of gigabytes rather than hundreds of kilobytes) and are frequently updated (such as in Grid information systems) it is suggested to partition the catalogue and store catalogue data where it is most needed [58].

The first question that arises is what is the analogous catalogue for Grid information systems? How are the information sources discovered and indexed? The domain of Web search engines is concerned with the discovery and indexing of documents, especially at scale. Although it is difficult to provide an accurate estimate of the size of the Web, today's Web search engines index billions of pages and handle hundreds of millions of queries per day. However, in order to achieve this, significant [2] resources are required which comes at a cost.

For Grid infrastructure a light-weight solution is required that uses a negligible amount of resources when compared to the overall resources available. This holds for components in general, as each resource required to provide the Grid infrastructure results in less resources that can be dedicated to primary purpose of the infrastructure, which in most cases is the storing and processing of scientific data. Grid information systems currently only need to index thousands of information sources and millions of queries per day [60]. In this respect the scale of Web search significantly surpasses that of Grid information systems. Whereas the Web is a vast collection of completely uncontrolled heterogeneous documents, the Grid is a managed infrastructure and information is structured. Web search is therefore a much more complex problem due to the scale and unstructured nature of the documents.

The Web search problem is split into two main areas; retrieving Web pages and index creation. Due to the decentralised nature of the Web, Web Crawlers have been created to discover Web pages [61, 62]. These traverse the Web by following the

[2]Due to commercial sensitivities, search companies have not published detailed information on their internal infrastructure and the resources required for Web search. However, we can infer from the vague information provided and the number of data centres that they operate that this is not insignificant.

HTML links between pages and is an active area of research. As the link concept between services is absent in the Grid environment, such an approach can not be used. This leaves the open question of how to discover Grid services in order to index them?

As the Grid is a managed entity based on policies, more structured approaches [63, 64, 65] have been adopted to discover services. The common aspect with all these approaches is that they provide a central point that can be queried. However there are subtle differences with how the information is aggregated and managed. Externally, from the perspective of a query, they are identical. The main issue is related to the authoritative source of the information that defines what services are participating in the infrastructure and to which administrative domain they belong. It should be highlighted that information about which services should and should not be in the infrastructure does not originate from the service itself. The subtleties of control and ownership for such information are of utmost importance in a global infrastructure. How information is propagated and managed is a question of policy. It is therefore possible to provide an abstraction of the difference approaches using the concept of a policy engine as shown in Figure 2.7. The policies identify the authoritative source of information and who manages it.

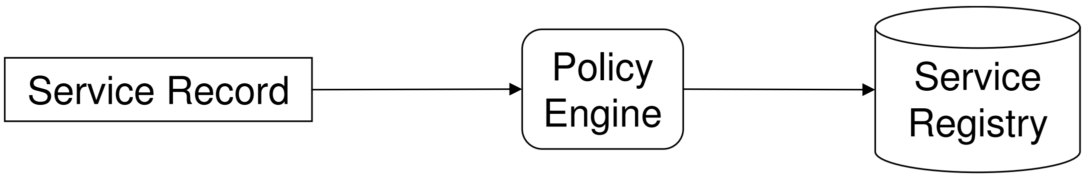

Figure 2.7: An abstraction of service discovery

In the Grid environment therefore, the catalogue is the service registry or index that provides the list of services along with their location in the form of a URL (the service record). Additional information may also be provided for internal use, such as time-to-live values, or to improve query performance via the use of an index. However for the global selection query, only the URL identifying the information source for that service is important. Other attributes, such as service type, are useful to index for improving query performance. The prerequisite of a Grid information system is therefore the existence of a catalogue that can be used to discover all the information sources and hence Grid services in a Grid infrastructure. While this work assumes that such a catalogue exists, it is an area that could benefit from further research.

In terms of query optimisation, the domain of distributed query processing provides a useful concept. This is whether to move the query to the data (query shipping) or to move the data to the query (data shipping) [66]. Query shipping does not scale well if there are many clients as the servers are potential bottlenecks in the system. Data shipping scales well as it makes use of the client machines, however, it can create very high communication costs if caching is not efficient resulting in redundant data being shipped to the clients.

If data shipping is used, this opens up the question of where copies of the data should reside so that the queries are executed in the most efficient way. Two types of data shipping have been defined; replication and caching [58]. Replication is typically coarse-grained: a whole table, a whole index, or a whole (horizontal) partition and propagation-based protocols (using a separate process) are used to keep replicas consistent and accessible [58]. Caching, on the other hand, is a by-product of query execution and typically uses a method that is based on invalidation which removes out-of-date copies from a client's cache so that copies of data are only available in a client's cache as long as the data has not been updated. Replication can occur at servers even if no queries are processed by these servers, whereas the cache of a client is empty if no queries have been processed by that client. As a consequence, caching decisions need to be made by the query processor while replication decisions can be made by a separate component that is established at every server and works independently of the query processor. Replicas are kept at servers until they are explicitly deleted whereas copies of data are kept in a client's cache until they are replaced by copies of other and more interesting data using a replacement policy or until they are removed from the cache because of invalidation.

In general, replication helps to move data near to a large group of clients so that these clients can access the data cheaply the first time they need the data. Caching makes it possible to access data cheaply when the data are used repeatedly by the same client [58].

Work on dynamic replication algorithms has focused on two aspects; reducing communication costs by moving data closer to the client and load balancing for servers. Put simply, replication is required if there are more queries than updates (expansion test) and should be avoided if there are more updates than queries (contraction test)[67]. For caching, the cache investment needs to be calculated; the benefits gained from caching and the cost of creating the cache. Typically a fault-in approach is used; that is to use the cache if available and only update the cache or store the query result if the cache is invalid.

2.4 Grid Information Systems

Since the advent of Grid computing, a number of Grid information systems have been developed that process queries in a Grid environment in order to discover the structure and state of the participating Grid services. This section presents an overview of the existing implementations that have been deployed in production Grid infrastructures.

2.4.1 Metacomputing Directory Service

The MDS [65] from the Globus project provides an implementation of the two information protocols (GRIP and GRRP) from the proposed Grid architecture and offers a clear separation between inquiry (data retrieval) and discovery (information retrieval). The MDS consists of two basic elements, information providers and information indexing services, which together can be used to build a hierarchy of query processing engines that can forward sub-queries to individual providers and merge the results (a query shipping approach).

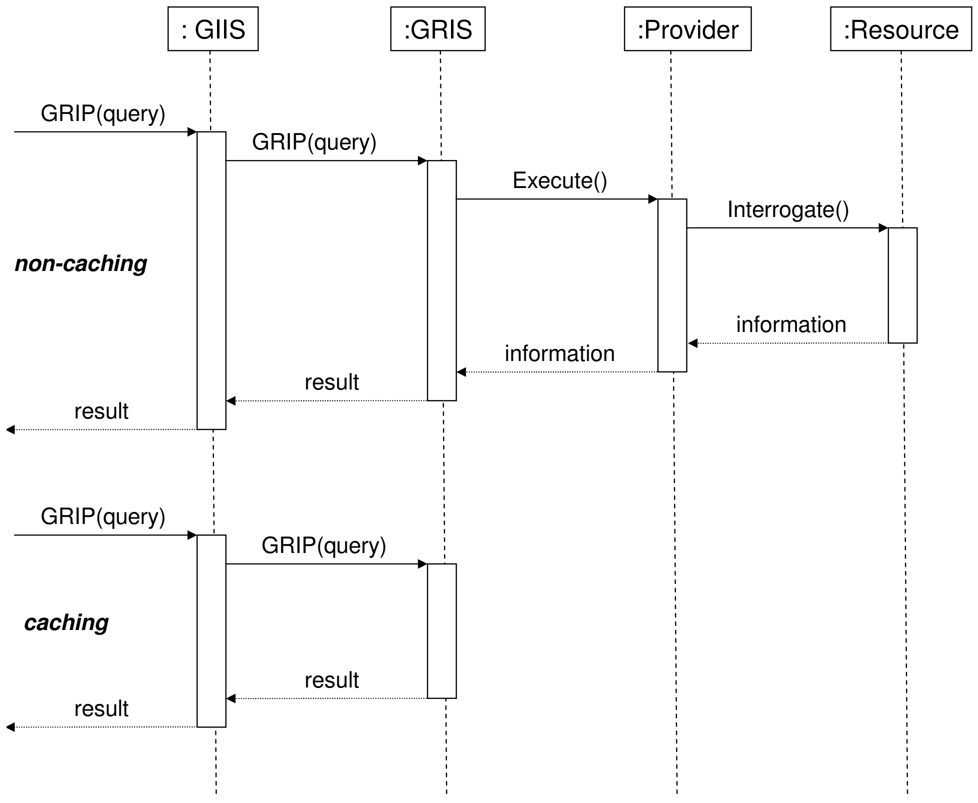

Figure 2.8: A sequence diagram of interactions between the MDS components

The interactions between the MDS components are shown in Figure 2.8. Information providers obtain information about the resource via local operations and the result is structured to conform to a predefined information model. The MDS provides a configurable information provider framework called a Grid Resource Information Service (GRIS). The GRIS parses each incoming GRIP request and then executes one or more information providers depending on the type of information requested. Results from the information providers are then merged and sent back to the client. To improve efficiency, an information provider is only considered if it is relevant for the query. A GRIS is configured by specifying the type of information that is produced by an information provider and the information provider itself. To control the intrusiveness of GRIS operation, improve the response time and for deployment flexibility, each information provider's results may be cached for a configurable period of time to reduce the number of information provider invocations. This is achieved by specifying the cache time-to-live (TTL) configuration parameter per information provider. The appropriate value depends upon both the dynamism of the resource and the cost of the information provider mechanism. Results returned by an information provider are filtered by the GRIS to eliminate any objects that do not match the query and it is necessary to ensure that the search semantics are implemented correctly. Filtering this way simplifies the implementation of the information providers and improves performance as a superset of results can be processed from the cache for each client request.

Indexing services, collect, manage, index, and respond to information provided by one or more information providers, and are analogous to the catalogue in the query processing architecture that was shown in Figure 2.6. Interactions between the information provider and the indexing service are described in terms of the GRRP. GRRP is a soft-state protocol [68], meaning that state established by a notification may eventually be discarded unless refreshed by subsequent notifications. Each GRRP message (a service record) contains the name of the service that is being described, a URL to which GRIP messages can be directed, the type of notification message, and timestamps that determine the interval over which the notification should be considered valid.

The MDS implementation provides an index in the form of an aggregate directory following a hierarchical structure (a hierarchical catalogue) using the Grid Index Information Service (GIIS). The GIIS accepts GRRP messages from a child GRIS or GIIS instance and merges these information sources into a unified information space. The GIIS framework comprises three components: generic GRRP handling, pluggable index construction, and pluggable query handling. The GIIS (and GRIS) implementations are configured so that GRRP can be used for both registration and invitation. With registration, a GRIS explicitly registers to a specific GIIS. In the case of invitation, a GRIS is asked to join by the GIIS or perhaps a third party. If a GRIS agrees to join, it then uses GRRP to register itself with the specified GIIS in a fault-tolerant manner. Grid Security Infrastructure (GSI) [27] is used to provide access control to information and authenticity of GRRP messages. A typical deployment scenario for the MDS can be seen in Figure 2.9.

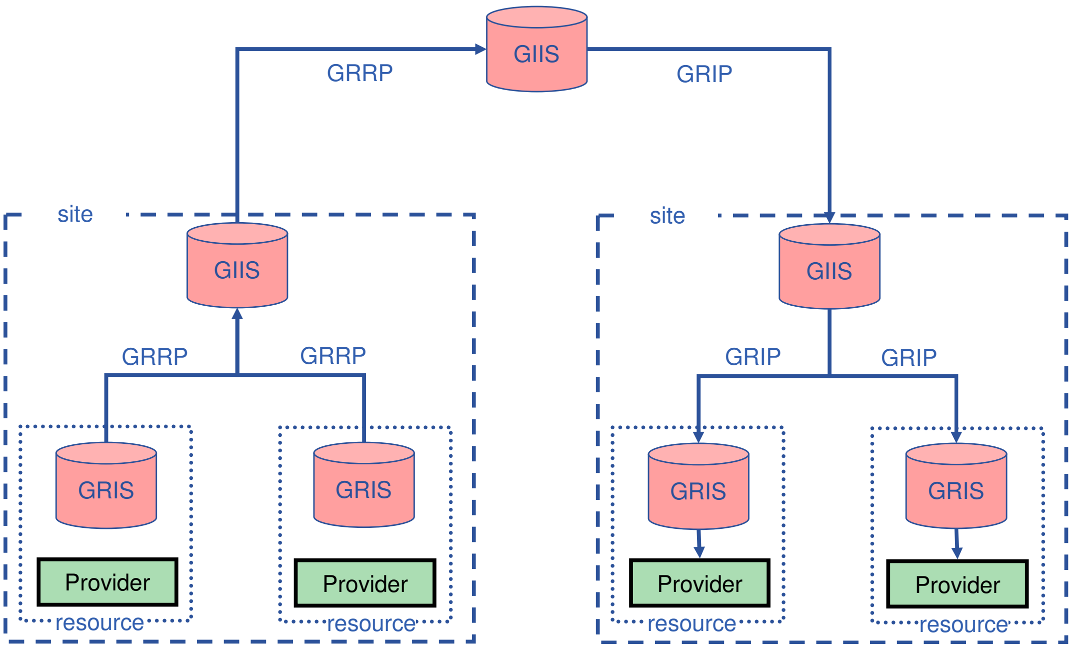

Figure 2.9: A typical deployment scenario for the MDS

The MDS implementation adopted the standard Lightweight Directory Access Protocol (LDAP) [69] as the GRRP transport mechanism, with GRRP messages mapped onto LDAP add operations of the standard LDAP protocol, and for the GRIP, where it is used to define the data model, query language, and transport protocol. Using a common protocol for both GRIP and GRRP simplified the implementation and promoted interoperability. This choice of LDAP was made for pragmatic reasons in order to simplify MDS development as its widespread use brought familiarity of the LDAP data formats, programming and deployment to developers and operators of the system. The information itself is structured to conform to the standard LDAP data model, the LDAP Data Interchange Format (LDIF) [70]. In LDIF, entries are described by a set of object classes which in turn define a set of attribute-value pairs and the MDS data model was defined within the scope of this framework. Object classes are used to define each type of entry and a tree-structured namespace called a directory information tree describes the hierarchical relationship between object classes.

The GRIS and GIIS functionality are both implemented as special purpose back-ends for an OpenLDAP server. A customized back-end can be used for specific information providers and these can be plugged into a standard protocol interpreter. The interpreter handles all authentication, data formatting, query interpretation, results filtering, network connection management, dispatch to the appropriate back-end and allows the OpenLDAP server to be used without modification. The OpenLDAP front-end can also decode GRRP messages and deliver them the appropriate back-end which performs any actions necessary to construct and maintain the GIIS's indices. As OpenLDAP also includes optional security bindings using the Simple Authentication and Security Layer (SASL) library, the public-key based GSI protocols also supported.

In summary, the MDS is an implementation of a query shipping approach where caching is available to improve performance. The GRIS provides a service-level interface that can be used to query the service directly. A distributed catalogue is provided using a hierarchical GIIS deployment strategy, where the GRRP is used as the mechanism to discover services. This GIIS structure also provides a hierarchy of query processing engines similar to the tree of query processing engines as suggested in [57].

The EDG Project

The MDS was used by the EDG project [7] to provide a Grid information system for a large-scale testbed encompassing 21 sites from 9 countries, which ran from 2001 to 2004. An MDS GRIS was deployed on the Computing and Storage resources and the GIIS was used to provide a three-tiered hierarchy of indexing services; site, country and top. During the initial evaluation of MDS it was found that the host-centric MDS information model did not describe the required properties of the Computing and Storage resources used in EDG, such as the queues in a batch system, and hence an alternative model was defined. As a consequence, new information providers were developed to provide this missing information in accordance to the EDG information model. The initial deployment [71] revealed instabilities which contributed to an unacceptably high failure rate for the end-users and frequent interventions were necessary to keep the testbed running, the most common being the restarting of the MDS daemons. During high query rates, the testbed falsely appeared to have no resources as both the GIIS and GRIS caused queries to timeout during these periods. Frequent problems were also observed with failed GRRP registrations, even in a stable testbed environment. The top-level GIIS would fail to respond when experiencing a high query load and the number of sites in the infrastructure also affected the performance. With four sites the MDS would respond to queries but the addition of a fifth site caused the top-level GIIS to become unresponsive.

To work around these issues a cache based on standard OpenLDAP server was used alongside the top-level GIIS. Periodically, information was extracted from the top-level GIIS and added to the OpenLDAP server. Queries would subsequently be directed to the OpenLDAP server instead of the top-level GIIS. This cache approach continued to be responsive under high query loads and in December 2002 became a standard component of the EDG Middleware. It was named the Berkeley Database Information Index (BDII) in order to distinguish it from the MDS GIIS. The BDII was enhanced to obtain information directly from the site-level GIIS and a file was used containing the LDAP URLS representing the site-level GIIS. An update script extracted these URLs from the file and queried each GIIS in turn. A timeout of 30 seconds was used for each query to prevent the script hanging if a GIIS failed to respond. When all the results had been returned, the OpenLDAP database was updated and the old entries removed. By relying on a static file for storing all the endpoints, this approach avoided the problems associated with the dynamic nature of the GRRP protocol. The effort required to manage the file was less than using the GRRP protocol, as the existence of the endpoints had to be communicated in any case to allow the GRRP registrations. In addition this approach enabled the infrastructure to be centrally managed which was desired by the testbed managers.

Updating the cache asynchronously from the query ensured that the cache was always available and there was never any significant increase in the query response time. Decoupling the query from updating the cache also ensured that the queries would never be propagated to lower levels of the information system hierarchy. This avoided the situation where problems with the lower levels of the hierarchy could manifest as problems in the upper layers. These changes resulted in a data shipping approach being adopted whereby the data is obtained from the lower level by querying for all information. The catalogue in this scenario is the file containing the LDAP URLs. The discovery process consisted of the testbed managers knowing the structure of the testbed and ensuring that the relevant LDAP URLs were included in the file.

The NorduGrid Project

The NorduGrid project [13] started in May 2001 and aimed to build a Nordic testbed for Grid computing. The project focused on the deployment of the Globus Toolkit and the higher-level services developed by the EDG project. However, at the beginning of 2002 the software from the EDG project was not considered mature enough to meet the requirements for a stable infrastructure and hence the NorduGrid project decided to create their own distribution for their production infrastructure [72].

The NorduGrid distribution provided an LDAP-based Grid information system that highly leveraged components from the MDS [72]. The two main differences with the use of MDS in NorduGrid were the information model and their use of the GIIS. NorduGrid, like the EDG project, found that the MDS information model did not describe the resources sufficiently and hence defined their own information model and new information providers were created that complied with the NorduGrid information model.

At the resource-level, the GRIS was used as envisaged to provide local information about the resource. Caching was enabled to improve the performance and to avoid the information provider being executed too frequently. The GRRP soft-state registration protocol was also used as envisaged for registration of both the GRIS and GIIS. The topology of the NorduGrid information system was a two-level hierarchical tree, based upon the geographical location of resources. Resources belonging to the same country were grouped together and registered to that country's GIIS. All the country GIISs were registered to a top-level NorduGrid GIIS and to avoid a single point of failure, multiple top-level GIISs were used.

However, similar to the EDG project, it was discovered that the caching capability of the GIIS back-ends was unreliable and did not scale, hence NorduGrid operated with the cache disabled in the GIIS. In such a configuration, the GIIS is used merely as a resource index containing the contact information from the soft-state-registrations (a catalogue) and resulted in a pure query shipping approach. The NorduGrid information system clients only use the GIIS to obtain the location of a GRIS, or a lower-level GIIS, and then query the GRIS directly. This configuration and use of MDS enabled the NorduGrid project to have a functional Grid information system in production by May 2002 and it is still in operation.

The WLCG Grid Information System

In January 2003 the WLCG project attempted to deploy a testbed using the software provided by the EDG project. By September 2003, 32 sites were integrated into the infrastructure which surpassed the scale achieved with the EDG testbed. This increase in scale uncovered a few performance issues with the BDIIs cache population mechanism. The cache was updated by querying each site GIIS for all information in series and hence the time taken to update the cache was proportional to the number of sites. The update time would be further increased by slow or failed responses for which a timeout was configured as a protection. To address this, the cache update method was modified to perform the queries in parallel.

Having only one single top-level BDII instance was seen as both a scalability bottleneck and a single point of failure. To deploy multiple top-level BDII instances, a method was required to synchronise the configuration file that provided the list of endpoints for all the site-level BDIIs in the infrastructure. The solution was to automatically update the configuration file from a central location; a Web server was used to host the master file, and the BDII was modified to periodically update the local configuration file from the Web-based master configuration file. This not only ensured that the configuration files were synchronized and but also provided a mechanism to centrally manage the infrastructure.

The move from a testbed to a production infrastructure brought with it an increased management overhead with policies, procedures and tooling. One such tool was the Grid Operations Centre Database (GOC DB) [63] which stored details about a site's existence, the services it provides and various other details required by the operational policy documents. One piece of information which the sites were required to provide was the LDAP URL for the site BDII. The Web-based master configuration file used by the top-level BDIIs would be automatically generated from the GOC DB information. This approach essentially replaced the original GRRP with a manual procedure based on the compliance with operational policy for site registration. In this scenario the Web-based master configuration file, along with the GOC DB, represents the catalogue.

Although WLCG drove the evolution of this information system, other virtual organisations that are not related to High Energy Physics are also supported. This has resulted in a complex relationships between services, sites, projects and virtual organisations. While the narrow view of the WLCG information system would be limited to only those sites and services that support one of the main WLCG VOs, in this thesis we consider the WLCG information system as the generic infrastructure and hence include all services that can be seen in the information system regardless of the virtual organisation that they support.

For over ten years since its initial deployment, the WLCG Grid information system has been in constant operation and has required regular re-engineering to cope with ever higher scalability demands. A recent study [73] showed that as the WLCG infrastructure expands, the total volume of information increases, along with the time-taken to update the top-level BDII. This adversely affects the freshness of information, although there was no mention of the resulting impact, and a differential update algorithm was evaluated to reduce the update time. It was suggested that further investigation is required into efficient transport mechanisms that will only propagate the changes.

2.4.2 The Relational Grid Monitoring Architecture

The Relational Grid Monitoring Architecture (R-GMA) [74], shown in Figure 2.10, was developed within the EDG project as a Grid information and monitoring system. It is based on the the Grid Monitoring Architecture (GMA) [75], which follows a simple consumer-producer model. The GMA consists of three components: consumers, producers and a registry. Producers register themselves with the registry and describe the type and structure of information that they provide. Consumers can query the registry to find out what information is available and locate producers that provide the information required. The consumer can then contact the producer directly to obtain the relevant information. The GMA also describes the registration of consumers, so that a producer can locate consumers. The main reason to register the existence of consumers is so that the registry can notify them about changes in the set of producers that interests them. This mechanism of registering producers and consumers makes use of soft-state registration; it is periodically refreshed in order to provide a heartbeat for the registry. If a heartbeat stops, the registrations are removed from the registry.

The GMA does not define any protocols nor the underlying data model and therefore the R-GMA is a relational implementation of the GMA which leverages the power and flexibility of the relational model. R-GMA creates the impression that there is one Relational Database Management System (RDBMS), however it is important to understand that the system is a way of using the relational model and not a general distributed RDBMS with guaranteed ACID[3] properties. Producers announce their existence to the

[3]Atomicity, Consistency, Isolation and Durability are a set of properties that guarantee that database

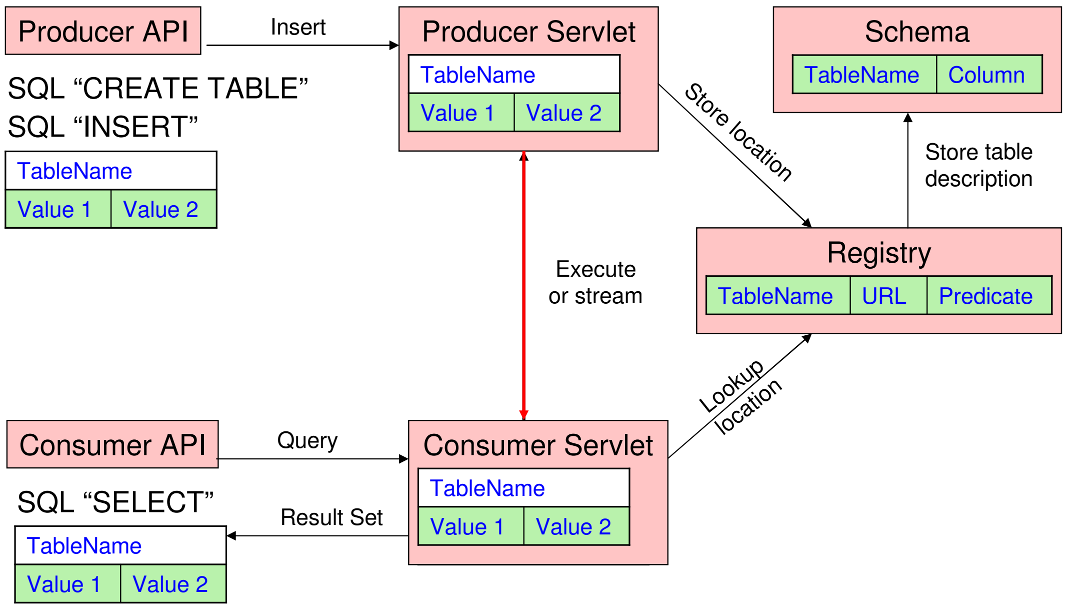

Figure 2.10: The Relational Grid Monitoring Architecture

registry with an SQL create table statement. They may in addition provide an optional where clause expressing a predicate which is true for all information that they produce. Producers then publish information with an SQL insert statement. Consumers use an SQL select statement to obtain the information they require. The predicate enables the registry to disregard any producers for which the SQL select statement from the consumer is invalid and hence reduces the number of producers which the consumer has to contact.

The R-GMA defined four different query types; continuous, history, latest and on-demand. Continuous queries are subscriptions to producers so that all new tuples that match the query are streamed to the consumer automatically. Latest queries are evaluated against the current information in the producer and the the latest tuple is streamed back to the consumer. History queries are evaluated against the current information in the producer and all tuples for the time period specified are streamed back the consumer. The purpose of on-demand producers is to make external information sources that are accessible through the R-GMA infrastructure. The query is evaluated against the external information source and the result is streamed back to the consumer.

Producers can be registered as primary or secondary producers. A primary producer inserts tuples into a storage maintained internally by the producer and autonomously

transactions are processed reliably [76].

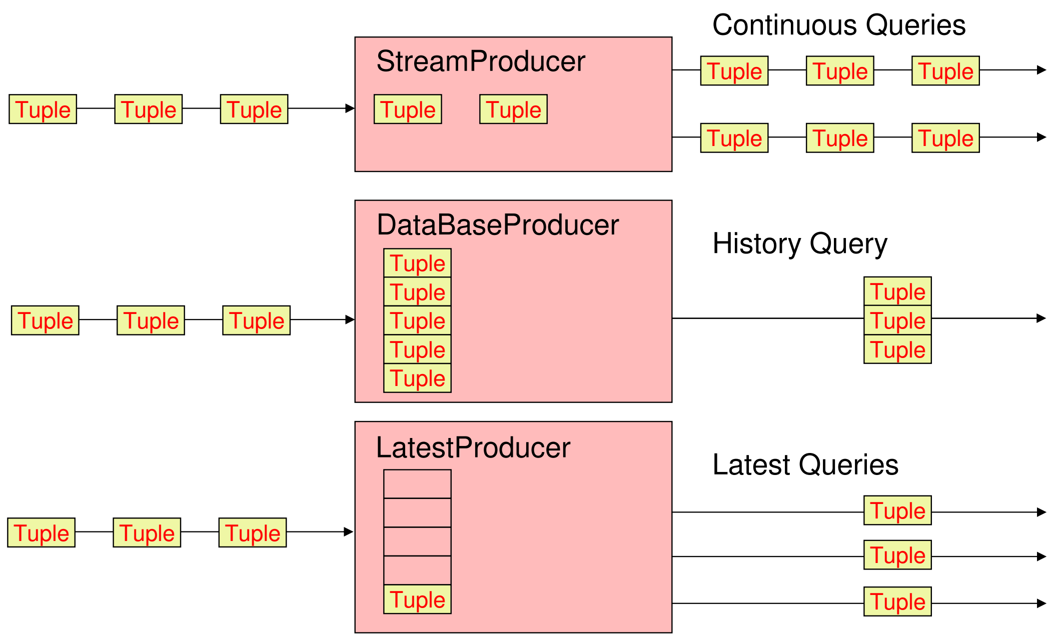

Figure 2.11: Query types used in the RGMA

answers consumer queries from this storage. A secondary producer also answers queries from its internal storage, but it populates this storage by running its own query against the R-GMA and the tuples typically originate from other primary producers. In essence secondary producers republished data from primary producers. This mechanism may be required to support different query types. For example, an archiver may be used to historically store tuples from primary producers and make this information available.

Although the GMA architecture was defined for monitoring, the R-GMA uses it as a basis for a combined information and monitoring system. The argument for this is that the only thing which characterises monitoring information is a timestamp. Within R-GMA the provision of timestamps is enforced so that all measurements have a timestamp for when that measurement was made or for when the statement represented by the tuple was true.

Within a Grid infrastructure one central registry is required to manage the registration of all the producers and consumers and can be seen as analogous to the catalogue. Each site provides an instance of the producer and consumer R-GMA servlets. APIs are used to publish information about a Grid service and this information is stored in the producer servlet for the site. Consumers can use the lightweight APIs to query for Grid services and their properties, however this requires an instance of a consumer servlet to be available. In order to improve efficiency of certain queries, archivers (secondary producers) can be deployed to aggregate information and provide a full result set to the consumer.

The query processing model used depends on how R-GMA is deployed and used. For example, a latest query to a primary producer would be an example of query shipping, however a similar query to a secondary producer that used a continuous query would be an example of data shipping.

R-GMA was evaluated [77] for the monitoring of data transfer rates and job state changes in the WLCG infrastructure. The tests compared the information extracted from R-GMA with the log files which were the actual information source. These tests revealed problems [78] with R-GMA which were investigated and fixed, resulting in improved reliability of the system. However, even after this period of production hardening, R-GMA would not reliably transport information and suffered from periods of downtime due to problems with the registry, which was still a single point of failure. In addition the consumer and producer servlets could not be load-balanced which resulted in a scalability limitation. The problems mentioned above contributed to the conclusion that the R-GMA implementation was not ready for mission critical applications on the WLCG infrastructure and hence the system was decommissioned in March 2011.

2.4.3 Monitoring and Discovery Service

The Monitoring and Discovery Service (MDS 4) [79] from the Globus project provides Web Service (WS) query, subscription and notification interfaces to XML-based information about Grid resources, and is characterized by its extensive use of the WS-Resource Framework and WS-Notification specifications. It differentiates between discovery and monitoring which are manifested as the Index service and a Trigger service respectively.

The Index service collects information about Grid resources and provides a query and subscription interface in the form of the WS-ResourceProperties. It represents a service registry that is analogous to the catalogue in the query processing architecture and similar to a UDDI registry, but in addition it maintains a cached copy of the resource properties using lifetime management mechanisms. Each entry has a lifetime and will be removed from the Index service if it is not refreshed before the lifetime expires. In more complex scenarios, the Index services can register to each other in a hierarchical fashion and thus aggregate information at multiple levels. The Trigger service collects information about Grid resources, compares it against a set of preconfigured conditions and when a condition is met an action is invoked.

The Index and Trigger services are both built upon the Aggregator Framework which collects information from an aggregate source and sends that information to an aggregate sink for processing. Aggregate sources distributed with the Globus Toolkit include modules that query the Grid resource for information, acquire information via subscription/notifications, and execute programs to generate additional information. Aggregator sinks include modules that implement the Index service interface and the Trigger service interface. Information providers publish information to an aggregator service using WS-ResourceProperty or WS-Notification mechanisms and it is the Trigger service that usually runs them to obtain information about the Grid resource.

MDS 4 was used as the production information system for the TeraGrid infrastructure [11]. TeraGrid was an open scientific discovery infrastructure combining resources at eleven partner sites in the U.S. to create an integrated, persistent computational resource. Within Europe, MDS 4 was also deployed as the production information system for the Distributed European Infrastructure for Supercomputer Applications (DEISA) [12], a consortium which brought together eleven national supercomputing centres to provide a distributed supercomputing platform operating in multi-cluster mode, similar to the original I-Way Metacomputing experiment. However, a number of simplifications of MDS 4 were made. Only the WS-RF query interface was used and only deployed on the resources in order to obtain information about them. As there were only eleven resources in DEISA, an indexing service was not necessary as the endpoints can easily be communicated manually via configuration or documentation.

Due to the verbosity of the XML data model, the performance of the MDS4 implementation is significantly lower than those based on either the LDIF and Relational data models. When comparing identical scenarios, [80], the average query response time for a relational database was 2-10 ms compared to 1-2 s for an XML database. While it has been demonstrated that MDS4 scales to eleven sites, scalability issues have been [81] found with larger infrastructures.

2.5 Concluding Remarks

Similar to metacomputing, Grid computing aimed to make using many computing resources spread across multiple organisations as easy as using a single computing resource. Whereas for metacomputing the focus was supercomputers, Grid computing extended this concept to include any networked computing resource, with a specific focus on computing clusters and data storage systems. While technological aspects evolved, such as the change from a protocol-based architecture to service-orientated architecture, the fundamental goals remain. The Grid is the infrastructure encompassing of the software, hardware, policies and operations that aims to seamlessly deliver geographically dispersed computing resources that span multiple organisations as a unified resource. The core aspect is the Grid security policy that facilitates single sign-on and interoperability with local policies by establishing a single framework of trust. A distinctive characteristic of Grid computing is the virtual organisation that is used to support multi-institutional scientific collaboration. Although the High Energy Physics community successfully built the WLCG infrastructure for the simulation, storage, processing and analysis of the data generated by the LHC, this was a heroic effort that took over ten years to realise. Furthermore, the original vision to seamlessly deliver coordinated resource sharing across multi-institutional virtual organisations has not been achieved as Grid computing remains too complex for the end-user. For WLCG, the experiment frameworks shield the individual user from this complexity and hence transparent access has been achieved. If the expectations for seamless Grid computing have not been realised, this suggests that the expansion of Grid infrastructures towards multi-science applications occurred too soon and at the wrong level. The functionality provided by the experiment frameworks needs to be understood, and generic solutions provided for use by the other communities if they are to avoid duplicating the effort required to provide these frameworks and to ensure smooth operations. One point to note is that the information from Grid services is not the only information required by the experiment frameworks and hence the virtual organisations have created their own information systems [82] to augment the information from services with their own topologies, naming and semantics. However, these are not specifically Grid information systems as described in this thesis; they act as configuration databases for the virtual organisation and aggregate information from multiple sources. Although the advent of cloud computing is sometimes seen as a successor to Grid computing, using multiple as-a-service providers presents a similar challenge that will benefit from the experiences of Grid computing.

To support coordinated resource-sharing and problem-solving, the virtual organisation needs a mechanism to obtain information about the structure and state of those resources. Resolving queries over distributed information sources is a form of data retrieval, more specifically the domain of distributed query processing. Grid infrastructures are composed of thousands of Grid services (information sources) that are widely distributed geographically and represent an extremely distributed query processing problem, i.e. each tuple (object instance) is found at a different location. In terms of query optimisation, the domain of distributed query processing provides the concept of query shipping (moving the query to the data) and data shipping (moving the data to the query). In a typical query processing architecture a catalogue is used to provide information about how the database has been partitioned, and physical information such as the location of copies of tables. In the Grid environment, the catalogue is the service registry or index that provides the list of services along with their location in the form of a URL. How this information is propagated and managed is a question of policy and it is therefore possible to provide an abstraction of the different approaches using the concept of a policy engine. The prerequisite for the processing of queries in a Grid environment is therefore the existence of a catalogue or implementation of a policy engine, that can be used to discover all the information sources in a Grid infrastructure. Even even if such a catalogue does not exist, the existence of services needs to be communicated if they are to be used and this information therefore defines the Grid computing infrastructure itself.

A number of Grid information systems have been developed that process queries in a Grid environment in order to discover the structure and state of the participating services. The most influential has been the MDS from the Globus project that provided an implementation based on query shipping and caching. Although there is some diversity with respect to Grid information system implementations, as their main purpose is to resolve queries, they can be modelled by an interface to which a query can be sent and a query result is returned.

The information model is the fundamental building block of not only the Grid information system but of the Grid infrastructure itself. It describes the real entities and the relationships between those entities in a Grid infrastructure along with their semantics, and the various interoperation initiatives have highlighted the importance of a common information model. The evolution of the Grid information model shows how the central view of Grid computing has changed from computers (Host in the MDS), through systems (the CE and SE in GLUE 1.1), to services (GLUE 2.0). This reflects the general trend of Grid computing evolving from a protocol-based architecture in the late 1990s to the Service-Orientated Architecture that exists today.

Chapter 3

The Nature of Grid Information

In one of the early papers on Grid information systems [83], information was described as being relatively static or more dynamic. The concept of static and dynamic information has since been widely used by the Grid community even though a detailed definition does not exist and its use may be misleading. We assert that all information is dynamic and it is the expected frequency of change that defines its relative dynamic nature. The aim of this chapter is to investigate this dynamic nature of Grid information in order to further understanding by measuring the expected frequency of change for Grid information. These measurements are used to measure the uncertainty of Grid information by means of a decay model, which will be a foundation for the subsequent chapters in this thesis.

Specifically, this chapter makes the following contributions:

- A method to measure the expected frequency of change of Grid information.
- The measurements for a widely-deployed Grid information model.
- A model for measuring uncertainty in Grid information.

The chapter is structured as follows; Section 3.1 describes a method that can be used to measure the dynamic nature of Grid information; Section 3.2 applies this method to obtain measurements for a widely-adopted Grid information model; Section 3.3 introduces a model for measuring uncertainty in Grid information and by using the measurements obtained in Section 3.2, demonstrates its application. The work in this chapter was published in the proceedings of the 11th IEEE/ACM International Conference on Grid Computing [18].

3.1 A method to obtain and analyse Grid information

3.1.1 Changes

Grid information is described according to an information model, which is expressed in terms of entities and relationships where the entities represent the real components of a Grid infrastructure. An instance of an entity (an object) is defined by the set of values for those attributes of which the object is composed. One attribute, or a small subset, is used to uniquely identify that instance. Relationships can either be defined as an attribute, for example a foreign key, or by an object if additional attributes are associated with that relationship. Just as real components change over time, the set of values used to describe that component will also change. We assert that an attribute's value v has changed if at time $t+\Delta t$, it is no longer equal to its value at time t. We can express this using Equation 3.1.

$$v_t \neq v_{t+\Delta t} \tag{3.1}$$

Alternatively, a value v has changed if there is a difference between the values during a time interval Δt. We can express this using Equation 3.2.

$$\frac{\Delta v}{\Delta t} \neq 0 \tag{3.2}$$

Similarly, as an object O is a collection of attributes, we assert that the object has changed if any of the values for the attributes from which that object is composed no longer meets the condition as defined in Equation 3.3

$$\frac{\Delta v}{\Delta t} = 0 \tag{3.3}$$

If Δt is large, it is possible that a value may change multiple times with the final value being equal to the initial value. This possibility can be reduced if Δt is also reduced, the limit being when Δt is infinitesimal, δt, and can be expressed using Equation 3.4

$$\frac{\delta v}{\delta t} \neq 0 \tag{3.4}$$

By observing a time-series of Grid information, either continuously or discretely, changes can be identified where Equation 3.4 defines the condition of change for the continuous case and Equation 3.2 defines the condition of change for the discrete case.

3.1.2 How Information Changes

The change of an attribute's value is only one way in which information may differ. A taxonomy of changes can be used to describe the different ways information may change. An instance of an object, such as a description of a Grid service, can undergo three transitions. A new Grid service joining the infrastructure would result in an object being added; the decommissioning of a Grid service would result in the object being deleted; and, during the lifetime of a Grid service, the object may be modified to reflect changes to the Grid service. Objects that are modified can be changed in three ways. A new attribute can be added to the object, an attribute can be deleted from the object, or an attribute's value can be modified. One additional transition that may need to be considered is the re-addition of an object. This can occur when a Grid information system implementation experiences failures, such as broken network connections or power outages at computing centres. Another reason for such behaviour is during a planned (or unplanned) intervention to the Grid service or information source. In such a scenario, the Grid service still exists but may not be usable. Whether the object should be removed from the information system or its status should be updated is a policy decision or an implementation detail. However, it is important to monitor such transitions in order to avoid double-counting of the added and deleted transitions. By considering all these possible changes, a taxonomy of changes has been defined and is shown in Figure 3.1.

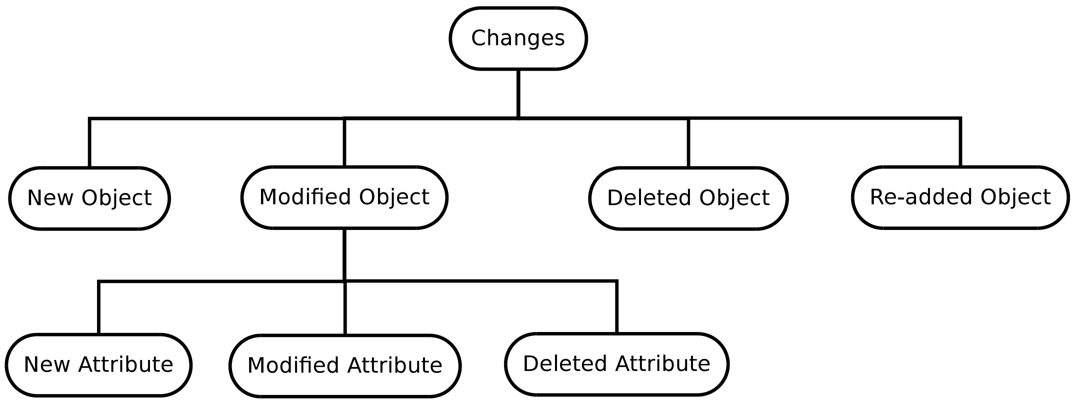

Figure 3.1: A taxonomy of possible changes for Grid information

3.1.3 When Information Changes

Identifying that a single change has occurred does not provide an indication of how dynamic the information is, only that it has changed i.e. not static. Multiple changes need to be observed and a metric is required for these changes so that comparisons can be made. Two metrics that are commonly used [84] are *lifetime* and *frequency*.

Lifetime is defined as the time t between changes and the mean lifetime τ, as described by Equation 3.5, is used as a metric for a number of changes n.

$$\tau = \frac{1}{n}\sum_{i=0}^{n} t_i \tag{3.5}$$

Frequency f is defined as the number of changes n per time interval Δt as described by Equation 3.6.

$$f = \frac{n}{\Delta t} \tag{3.6}$$

The mean (expected) frequency $\overline{f}$ can be calculated for N periods of Δt as described by Equation 3.7.

$$\overline{f} = \frac{1}{N}\sum_{i=0}^{N} \frac{n_i}{\Delta t_i} \tag{3.7}$$

The mean lifetime can be used in the continuous case where it is possible to calculate the lifetime of a value, i.e. when exact time at which each change occurred is known. For the discrete case, unless the time resolution is sufficiently small so that the exact time for each change can be known, the frequency can be used. In either case, the metrics can be used to provide an absolute measure for the dynamic nature of the information. If the mean lifetime is infinity or the mean frequency is zero, the information can be described as static. These conditions are defined by Equations 3.8 and 3.9 respectively.

$$\tau = \infty \tag{3.8}$$

$$\overline{f} = 0 \tag{3.9}$$

The relationship between the mean lifetime τ and the expected frequency $\overline{f}$ is given by Equation 3.10.

$$\tau = \frac{N}{\overline{f}} \tag{3.10}$$

3.1.4 Sampling Methods

For the discrete case, if it is not possible to obtain an accurate timestamp for when a change occurs, a sampling method can be employed. With such a method, the current values for attributes are compared with a previous state in order to detect and count the number of changes that have occurred during that period. A caveat with this approach is that the change observed is only the most recent change that took place and will lead to the under-counting of changes. One method to minimize the under-counting is to reduce the sampling period, however, there may be a technical limit which imposes a minimal period. If this period is too large, multiple changes may be missed and the number of changes observed will be less than expected. However by using a stochastic model of the system, it is possible to estimate this under-count using standard probability theory and can then be compared with observations.

Consider a collection of n values of the same attribute type where the probability p that a value will change in a given time period is constant. The number of values which changed after that time period is given by Equation 3.11.

$$c_1 = np \tag{3.11}$$

The probability that a single value is changed twice in that time period is given by Equation 3.12.

$$p_2 = p \times p = p^2 \tag{3.12}$$

Likewise for three changes, Equation 3.13

$$p_3 = p \times p \times p = p^3 \tag{3.13}$$

This can be generalised for k changes using Equation 3.14.

$$p_k = p^k \tag{3.14}$$

The total number of changes, c_∞, will be given by the number of values that have changed plus the number that experience two changes, three changes etc. This is described in Equation 3.15.

$$c_\infty = np + np^2 + np^3 + \ldots + np^k + \ldots \tag{3.15}$$

Equation 3.15 is an example of a geometric series. If $0 \leq p \leq 1$ this series is convergent, the final value can be calculated using Equation 3.16.

$$\sum_{i=k}^{\infty} np^i = \frac{np^k}{1-p} \tag{3.16}$$

The total number of changes can then be calculated using Equation 3.17, i.e. where $k = 1$ in Equation 3.16.

$$c_\infty = \frac{np}{1-p} \tag{3.17}$$

The under-count can then be calculated using Equation 3.18, i.e. where $k = 2$ in Equation 3.16.

$$\sum_{i=2}^{\infty} np^i = \frac{np^2}{1-p} \tag{3.18}$$

The value of p can be found by observing the number of changes c_1 within a given period for a collection of n values and calculated using Equation 3.19, which is a rearrangement of Equation 3.11.

$$p = \frac{c_1}{n} \tag{3.19}$$

3.1.5 The Method

In order to measure the dynamic nature of Grid information, it must first be understood which approach can be used. If it is possible to obtain a timestamp for when a change occurs, the mean lifetime metric can be used. If this is not possible, a sampling method can be used to measure the expected frequency. When using such a sampling method, the possible under-counting of the result must be taken into consideration. For each type of change outlined in the taxonomy, the chosen metric can be calculated. This can be expanded to include each object type and each attribute type. The metrics obtained will provide a measure for the dynamic nature of that information.

3.2 Measuring the dynamics of Grid information

To measure the dynamic nature of Grid information, the changes from a production Grid infrastructure need to be observed over a long period of time. For this purpose, real information was used from the WLCG infrastructure, which has adopted the GLUE 1.3 information model [85]. As a reminder of Section 2.2.5, this model is composed of fifteen object types that are split into three main groups: core (structural) objects (e.g. Site and Service); compute service-related objects (e.g., Cluster); and storage service-related objects. In addition, two relationship objects also exist that describe the relationship between the compute service and the storage service. A list of these objects, listed by group, is given in Table 3.1.

Object Type	Description
Site	The organisation that manages the Grid service
Service	An abstract, logical view of a Grid service
ServiceData	Key/value pair extension for the service object
Cluster	A computing cluster managed by a single batch system
SubCluster	A homogeneous set of machines within the cluster
CE	A single queue within the batch system with associated policies and state
VOView	The policies and state of a queue from the VO's perspective
Location	The name, version and path of installed software
SE	A file-based storage system
SA	A logical storage area within the storage system
VOInfo	VO-specific information for a storage area.
SEAcessProtocol	A file transfer protocol supported by the storage system
SEControProtocol	A control protocol supported by the storage system
CESEBind	An association object to describe the cluster to storage system relationship
CESEBindGroup	A grouping object for all storage system relationships for a cluster

Table 3.1: Description of the fifteen object types in the GLUE 1.3 information model

Information from the WLCG information system is archived by the Grid Observatory project [86]. The Grid Observatory project collects data on the behaviour of the WLCG infrastructure and gives access to a database of Grid usage traces to the wider computer science community. As the WLCG information system does not archive its information, the project defined an architecture for its digital curation. As a previous study [73] showed that more than 97% of the changes are confined to 14 attributes, the volume of the logging process is reduced by only archiving a complete reference snapshot each day and the differences with this original file are recorded every 15 minutes. The Grid Observatory provides a portal which can be used to request the required traces and URL is returned to where this information can be downloaded. On investigating these traces, it was found that only 59 attributes out of the 173 published were archived by the Grid Observatory. This was due to the fact that the initial aim of the Grid Observatory was to understand job behaviour and hence only attributes relating to job scheduling were obtained. As such, an alternative implementation was developed to record daily snapshots for the full set of GLUE 1.3 attributes published in the WLCG information system.

Daily snapshots of the information system were recorded for the month of June 2011 by using a daily cron job that executed a query which requested all objects and wrote the result to an LDIF file. During this period, there was an average of 169,005 entries (i.e. unique object instances) in each snapshot, which represented an average data size of about 121MB in LDIF [70]. The composition of this information per object type is shown in Figure 3.2. The average number of objects and average size for each object type ($\pm\sigma$) are provided in Table 3.2.

Object Type	Objects	Size (Bytes)	Adds	Deletes	Modifications
Site	368.5 ± 2.5	894 ± 129	0.45 ± 0.72	0.48 ± 0.72	1 ± 1.1
Service	4150 ± 33.5	1500 ± 1080	10 ± 12.9	12 ± 11.9	1004 ± 135
ServiceDataKey	15099 ± 107	439 ± 57	44 ± 45	51 ± 46	71 ± 92
Cluster	667 ± 5	1610 ± 1369	1.2 ± 1.3	1.1 ± 1.4	3.2 ± 2.9
SubCluster	755± 7	15574 ± 16077	1.8 ± 2.2	1.8 ± 2.2	213± 84
CE	5081 ± 57	1995 ± 326	11.9 ± 16.6	12.2 ± 16.6	3966± 134
VOView	11081± 118	854 ± 94	42.1 ± 65.6	42.2 ± 65.6	7942 ± 260
Location	104188 ± 3592	610 ± 57.5	713.8 ± 1052	534 ± 1067	459 ± 80
SE	499± 4	668 ± 66	0.76 ± 0.97	0.83 ± 0.97	211 ± 15
SA	2957 ± 51	1158 ± 491	6 ± 10.3	6.5 ± 10.4	1021 ± 86
AccessProtocol	2334 ± 21	636 ± 100	3.6 ± 6.6	4.3 ± 6.6	0.41 ± 0.89
ControlProtocol	728 ± 6.7	550 ± 70	1.3 ± 2.0	1.6 ± 2.0	0.069± 0.25
VOInfo	6571 ± 153	597 ± 75	21.7 ± 44	21 ± 44	0.10 ± 0.30
CESEBindGroup	5205 ± 69	436 ± 80	9.9 ± 14.3	12.4 ± 14.5	2.2 ± 4.1
CESEBind	9323 ± 411	479 ± 33	15.8 ± 20.8	49.9 ± 39.9	0.28 ± 1.46

Table 3.2: The average number of changes per day by object type, see Table 3.1 for object descriptions

Figure 3.2 shows that most of the information, 45.3% by number of entries and 35.1% by data size, describes installed software (Location), which within WLCG is under the control of the virtual organisation. The second largest, 9.5% by number of entries and 10.7% by data size, is the VOView, which describes the state of a queue in a batch system from the virtual organisation's perspective. The information describing Grid services, 3.8% by number of entries and 6.9% by data size, and sites, 0.35% by number of entries and 0.39% by data size (included under 'Other' in the figure), is in comparison relatively small.

The daily snapshots were compared to find differences and hence calculate the number of changes that had occurred. These changes were grouped according to each

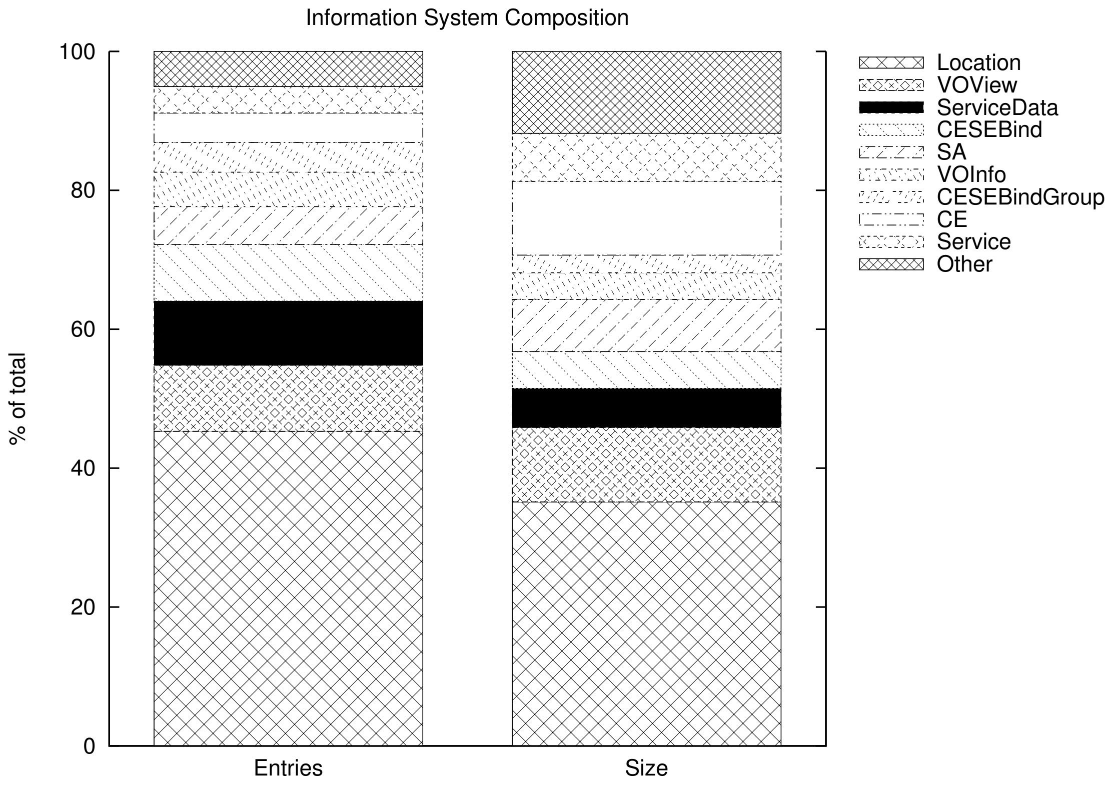

Figure 3.2: The composition of information (for different object types) in the WLCG information system by number of entries and data size

major type of change (added, deleted, modified), as outlined in the taxonomy in Figure 3.1, and object type. The average number of objects changed per day for each object type and each of the three types of change were calculated and the result is provided provided in Table 3.2. The number of objects added and deleted per day as a percentage of the total number of objects for each type is shown in Figure 3.3.

For all object types, this percentage is less than 1% and there are no big variations for different object types. In fact, the slightly higher values for some object types are also due to the fact that there exists a one-to-many multiplicity between objects in the information model. For example, if a VOView object is added for each virtual organisation that is configured to use a specific Cluster, the addition of one new Cluster object will also result in the addition of multiple VOView objects. Overall, the small percentages suggest that there is a more than a 99% chance that an object will not be added or deleted within a day, which, in turn, is an indication of a lifetime significantly longer than the sampling period. However, the situation is different when we consider

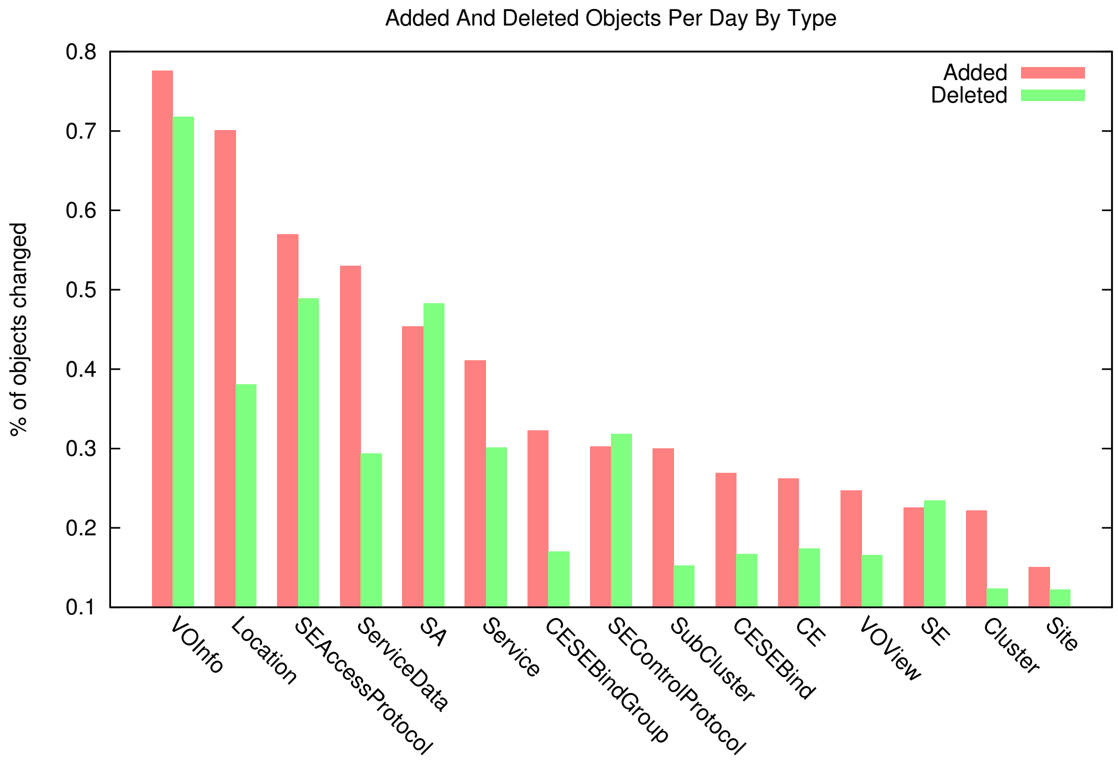

Figure 3.3: Percentage of added and deleted objects per day by object type.

the number of modified objects per day as a percentage of the total number of objects for each object type. This is shown in Figure 3.4 for different object types (x-axis) sorted in descending order of the percentage value.

The graph clearly suggests that different object types can be grouped into two categories; those that experience on average less than 1% changes per day (which we can call *low frequency object types*) and those that experience over 1% changes per day (*high frequency object types*). A line separating the two categories can be drawn between the Location object type, where 0.66% of objects were changed, and the SubCluster object type, where 19.7% of objects were changed (note that these percentages can easily be derived from the values in Table 3.2). This separation suggests a possible measurable way to differentiate between what has been termed as *static* and *dynamic* information in the past [83], as noted in the introduction to this chapter, even though we would rather avoid the use of the word static, as it may imply that the information does not change, which is not true.

Regarding the category of the so-called high frequency object types, the rather high percentage of objects changed, which at one extreme approaches 82.7% for the CE

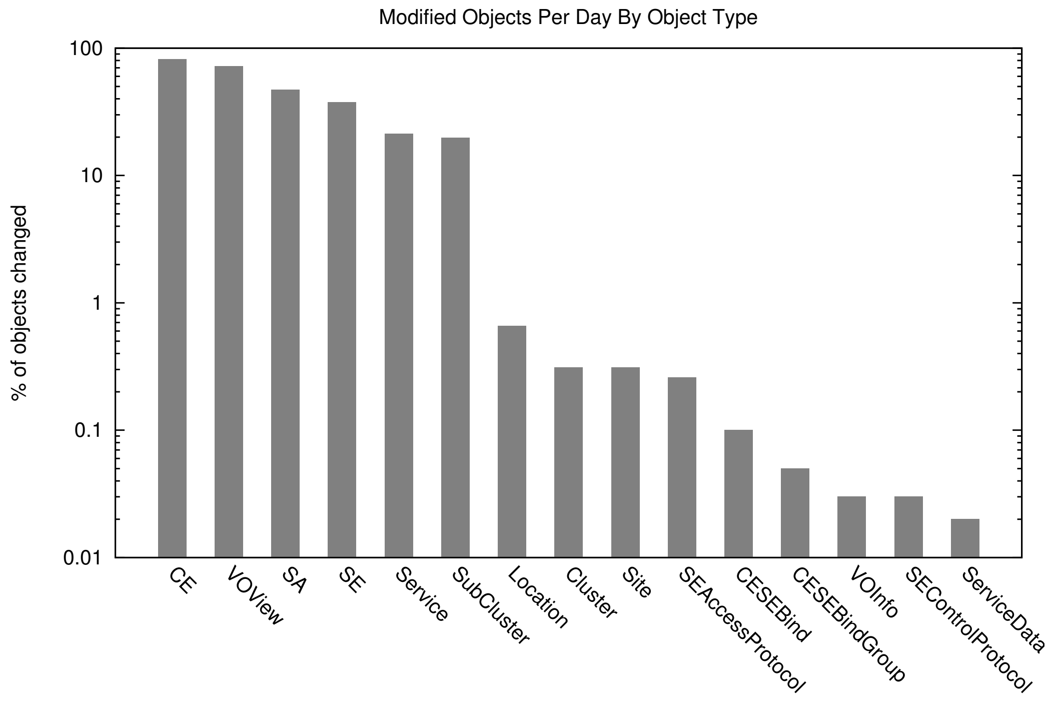

Figure 3.4: Percentage of modified objects per day by object type

object type, suggests that the expected lifetime for objects of this type may be smaller than the sampling period. Therefore, it would be prudent to calculate the under-count to check whether, for some object types and types of change, there are indications that the sampling period of a day may be too high and a smaller sampling period may be necessary to obtain more precise numbers. The resulting under-count for each change type and object type is given in Table 3.3. The under-count as a percentage of the number of changes observed is also provided.

It can be seen that for the addition and delete change types, the under-count is less than 1%. This suggests that the values obtained are reasonably representative as the under-count of a similar order to the variance. However looking at the modifications, the percentage undercount for the Service, SubCluster, CE, VOView, SE and SA, is significant with 31.9%, 356.1%, 253%, 73.1% and 52.7% respectively. This suggests that for these object types, a shorter sampling rate is required, as expected.

As mentioned in Chapter 2, the WLCG information system has a hierarchical structure with three levels. The fundamental building block used in this hierarchy is the BDII, which can be visualized as an LDAP database that is periodically refreshed. The first level of the hierarchy is the resource-level which is co-located with the Grid

	Addition		Deletes		Modification	
Object Type	Undercount	(%)	Undercount	(%)	Undercount	(%)
Site	0.00055017	0.12	0.00062602	0.13	0.00312	0.29
Service	0.02552964	0.25	0.03302443	0.28	320.618	31.9
ServiceDataKey	0.12801393	0.29	0.17028107	0.34	0.332	0.47
Cluster	0.00219871	0.18	0.00194462	0.17	0.0158	0.49
SubCluster	0.00425178	0.24	0.00439567	0.24	83.836	39.3
CE	0.02774915	0.23	0.02922052	0.24	14126.826	356
VOView	0.16056533	0.38	0.16163705	0.38	20100.17	253
Location	4.92390184	0.69	2.74990463	0.51	2.034	0.44
SE	0.00115872	0.15	0.00137554	0.17	154.03	73.1
SA	0.01232339	0.20	0.01454302	0.22	538.23	52.7
SEAccessProtocol	0.00549985	0.15	0.00782652	0.183	0.000072	0.018
SEControlProtocol	0.00261942	0.19	0.00343464	0.22	0.0000065	0.01
VOInfo	0.07183830	0.33	0.06688443	0.32	0.0000015	0.0015
CESEBindGroup	0.01897923	0.19	0.02951388	0.24	0.00096	0.04
CESEBind	0.02678819	0.17	0.26841093	0.54	0.0000084	0.003

Table 3.3: Calculating the undercount of changes for daily snapshots

service and provides information about that Grid service. The second level is the site-level which aggregates the information from all the resource-level BDIIs running at a particular site. The third level is the top-level where all the information from all the site-level BDIIs is aggregated and hence contains information about all Grid services in the infrastructure. As the top-level BDII contains the global view of the infrastructure, it can be used to provide a snapshot of the information in the information system. However, as the top-level BDII is updated every five minutes, this is the shortest possible sampling rate with this approach. By querying the site-level BDIIs directly, it is possible to create a snapshot every minute. It is not possible to have a shorter sampling rate by directly querying the resource-level BDIIs, as many of the Grid services are behind firewalls.

Table 3.4 shows the under-count for one-minute snapshots that were recorded for a 24-hour period on July 21, 2011.

The shorter sampling period has reduced the under-count, however, with the exception of the SubCluster, this under-count is still significant. As it is not possible to reduce the sampling period further using the existing technique, the calculated value for c_∞ is the best approximation that can be obtained. This result suggests that the 15-minute differences archived by the Grid Observatory [86] have limited use. The daily snapshots are sufficient for observing the low frequency changes, however for

Object Type	n	c	p	c_∞	Under-count	(%)
Service	4150	345.11	0.0832	376.4	31.30	9.07
SubCluster	755	1.656	0.00219	1.66	0.00364	0.22
CE	5081	1033.08	0.2033	1296.75	263.666	25.52
VOView	11081	1871.69	0.169	2252.10	380.414	20.32
SE	499	50.83	0.102	56.60	5.762	11.34
SA	2957	135.99	0.0460	142.55	6.557	4.82

Table 3.4: Calculating the under-count of changes for one-minute snapshots

the high frequency object types, the 15-minute sampling rate is not sufficient to accurately measure the changes. Even with a one-minute sampling period, there are still many modifications for the high frequency object types, shown as a percentage of the total number of objects for each object type in Figure 3.5.

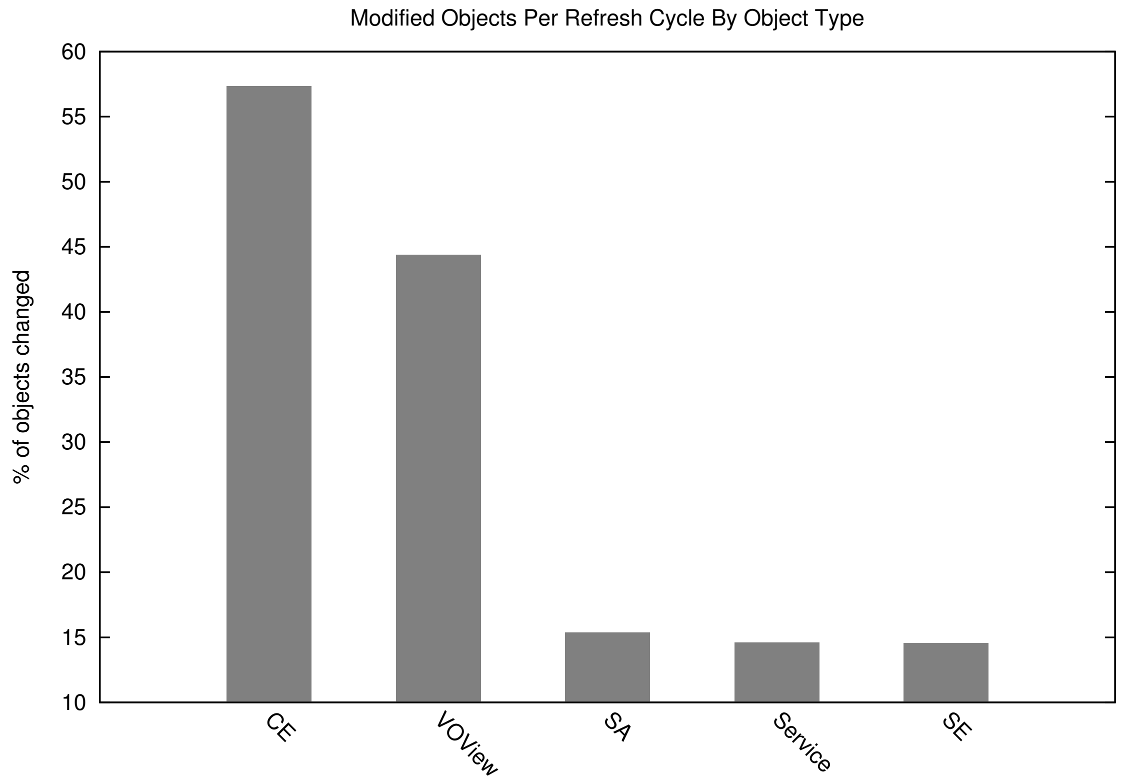

Figure 3.5: The average number of modified objects every minute by object type

Using data on the number of changes per object type, the expected frequency of change can be calculated using Equation 3.7. The resulting values per change type and object type is shown in Table 3.5.

Object	Add (s^{-1})	Delete (s^{-1})	Modify (s^{-1})
Site	5.20×10^{-4}	5.56×10^{-6}	1.24×10^{-5}
Service	1.19×10^{-4}	1.35×10^{-4}	6.27
ServiceDataKey	5.08×10^{-4}	5.86×10^{-4}	8.17×10^{-4}
Cluster	1.40×10^{-5}	1.31×10^{-5}	3.75×10^{-5}
SubCluster	2.07×10^{-5}	2.1×10^{-5}	2.7×10^{-2}
CE	1.37×10^{-4}	1.41×10^{-4}	21.61
VOView	4.87×10^{-4}	4.89×10^{-4}	37.54
Location	8.26×10^{-3}	6.18×10^{-3}	5.32×10^{-3}
SE	8.79×10^{-6}	9.58×10^{-6}	0.94
SA	6.98×10^{-5}	7.58×10^{-5}	2.38
SEAccessProtocol	4.14×10^{-5}	4.94×10^{-5}	4.75×10^{-6}
SEControlProtocol	1.6×10^{-5}	1.83×10^{-5}	7.99×10^{-7}
VOInfo	2.51×10^{-4}	2.42×10^{-4}	1.16×10^{-6}
CESEBindGroup	1.15×10^{-4}	1.43×10^{-4}	2.59×10^{-5}
CESEBind	1.83×10^{-4}	5.77×10^{-4}	3.24×10^{-6}

Table 3.5: The frequencies of change for the object types in the GLUE 1.3 information model

As objects are a collections of attributes, the snapshots can be analysed in more detail by calculating the number of changes for individual attributes. The daily snapshots for the month of June 2011 were compared to find differences and hence calculate the number of changes that had occurred per attribute type. These changes were grouped according to each major type of change (added, deleted, modified), as outlined in the taxonomy in Figure 3.1, and attribute type. This investigation showed that attributes are very rarely (i.e. one per month) added or removed from objects. These types of change only occur where there is an evolution of the information model or an improvement in the implementation. The average number of attributes changed per day varied considerably depending on the attribute. As for objects, attributes can be grouped into low frequency attributes, i.e. those that experience on average less than 1% changes per day, and high frequency attributes, i.e. those that experience over 1% changes per day. A list of the high frequency attributes together with the average percentage which change per day is provided in Table 3.6.

It can be seen that the majority of high frequency attributes are related to state information such as the number of jobs or storage size. Although using daily snapshots will under-count the number of changes, this method is sufficient for grouping the attributes.

Attribute	Object	Percent Changed
FreeCPUs	GlueCE	61.34%
FreeJobSlots	GlueVOView	53.21%
FreeJobSlots	GlueCE	40.82%
UsedOnlineSize	GlueSE	40.30%
SizeFree	GlueSE	40.24%
EstimatedResponseTime	GlueCE	34.86%
TotalJobs	GlueCE	32.13%
WorstResponseTime	GlueCE	30.89%
AvailableSpace	GlueSA	28.73%
UsedSpace	GlueSA	28.13%
ApplicationSoftware	GlueSubCluster	26.90%
EstimatedResponseTime	GlueVOView	26.88%
RunningJobs	GlueCE	26.65%
TotalCPUs	GlueCE	25.02%
WaitingJobs	GlueCE	24.98%
WorstResponseTime	GlueVOView	23.65%
FreeOnlineSize	GlueSA	22.80%
StartTime	GlueService	22.69%
UsedOnlineSize	GlueSA	22.53%
AssignedJobSlots	GlueCE	19.95%
Totaljobs	GlueVOView	18.33%
WaitingJobs	GlueVOView	14.75%
RunningJobs	GlueVOView	14.38%
MaxRunningJobs	GlueCE	8.88%
StatusInfo	GlueService	8.02%
SizeTotal	GlueSE	5.07%
TotalOnlineSize	GlueSE	4.63%
Capability	GlueSA	2.69%
TotalOnlineSize	GlueSA	2.29%
UsedNearlineSize	GlueSE	1.44%
ProcessorOtherDescription	GlueSubCluster	1.2%
BenchmarkSI00	GlueSubCluster	1.18%
Status	GlueCE	1.00%

Table 3.6: The percentage changes per day by attribute type

This investigation has provided evidence to support a number of remarks with respect to Grid information from the WLCG infrastructure. These remarks can be summarised as follows:

i. information describing the main entities in a Grid infrastructure, the site and service objects represents a small proportion, 4.17%, of the total information;

ii. the number of adds and deletes per day is quite low in comparison to the modifications, even for the Location object (remember that most of the information in the snapshots relates to this);

iii. it is misleading to characterize some Grid information as static, as all objects types do experience change. However, the rates of change vary for each object type;

iv. looking in more detail into high frequency object types, their actual frequency of change may be so high that a 1-minute sampling period may not be small enough to estimate it with good precision;

v. within these highly dynamic objects, the attributes that mostly contribute to the dynamic nature of the object are related to state information, such as the number of jobs or storage sizes.

Finally, it should be stressed that as this investigation relied on data from a Grid infrastructure following a specific information model, therefore specifying particular types of objects, these results would not be directly comparable with results obtained from platforms where a different information model was used.

3.3 Modelling Grid information decay

Related work [84] investigating the dynamic nature of Web pages has introduced the concept of *(α,β)-currency* to measure how up-to-date a Web search engine is. Roughly speaking, this provides a probability, α, that a search engine is *current* (*up-to-date* or *fresh*) relative to a grace period, β, for a given Web page. Estimating this probability for different types of Grid information could be used to indicate freshness and, hence, provide a quality metric for Grid information systems.

The cached data for a given Web page that is used by a search engine is said to be β-current if the Web page has not changed between the last time it was indexed and β time-units ago. A search engine for a collection of pages is then said to be (α,β)-current if a randomly-chosen page in the collection has a search engine entry that is β-current with a probability at least α. To quote an example from [84], a daily newspaper is (0.90, 1 day)-current when it is printed, meaning that the newspaper has at least 0.9 probability of containing one-day current information on topics of interest to its readers. By considering the analogy between a Web page and a Grid service (i.e. they are both information sources), and a Web search engine and a Grid information system (they are both snapshots; the former of the Web, the latter of the Grid), it should be possible to apply the concept of (α,β)-currency onto the Grid to describe how up-to-date or fresh the information is for a given object or attribute of a Grid information service. Applying the concept of (α,β)-currency requires an approach to calculate the probability α, that what is contained in the snapshot for some information source is current, relative to a grace period β. In [84], the changing nature of Web pages is modelled using the same ideas underpinning reliability theory [87] in industrial engineering. When a component in a system fails, it is replaced and the period between when it was replaced and when it failed is defined as the lifetime. For Web pages, lifetime, defined as the period between changes of a Web page, can be easily found by using the LAST-MODIFIED attribute in the HTTP header. However, as no such attribute is available in Grid information models or Grid information system implementations, this approach cannot be used. Instead, lifetimes are inferred, and essentially rate of change, by observing the number of changes for a specific object over a period of time. Then, by feeding this information into an exponential decay model, as described in [87], the probability α, that some information (i.e. a specific object type) obtained from the Grid information system is current with respect to the information source, as a function of a grace period β can be approximated. The model assumes that changes are events controlled by an underlying Poisson process, where the probability of observing a change for any given time period does not depend on previous changes or the specific time period, only the length of the time period. More specifically, each value is considered to be a random variable where the probability distribution that the value has changed is stationary and has an exponential distribution. Given a time period t and a change rate λ, the probability that a change is observed is given by Equation 3.20.

$$P(\text{change observed}|\lambda,t) = 1 - e^{-\lambda t}. \tag{3.20}$$

Equation 3.20 expresses the probability that a change has occurred. Therefore, the probability that a change has not occurred is given by Equation 3.21.

$$P(\neg(\text{change observed})|\lambda,t) = e^{-\lambda t}. \tag{3.21}$$

In order to apply the same technique to Grid information systems, it must be understood if they can be modelled using the same process.

Consider object x at the information source and its image x' in the Grid information system. When the information is obtained from the information source at time $t = 0$, it is certain that image x' is identical to the original object x. Assuming that all information will change if the observation period is sufficiently large, we can also assert that image x' is not identical to the original object x when $t = \infty$. During the period $0 < t < \infty$ the probability P that image x' is identical to the original object x is described by the function $f(t)$. These boundary conditions can be expressed as follows:

$$P(x' = x, t) = \begin{cases} t = 0 & P = 1 \\ 0 < t < \infty & P = f(t) \\ t = \infty & P = 0 \end{cases} \tag{3.22}$$

Assuming that each object x is of the same type, the probability that the object will experience a certain change type in time period Δt is identical for all objects and can be described using Equation 3.23.

$$P(x \to \neg x', \Delta t) = constant \tag{3.23}$$

As the probability of change is constant for all objects of the same type, the change events can be modelled by a Poisson process. The probability of an object changing in time period $\triangle t$ can be found by dividing the total number of changes n during that period by the total number of objects N, as shown in Equation 3.24.

$$P(x \to \neg x', \Delta t) = \frac{n}{N} = \lambda \tag{3.24}$$

Combining Equation 3.6 with Equation 3.24 we obtain Equation 3.25, assuming that the rate refers to the same period of time used to calculate frequency.

$$\lambda = \frac{f}{N} \tag{3.25}$$

Using Equation 3.21 and the value for λ given by Equation 3.25, it is possible to find the probability α that Grid information is current, for a specific grace period β for a randomly selected object by knowing the expected frequency of change. The value of β is equivalent to t in Equation 3.21 and the probability α is equivalent to $P(\neg(\text{change observed})|\lambda, t)$. Hence, the fundamental equation underpinning our model is Equation 3.26.

$$\alpha = e^{-\lambda\beta}. \tag{3.26}$$

The values for frequency of change can be used to obtain a value for λ using Equation 3.25. Then, the value of λ allows us to use Equation 3.26, which models the probability α that a randomly selected object of a specific object type is current after a grace period β.

To support the validity of the decay curve produced by Equation 3.26, we considered three different cases for further study which can be regarded as representative. The rationale for choosing these was as follows. First, with respect to changes due to additions and deletions, as we noted from Figure 3.3, the percentage of changes is similar for all object types. As such, the Service object was chosen as representative of all object types with additions/deletions. For changes due to modifications, we chose the Site object as representative of a low frequency object as it does not contain any high frequency attributes and the VOView object as representative of a high frequency object at it contains the attributes with the highest frequency of change. The values of f and λ for these object types and change types were calculated and the results are shown in Table 3.7.

Object	Objects	Changes	f (sec^{-1})	λ
Service	3851	11.6 / day	1.34×10^{-4}	3.5×10^{-8}
Site	351	1.1 / day	1.27×10^{-5}	3.6×10^{-8}
VOView	9736	4320 / 5 mins	14.4	1.48×10^{-3}

Table 3.7: The value of λ for the object types and change types described

The decay curve produced by Equation 3.26 models the change in probability α that a randomly selected object of a specific object type is current with respect to the grace period β. To compare this with the data obtained from the snapshots, we need to measure the decay of a snapshot over time by comparing subsequent snapshots to the first snapshot. Then the probability that a randomly selected object of a specific object type is current at a given time t, that is at time t it has not changed with respect to the first snapshot, is given by Equation 3.27. In this equation, n is the number of objects of this type that are still current at time t and N is the total number of objects of this type in the original snapshot.

$$P(x' = x, t) = 1 - \frac{n}{N}, \tag{3.27}$$

For the Service object type, a comparison of the probability computed from the snapshots (raw data) with the outcome of the decay model produced by Equation 3.26 using $\lambda = 3.5 \times 10^{-8}$ (see Table 3.7) is shown in Figure 3.6. We can see that the values predicted by the decay curve, shown in Figure 3.6, do indeed correlate with the observed decay pattern. In fact, the standard deviation of the difference between the two is 0.0145.

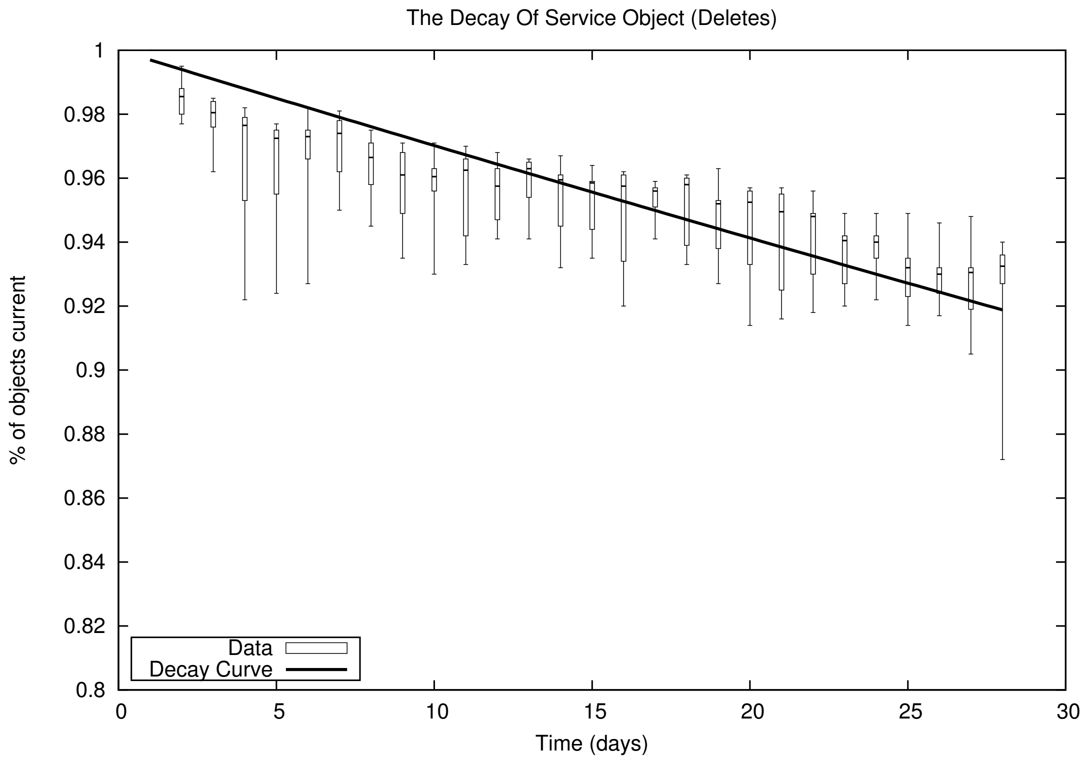

Figure 3.6: The decay curve for deleted Service objects along with data showing the actual decay of a snapshot

One interesting point to note is that the data shows that the probability of a randomly selected object of a specific object type being current in some instances increases. This is due to services dropping out of the information system and then re-appearing due to a scheduled or unscheduled outage. However, as this has been taken into consideration when measuring the frequency of change, it does not affect the overall correlation.

For the Site object type, the unique identifiers found in the first snapshot were stored in a list along with the object itself. These unique identifiers were used to find the objects in the subsequent snapshots. These were compared to the original object and the number of objects that were still identical was recorded along with the time since the snapshot was first made. The probability that a randomly selected object of a specific object type is current was calculated using Equation 3.27 and the result can be seen in Figure 3.7, along with the decay curve given by Equation 3.21 and the value of λ.

The same comparison is made for the Site object type and the VOView object

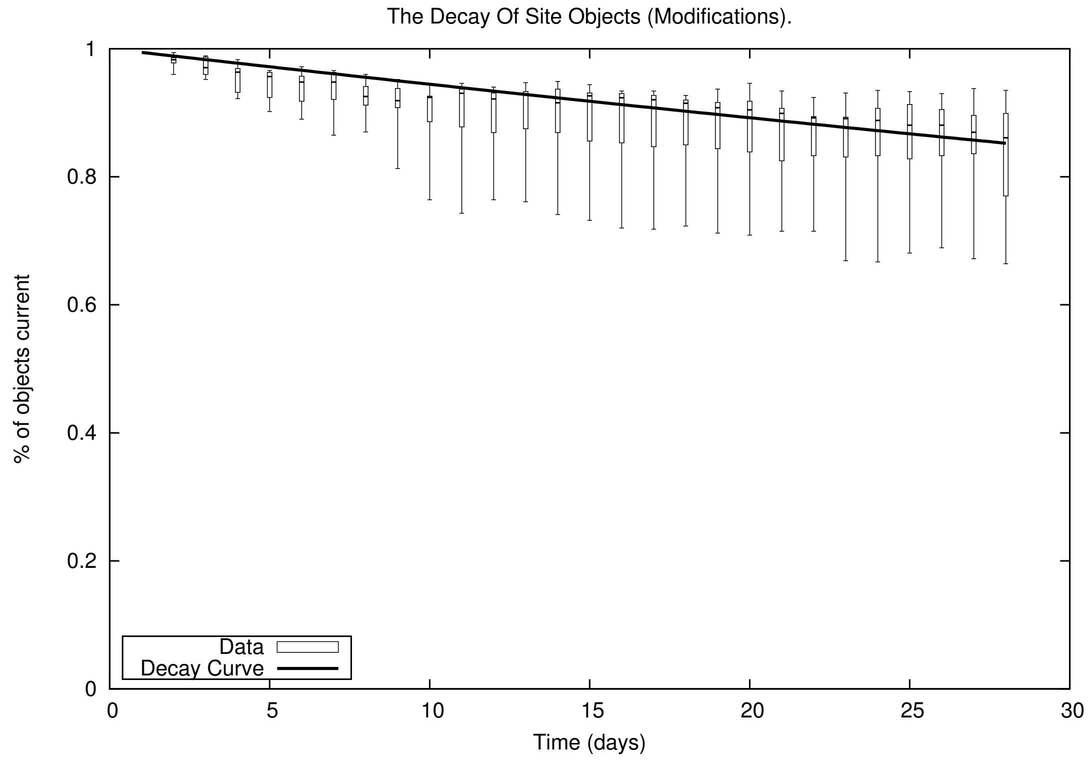

Figure 3.7: The decay curve for modified Site objects along with data showing the actual decay of a snapshot

type in Figures 3.7 and 3.8, respectively. Again, the values predicted by the decay curve correlate with the observed decay pattern from the snapshots. The standard deviation of the difference between the two is 0.00113 for Site and 0.0145 for VOView. It should be noted that the values for VOView decay very fast (as remarked previously in Section 3.2, a large number of changes was observed for this object type) and thus, the daily snapshots can not be used. To compare the decay curve with more raw data, additional snapshots using a sampling period of 30 seconds were obtained by querying the resource-level BDIIs directly.

Again, Figure 3.7 indicates that the values predicted by the decay curve do indeed correlate with the observed decay pattern. The same instability effect can be seen, however a systematic error is present as the objects could have been added or deleted. During this period the information system contained an average of 351.9 Site objects with a standard deviation of 2.16. In essence the object experiences decay by two processes. This could be taken into consideration by adding the decay constants, however, in this case one decay process significantly dominates.

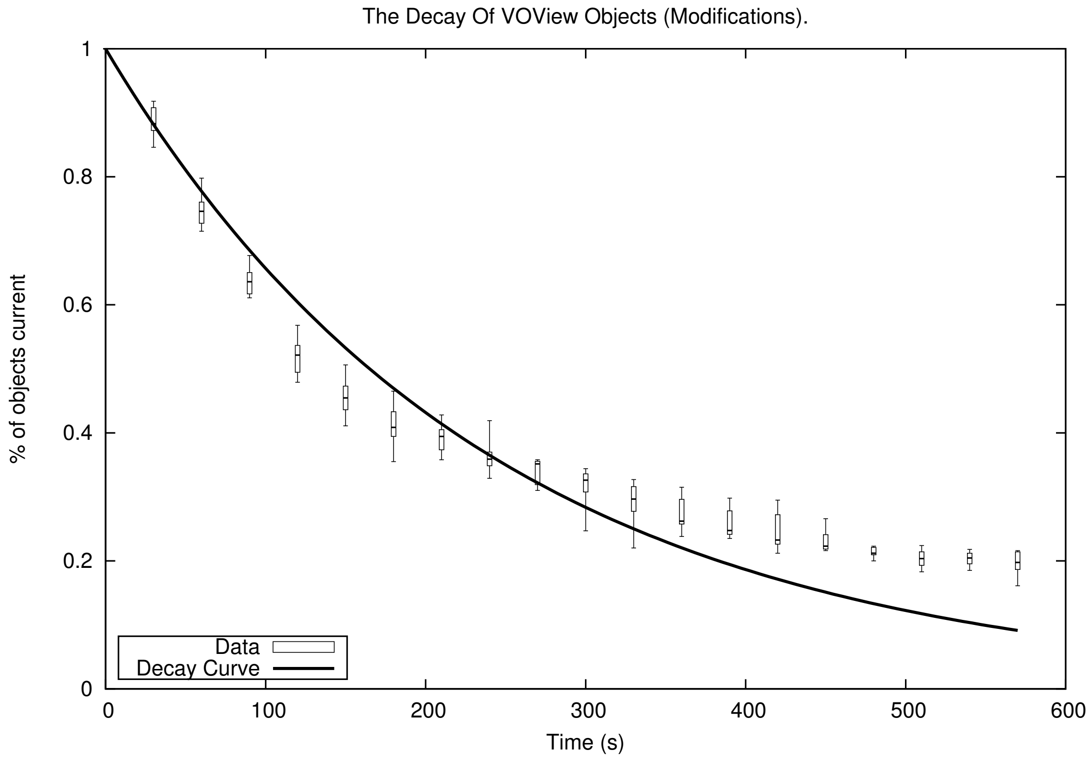

Figure 3.8: The decay curve for modified VOView objects along with data showing the actual decay of a snapshot

Measuring the frequency of change for the VOView in the previous section highlighted the limitation of the sampling method. To achieve a sampling period of 30s, the resource-level BDIIs were queried directly for the VOView objects. The refresh mechanism employed in the top-level BDII was reused, however, it was customized to query the resource-level BDIIs rather than the site-level BDIIs and to only obtain the VOView objects in order to reduce the time required for the query. Twenty snapshots were created every 30s, which represents a period of 10 minutes. The VOView objects found in the first snapshot were stored in a list and these were compared to the objects found in the subsequent snapshots. The probability that a randomly selected object of a specific object type is current was calculated using Equation 3.27 and the result can be seen in Figure 3.8, along with the decay curve given by Equation 3.21 and the value of λ for the VOView from Table 3.7.

In Figure 3.8, the systematic error due to added or deleted objects is negligible due to the very short time period however, another systematic error exists. As mentioned in Section 3.2, the VOView object can be active or inactive and hence will experience different rates of decay. This results in an offset error which must be taken into account. The probability that a randomly selected object of a specific object type is current must be calculated by considering only active objects, which for the time period was calculated to be 61% of objects. By considering only the decay of active VOView objects, the values predicted by the decay curve do indeed correlate with the observed decay pattern.

The above suggests that the decay model for a snapshot given by Equation 3.26 is supported by real data from a the WLCG Grid infrastructure. This decay model can therefore be used to describe the (α, β)-currency of Grid information for a specific object type and change type. For example, Grid services are (0.95, 15 day)-current, meaning that a snapshot of Grid services objects has a 0.95 probability of containing current (or fresh) information on Grid services fifteen days after it was taken. However, VOView objects are (0.95, 35 second)-current, meaning that a snapshot of VOView objects has a 0.95 probability of containing 35-second current information on the state of a computing service.

Using this metric, information freshness in a Grid information system, and hence the quality of the system itself, can be assessed. By measuring the age of the information, the time period (β) that has passed since the information was refreshed from the information source, the probability (α) that a specific object type is fresh can be calculated using Equation 3.26. In addition to its value as a quality metric, this probability may also be used to dictate the desired refresh cycle for different types of information.

3.4 Concluding Remarks

This chapter investigated the nature of Grid information. A method to obtain measurements on the expected frequency of change of Grid information was presented. This included a definition for the condition of change along with equations to calculate the mean lifetime and expected frequency, the two metrics for describing the dynamic nature. In addition, the limitation of the sampling method to obtain the expected frequency was discussed and an approach to overcome this limitation was incorporated into the method. The method was then used to calculate the expected frequency of change for information from a production Grid infrastructure.

The (α, β)-currency metric from the domain of Web search was introduced and enhanced appropriately to provide a quality metric for Grid information systems. The values for the expected frequency of change were used to model the decay of Grid information and this was compared to observations of information decay for a production Grid infrastructure such as WLCG.

From these investigations, the following conclusions can be made:

- Information describing the main entities in a Grid infrastructure, the site and service objects, represents a small proportion, 4.17%, of the total information.
- The number of adds and deletes per day is quite low, less than 1%, in comparison to the modifications.
- It is misleading to characterize some Grid information as static, as all objects types do experience change; however, the rates of change vary for each type.
- Looking in more detail into high frequency object types, their actual frequency of change may be so high that a 1-minute sampling period may not be small enough to estimate it with good precision.
- Unsurprisingly, snapshots of such high frequency object types decay quickly.
- Within these highly dynamic objects, the attributes that mostly contribute to the dynamic nature of the object are related to state information.
- Information freshness in a Grid information system can be described using the concept of (α, β)-currency.

Regarding the concept of static and dynamic information, all information has some form of dynamic behaviour, therefore the division into static and dynamic may be misleading. At some point in time an object instance is added to the system and it will eventually be deleted. As for attributes, it was noticed that the composition of objects changed infrequently, with only one or two attributes being added or deleted per day. However when it comes to modifying attributes, some will remain constant throughout the lifetime of the object (e.g. unique identifiers), some will change slowly (e.g. software versions) and some will change quickly (e.g. state attributes). The difference between the low frequency object types and high frequency object types is considerable. Whereas snapshots of low frequency object types will decay within the order of days, snapshots of high frequency object types will decay within the order of seconds. This contrast between object types may have an implication for the design of Grid information systems.

Chapter 4

Benchmarking Grid Information Systems

As large-scale distributed infrastructures can exist for many years, deployment decisions should not be taken lightly as it can be difficult to migrate between different implementations once in operation. A benchmarking method for Grid information systems is an important tool for comparing and evaluating different designs, implementations and deployment scenarios. Such a methodology should evaluate the Grid information system in order to understand its scalability limitations to ensure that they will meet the future requirements in terms of information volume and query load. The view in this thesis is that existing benchmarking methodologies are deficient as they focus mainly on performance metrics, and neglect important aspects such as the nature of the queries as well as the quality of the answers delivered.

Taking this into account, this chapter presents a comprehensive benchmarking methodology for evaluating Grid information systems, which is based on the assessment of three different components:

i. a set of commonly-executed queries;

ii. query response time;

iii. quality of information returned.

To the best of our knowledge, this is the first time that such a comprehensive benchmarking methodology for Grid information systems, which incorporates the use of a quality metric, has been defined. The proposed benchmarking methodology is then used to evaluate two different Grid information system implementations; the MDS

[65] from the Globus project (the pioneering approach) and the BDII [88], which is the alternative implementation used in the WLCG infrastructure.

Specifically, this chapter makes the following contributions:

- A critical evaluation of existing Grid information system benchmarking methods that highlights the absence of a quality metric for Grid information.
- An improved benchmarking method that includes (α, β)-currency as a quality metric and considers the use cases from a production Grid infrastructure.
- An evaluation of two different Grid information system implementations.

This chapter is structured as follows; Section 4.1 provides a critical evaluation of previous, related work; Section 4.2 describes in detail the components of the benchmarking methodology and how the metrics included can be measured. The evaluation of the two Grid information systems using the benchmarking methodology presented is given in Section 4.3, along with some concluding remarks in Section 4.4. The work in this chapter was published in the proceedings of the 17th International Euro-Par Conference [19].

4.1 Related Work

In a frequently-cited study [89, 90, 91], the functional components of three Grid information systems were grouped into similar types and the performance of each component type was evaluated against those of their counterparts. The focus was to evaluate the effect of a large number of users and information sources for each implementation. In addition to the average query response time, a throughput metric (average number of queries per second) was also used. Both of these metrics were measured for different numbers of concurrent queries and information sources. However, in this study little attention was given to the information itself. A single query that returns all information for a specific information source was used and it was assumed that the response was always correct (i.e., up-to-date, or current, or fresh). In addition, the information volume used was small (10Kb) compared with today's infrastructures (100MB).

A study investigating the scalability and performance of a Grid information system [60] addressed some of these deficiencies by using real data from a large-scale Grid infrastructure. It also investigated how the performance is affected by the size of the query response, however only the average query response time was used as a metric and again it was assumed that all responses were correct.

A Grid information benchmark has previously been proposed [26], based on a set of queries and scenarios, which were used to compare the access language and platform capabilities for three different database platforms serving Grid information. Although in total thirteen different queries were used, the rationale for using this set was not made. The databases were populated with synthetic information about Grid services, which conformed to the GLUE 1.1 information model [85]. Short synthetic workloads (scenarios) were used to measure the query response time of concurrent query requests to the repository. The major difference with our work is that we include a quality metric and actual information from a production Grid infrastructure.

Of particular interest is the study in [92] which in addition to the query response time, proposed two further metrics, network overhead and information freshness, for evaluating Grid information systems. Network overhead is defined as the number of bytes that a crawler downloads to update the information in the system. The crawler in this context is a specific implementation detail of this system and although important, it is not a concern for the user from a quality of service perspective. Information freshness, on the other hand, is of great concern to the user. Information from the Grid information system is considered fresh (or correct), at a given point in time, if it is synchronized with its real-world equivalent value at the information source, that is, both values are the same. This concept of freshness is useful as a quality metric for the information returned, and it is missing from other studies. However, the proposed calculation of freshness may not be feasible in a real system as it requires continuous comparison of all values.

All of the above studies demonstrate the need to evaluate Grid information systems. They show that the query response time is a key metric and should be a core part of any proposed benchmarking method. The studies also suggest that the query response time can be affected by four main factors; the total information volume, the type of query (both due to the complexity and size of the response), the number of concurrent queries (query loading) and the implementation (hardware, software and internal architecture). In order to compare different Grid information system implementations, it is therefore necessary to compensate for the other factors.

The information model used is a key part of any such study as it describes the information and the types of queries that can be made. The Grid information benchmark method [26] suggests using the GLUE 1.1 information model, although the reason for this choice was not given. However, a recent survey of production Grid infrastructures [41] revealed that the GLUE 1.3 information model is used by the majority of infrastructures and hence resources. The volume of information used should reflect a typical production Grid infrastructure in order for the results to be relevant to practitioners and as infrastructures grow, this value may need to be revised.

The Grid information benchmark method [26] also suggests measuring the response time for a defined set of queries. Although in total thirteen different queries were used, the specific case for using this set was not made. The study investigating the scalability and performance analysis of an information system [60] sampled the queries actually made to find the type of queries, the frequency of the queries and the size of the query responses. This information could be used to improve the composition of the query set used in the benchmarking method.

Extending the current state-of-the-art further, the benchmarking methodology proposed in this chapter takes into account the above, also including for the first time a metric to measure the quality of the query response.

As stated above, measuring the query response time only is not enough. It is also necessary to describe the quality of the query response itself, and information freshness was introduced as an additional metric above. In the previous chapter we proposed an alternative quality metric, (α, β)-currency, that originally was used to measure how up-to-date a Web search engine is. Roughly speaking, this provides a probability, α, that the information is current (up-to-date or fresh) relative to a grace period, β. This investigation then estimated, by means of an empirical study, this probability for different types of Grid information based on the GLUE 1.3 information model. It then demonstrated the use of (α, β)-currency as a quality metric for Grid information systems. We propose to include this metric in an updated benchmarking methodology, and hence provide a measure for the quality of the information returned.

4.2 Methodology

The benchmarking methodology will be based on the GLUE 1.3 information model [85] as this information model is used by the majority of today's production Grid infrastructures [41]. The information model used is a key part of any benchmarking study as it defines a number of constants on which the benchmarking method is based. These include the object types, and hence expected frequency of change for those object types, the composition of the queries and the size of the query response. If an alternative information model is used, these constants will have to be measured for that information model. Due to the different constants being used, benchmarking results cannot be compared for different information models.

The volume of information and the use cases for the benchmarking methodology will be taken from a large-scale production Grid infrastructure, the WLCG infrastructure [9]. This is a fair choice to use as a reference for the benchmarking methodology as WLCG operates the largest Grid infrastructure in the world and in addition, it also makes use of the GLUE 1.3 information model. As of November 2010, the WLCG information system contained information representing 4102 Grid services deployed between 375 sites. This information represents an information size of 102MB in the LDIF [70] format, which corresponds to an average of 272KB per site or 25KB per service.

Set of queries: In order to consider queries commonly made to the WLCG information system, we used the information provided in [60] to select the top ten queries. The type of each query and its frequency as a percentage of the total number of queries made to the system are shown in Table 4.1.

	Query	%
Q1	Find the Closest Computing Service to a specified Storage Service	19.6
Q2	Find the VO's Storage Area for a Storage Service	17.7
Q3	Find all Storage Services	16.3
Q4	Find a particular Storage Service	15.5
Q5	Find the Closest Storage Service to a specified Computing Service	7.8
Q6	Find all Services for a VO	6.8
Q7	Find all Computing Services for a VO	2.1
Q8	Find all Storage Areas for a VO	2.1
Q9	Find all Sub Clusters	1.5
Q10	Find the VO's Computing Share for a Computing Service	1.4

Table 4.1: The top ten queries made to the WLCG infrastructure as a percentage of the total number of queries

It is interesting to note that none of these queries require the object with the largest contribution to the total information volume, the Location (installed software). The characteristics of each query in terms of the query response size and the object type that they use were further analysed. As different objects experience a different frequency of change, the object type being used by each query may indicate how up-to-date the results of this query are expected to be. These characteristics are shown in Table 4.2.

A few observations are interesting to note. All queries have a similar level of complexity: they either return all the objects of a particular type or filter by only one predicate. As such, the two key differences between the ten queries are the size of the query response and the object type that is queried.

Query	Size(B)	Size	Object	Frequency (s^{-1})
Q1	2858	small	CESEBind	Low
Q2	5436	small	SA	High
Q3	315225	large	SE	High
Q4	642	small	SE	High
Q5	1875	small	CESEBind	Low
Q6	49801	varying	Service	High
Q7	16607	varying	CE	High
Q8	12348	varying	SA	High
Q9	8959015	large	SubCluster	High
Q10	6009	varying	VOView	High

Table 4.2: Characteristics of each of the top ten queries

The size of the query response allows a classification of the queries into three groups: (i) queries that return information about one service (denoted by *small* in the table); (ii) queries that return all information for an object type (denoted by *large* in the table); and (iii) queries that filter information for one object type (denoted by *varying* in the table). For queries of the latter group the actual size of the response varies depending on the filter being used. To ensure that the filter used for a particular query provides a good representation of the query response size for that query, all possible values for that filter were evaluated and the response size in each case was calculated. The median size was then chosen (and is shown in the table) to avoid the average being skewed by the few large query responses. Regarding the object type used by each query, these objects are classified according to the classification in Chapter 3 into high-frequency objects (those that experience over 1% changes per day) and low-frequency objects (those that experience fewer than 1% changes per day).

Query response time: Each query is executed 50 times against the deployed instance of the Grid information system under evaluation. Each time, the query response time (QRT) is measured from the start of the client connection to when all the data for the query response has been received by the client. The average query response time can be calculated using Equation 4.1, where $n = 50$.

$$QRT_{average} = \frac{\sum QRT}{n} \tag{4.1}$$

The throughput, i.e. queries per second, for a number of concurrent queries n, can be calculated from the average query response time using Equation 4.2.

$$throughput = \frac{1}{QRT} \times n \tag{4.2}$$

Quality of information: For each query, the (α, β)-currency will be used to assess the quality of information returned. This metric, first investigated in the context of Grid computing in Chapter 3, can be used to calculate a probability, α, that a selected object's values stored by the Grid information system are current with respect to a grace period β. The probability α can be calculated by using Equation 4.3, where β is the age of the information and λ is a constant for each object type.

$$\alpha = e^{-\lambda\beta} \tag{4.3}$$

The method used to measure β, the age of the information, depends on the specific Grid information system implementation. If a timestamp is available for when the information was created, this can be compared to the time when the query was executed. Alternatively, the time between when the information was created and when the query was executed will need to be measured. The latter will be the method used later in our benchmarking comparison of MDS and BDII. We assume that the information is fresh and the query is executed as soon as the cache has been updated. Hence, we use the time it takes to update the cache as the time between when the information was created and when the query was executed; this provides a value for β.

The constant λ can be calculated using Equation 4.4, where f is the frequency of change for a specific object type and N is the number of objects of that type. Further information regarding the frequencies of change for the GLUE 1.3 object types, as well as the number of objects for each type, can be found in Section 3.2.

$$\lambda = \frac{f}{N} \tag{4.4}$$

4.3 Benchmarking MDS and BDII

This section benchmarks two Grid information system implementations, the MDS and the BDII, using the benchmarking methodology described in the previous section.

4.3.1 Background

The MDS query interface hides the details [83, 65] of the MDS architecture and implementation from the client. This interface is provided in MDS by the GIIS using the GRIP, which is implemented as a special purpose back-end for an OpenLDAP server. The GIIS maintains a dynamic registry for information source locations populated from registration notifications received via the Grid Resource Registration Protocol [65]. The GIIS parses each incoming GRIP request and then forwards this request to one or more information sources found in the registry depending on the type of information requested. To improve performance, each response may be cached for a configurable period of time by specifying the cache time-to-live (TTL) per information source as part of the registration.

The BDII is a simplified implementation of the MDS GIIS. The main simplification is the absence of the Grid Resource Registration Protocol [65] with the dynamic registry functionality being replaced by a static registry where new information source locations are added manually by a system administrator. The other simplification is that the BDII maintains its information cache using a parallel process rather than caching query results. This serves to decouple the cache update process from incoming queries; therefore, the information sources are isolated from the queries and this results in a predictable behaviour of the system.

In both implementations, the information is cached and queries are evaluated using this cache. The implementations diverge in the method employed to refresh that cache. Further discussion on the impact of the cache update method can be found later in Section 4.3.6.

4.3.2 Experiment Set-up

The MDS and BDII software was installed and configured on a physical machine. The machine used had a 1.80GHz Intel Pentium CPU and 1GB of memory. The MDS version used was 2.4, which is based on OpenLDAP 2.0 and the BDII version used was 5.1.9, which is based on OpenLDAP 2.3. A second machine was used to run

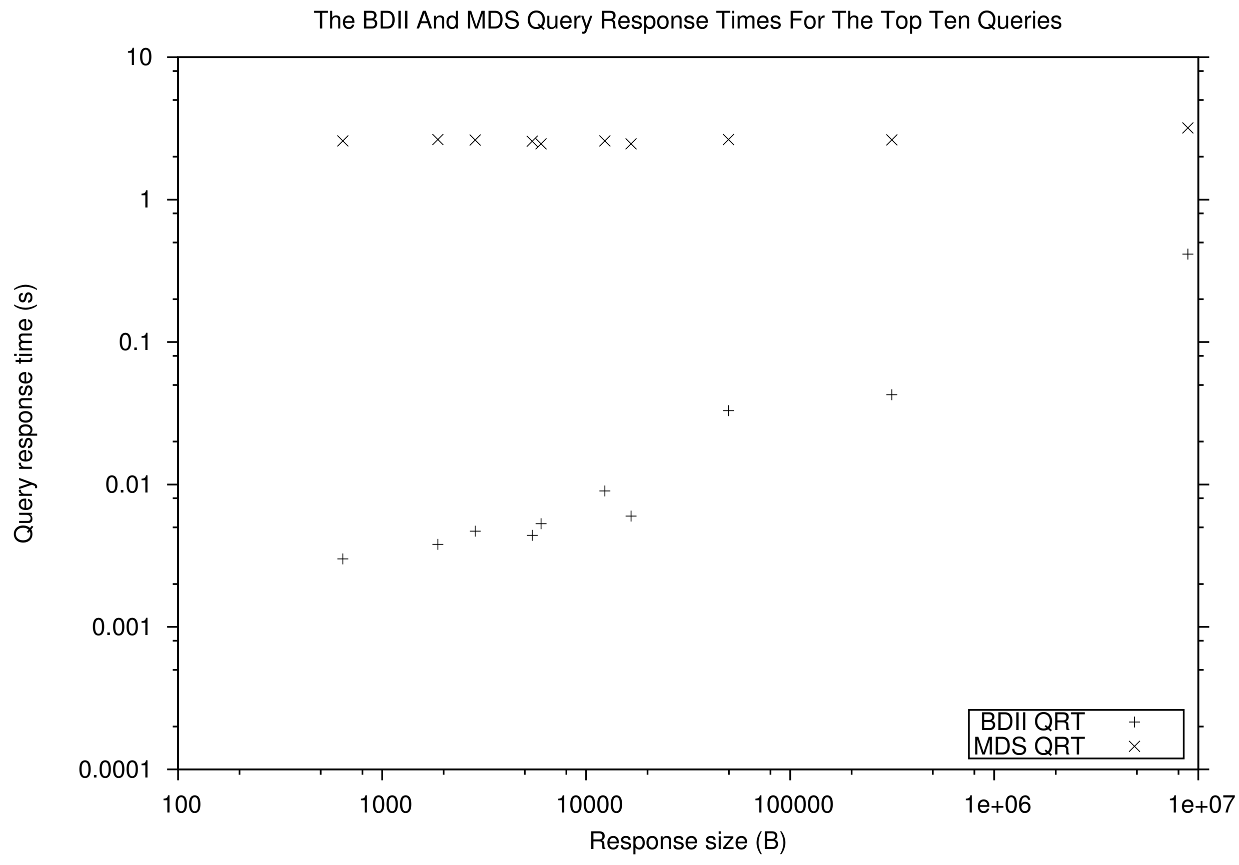

Figure 4.1: The average query response time for the top ten queries made to MDS and BDII (variance suppressed for clarity)

client queries. This machine had four 2.66GHz Intel Xeon CPUs and 8GB of memory. The two machines were connected through a 100 Mbps LAN.

The caches were populated with data representing the information from the WLCG infrastructure, using a snapshot from Section 3.2 and the cache update processes were disabled. While this approach assumes the existence of caching in MDS, it has been adopted so that the two systems are comparable. This assumption does not reflect the full distributed set-up but this will be revised later.

4.3.3 Query Response Time

Each query was executed in a single thread that iterated fifty times over the same query. The average response time for each query type made to the MDS GIIS and the BDII can be seen in Figure 4.1.

These results show that for MDS GIIS the query response time is almost constant, approximately 2.6s, for all query result sizes, with the exception of the 8.9MB result size, which causes the query response time to increase to 3.2s. The increase of 0.6s is comparable with the time taken to transfer the 8.9MB result over the 100Mbps LAN connection, 0.71s. This suggests that the query execution time is the main factor that affects the query response time for the MDS GIIS. The query response size, and hence network transfer cost, only has an impact for large query response sizes. By contrast, the BDII implementation has much lower query response times, which seem to be affected by the query response size. This suggests that the query execution time is much lower for the BDII implementation and that the network transfer cost is the main factor that affects the query response time.

4.3.4 Throughput

The same experiment was repeated for the MDS GIIS using 2, 4, 6 and 8 parallel query threads. The average response times can be seen Figure 4.2.

In order to protect the test framework from hanging queries, a timeout value was used to terminate the query if it exceeded a certain threshold. As the MDS query response time is between 2.6s and 3.2s depending on the response size, the timeout value was set to 20s. The results show that for MDS doubling the number of query threads essentially doubles the query response time. In addition, with 4 query threads, a few queries timed out. With only 6 query threads, up to 35% of the queries timed out and with 8 query threads up to 52% of the queries timed out.

For the BDII 10, 25, 50, 75 and 100 parallel query threads were used, as provisional tests showed that it could handle much more that 8 query threads. The average response times can be seen in Figure 4.3.

The BDII seems to be much more scalable with respect to the number of query threads than the MDS GIIS. Even with 100 query threads, many of the queries return a result in less than 1s. The exception is Q9, which returns 8.9MB. With 50 query threads, the query response time is 19.3s and for 75 and 100 threads, all the queries timed out.

4.3.5 Quality of Information

As already mentioned in Section 4.2, for the value of β we can use the time it takes to update the cache. However, we note that the BDII and the MDS GIIS differ in

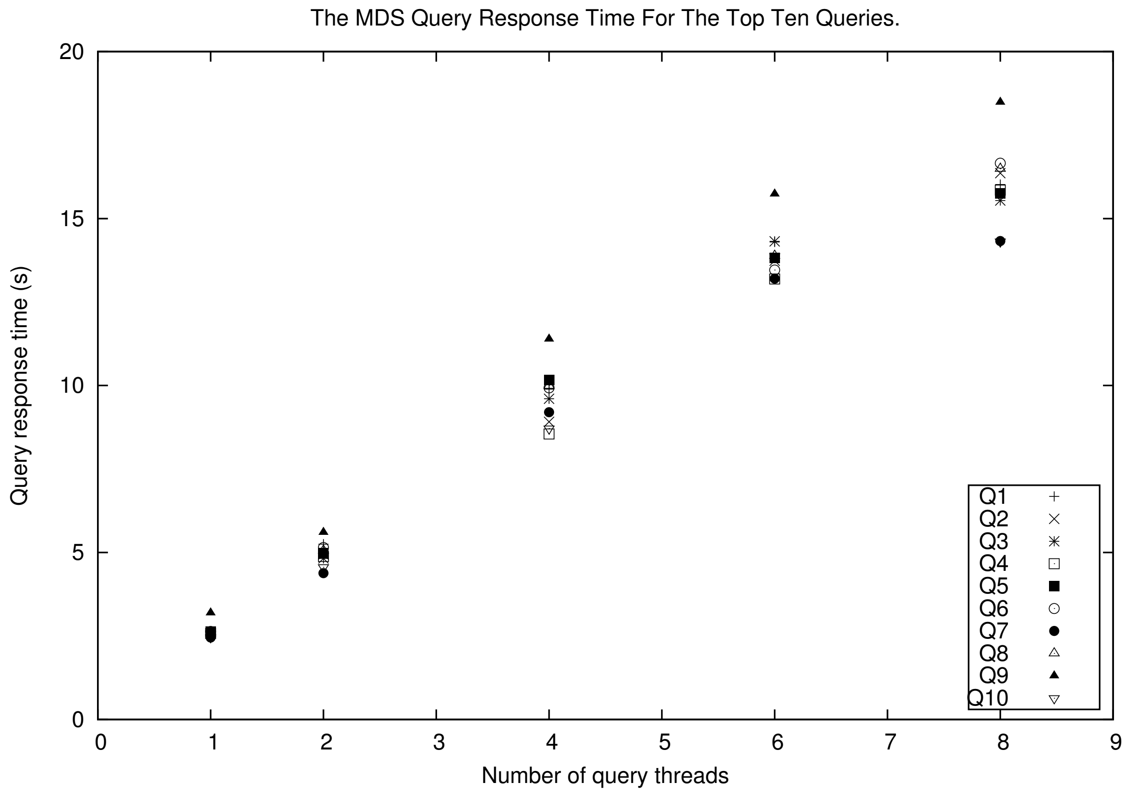

Figure 4.2: The average query response for the top ten queries under different query loads for queries made against the MDS GIIS

the way the cache is updated. In the MDS GIIS, the cache is updated synchronously with the query, whereas with the BDII this update is done asynchronously. This means that for the MDS GIIS, when the cache is stale, the next query will trigger the update process and subsequent queries will block until this process has finished. In the BDII a parallel process updates the cache while the cache is still responding to queries. In both cases, the time to update the cache is the minimal value for β in Equation 4.3, with the maximum value being defined by the TTL of the cache in the MDS GIIS or the delay between update cycles in the BDII. The best-case scenario, which will be measured here, is where these values are zero.

To measure the query response time for the MDS GIIS, the cache TTL was set to zero so that every query made to the cache would trigger the update process. A simple query that returned zero results was used to trigger the update. The query response time for ten successive queries was measured and the average query response time was calculated. The average query response time when the cache is valid was also

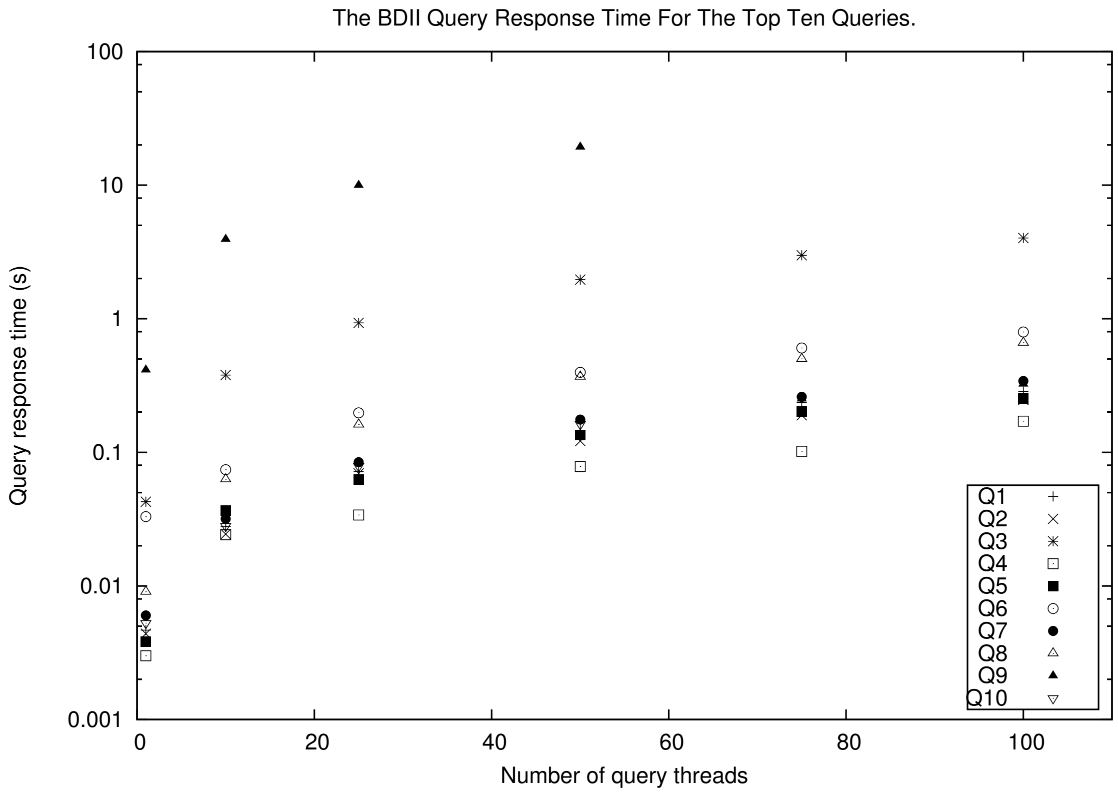

Figure 4.3: Average query response for the top ten queries under different query loads for queries made against the BDII

calculated. The difference between the query response time when the cache is valid and invalid gives the cache update time. The cache update time for the MDS GIIS was 2496s with σ=13.6s.

For the BDII, the update process is instrumented and the update time is recorded in the log file of the update process. Thus, the average time for ten updates was calculated. The cache update time for the BDII was 201s with σ=14.9s.

Using the cache update times for β (that is, β_{MDS}=2496 and β_{BDII}=201) and the frequencies of change from Chapter 3 to calculate λ using Equation 4.4, the value of α was calculated using Equation 4.3 where λ is the corresponding value for the object being queried. The results in Table 4.3 show the probabilities α_{MDS} and α_{BDII} that a random selected value of a particular object type is current for the MDS GIIS and BDII respectively, immediately after the cache has been populated.

Query	Object	$\lambda(s^{-1})$	α_{MDS}	α_{BDII}
Q1	CESEBind	1.14×10^{-08}	1.000	1.000
Q2	SA	5.11×10^{-04}	0.279	0.902
Q3	SE	4.85×10^{-04}	0.298	0.907
Q4	SE	4.85×10^{-04}	0.298	0.907
Q5	CESEBind	1.14×10^{-08}	1.000	1.000
Q6	Service	4.86×10^{-04}	0.298	0.907
Q7	CE	1.91×10^{-03}	0.008	0.681
Q8	SA	5.11×10^{-04}	0.279	0.902
Q9	SubCluster	2.28×10^{-06}	0.994	1.000
Q10	VOView	1.48×10^{-03}	0.025	0.743

Table 4.3: The result of calculating (α, β)-currency for the MDS GIIS and the BDII.

4.3.6 Discussion

As both the BDII and the MDS GIIS are based on OpenLDAP, a comparison of the the query results will mainly evaluate the efficiency of the different versions used and their configuration. When comparing the query response times of the BDII and the MDS GIIS implementations for a single query thread, the result depends on the query used. For Q4, the query with the smallest response size (642 bytes), the query response times are 0.003s and 2.58s for the BDII and the MDS GIIS, respectively. For Q9, the query with the largest response size (8.5MB), the query response times are 0.41s and 3.19s, respectively. The BDII implementation delivers much better performance than the MDS GIIS for queries with a small response size (Q4), however, this advantage is reduced for larger response sizes (Q9).

As stated previously, looking at the query response time is only one aspect of the overall quality of service. The quality of the information returned also needs to be considered. In the case of querying a low frequency object type, for example Q1 (CESEBind), it is almost certain that all information is current. However, for the case of querying a high frequency object type, Q7 (CE), the probability that the information is current is 0.008 for the MDS GIIS, while, for the BDII, the probability that the information is current is 0.681. We note that these figures represent the best-case scenario as they assume that the original information is current. The value for α will degrade further during the period after the caches have been updated. The length of this period can be defined by a configuration parameter in both instances. Knowledge of (α, β)-currency can be used to set an optimal value for this parameter, however, the limit is reached when the value is set to zero.

Comparing the query response times will show which implementation gives a faster response, however it does not indicate whether the query response time is acceptable. In both implementations, the query response time may be acceptable for a particular deployment scenario. Including the (α,β)-currency metric gives additional insight by providing a measurement of the quality of the information returned. What values are acceptable will be dependent on the particular deployment scenario, however, we can make some general statements about the result. Any value less than 1 would result in the client having to include a probabilistic approach when using this information. Any value less than 0.5 means that the information returned is more likely to be incorrect than correct. With this interpretation, the MDS GIIS is returning more incorrect information when querying high frequency object types than correct information, and although the BDII is returning more correct information than incorrect information when querying high frequency object types, a probabilistic approach to using that information would be required.

It should be noted that when the cache in the MDS GIIS is stale, the next query will trigger the cache to be updated synchronously with the query and subsequent queries will be blocked until this update process has finished. The result is that the MDS GIIS is unresponsive for 41 minutes 36 seconds it took to update the cache, which has an adverse affect on the average query response time and results in the MDS being unusable in a production Grid infrastructure.

Unlike the MDS GIIS, the BDII can be queried while the cache is being refreshed asynchronously to query. To see how the query load affects the cache update time and hence β, the query Q3 was used to produce a query load using a single thread. This query load caused the cache update time to increase to 1100s with σ=145 while the average query response time for the query increased from 0.0427s with σ=0.00114 to 0.0472s with σ=0.0339. Although for low frequency object types, it is still almost certain that the information in the cache is current, the value of α for high frequency object types was reduced from 0.907 to 0.58 (for Q3). Also, the values for Q7 and Q10 were reduced from 0.681 and 0.743 to 0.12 and 0.2, respectively. As a result, under a query load the BDII is returning more incorrect information than correct information when querying high frequency object types.

Our experimental results have also indicated that the existing Grid information system used in the WLCG infrastructure is not particularly suited to high frequency object types. It also highlights the challenge of obtaining accurate dynamic information in a globally-distributed infrastructure and hence that there is a limitation for global scheduling which is based on such information. It is for this reason that the experiment frameworks used in WLCG have adopted a late-binding approach [93], where available resources announce themselves when they are available, along with a description of their environment, and has removed the virtual organisations dependence on some of the dynamic attributes.

4.3.7 Changing the Information Model

The results demonstrate the difficulty for Grid information systems to handle highly dynamic information from many different information sources. One approach to improve the value of α for the (α, β)-currency metric would be to remove the dynamic attributes from the object type. An alternative approach for dealing with dynamic information could then be considered which is better suited to this kind of behaviour, or an alternative description of that attribute could be used that is more coarse-grained. For example, rather than using the actual value for the number of running jobs, the used capacity could be specified using the enumeration HIGH, MEDIUM and LOW with reference to the total capacity, which changes much less frequently. The same knowledge of the system would be extracted with the attribute being less dynamic. Such issues should be considered during the design of the information model.

In order to select which attributes to remove, a method is required to identify the highly dynamic attributes. By considering the real system that the GLUE information model describes, we can identify the main actions that contribute to information changes. The first action is either commissioning or decommissioning a new site or service. This action will create or delete new objects in the information system, i.e modify the structure. A service update, such as a configuration change or software upgrade, may also create or delete new objects, however it may also modify an existing object. Finally, while the service is running, objects may be modified to reflect the changing state of the service.

By considering such changes we can classify attributes into three different groups; *static* attributes, *mutable* attributes and *dynamic* attributes. Static attributes are attributes that do not change during the lifetime of the object. Mutable attributes are attributes that may change during the lifetime of the object but at a rate that is comparable to service updates. Dynamic attributes are attributes that change at a rate that is not comparable to the service updates i.e. more frequently.

In order to define the boundaries between these groups it is necessary to measure the average lifetime of a service and the average time between service updates. The relationship between the mean lifetime τ and the constant λ is described in Equation 4.5.

$$\tau = \frac{1}{\lambda} \tag{4.5}$$

In Chapter 3 the value of λ for service deletes (decommissioning) was found to be $3.5 \times 10^{-8}(s^{-1})$. Using Equation 4.5, this equates to a mean lifetime of 330.7 days. In order to calculate the average lifetime between service updates, the attribute Service Start Time from the GlueService object can be used. This approach is similar to the approach taken to measure the age of a Web page by using the last modified timestamp [84]. The assumption made is that a service update results in the service being restarted, which is typically the case. The snapshot from the 1st March 2011 was used to obtain all the Service Start Times and the ages of the services were found. The medium age for the service was found to be 15 days.

We define *static* attributes as those attributes which have a lifetime greater than the mean lifetime of a service (330 days), *mutable* attributes as those attributes that have a mean lifetime greater than the update lifetime (15 days) but less than the service lifetime and *dynamic* attributes as those attributes with a lifetime less than 15 days. The result was that out of 228 attributes, 68 were classed as static, 101 as mutable and 59 as dynamic.

Now that the dynamic attributes have been identified, the consequence of removing these attributes from the object type can be calculated. Taking the VOView as an example, a new value for λ can be calculated. The same data for March 2010 from Chapter 3 was used to recalculate λ without considering the dynamic attributes. The new value of λ is $2.6 \times 10^{-9}(s^{-1})$ and if the period between updates is one hour, the probability that the information is current would be 0.99999. This means that only one value out of 100K values for that attribute would change during the period and hence no longer be identical to the value at the information source.

4.4 Concluding Remarks

This chapter investigated the benchmarking of Grid information systems and proposed a comprehensive benchmarking methodology, suggesting that the addition of (α, β)-currency as a metric for describing the quality of a Grid information system would give additional insight. A Grid information system benchmark methodology that incorporated this metric was defined, along with additional constraints that make the results relevant for today's production Grid infrastructures. Two Grid information system implementations that have had production exposure, the MDS and the BDII, were then benchmarked using this methodology. The results show that in terms of response time, the BDII implementation gives better performance than the MDS GIIS for queries with a small response size, 0.003s and 2.58s for the BDII and the MDS GIIS respectively for Q4 (642 bytes). However, this advantage is reduced for larger response sizes, giving 0.41s and 3.19s for the BDII and the MDS GIIS, respectively, for Q9 (8.5MB). This suggests that although the connection and query execution overhead is less for the BDII, for larger response sizes, the data transfer component provides a significant contribution to the overall query response time.

While this comparison showed that the query response time for the BDII was better (faster) than for the MDS GIIS, a statement could not be made on whether or not this result was acceptable for deployment in a production Grid infrastructure. Including the (α,β)-currency metric gave additional insight by providing a quality measurement for the information returned. For low frequency object types, it is almost certain that the information in the cache is current. However, for the high frequency object type in Q7 (CE), the probability that the information is current is 0.008 and 0.681 for the MDS GIIS and the BDII, respectively. This suggests that for Q7, the MDS GIIS would mainly be returning incorrect values and that a probabilistic approach would be required for the information returned from the BDII.

Looking at the methods used to refresh the cache, the MDS GIIS is unresponsive for 41 minutes 36 seconds, which has an adverse affect on the query response time and results in the MDS being unusable in a production Grid infrastructure. For the BDII, a single threaded query load using Q3 resulted in the cache update time increasing to 1100s with only a small increase in the response time, 0.0427s vs 0.0472s. The longer cache update time for the MDS and hence the higher value for β resulted in the value of α for high frequency object types being reduced from 0.90 to 0.58 and the values for Q7 and Q10 were reduced from 0.68 and 0.74 to 0.12 and 0.2 respectively. This suggests that for these queries the BDII would also mainly be returning incorrect values.

The measurement of (α,β)-currency is highly dependent on the information model. The value of α can therefore be improved by considering changes to the information model which will reduce the dynamic nature of the attributes. Such issues should be of consideration during the design of the information model. If use cases do require the fine-grained dynamic information, it may not be possible to improve the quality by changing the nature of the attribute.

In conclusion, this chapter has presented a comprehensive methodology for benchmarking Grid information systems. This methodology demonstrated that including the concept of (α, β)-currency gives additional insight over using the query response time alone. The investigations revealed that the existing Grid information system used in the WLCG infrastructure is not suited to high frequency object types. It has also highlighted the challenge of obtaining accurate dynamic information in a globally-distributed infrastructure and hence that there is a limitation for global scheduling which is based on such information.

Chapter 5

Processing Queries in a Grid Environment

This chapter re-visits the fundamental concepts relating to the processing of queries in a Grid environment in order to identify the different possible approaches. Each approach is explored in further detail and evaluated to the best possible extent using the benchmarking metrics described in Chapter 4 and the measurements relating to the nature of Grid information from Chapter 3.

Specifically this chapter makes the following contributions:

- The identification of the main approaches for processing queries in a Grid environment.
- An analysis of these approaches with respect to how well they are suited to the Grid environment.
- An evaluation of the main approaches with respect to the Grid information system benchmarking method.
- A statement on the best approach for processing queries in a Grid environment.

The chapter is structured as follows; Section 5.1 revises the fundamental concepts for processing queries in a Grid environment and identifies the main approaches; a more detailed analysis of their suitability is provided in Section 5.2. An evaluation of the different approaches with respect to the Grid information system benchmarking method can be found in Section 5.3. Some concluding remarks are given in Section 5.4 along with a statement on the best approach for processing queries in a Grid environment.

5.1 Background

As stated in the Chapter 2, a fundamental aspect of Grid computing is the need to obtain information about the structure and state of Grid services which are widely distributed geographically. Information describing each Grid service is provided by the service itself and hence the Grid service is the primary information source. The information provided conforms to an information model and it is assumed that the contents of the information source are up-to-date, that is the values represent the real state of the Grid service. Queries need to be resolved that may consider thousands of information sources in order to enable efficient Grid functions that may utilise multiple cooperating services. The goal is to efficiently execute many queries, from many clients, for many information sources in order to minimise the query response time, i.e. a scalability problem.

In the absence of an information system, the burden of information retrieval (i.e. discovering the information sources) and data retrieval (i.e. executing the query) falls on the client initiating the query. An information system aims to remove the burden of information retrieval and data retrieval from the client by providing a central point, an information service, to which a query can be sent and the result is returned. The information system in this context is the complete distributed infrastructure that enables the information service to function. The provision of an information service transfers these burdens from the client to the information service; however, the information sources still need to be discovered and the query still needs to be executed.

To resolve a query over all the information sources (data retrieval), it is first necessary to discover those information sources (information retrieval). In a typical architecture for query processing, information related to the location of information sources is stored in a catalogue. In the Grid environment, this catalogue is the service registry or index that provides the list of services along with their location in the form of a URL, the service record. Other attributes, such as the service type, can be used to avoid contacting services that may be irrelevant for the query, however for the global selection query, only the URL identifying the information source is required. It should be highlighted that information about which services should and should not be in the infrastructure does not originate from the service itself. The authoritative source for information about which services are participating in the infrastructure and to which administrative domain they belong is dependent on the management policies of the Grid infrastructure. The ownership and control of such information is of utmost importance in a global infrastructure and how this information is managed is a question of policy. Therefore the prerequisite of a Grid information system is the existence of a catalogue in the form of a service registry which is a result of those management policies, that can be used to discover all the information sources in a Grid infrastructure. The details of how these policies are used to create the service registry is out-of-scope for this thesis but it is an area that would benefit from further investigation.

In terms of querying the information sources, the question is whether to move the query to the data (*query shipping*) or to move the data to the query (*data shipping*) [94]. Two approaches for data shipping have been introduced [94]; caching and replication. In this context caching is triggered by query execution (i.e. client-initiated [95]), and typically uses a synchronous protocol that is based on invalidation [94]. Replication is server-initiated [95] (i.e. in this context triggered by the information source itself) and aims to maintain the consistency of a *quasi-copy* [96] of the information using an asynchronous protocol, which can occur even if no queries are processed. In general, replication moves the information close to a large group of clients so that they can access it cheaply the first time it is queried. Caching enables a client to access information cheaply when that information is used repeatedly. Put simply, replication is required if there are more queries than updates (expansion test) and should be avoided if there are more updates than queries (contraction test)[67]. For caching, the cache investment needs to be calculated, that is the difference between the benefits gained from caching compared to the cost of creating the cache. Caching in this context typically uses a *fault-in* [94] approach; that is to use the cache if available and only update the cache or store the query result if the cache contents are invalid.

Both caching and replication make it possible for queries to access data cheaply. The first question that needs to be answered is do queries benefit from cheap access to information, i.e. in a Grid environment should a query shipping or data shipping approach be used? The second question is that if a data shipping approach is used, which method for managing the cache will result in the lowest value of β, and hence maximum value of α, for the lowest network transport cost.

The benchmarking method outlined in Chapter 4 can be used to compare these different approaches. The benchmarking method defines two metrics; the query response time and the (α,β)-currency metric. These should be measured for a scenario that resembles a production Grid infrastructure. One of the main features of a Grid computing environment is that the information sources (Grid services) are widely distributed geographically. This suggests that the underlying network which links the Grid services, provides physical limitations for data movement and hence may be a significant component of β. As such we will assert that the approach which requires the least amount of data to be transferred should be the more efficient approach, with the lowest values for β, and hence α, and the query response time.

5.2 The Different Approaches

5.2.1 Query Shipping

The Grid information system environment consists of:

i. many thousands of Grid services as information sources

ii. a single point to which a query is sent (the information service)

iii. many thousands of clients initiating queries.

For a single query of n information sources, the total data returned, v, is given by Equation 5.1 where d is the result size returned by the individual information source.

$$v = \sum_{i=1}^{n} d_i \tag{5.1}$$

If the catalogue for the information sources contains additional metadata, the number of information sources that need to be contacted n, may be reduced if it can be deduced from the metadata that an information source is not relevant for the query. As Grid information may be extremely distributed, i.e. for some queries each tuple (object instance) is found at a different location, the main purpose of the Grid information system is to aggregate information. The result is that the total volume of information transported between the Grid services and the information service is equal to that transported between the information service and the client.

The query response time t for a single query to an information source is given by Equation 5.2, where q is the time to transfer the query to the information source, e is the execution time for the query and r is the time to transfer back the query result.

$$t = q + e + r \tag{5.2}$$

Assuming no resource limitations, Equation 5.3 can be used to calculate the query response time for all information sources if the queries are executed in parallel, where p_c and p_s are the client and server processing times respectively.

$$t = p_c + \max(q_i + e_i + r_i) + p_s \tag{5.3}$$

As a protection from queries that take too long or hang indefinitely, a timeout is usually specified for each query. It was mentioned in Chapter 2 that unavailability is an issue as scale grows. The chance of failure in at least one component increases, which means that at least some of information sources will be unavailable if there are many. If p is the average probability across all information sources that a specific query will timeout within a given time period, the probability that at least one will timeout out of n parallel queries, is given by Equation 5.4.

$$p_n = 1 - (1-p)^n \tag{5.4}$$

As $0 \leq p \leq 1$ and $0 \leq (1-p) \leq 1$, $\therefore$ the limit of $(1-p)^n$ as $n \to \infty$ is 0, hence as the number of parallel queries increases, so does the probability that a specific query will timeout in a given time period. This suggests that for a large number of information sources, and hence parallel queries, the timeout value may significantly contribute to $\max(q_i + e_i + r_i)$.

However, as such scenarios are typically resource-limited, the average query response time $\bar{t_p}$ for the parallel queries can be found empirically and used in Equation 5.5 to give the the query response time.

$$\bar{t} = p_c + \bar{t_p} + p_s \tag{5.5}$$

The value for the query response time t in Equation 5.5 can be used as the grace period β in the (α, β)-currency equation.

As each query contacts all the relevant information sources, each information source is required to handle as many queries that are initiated and the amount of data transferred v is given by Equation 5.1, where d is the result size returned by the information source for a specific query and n is the number of queries initiated.

5.2.2 Data Shipping

In the data shipping scenario the query is resolved directly by the Grid information service and the query response time is defined by Equation 5.2. Again, as Grid information is extremely distributed and the queries are not complex, no more than one predicate and no joins, the query result represents a sub-set of the total information (a partition) and hence can be quickly obtained. As the main role of the query shipping approach is to aggregate information, it follows that the advantage of the data shipping approach with respect to the query response time is comparable to the cost of executing the parallel queries, which is avoided. The data transferred between the information service to the client is the same in both cases and is given by Equation 5.1. Therefore the difference in query response time between the two cases can be expressed using Equation 5.6, where $\bar{t_p}$ is the average query response time for the parallel queries found empirically, p_{qs} is the processing time for the query shipping approach and p_{ds} is the processing time for the data shipping approach. The processing times p_{qs} and p_{ds} can be measured by instrumenting the mechanisms used to execute the queries. p_{qs} is the time from the start of the execution to when the first parallel query is sent and p_{qs} is the time taken from when the last parallel query returns to when the execution terminates.

$$\Delta QRT = \bar{t_p} + p_{qs} - p_{ds} \tag{5.6}$$

This difference describes one benefit of using a data shipping approach over the query shipping approach. However other factors such as the amount of data transferred as well as the measurement of (α, β)-currency will depend on the specific design decisions such as the data format and protocols used.

Caching

Caching makes it possible to cheaply return a query response, reduces the query response time, when the same query is repeatedly instantiated. The concept of cache investment [97] can be used to understand if a cache should be used. This compares the cost (i.e., investment) of creating/using the cache versus the benefits of the cache. Put simply, the first query will experience a cost overhead while all other subsequent queries will benefit. In most cases, if $cost << benefit$ and there are two or more queries i.e. at least one query that will benefit, a cache should be used. The benefit of using a cache is defined in Equation 5.6, that is the cost to execute parallel queries is avoided.

The cost to create the cache will result in the query response time t increasing by t_c, as shown in Equation 5.7.

$$t = p_r + \bar{t_p} + p_o + t_c \tag{5.7}$$

As long as $t_c << \Delta QRT$ and that there will be multiple queries of the same type during the period where the cache is valid, caching should be used.

Caching in this context typically uses a fault-in approach where the information sources are contacted to populate the cache if the information is not already available in the cache. The cache time-to-live, i.e. when to invalidate the cache and fetch fresh data, should be set using the (α, β)-currency equation so that α does not decrease below a desired *quality level*. The cache TTL defines the period over which the cache can be considered valid and hence it is possible to calculate the number of queries that will benefit from the cache using Equation 5.8 where q is the number of queries per unit time and Δ QRT is given by Equation 5.6.

$$benefit = ((q \times TTL) - 1) \times \Delta QRT \tag{5.8}$$

The -1 in Equation 5.8 takes into consideration that when using a fault-in approach the first query to arrive after the TTL has expired will experience an increased query response time due to the cache being invalid. The amount of data transferred for the queries that query the information sources directly when the cache is invalid is given by Equation 5.8, where q is the number of queries per second.

In terms of reduction in the amount of data transferred, the benefit for the amount of data transfer avoided when the cache is used is given by Equation 5.9 where v is the data volume for a single query as defined in Equation 5.1.

$$benefit = ((q \times TTL) - 1) \times v \tag{5.9}$$

Replication

One of the problems with the fault-in approach is that one query will always experience an increased query response time due to the cache invalidation triggering the update via query shipping. By anticipating what information is required in the cache, the information could be transferred asynchronously (replicated) to the query and hence the cache could always remains valid. The caching approach requires a request-driven data shipping model whereby the client is active (initiates queries) and the information sources are passive (process queries upon request). The provision of a persistent information source enables alternative approaches for data shipping to be considered, in particular where those information sources actively transfer all the information to the information service. There has been a great deal of interest in push technology for distributed query processing [98]. The challenge is to anticipate what information needs to be replicated.

If p is the probability that a specific query will occur within a time period then it follows that for n queries during that period, the probability that a specific query occurs is given by Equation 5.10.

$$p_n = 1 - (1 - p)^n \tag{5.10}$$

As $0 \leq Q(t) \leq 1$, $0 \leq (1 - Q(t)) \leq 1$, $\therefore$ the limit of $(1 - Q(t))^n$ as $n \to \infty$ is 0, hence as the number of queries increases, so does the probability that a specific query will occur in a given time period. This means that with enough queries in that period it is highly-likely that a specific query will occur and hence the corresponding object or attribute will be requested, hence all information from all services needs to be transferred. In this scenario, with a sufficiently large number of queries, the caching approach would in any case ship all information from the information sources. Hence, the focus should be on efficiently replicating the information from the information source to the information service.

This can be achieved by copying all information from all information sources by using a query that requests all objects. Periodically executing this query before the time-to-live has expired will ensure that the cache is always valid. The required frequency of this query can found by using the (α, β)-currency equation so that α does not decrease below the desired quality level. One method to optimize this approach is to use different queries so that only information that needs to be updated is obtained.

In general, cache update techniques for maintaining the consistency of a quasi-copy of the information source at the information service would be an area for further investigation. However, a simple method is for all the information from the information source to to be transferred when a change is detected. The total data transferred v for n information sources is given by Equation 5.11, where c is the number of changes to the information source and d is the size of the individual information source.

$$v = \sum_{i=1}^{n} c d_i \tag{5.11}$$

If only the changes are transferred, the total data transferred is given by Equation 5.12 where d_{ij} is the size of update for the individual information source.

$$v = \sum_{i=1}^{n} \sum_{j=1}^{c} d_{ij} \tag{5.12}$$

In both cases, the query response time is given by Equation 5.2. The value for β in the (α, β)-currency equation is the latency between when the information source changed and when the information service was updated. If there are multiple changes within a short period, changes can be bunched which will reduce the number of transfers that occur, although not the volume of data moved. Such a grace period would increase the latency and hence the value for β in the (α, β)-currency equation.

5.3 An Evaluation Of The Different Approaches

In order to evaluate the different approaches with respect to the benchmarking method outlined in Chapter 4, information corresponding to a production Grid infrastructure is required. By analysing this information, the necessary parameters can be found which are transposable to other testing environments.

5.3.1 Query Response Time

The MDS is an example of an implementation that has adopted the query shipping approach. Its performance has been the focus of a number of studies [89, 90, 91]. These evaluated the effect of a large number of concurrent queries and information sources on the average query response time. However, in these studies little attention was given to the information or queries themselves. A single query, return all information for a specific information source, was used and it was assumed that the response was always correct i.e., up-to-date, or current. In addition, the information volume used was small (10KB) compared with today's infrastructures (100MB). The benchmarking of the MDS in Chapter 4 addressed many of these issues and provides some parameters that can be used here.

What these studies show is that when there are a large number of concurrent queries, the query response time degrades dramatically without the use of caching. The benchmarking of the MDS in Chapter 4 showed that the query response time when the cache is invalid (query shipping approach) was 41 minutes 36 seconds, compared to approximately 2.6s for when the information is returned from the cache (data shipping). Further investigation indicated that the long query response time for when the cache is invalid was due to an inefficient method for updating the cache rather than the query shipping approach itself. When caching is based on a synchronous fault-in approach, the query is blocked while the cache is updated resulting in a significant degradation to the query response time. In this scenario, the data shipping approach reverts back to a query shipping approach. All subsequent queries would also be blocked until the cache update process has been completed, and hence will also experience an increased query response time.

Updating the cache asynchronously to the query can overcome this drawback and is a form of replication that is based on obtaining the information via query shipping. Previous work [73, 88, 77] has attempted to address the limitations of the caching approach by updating the cache asynchronously to the query, using a query shipping approach that requests all information from all information sources.

As the query response time is linked to the specific design decisions, such as query language used and cache data structure, it has to be measured. Using a data shipping approach (implemented using an LDAP database) the query response time for the top ten queries made to the WLCG infrastructure was measured in Chapter 4 and can be seen in Table 5.1 along with other relevant metrics.

Query	Size (B)	Object	Frequency (s^{-1})	QRT (s)
Q1	2858	CESEBind	19.6	2.62
Q2	5436	SA	17.7	2.57
Q3	315225	SE	16.3	2.63
Q4	642	SE	15.5	2.58
Q5	1875	CESEBind	7.8	2.64
Q6	49801	Service	6.8	2.64
Q7	16607	CE	2.1	2.46
Q8	12348	SA	2.1	2.58
Q9	8959015	SubCluster	1.5	3.19
Q10	6009	VOView	1.4	2.46

Table 5.1: The top ten queries made to the WLCG infrastructure

An investigation in the next chapter endeavours to find good values for the timeout and number of threads when querying all information sources for all information, which is the query shipping approach used to populate the cache in the BDII. It finds that the time taken is between 20.25s and 22.08s depending on the timeout value used. This compares to 4.8s with σ=0.4 when querying for all objects using a cache/replica of the information.

What is clear in all these studies is that the query response time is less for the data shipping approach than the query shipping approach.

5.3.2 (α, β)-currency

The value of α for the query response in the query shipping case can be calculated using the (α, β)-currency equation where β is the query response time for the query. Taking 21s as the value for β, which represents the worst case for the queries when all information is returned, the values of α for the top ten queries are shown in Table 5.2.

Query	Object	λ (s^{-1})	β (s)	α
Q1	CESEBind	1.14×10^{-08}	21	0.999999
Q2	SA	5.11×10^{-04}	21	0.989326
Q3	SE	4.85×10^{-04}	21	0.989866
Q4	SE	4.85×10^{-04}	21	0.989866
Q5	CESEBind	1.14×10^{-08}	21	0.999999
Q6	Service	4.86×10^{-04}	21	0.989845
Q7	CE	1.91×10^{-03}	21	0.960683
Q8	SA	5.11×10^{-04}	21	0.989326
Q9	SubCluster	2.28×10^{-06}	21	0.999952
Q10	VOView	1.48×10^{-03}	21	0.969398

Table 5.2: The (α, β)-currency for the query shipping approach

From the results in Table 5.2, even with a query shipping approach, due to the length of time it takes to execute the query, seven of the ten query responses have a value of α less than 0.999. For Q7 (CE) and Q10 (VOView) α is as low as 0.961 and 0.969 respectively.

When using a cache the value of α is predetermined by the cache TTL value. The (α, β)-currency concept can be used to calculate the optimal TTL value. For a given quality level α, the maximum TTL β can be calculated using Equation 5.13.

$$\beta = \frac{ln\alpha}{-\lambda} \tag{5.13}$$

Table 5.3 shows the optimum value of β for each object type where α=0.999. quality level. For simplicity, the values of β have been grouped into second, hour and day. These are the smallest time units for a specific object type where α is greater than 0.999.

The results in Table 5.3 suggest that a single value for the TTL, which was used in the original MDS deployment, may not be optimal. For many of the queries, the TTL value of 30s used is too conservative. The TTL could be increased to 3600s (1 hour) and α would still be greater than 0.999. However, for the queries that query objects

Object Type	λ (s^{-1})	β (s)	Grouping
CE	1.91E-03	0.52	Second
CESEBind	1.14E-08	88086.72	Day
CESEBindGroup	5.71E-09	175240.31	Day
Cluster	3.54E-08	28294.81	Hour
Location	7.66E-08	13069.01	Hour
SA	5.11E-04	1.96	Second
SE	4.85E-04	2.06	Day
SEAccessProtocol	2.99E-08	33423.77	Hour
SEControlProtocol	3.29E-09	303847.95	Day
Service	4.86E-04	2.06	Second
ServiceData	1.99E-09	503153.62	Day
Site	3.63E-08	27583.25	Hour
SubCluster	2.28E-06	437.98	Second
VOInfo	3.86E-09	259381.45	Day
VOView	8.35E-06	0.68	Second

Table 5.3: The optimum TTL value of β for a *three nines* tolerance level

which have a high frequency of change, the TTL value would need to be reduced to 1-2 seconds.

This suggests that an approach where the TTL is set per object type may be better.

What also needs to be considered is cost in terms of time required to create the cache, which will increase the query response time by t_c as shown in Equation 5.7. If the time taken to obtain the information using a query shipping approach t_c is greater than the TTL value, the cache will be immediately invalidated and hence it is not possible to provide a cache with the desired quality level. If the $t_c << TTL$, then this approach can be considered and the information should be cached. In any case, it can be seen from the query shipping approach that the query response time already results in a value of α that is less than 0.999 and as such, a data shipping approach based on caching information cannot improve upon the query shipping approach.

If an asynchronous cache update approach is used, the optimal period after which the cache should be refreshed is identical to the optimal values for the cache TTL as shown in Table 5.3. The current value used in the WLCG Grid information system is 120s. It is clear from using the (α, β)-currency equation that for many of the queries, this value is too conservative. It could be increased to 3600s (1 hour) and α would still be greater than 0.999. However, for the queries that query objects which have a high frequency of change, this value would need to be reduced to 1-2 seconds.

Again the cost in terms of time taken to update the cache needs to be considered as this will increase the grace period β and hence α. If the time taken to obtain the information needs to update the cache β is greater than the TTL value, then the information in the cache will not have the desired quality level.

To reduce the time taken to update the cache asynchronously, a differential algorithm was employed to refresh the cache [73]. This differential update algorithm compares the newly obtained information with the cache content, generates a set of update operations that represents the differences and applies these operations to the cache. The differential update algorithm can be divided into three distinct parts; query shipping for all information from all information sources, generating the update operations and executing those operations. The BDII was instrumented in order to obtain detailed timing measurements of the differential update algorithm. The tests were carried out in a virtual machine (SL5, 2GB, 2.33GHz Intel Xeon CPU, 100Mbps LAN) in a production top-level BDII deployment over the course of one day. On that day the volume of information representing the infrastructure had a size of 114MB in the LDIF format and represented information from 4040 services spanning 373 sites. The average time taken for each part of the update algorithm was calculated. The query shipping took 96.5s, generating the operations took 41.6s and updating the cache took 46.9s. It is clear that the time for the update algorithm to run is significantly greater than the 1-2 seconds refresh period required by the high frequency object types.

It can be seen that the first two parts of the differential update algorithm which generates the update operations take 138.1s. A method that significantly reduces the time it takes to generate the list of update operations would reduce the overall update time.

In general, the value of β in (α, β)-currency for the data shipping approach represents the latency between when information was last verified as current and the time when the status is reflected in the cache/replica. This will be technology- and implementation-dependent, and hence must be found by experiment.

5.3.3 Data Transfer

As one of the main features of a Grid computing environment is that the information sources (Grid services) are widely distributed geographically, this suggests that data movement is an important factor that contributes to the query response time and (α, β)-currency, as time is a component in both cases. The underlying network that links the Grid services provides physical limitations on the movement of data on top of which technology choices for the implementation will provide additional constraints. We assert that the approach which requires the least amount of data to be transferred should therefore be the more efficient approach.

The top ten queries made to the WLCG infrastructure, which together represent 90.8 % of the queries made, has already been shown in Table 5.1. The total data transferred v for 90.8 queries can be calculated by using Equation 5.14, where the values for d, the query response size, and f, the frequency of occurrence, are provided in Table 5.1

$$v = \sum_{q=1}^{10} f_q d_q \tag{5.14}$$

The result is that 18.3MB data is transferred for those 90.8 queries. According to [60], the CERN top-level BDII service in the WLCG information system experiences approximately 2 million queries per day. Taking the top ten queries as our sample, the total data transferred per day would be 1,748GB. This represents both the volume for the query shipping approach and for the information service itself. The location of the information service with respect to the client is not really a concern (assuming an even distribution across Grid services and clients) as the same amount of data will need to be transported. In the absence of a catalogue that provides additional metadata for reducing the number of information sources that needed to be contacted, each information source would have to handle 2 million queries and transfer 447MB, assuming there are 4000 information sources.

For a cache-based data shipping approach, the total data transferred per day, v, for the top ten queries can also be calculated by using Equation 5.14 where d is the query response size, and f is the number of times the cache will be invalidated per day.

In the initial MDS deployment scenario, the cache TTL was set to 30s. With a total data size of 102MB, the throughput is 28.7GB per day, which is 1.6% of the amount transferred when using the query shipping approach.

Adopting a policy-based approach, where the cache TTL is based on the optimal values as defined in Table 5.1, the data transferred is 364GB, which is less than the query shipping approach, but higher than a caching approach that uses a 30s TTL.

With the differential update algorithm, all information is transported. By introducing an active information source, changes can be identified at the information source and the updates can be shipped to the information service. This essentially parallelises part of the differential update algorithm and further reduces the amount of information that is transported.

The changes observed translate into the information that would be sent in such a scenario. Table 5.4 shows the number of updates that would be sent per day based on the analysis of the snapshots in Chapter 3. The values have been rounded to the nearest whole number (i.e. complete update).

Object Type	Size (B)	Add	Delete	Modify	Total	Rate (KB)
Site	894	1	1	1	3	2.6K
Service	1499	10	12	542016	542038	813,000
ServiceDataKey	439	44	51	71	166	73
Cluster	1610	1	1	3	5	8.05
SubCluster	15575	2	2	2390	2394	37,000
CE	1995	13	12	1867320	1867345	3,700,000
VOView	854	42	42	3243024	3243108	2,800,000
Location	610	714	534	459	1707	1,000
SE	668	1	1	81504	81506	54,000
SA	1158	6	7	205272	205285	238,000
AccessProtocol	635	4	4	1	9	5.7
ControlProtocol	550	1	2	1	4	2.2
VOInfo	596	22	21	1	44	26.22
CESEBindGroup	436	10	12	2	24	10
CESEBind	479	16	50	1	67	32
		887	752	5942066	5943705	7,600,000

Table 5.4: Estimated number of changes per day

Based on these observations, this approach would generate nearly 6 million updates a day (69 per second) and have a throughput of 7.6GB a day (88KB per second). This can be compared to 28.7GB for the simple caching approach with a 30s TTL. The interesting aspect of the data in Table 5.4 is that the number of add and delete changes is less than 0.03% of the total number of changes. In fact the updates for six high frequency object types are responsible for 99.96% of the total changes.

This simulation suggests that this is technically feasible as a data rate of 88KB per second is much less that the Gbps WAN connections that are provided. However, one issue with this approach is that only modified information is sent. This presents a bootstrapping issue with initial population. Using a similar approach to (α, β)-currency, the fraction of values populated in the cache p after time t can be modelled with Equation 5.15.

$$p = 1 - e^{-\lambda t} \tag{5.15}$$

This suggests that a hybrid approach of query shipping to initially bootstrap the cache, then shipping the updates to keep it up-to-date, may be a good approach. Further thoughts will be required to ensure synchronisation when experiencing temporary network failures.

An alternative approach would be to periodically publish all the information which could also act as a heartbeat for system monitoring.

5.4 Concluding Remarks

This chapter re-visited the fundamental concepts of processing queries in a Grid environment in order to identify the different approaches. The fundamental problem of resolving a query over many information sources is a data retrieval problem and in particular the field of distributed query processing. There are two main approaches for distributed query processing; query shipping (moving the query to the data) and data shipping (moving the data to the query).

Previous studies have shown that query shipping does not scale well if there are many concurrent queries. It can be seen from the earlier performance investigations into the MDS in Chapter 4 that this is indeed the case. Not surprisingly, the query response time is lower for the data shipping approach than for the query shipping approach, although the actual performance difference may depend on other factors including the implementation. If a fault-in approach is used to update the cache, some queries will experience higher response times due to the cache being invalidated and a query shipping approach being used to perform the update. If the cache refresh is synchronous with the query, the query and all subsequent queries will block until the cache has been refreshed and hence will also experience an increased query response time. The data shipping approach based on either an asynchronous cache update process or replication will give a predictable query response time for all queries. Due to the query scalability demands of Grid computing, a data shipping approach should be used to provide the best (lowest) query response time. However, it must be understood that the data shipping approach used does not decrease the quality of information returned with respect to the value of α in the (α, β)-currency metric.

Even with a query shipping approach, the value of α for seven out of the top ten queries falls below the 0.999 quality level. The optimum value for the TTL value to be used in a fault-in caching approach has been calculated using the (α, β)-currency equation which shows that dynamic information would be invalidated every 1-2 seconds. As it takes longer to obtain this information using a query shipping approach (a standard query takes approximately 2.5s), the result is that the cache will always be invalid. The current method of asynchronously updating the cache by using query shipping to obtain all information from all information sources and performing a differential update takes 138.1s, which far exceeds this 2 second threshold.

Considering the data transferred for the WLCG use case, a query shipping approach would transport 1,748GB due to the duplication of the query response. Note that the total size of the data from all information sources is only 104MB.

Due to the large number of queries all information needs to be shipped. This suggest that the fundamental problem is to replicate the data from the information source at the information service. Replicating via a simple copy, as implemented in the current Grid information system, is not efficient as all object types are considered identical. This means that for some object types, redundant data being shipped and for other object types there is a low value for α as data is not being shipped frequently enough. By introducing an active information source, changes can be identified at the source and the updates can be shipped to the information service. Based on a simulation, an approach that ships the changes would generate nearly 6 million updates a day (69 per second) and have a throughput of 7.6GB a day (88KB per second). This indicates that this approach is technically feasible as a data rate of 88KB per second is negligible when compared to the Gigabit WAN connections that are available today. The value of β in (α, β)-currency for this approach represents the latency between when information changed and the information service is updated and must be found by experiment.

Finally, referring back to the expansion and contraction tests as mentioned earlier, for the WLCG use case there are 2 million queries and 6 million updates, which suggests that replication should be avoided. However, looking at the data transferred, 1748GB for the query shipping and only 7.6GB for replication, suggests that replication should be used.

Chapter 6

An Architecture for Grid Information Systems

This chapter presents an architecture for a Grid information system that follows the data shipping approach based on the replication of information. An implementation of this architecture is described and evaluated using the benchmarking method as defined in Chapter 4.

Specifically, this chapter makes the following contributions:

- A Grid information system architecture that follows a data shipping approach based on the replication of information.
- An implementation of this architecture based on enterprise messaging technology.
- The use of (α, β)-currency to guide implementation decisions.
- The use of a data shipping approach that sends changes as soon as they occur at the information source to improve the currency of highly-mutable (dynamic) information.
- An evaluation of this implementation using the benchmarking method as defined in Chapter 4.

This chapter is structured as follows; the Grid information system architecture is presented in Section 6.1 with the implementation described in Section 6.2; the evaluation of this implementation using the benchmarking method is documented in Section 6.3 along with some concluding remarks in Section 6.4.

6.1 An Architecture For Grid Information Systems

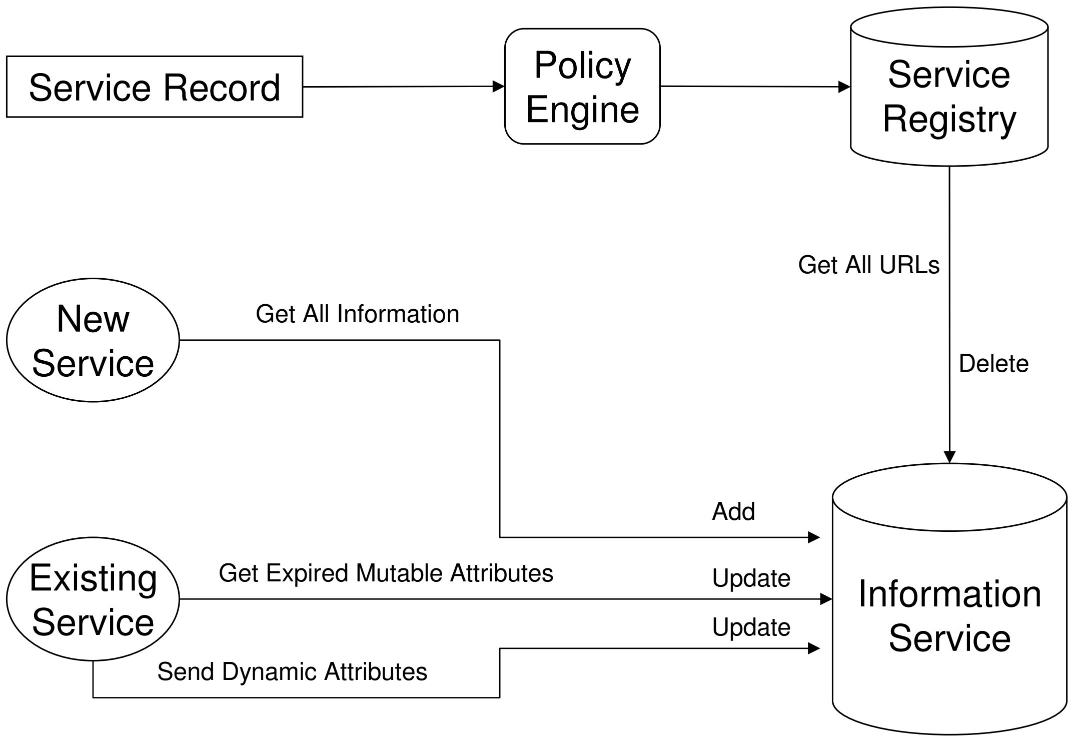

Figure 6.1: The inclusion of data shipping in a Grid information system architecture

In this section two related and complementary approaches are presented. One is an incremental improvement over the existing architecture that makes use of parallel mechanisms for the update process and is shown in Figure 6.1. Information is obtained once from the information source when a new service is discovered. Mutable information is periodically updated at the optimal frequency for each object or attribute type. Updates for the dynamic attributes are obtained from the information source via a transport mechanism. The other is a simplified approach that is solely based on data shipping and is shown in in Figure 6.2. Both represent a major change for Grid computing as they require the service to be an active information source and explicitly use data shipping rather than query shipping. In particular, the novelty of this approach is the introduction of a publish-subscribe model in order to send changes as soon as they occur at the information source and hence improve the currency of highly-mutable (dynamic) information.

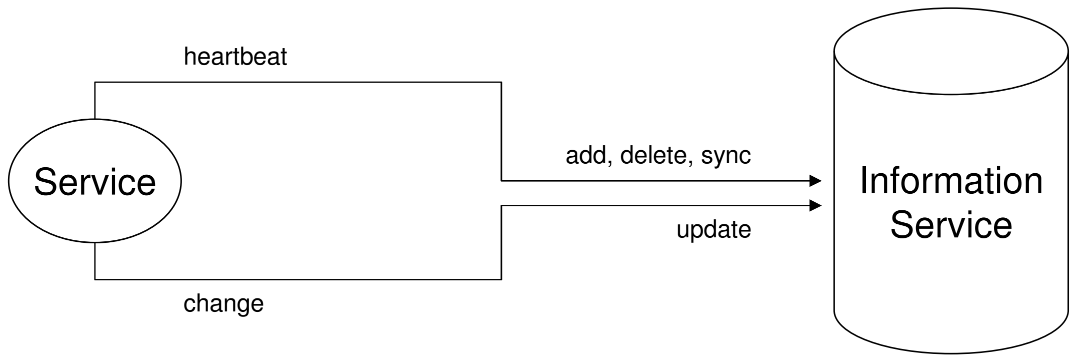

Figure 6.2: A data shipping architecture for Grid information systems

The prerequisite of these architectures is the existence of a catalogue in the form of a service registry which is a result the management policies underpinning the Grid infrastructure, as mentioned in Chapter 2. The service registry provides a list of URLs for the services that exist in the infrastructure. A URL, as specified in RFC 1738 [99], is a generic syntax that is used to locate resources by providing an abstract identification of the location and is not limited to the domain of Grid computing. However, it is assumed that the URLs returned identify where further information about the service can be found. The concept of existence can differ between different designs. Those that make use of a soft-state protocol to register a service show what *is* there. If a service is down, the registration is removed and hence not found. Others that link the registration to operational procedures and policies show what *should* be there. If there are problems with the Grid infrastructure or the service itself, the registration is still available and hence can be discovered.

The main question is how to deal with the decommissioning of a service so that service registry does not serve stale information. With a soft-state approach if a service is decommissioned, it will no longer update the registration which will automatically be removed from the service registry after the time-to-live for that registration has expired. If a more persistent approach is used, the registration will still be discoverable until it is explicitly removed. Both approaches result in undesired consequences. If a service exists but the registration is not available, inconsistencies can arise where persistence is required. For example a Storage URL (SURL) specifies the logical location of a file and is used in file catalogues. From obtaining a SURL, further information about the service may be needed in order to obtain the file, or at least to know that the service is unavailable. On the other hand, if an action is required when a service is decommissioned, stale information will be returned if this action fails to be carried out. The result is that some URLs will not return information about a service, similar to the HTTP 404 error found on the Web. A related issue with existing approaches is with regard to historical information. It is currently not possible to obtain information about the existence of a service in the past. We conclude that the definition of existence and the approach to decommissioning is a policy issue to which the service registry adheres.

In Chapter 3 the concept of static and dynamic information, which has been widely used by the Grid community, was discussed. It was found that this concept is misleading as all objects types experience change, although the rates of change vary for each type. We will build upon the taxonomy of possible changes for Grid information to introduce an alternative, more accurate description of information. The term *static* will be used to describe objects or attributes that only support the add or delete operation and hence are persistent throughout the lifetime of that service. The new term *mutable* will be introduced to describe objects or attributes that support the update operation. The term *dynamic* will be used to refer to the specific case of highly-mutable objects or attributes. From the investigation to find the optimum TTL value of β (where $\alpha = 0.999$) we can equate these terms to the update periods of daily, hourly and per second. Using this terminology, the information obtained from the service registry system is static. When a service joins the infrastructure, information is added and when the service is decommissioned that information is deleted. As such the URLs for all services will be obtained from the service registry at least once per day. The uncertainty could be reduced by obtaining this information more frequently or using a notification mechanism whereby the service registry publishes the latest changes. The actual frequency used will depend on the cost of obtaining the information from the service registry and notification is only an option if the service registry supports this functionality.

From benchmarking the existing BDII-based information system in Chapter 4, the approach used for static information resulted in $\alpha = 1.0$. Therefore the accuracy of static information is determined by the completeness of the URLs returned from the service registry. Any latency for adding a new service does not affect the quality of existing results, only the completeness. The latency for decommissioning a service does affect the quality as the information returned will not be current. However this is mainly an issue for the service registry as the overhead for deleting information is negligible.

For mutable information, excluding dynamic information, the approach can be improved upon as $\alpha < 1.0$. As information is currently updated at the same frequency, the update frequency for each object, or in addition each attribute, can be calculated. This can be done by either using the results from Section 3.2 or can be found dynamically though a process of continuous monitoring of the changes and calculating the (α, β)-currency based on results the last or last few updates.

The overhead of contacting a service multiple times at different, yet similar frequencies may be more expensive than contacting it once. The novel approach suggested here is that the concept of (α, β)-currency is used to evaluate the optimal frequency and each object, or in addition each attribute, is considered independently.

For highly-mutable dynamic information, the BDII approach evaluated in Section 4.3 is inadequate as $\alpha < 0.9$. An approach to ship the information from the service will be evaluated. Three components are therefore required. One at the information source that can detect when a change has occurred and take the appropriate action. A second to transfer this information from the information source to the information service, and a third at the information service to do the necessary update. One constraint is that reverse discovery should not be necessary, i.e. a service should not have to discover the information service. Therefore the transfer mechanism has to ensure that information reaches the information service, even if the service does not specify the final location.

As mentioned in Chapter 5, if information is only sent when a change occurs, there is a bootstrapping issue with the initial population. This can be solved by obtaining the initial values directly from the information source. Periodically cross-checking with the information source directly may be required to maintain the consistency when experiencing temporary network failures. An alternative approach would be to periodically publish the information which could also act as a heartbeat for system monitoring. Publishing information this way could also be useful from a service discovery perspective. By monitoring the heartbeats and extracting the unique identifier for the service, it would be possible to discover all the services. The main difference with such an approach is with respect to the Grid infrastructure management policies used. The Grid infrastructure management policies implemented define rules regarding which services should be in the infrastructure. This suggests that the information service should compare the heartbeats with the service registry and reject those that are not found. The main advantage with this approach is that new services would be discovered quickly, they would send a heartbeat immediately on starting, however the disadvantage is with persistence i.e. services that no longer send a heartbeat are considered defunct. This approach could also be adopted for mutable information. The advantage would be receiving prompt updates about changes, and persistency is not relevant as it is a service discovery issue.

6.2 A Messaging-based Grid Information System

The service discovery method used in WLCG today is entangled. A site must follow a number of procedures in order to be added into the GOC DB, or for U.S. sites, the OIM (OSG [10] Information Management) system. Each service is manually registered by a site manager along with a few attributes such as the service type and host name. The GOC DB and OIM can be queried directly for registered services but this does not necessarily reflect what is deployed. To discover what services are deployed, the site-level information service is queried, or OSG information service in the case of OSG. The URL used to contact the site-level information service is mandatory information required for registering a site.

The EMI Registry (EMIR) [64] presents an elegant approach that suits the federated management style of the WLCG infrastructure. The foundation is that the services to be discovered are on-line services that are connected to the Internet. The primary use case is Grid services but as a generic approach towards services has been adopted, this could be extended to other use cases such as discovering cloud-based services. Services are grouped primarily with other services that are managed by the same autonomous organisation and such grouping is call a domain. Domains can be organised in a hierarchical structure, however there is no single top-level organisation. Domains can also be organised to form a federation, whereby each federation is the authoritative source for services within its own federation and shares this information in a peer fashion with the other federations.

EMIR provides two main building blocks; the Domain Service Registry (DSR) and the Global Service Registry (GSR). It is envisaged that these base components can be used to support different topologies that map to the real operational and deployment models of production infrastructures, and an example is provided in Figure 6.3. While the ability to form a hierarchicy of registries is similar to MDS (Figure 2.9), EMIR in addition offers the ability to form other grouping of services by using a peer-to-peer approach. A Service Endpoint Record Publisher (SERP) periodically sends information about the service to the DSR and is a soft-state protocol, similar

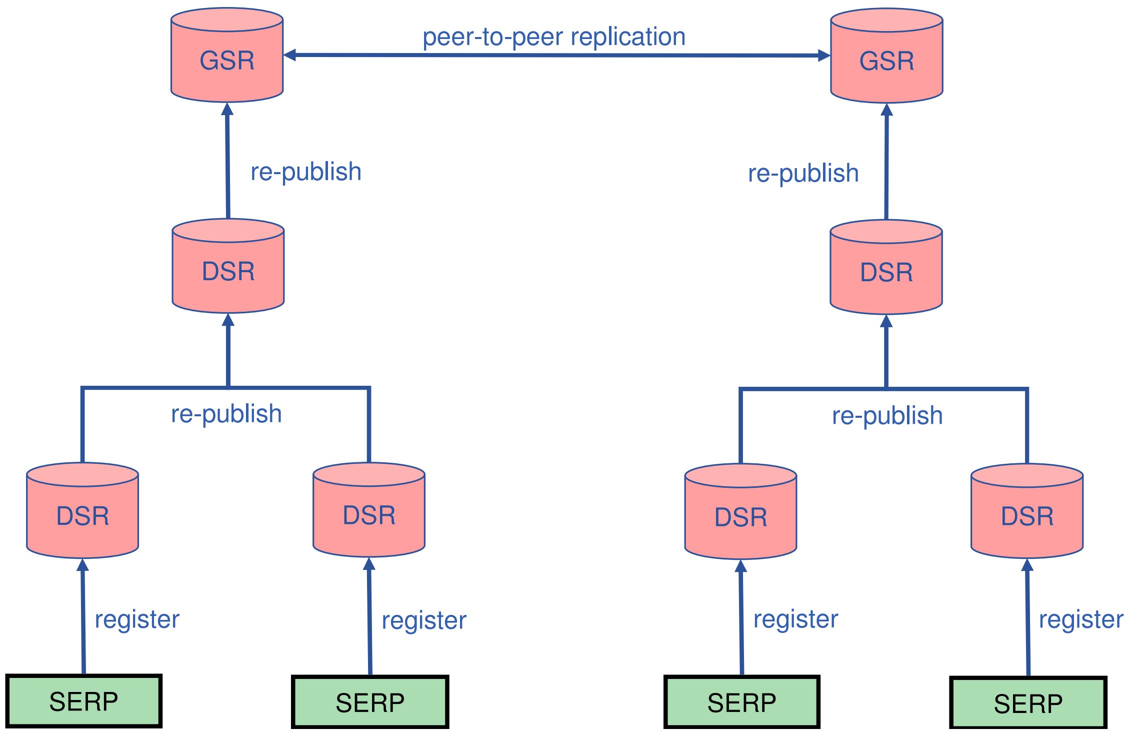

Figure 6.3: An example deployment of the EMIR

to the Grid Resource Registration Protocol [65]. The DSR may re-publish this information to a parent DSR/GSR, or in the case of the GSR, replicate this information n-times (where n is the number of GSRs) to the other GSRs. The peer-to-peer network is formed from GSRs that are configured using a static list, as it is assumed that the placement of the GSR will be predetermined when the topology is defined and that this will not undergo significant changes. An authentication and authorisation mechanism is adopted by all components to ensure that only authorised registration can occur. The core aspect of the system is the service endpoint registration record. It contains static information that does not change during the lifetime of the service. The goal is to avoid the need to update service records, which would add additional complexity to the system. However, updates to service records are supported in the implementation, and hence semi-static information such as version information is available. Highly-dynamic information, state information that changes frequently, is not available in the service record

One issue with this approach for WLCG is the overhead with respect to the effort required to roll out a new system across a large-scale distributed infrastructure. Even if it only takes one day of effort to set-up an EMI DSR, integrated over approximately

400 sites, this represents more than one year of full-time effort. Being able to achieve a similar result with the existing infrastructure would therefore be desirable as it would minimise the operational overhead. As each site (domain) is already running an LDAP server in the form of a site-bdii, this could be used to store and manage the service records for a site. Services could register themselves to their LDAP-based DSR using the ldapadd operation, similar to the soft-state registration protocol GRRP used in the MDS. A higher-level domain could be introduced at the national-level to represent a NGI's Domain. Records could be synchronized between the site-level and national-level using the OpenLDAP replication method *(*sncrepl). A GSR would be an extension to the national-level DSRs that obtains information from itself and all other national-level DSRs. The policies are defined by the domain and result in the configuration of the DSR or GSR respectively. The main difference between this approach and EMIR is the technology used to implement the architecture.

The existing BDII update process can be improved by updating mutable information at different frequencies. One improvement is that the top-level BDII will use the service discovery system to obtain the URLs for the services and contact them directly. There is no operational, performance or policy reason why information must be obtained using a hierarchical approach. Using the (α,β)-currency concept, the maximum grace period β for a specific value of α can be defined for groupings of attributes. The update process will be modified so that all information is only obtained for the initial cycle. The time since the last update for each attribute grouping will be recorded. For subsequent cycles the attribute grouping will only be updated if they have exceeded their lifetime by the following update. The service discovery system will be periodically queried to discover new services. When a new service is found, all information will be obtained and then updated as described above. This can be achieved by modifying the existing update mechanism used in the BDII. Rather than obtaining the URLs of the site-BDII services from the GOC DB, the URLs for all services can be discovered. An improved method to query all those URLs will be required to take into account that the services will be queried at different frequencies. The interactions between the main components of the information system are shown in Figure 6.4.

Using the existing BDII update process for updating mutable information would require the update cycle to obtain information every few seconds and may result in many information services being contacted unnecessary. Hence, an alternative approach that uses an enterprise messaging system will be implemented for shipping the dynamic

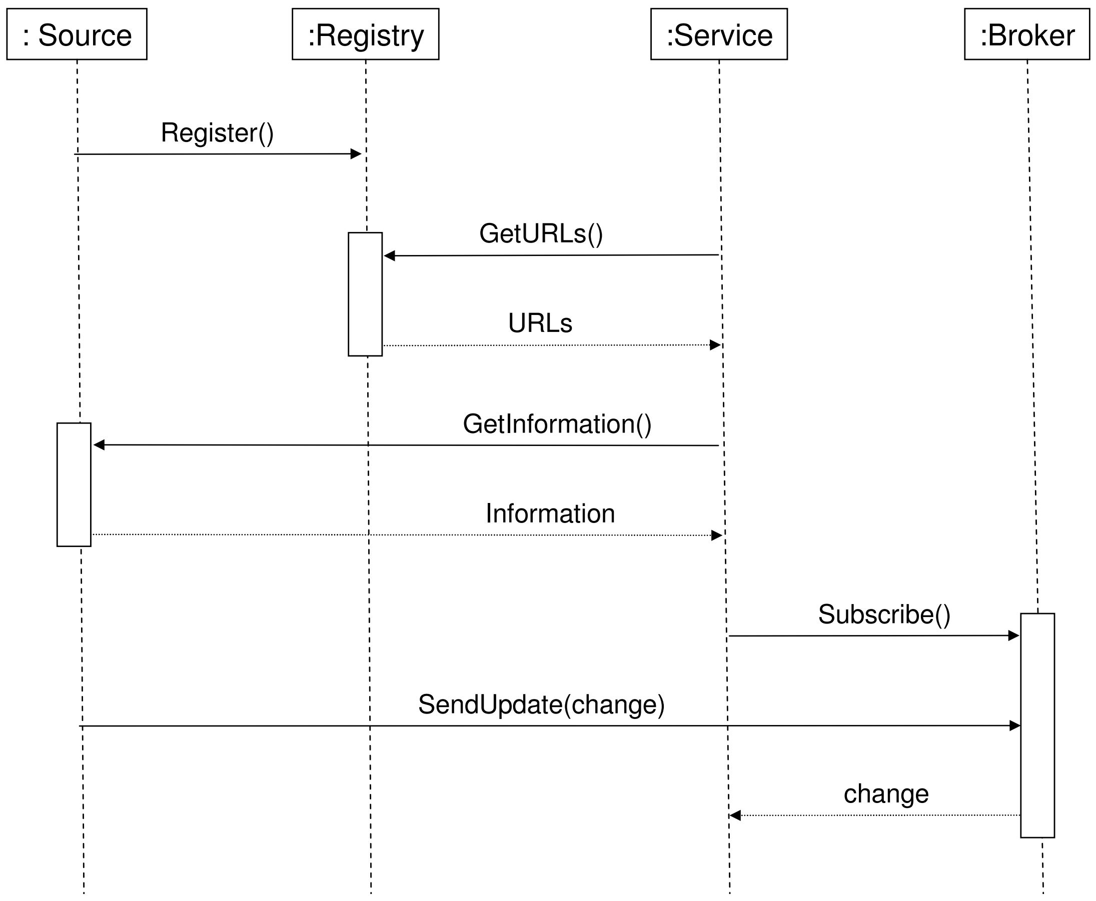

Figure 6.4: A sequence diagram showing the interactions between main components of the information system

information. An enterprise messaging system enables messages to be exchanged between peers where the exchange is mediated by a messaging broker [100]. It is the role of the message broker to route messages between multiple peers and hence neither the sender (information source) nor receiver (information service) needs to be aware of each other. The message itself consists of two parts; a header and a body. The header contains meta-information about the message which is used by the enterprise messaging system to correctly route the message. The body contains the information that is intended for the receiver. There are two main message delivery models: point-to-point and publish/subscribe. Point-to-point uses the concept of message queues. Messages are queued until they are consumed or expired. Each message is received by only one receiver and delivery is guaranteed. With the publish/subscribe model, messages are sent to a topic from where they are consumed by all available consumers (a broadcast approach). Each message can have multiple consumers, but consumers can only consume messages sent after they have subscribed to a topic and the message is lost if there are no consumers. In either case the sender of a message does not need to know the identity nor location of the receiver. It simply connects to a message broker and sends the message to a topic or queue. Multiple message brokers can be used in different ways to provide a reliable and scalable infrastructure for message delivery. In such a scenario a message can be sent to any broker, and the network of brokers will route the message to all interested consumers who might be connected to other brokers.

The use of enterprise messaging systems in Grid infrastructures to simplify and improve the information exchanged between several loosely coupled components has recently been investigated [100]. It has been adopted as an integration framework for service monitoring and other operational tools including accounting, ticketing and operational dashboards [100]. One early adopter of enterprise messaging systems was the Service Availability Monitoring (SAM) system [101] as the requirements for security, scalability and reliability were low. The original SAM architecture followed a centralised model where monitoring results from agents (probes) were returned using a client/server model. Due to the short life-time of the agents, any downtime of the central service resulted in lost monitoring results. As the monitoring results are used to calculate service availability metrics for comparison with service level agreements, lost metrics affect the perceived availability. Messaging queues that guaranteed the delivery of the monitoring results were seen as a solution. In addition, the general trend towards a de-centralised operational model required monitoring results to be aggregated from regional testing initiatives. The experience gained over the past few years has been summarised [102] along with some recommendations on the use of messaging in Grid infrastructures. One recommendation is that each application should use its own dedicated messaging infrastructure as they may require incompatible configuration to achieve optimal performance. Dedicated messaging services should be used in a *send-to-one read-from-all* model, as it was found that in some cases e.g. software upgrades or major configuration changes, the whole network of brokers needed to be stopped and therefore it was not possible to deliver continuity of service. A messaging service could just be one process on one machine, or a fully distributed service spanning tens of dedicated servers using tailored hardware and increased network capacity.

Building upon this experience and considering the estimated number of messages per day from Chapter 3, only a single dedicated message broker would be required for the information system use case. In a production deployment, more than one broker would be needed to ensure service availability and service continuity but it is not necessary for evaluation purposes. If more brokers are required due to scalability or policy reasons e.g. a NGI wishing to control their own service, the identification of all brokers would be required. This information is something that needs to be known and managed by the messaging service providers. The send-to-one read-from-all approach is suitable as there are many more Grid services than information services. The placement of brokers is not a major concern as the throughput of the system is the same, brokers just route messages. For example, the number of messages between the Grid service and broker is equal to the number of messages between the broker and the information service. However, if there are multiple information services, this multiplies the messages on the consumer side so the network connectivity on the consumer side of the message broker may be a concern. To avoid reverse discovery, the Grid services themselves should be configured to only use one broker and a DNS alias can be used to avoid services having to be reconfigured due to changes with the broker infrastructure. This alias could also act as a load balancer, redirecting the connection requests to the best possible broker, and be under the control of the operations team managing the messaging infrastructure. A topic-based approach will be used to send the updates for the dynamic information to the top-level BDII via the message broker. To generate the message, a watch-dog process will be added to the Grid service to detect information changes. When the information changes, a message describing the change will be sent to the message broker using Simple (or Streaming) Text Orientated Messaging Protocol (STOMP) [103], as recommended in [102]. The updated process of the BDII will be augmented with a process that listens for new messages on the topic and updates the database on receipt of the message.

As mentioned in Section 6.1, if a heartbeat is used to address the bootstrapping problem and synchronisation issues, there is no need for a service discovery system. As a consequence, the full information system could be implemented using messaging alone. Heartbeats can be sent on one topic and updates on another topic. A new service (or object) would be identified by its first heartbeat. A special flag in a heartbeat such as the time-to-live value being equal to zero could represent the decommission of a service. As topics are used to group related messages, they can be used by clients to essentially filter messages by only registering to topics in which they are interested. A topic will therefore be defined for the main abstract object type (site, service, computing element and storage element). Although this is irrelevant for the information service as it will consume from all topics, splitting the messages this way may be useful for other applications.

One consideration of such an approach is how to control what services are allowed to join the infrastructure. There are two possible options; either control what can be published, or control what is consumed. If the messaging system is open then there is no control over what messages are published. Who can publish to the messaging system needs to be restricted to avoid misuse of the infrastructure. Using SSL connections provides one obvious advantage, as a trust framework for Grid computing has already been established in the form of the IGTF. This would restrict misuse of the messaging system to within that community and could be addressed through an acceptable use policy. This would not prevent accidental misuse and further authorisation may be required if this risk is to be mitigated. An authorisation method would require an understanding of the policies that describe who is and who is not allowed to use the system, similar to the service discovery problem. For example if the authorisation used the DN in the certificate, the DNs (an attribute of the service) for all services need to be discovered. This could be avoided by adding attributes to an x509 proxy, similar to how the Virtual Organisation Membership Service (VOMS) [104] works for virtual Organisations. Each management domain could set-up a service similar to VOMS that receives requests from Grid services to add an attribute. The service proxy could then be used to connect to a message broker. The message broker would have a list of the public certificates for each of the management domains it trusts. It could then verify the attribute and allow or reject the connection. Using this method, only endorsed services could publish messages and the service discovery problem would be avoided. The collection and management of domain certificates would be an infrastructure operations issue. However, as the impact of publishing information about services that should not be in the infrastructure is low, this mechanism will not be implemented in the prototype. To ensure that a message did originate from the service it should be signed by that service. To verify this, it is necessary to have the certificate (an attribute of the service) for the service, which again leads to the service discovery problem. In our prototype we will implement SSL connections to restrict misuse within the community. Message signing and verification will not be implemented at this stage but is something that could be considered for the future.

6.3 Evaluation

This evaluation benchmarks prototypes of the two approaches described in Section 6.2. The results are then compared to the benchmarking in Section 4.3 of the existing Grid information system used in the WLCG infrastructure to understand if there is any improvement.

In both approaches an LDAP server is used to provide the query interface for the information service. This has been done to ensure backwards compatibility for the clients of the existing system. As the Grid infrastructure supports many different user communities, each with their own priorities and timelines, any change to the fundamental interfaces of the infrastructure requires a migration strategy and will take time to implement. Therefore, the fundamental interfaces should only be changed if there is a major advantage that justifies the cost of the migration. As the query interface is LDAP, the query response time is essentially the query response time of an LDAP server populated with the Grid information. This has already been thoroughly evaluated in Chapter 4, so will not be repeated here.

Querying Services Directly

Each additional caching layer in a hierarchical topology increases the value of β and hence will decrease the probability α that the information is current. Therefore, the first improvement that will be evaluated is querying the services directly rather than querying the site-level information service. The evaluation will be performed against the WLCG production infrastructure as this is real operational environment for the proposed architecture. Although this presents additional difficulties due to the dynamic nature of the infrastructure (loaded machines, networking issues, incorrect configurations etc.) it is difficult to simulate these unexpected events and their inclusion is necessary to gain a real understanding of production operations. As the OSG [10] does not offer either site-level or service-level interfaces, the subsequent tests do not include resources from this infrastructure.

A fundamental component of WLCG is the network infrastructure that provides the connectivity between the Grid services. In order to understand the network performance, the Linux *ping* command was used to measure the round-trip time between a Virtual Machine at CERN and each site-level BDII. As each site publishes its location, the distance between the two geographical locations can be calculated from their longitudes and latitudes. The comparison of the average round-trip time for 64B and physical distance from CERN is shown in Figure 6.5.

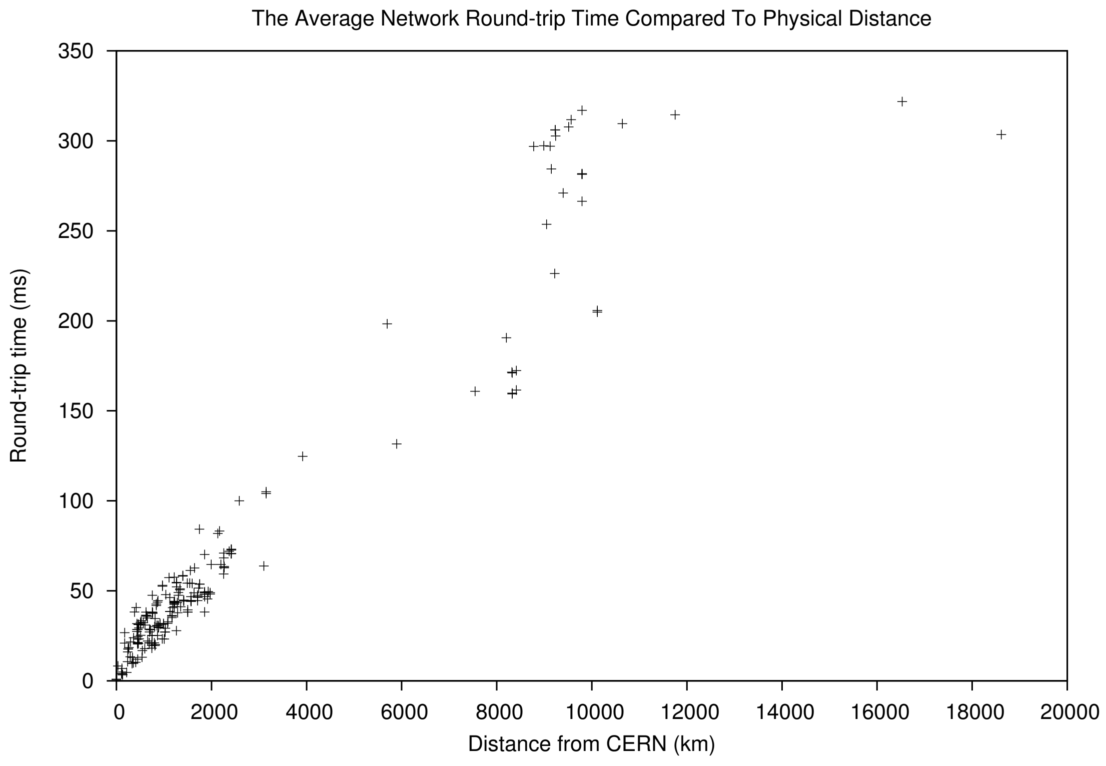

Figure 6.5: The average network round-trip time compared to physical distance from CERN (variance suppressed for clarity)

As expected, it can be seen that greater physical distance corresponds to higher round-trip times. The average round-trip time for the LAN (to the CERN site-level BDII) is less than 1ms. For site-level BDIIs within the same country (Switzerland) or physically close (less than 200km), the average round-trip time was less than 10ms, and within Europe the average round-trip time was less than 100ms. Average round-trip times greater than 100ms correspond to site-level BDIIs in North America, South America and the Asia-Pacific region. The site-level BDII for The University of Melbourne in Australia had the highest average round-trip time, 321.8ms with σ 1.7. At a physical distance of 16,500 km from CERN, it is the second furthest after The University of Auckland in New Zealand, which is 18,600 km away and had the ninth highest round-trip time of 303.5ms with σ 0.4. The University of Melbourne is therefore a good site to select for evaluating the effects of high network latency.

Before evaluating the direct querying of services, the current method of querying the site-level BDIIs must be understood. Each site-level BDII was queried for all objects 100 times to measure the average query response time. The result is shown in Figure 6.6, which compares the average query response time to the average round-trip time.

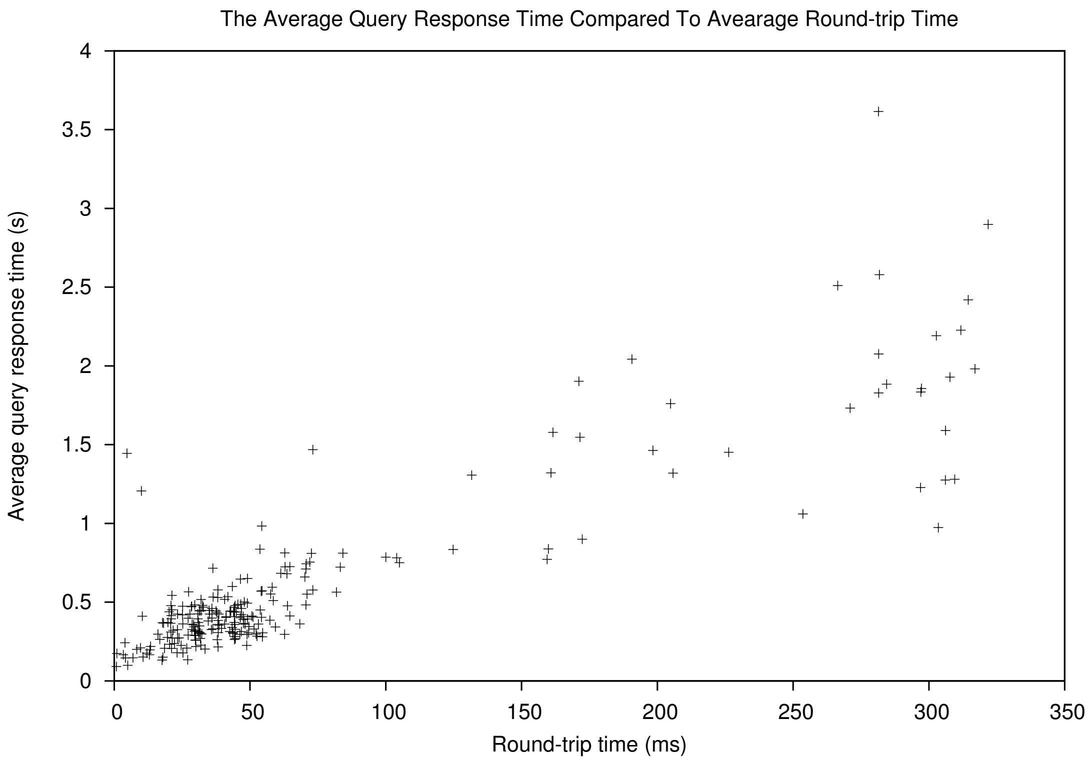

Figure 6.6: The average query response time for all objects compared to the average round-trip time (variance suppressed for clarity)

Again, as expected it can be seen that in general, higher query response times correspond to higher round-trip times, however it also suggests that other factors (the size of the query result, machine load, link quality etc.) may influence the result. For most site-level BDIIs within Europe (a round-trip time $<$ 100ms), the query response time is less than 1s. The maximum query response time for a successful query (one that did not timeout), was 3.6s with σ 0.96 for Academia Sinica in Taipei. The University of Melbourne (the reference site) was the second slowest, responding in 2.9s with σ 0.24.

Ignoring the 7 sites that either failed to respond or timed out, the time to query serially the 318 site-level BDIIs would be 185.76s with σ 0.27. This time can be reduced by using parallel queries and is the mechanism currently in use today. There are two main tuning parameters for parallel queries; the maximum number of queries to run concurrently and the timeout value for each query. These parameters are currently set as fifteen maximum concurrence queries and timeout value of 30s. The measurements of the average query response time suggest that the timeout value is conservative. To investigate the optimal settings for the experiment set-up, all sites were queried using different values for the maximum number of concurrent queries and timeout values. Two different measurements were made. One is the total time t that includes the queries that timed-out. The other is the query time q, which is that time until the last successful query finishes. The results are shown in Figure 6.7, which aims to show the point of inflection and hence the optimal number of concurrent queries for the specific scenario.

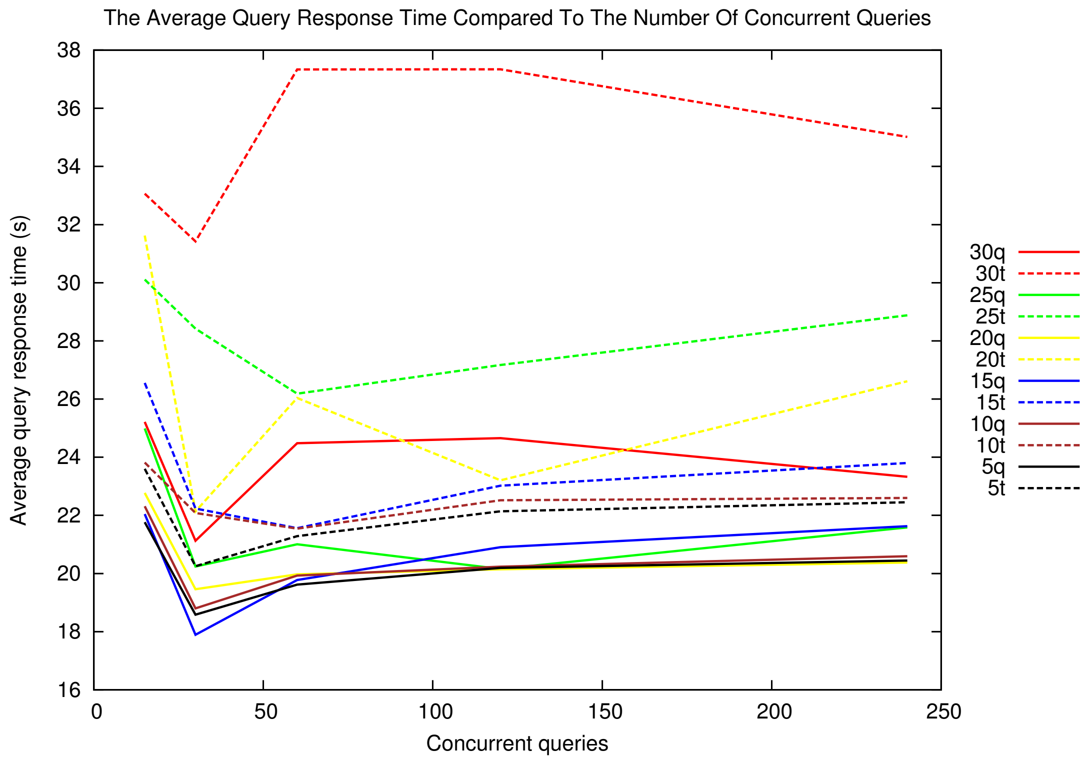

Figure 6.7: The query response time compared to the maximum number of concurrent queries for different timeout values

It can be seen from the results that decreasing the timeout reduces the total query time t closer to the the query time q. For most timeout values, 2-3 queries timed out, however with short timeout values (5 and 10 seconds), 4-5 queries timed out. It also seems that having a maximum of around 30 concurrent queries is optimal for this specific case. With a 5s timeout value and a maximum of 30 concurrent queries, the total time t was 20.25s with σ 0.21 and the query time q was 18.58s with σ 0.24 and an average of 5.2 timeouts with σ 0.62. With a 10s timeout and a maximum of 30 concurrent queries, the total time t was 22.08s with σ 0.22 and the query time q was 18.80s with σ 0.37 and an average of 4.0 timeouts with σ 1.56.

With the release of the EMI Resource Information Service (ERIS) [105] from the EMI project, querying services directly is now officially supported in the WLCG. The discovery part of the method used above was modified to obtain the host names for each service rather than the LDAP URL for the site-level BDII. An LDAP URL was constructed from the hostname using the standard port (2170) and the standard bind point (mds-vo-name=resource,o=grid) for the ERIS. The maximum information returned by one service was 407KB and the minimum was 72B, which represents the base entry. The median size of information returned from all services was 7.4KB with the total information from all services being 40.7MB.

The time taken to query each service was measured and the mean query response time was found to be 0.68s with σ 0.85. If the queries were executed in series, it would take 903 seconds to obtain the information from all services. The average query response time was measured using different values for the maximum number of concurrent queries and with a fixed timeout value of 5s. It can be seen from the results shown in Figure 6.8 that using a maximum of 20 concurrent queries returns the best results with the information from all services being obtained in 61.6s with σ 1.1.

To understand if there is an advantage gained by querying for only one attribute, the same test was repeated but using a query that only returned one attribute from the base entry (55B). The results can also be seen in Figure 6.8. Again using a maximum of 20 concurrent queries was gives the best result with an average total query time of 59.8s with σ 0.9. It is clear from this result that there is only a small advantage gained by querying for only one attribute. This suggests that the connection overhead for the query is much greater than returning the small amount of data (7.4KB on average).

As the data returned from the base entry is similar to the information used by the ping command (55B and 64B respectively), the averaged query response time can be compared to the average round-trip time. It can be seen in Figure 6.9 that the average round-trip time does not give an accurate indication of the average query response time, which suggests that the performance of the ERIS is an important factor. There is however a clear trend for the minimum query response time whereby there is no query that is less than 3 times the round-trip time and the medium is 5.6 times.

From the above results we can conclude the following.

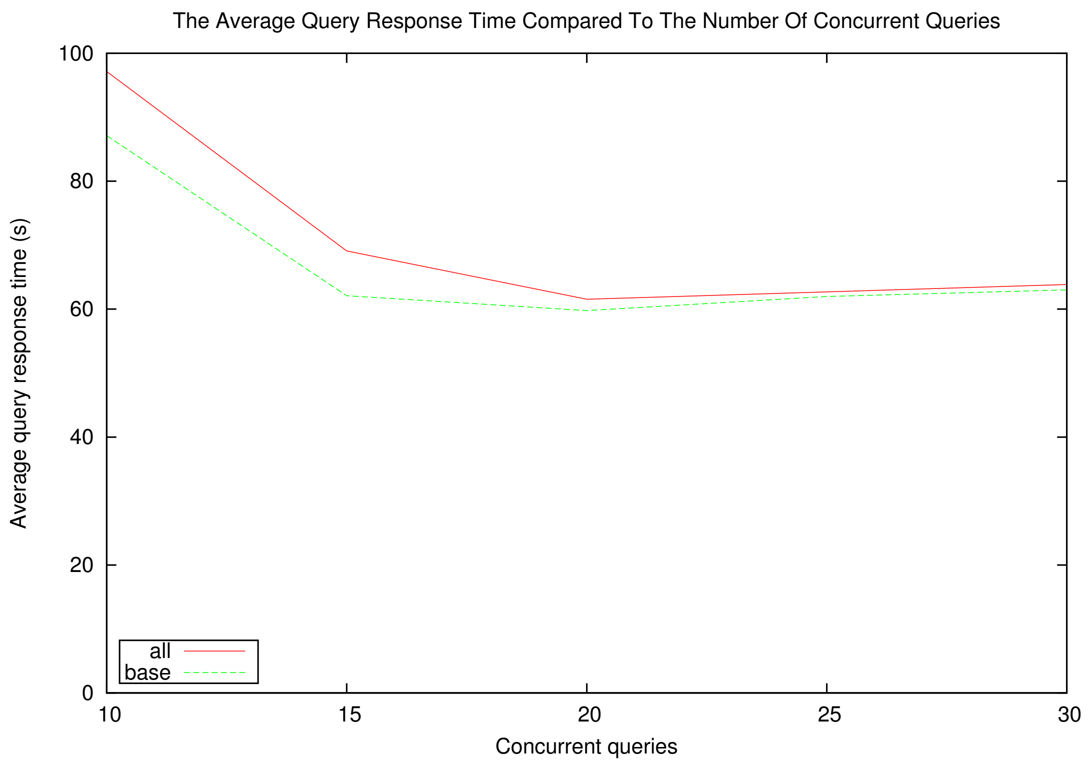

Figure 6.8: The average total query response time compared to the maximum number of concurrent queries

- Although querying all services directly takes more time than querying the site-level BDIIs, it would still be an acceptable approach for mutable information as 60s is much less than the defined grace period β of 1 hour.
- The overhead for obtaining all information is so small that unless network usage is an issue (7.4KB per service on average verses 55B for one attribute), the information from a service should be considered as an atomic unit rather than handling each attribute individually.
- The query response time is lower if more data is obtained from fewer sources.

Frequency of Updates

As has just been demonstrated, obtaining information directly from the services is an acceptable approach. The next question is with what frequency should those services be re-checked for any changes? Each service describes itself using the Service object type. In addition there are two specific services types (the Computing Service and

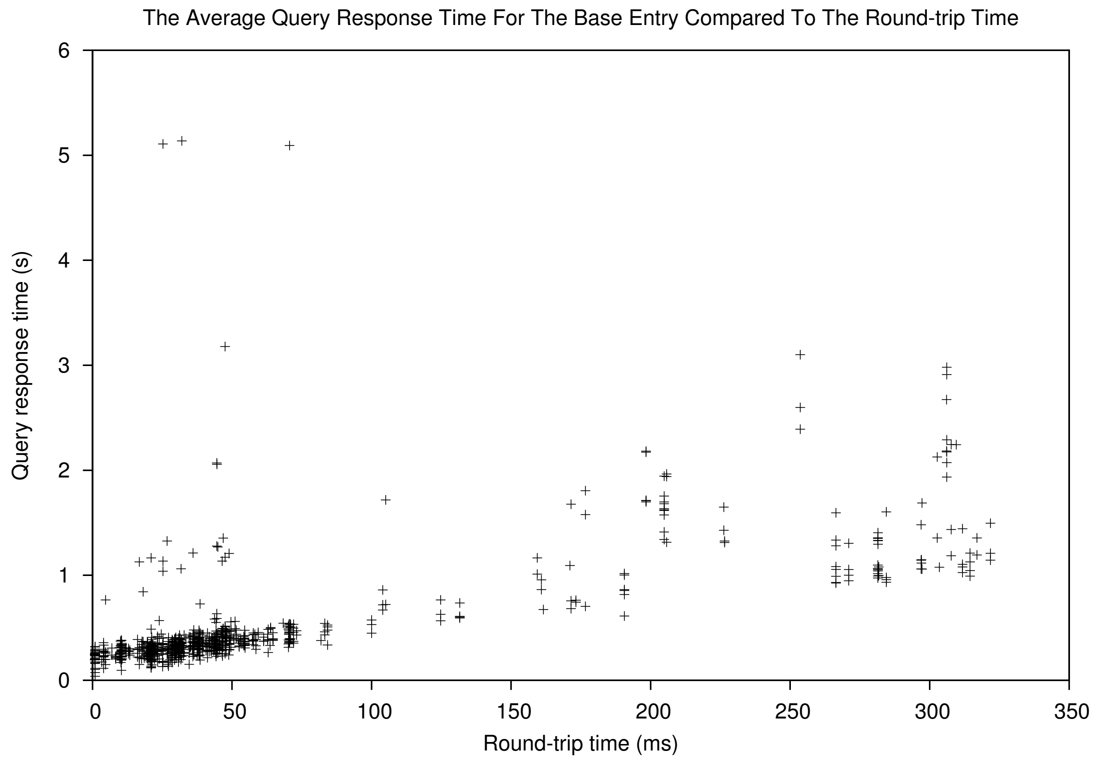

Figure 6.9: The average query response time for the base entry compared to the ping time (variance suppressed for clarity)

Storage Service), which provide more detailed information about those respective service types. As all information will be obtained, the frequency used will be the one for the object type that is required to be checked most frequently. Table 6.1 shows the re-calculation for the frequencies of change, ignoring the dynamic attributes and using the same daily snapshots of the information system that were recorded during the month of June 2011.

Table 6.2 shows the value of α for each object type immediately after the information has been retrieved from the service and the optimal value for β to ensure that $\alpha > 0.999$.

From the results in Table 6.2, a computer service should be queried every 1.36 hours, a storage service every 1.48 hours and any other service, every 3 hours.

Object	Total	Modifications	frequency (s^{-1})	$\lambda(s^{-1})$
CE	5081	88	1.02×10^{-3}	2.01×10^{-7}
CESEBind	9323	0.28	3.2×10^{-6}	3.5×10^{-10}
CESEBindGroup	5205	2.2	2.6×10^{-5}	5.0×10^{-9}
Cluster	667	3.2	3.8×10^{-5}	5.6×10^{-8}
Location	104188	459	5.3×10^{-3}	5.1×10^{-8}
SA	2957	32	3.7×10^{-4}	1.2×10^{-7}
SE	499	8.0	9.2×10^{-5}	1.8×10^{-7}
SEAccessProtocol	2334	0.41	4.7×10^{-6}	2.0×10^{-9}
SEControlProtocol	728	0.07	8.1×10^{-7}	1.1×10^{-9}
Service	4150	33	3.8×10^{-4}	9.2×10^{-8}
ServiceData	15099	71	8.2×10^{-4}	5.4×10^{-8}
Site	369	1.0	1.2×10^{-5}	3.3×10^{-8}
SubCluster	755	5.2	6.0×10^{-5}	7.9×10^{-8}
VOInfo	6571	0.1	1.2×10^{-6}	1.8×10^{-10}
VOView	11081	2.6	2.9×10^{-5}	2.7×10^{-9}

Table 6.1: The frequencies of change ignoring the dynamic attributes

The Latency of Shipping Updates

For the dynamic attributes a data shipping approach based on enterprise messaging technology will be used to send the updates directly to the information service. In order to calculate the probability α that the information is current, the value of β needs to be measured. In this scenario the value of β is the latency between when the change occurred at the information source and when it was available in the information service. It is not possible on a reasonable time scale to deploy a test on each service in the WLCG Grid infrastructure to capture the updates of the information source. In addition, due to the granularity for the maximum resolution of the snapshots available from querying the services directly, there is an undercount in the number of changes. The updates will therefore be simulated and a specific test will measure the best- and worst-cases scenarios respectively.

To simulate the changes, the LDAP DNs (Unique Identifiers) for all objects were harvested from the WLCG Grid information system and grouped by object type. Using the information on changes from Section 3.2, the changes as a percent for each object type were described cumulatively as shown in Table 6.3.

A random number was generated between 1 and 100 which was mapped to an object type using the information in Table 6.3. A random LDAP DN was then selected

Object	α	$\beta_{0.999}$ (s)	$\beta_{0.999}$ (h)
CE	0.999988	4915	1.375
CESEBind	0.999999	2878200	799.5
CESEBindGroup	0.999999	200818	55.78
Cluster	0.999997	17738	4.927
Location	0.999997	19546	5.429
SA	0.999993	7962	2.212
SE	0.999989	5355	1.487
SEAccessProtocol	0.999999	492013	136.7
SEControlProtocol	0.999999	899456	249.8
Service	0.999995	10820	3.006
ServiceData	0.999997	18430	5.119
Site	0.999998	29712	8.253
SubCluster	0.999995	12473	3.465
VOInfo	0.999999	5679710	1578
VOView	0.999999	369766	102.7

Table 6.2: The optimal refresh period to ensure $\alpha > 0.999$

Object	Cumulative %
VOView	44.65
CE	86.27
SA	94.79
Service	98.19
SE	99.41
SubCluster	100

Table 6.3: Cumulative changes as a percent per object type

for that object type and an update message is constructed. The update message consisted of the LDAP DN, the relevant dynamic attributes for that object type and the values, which were set to the current time. The average size of the message generated for each object type is shown in Table 6.4. The message was then sent to a topic on a messaging broker hosted at CERN using a python client. A delay was added to enable the frequency of messages to match the expected frequency of changes.

To measure the latency, a message consumer was created that subscribes to the topic. This was run on a machine that has the time synchronized with the messaging client. When a message is received, the values for the attributes are compared with the current time to obtain the latency for that message.

The first test evaluated the best-case LAN scenario with the message publisher hosted at CERN. The average round-trip time between the publisher and the broker for 64B of data was 2.42ms with σ 1.9ms. The second test evaluated the worst-case WAN scenario with a message publisher at The University of Melbourne in Australia. The average round trip time between the publisher and the broker for 64B of data was 340ms with σ 1.1ms. The latencies for each object type are show in Table 6.4.

Object	Size (B)	β_{CERN} (ms)	$\beta_{Melbourne}$ (ms)
CE	562.6	2.60	220.10
SA	397.8	2.58	215.20
SE	278.2	2.13	218.27
Service	316.2	2.60	219.14
SubCluster	295.1	2.77	220.56
VOView	375.6	2.37	217.63

Table 6.4: Average message latency

Calculation of the (α, β)-currency

Table 6.5 shows the re-calculation for the frequencies of change considering the dynamic attributes only, using the same daily snapshots of the information system that were recorded during the month of June 2011. Due to the under-counting problem, the probability p that a modification occurs was used to calculate the actual number of changes.

Object	Total	Modifications	p	Changes	f (s^{-1})
CE	5081	3964	0.78	18024	0.2086
SA	2957	1012	0.34	1540	0.0178
SE	499	209	0.42	361	0.004174
Service	4150	989	0.24	1299	0.01503
SubCluster	755	210	0.28	292	0.003376
VOView	11081	7942	0.72	28033	0.3245

Table 6.5: Frequencies of change for dynamic attributes

The frequencies of change from Table 6.5 were then used along with the values for β from Table 6.4 to calculate the values for α, with the result shown in Table 6.6.

It can be seen that for the worst-case scenario (the CE object type published from The University of Melbourne) α has value greater than 0.99999.

Object	$\lambda(s^{-1})$	α_{CERN}	$alpha_{Melbourne}$
CE	4.1×10^{-5}	0.9999999	0.9999910
SA	6.0×10^{-6}	0.9999999	0.9999987
SE	8.4×10^{-6}	0.9999999	0.9999982
Service	3.6×10^{-6}	0.9999999	0.9999992
SubCluster	4.5×10^{-6}	0.9999999	0.9999990
VOView	2.9×10^{-5}	0.9999999	0.9999936

Table 6.6: α for the local and remote scenarios

6.4 Concluding Remarks

This chapter highlighted the importance of the catalogue in the form of a service registry as a prerequisite for any Grid information system. The key aspect of a service registry is the management policies that it implements and hence can be abstracted as a policy engine. It is used to discover the information sources to which queries can be sent or from which to accept information if using a messaging-based approach.

The current mechanism for obtaining information from the site-level BDIIs was investigated and compared to an approach querying the services directly. The two advantages of querying the services directly are that the information is current and the site-level BDII would not be needed, resulting in a reduced operational overhead. The disadvantages are that it takes three times longer to query and the information source would be exposed to unpredictable query loads that may overload the service. It was found that although querying all services directly takes 40s longer than querying the site-level BDIIs, it would still be an acceptable approach for mutable information. As the overhead for obtaining all information is less than 1s, the information from a service should be considered as an atomic unit rather than handling each attribute individually.

The advantages of querying the site-level BDIIs are that the query response time is lower and they protect the resource-level BDIIs, which are typically co-hosted with the Grid service, from being overloaded with queries. This hierarchical approach essentially parallelises the query across multiple machines (318 site-level machines querying a few services each), with the top-level aggregating the result. Such an approach could also be done centrally with parallel queries being executed across multiple machines, however that is not the focus of this investigation.

Ignoring the dynamic information, the optimal frequency to re-check the service for changes was found for each service type. A computer service should be queried every 1.36 hours, a storage service every 1.48 hours and any other service, every 3 hours.

For the dynamic information, a simulation was used to evaluate the best- and worse-case scenarios, for shipping the changes from within a LAN and over a high-latency network connection respectively. It was found that that for the worst-case scenario (the CE object type published from The University of Melbourne) α has value greater than 0.99999.

As the number of messages sent for the dynamic information is much greater than the number of changes for mutable attributes by definition, using data shipping for mutable attributes in addition would also be possible.

Chapter 7

Conclusion

This chapter provides a brief summary of the problem investigated in this thesis along with the main proposal (Section 7.1), a review of the contributions (Section 7.2), and directions for future work (Section 7.3).

7.1 Problem and Thesis Summary

This thesis investigated the scalability of existing Grid information systems. That is their ability to respond to millions of queries per day regarding the structure and state of the thousands of Grid services that exist in a Grid infrastructure, and are the primary information sources for those queries. While the query response time for the existing Grid information system implementation used in the WLCG infrastructure is between 2.46s and 3.19s, depending which one of the top ten queries is considered, the time taken to update the information service where queries are resolved results in incorrect information (i.e. values which are no longer equal to the values at the information source) being returned. Existing approaches for evaluating Grid information systems focused on the query response time alone and assumed that the response was always correct i.e. the value returned by the query is identical to the real value at the information source. In doing so the accuracy of the information returned has been compromised, especially with respect to highly-dynamic state information, in order to improve the query response time.

This thesis addresses the problem by proposing the adoption of a data shipping approach that only transports the changes so as to reduce the latency between when a value changes at the information source, and when that change has been reflected in the information service.

Specifically, the thesis has proposed solutions to the Grid information system problem, by providing:

- a way to quantify the accuracy of the information returned by a Grid information system with respect to the expected frequency of change for that information;
- an improved benchmarking method that includes a quality metric for the accuracy of the information returned by a query;
- a study on the different possible approaches for processing querying in a Grid environment;
- a novel architecture, along with an implementation, for a Grid information system that improves on the existing system using the WLCG.

7.2 Review of Contributions

This thesis has enhanced the existing knowledge of Grid information systems by providing the following contributions.

- A review of the underlying problem addressed by Grid information systems:
  - Highlighted the importance of the information model, which mirrors the fundamental concepts of the Grid itself.
  - The identification of the Grid service as the primary information source for Grid information systems.
  - The identification of a catalogue in the form of a service registry for Grid information systems and Grid infrastructures in general.
- An investigation into the nature of Grid information that revealed for the first time the composition and frequencies of changes for information from a production Grid infrastructure.
- A demonstration of how the (α, β)-currency concept from the domain of Web search can be applied to Grid information.
- An improved benchmarking method for Grid information systems that includes (α, β)-currency as a quality metric.

- The identification of the different possible approaches for processing queries in a Grid environment:
  - An evaluation of the different approaches for processing queries in a Grid environment.
  - A demonstration that a data shipping approach which focuses on replicating information from the information source at the information service is the best approach when considering scalability, both in terms of the number of queries and number of information sources.
- The presentation of two related architectures that improve on existing approaches:
  - An incremental improvement to the existing system used in the WLCG.
  - An implementation based solely on messaging technology.

7.2.1 The Nature of Grid Information

An investigation into the nature of Grid information revealed that all information changes. At some point in time a Grid service joins the infrastructure, hence information is added, and it will eventually be deleted when the service is decommissioned. The concept of static and dynamic information, which has been widely used by the Grid community, was discussed. It was found that this concept is misleading as all objects types experience change; however, the frequency of change varies for each object type. By building upon the taxonomy of possible changes for Grid information, an alternative, more accurate description of information is provided. The term *static* is used to describe objects or attributes that only support the add or delete operation and hence are persistent for the lifetime of that service. The new term *mutable* is introduced to describe objects or attributes that support the update operation. The term *dynamic* is used to refer to the specific case of highly-mutable objects or attributes that change much more frequently and are mostly related to state information.

Specific conclusions relating to the information from the WLCG infrastructure revealed that information describing the main entities, the site and service objects, represents a small proportion, 4.17%, of the total information and the number of adds and deletes per day is quite low, less than 1%, in comparison to the modifications. This suggests that the structural information and hence Grid infrastructure itself does not experience frequent changes.

7.2.2 A Quality Metric for Grid Information

The use of (α,β)-currency as a quality metric for Grid information was evaluated. The values for the frequencies of change for attributes and objects types can be used to obtain a valve for λ in the (α,β)-currency equation. A sampling method was employed to measure the expected frequencies of change for the different attributes and objects types in the GLUE 1.3 information model using real information from the WLCG information system. It was found that for the dynamic object types, their frequency of change was such that a 1 minute sampling period was not sufficient to measure it with a reasonable precision. The surprising result is that the dynamic information is much more dynamic than was initially thought and this was not considered in early implementations of Grid information systems. There was also a considerable difference between the frequencies of change for dynamic and mutable information.

It was demonstrated how the concept of (α,β)-currency can be used to measure the probability that a randomly selected value from a snapshot is current after a given period of time and hence provide a quality metric Grid information. The unsurprising consequence of this is that snapshots of dynamic information decay quickly with respect to this metric. It was found that whereas snapshots of mutable information will decay within the order of days, snapshots of dynamic information will decay within the order of seconds.

7.2.3 A Benchmarking Methodology for Grid Information Systems

A comprehensive benchmarking methodology which incorporated the concept of (α,β)-currency as a quality metric for Grid information systems was proposed and includes additional constraints to make it relevant for today's production Grid infrastructures. Two Grid information system implementations that have had production exposure, the MDS GIIS and the BDII, were benchmarked using this methodology. It was found that in terms of the query response time, the BDII implementation offers better performance than the MDS GIIS. With a small response size (642 bytes) the query response times for the BDII and the MDS GIIS were 0.003s and 2.58s respectively. For a larger response size (8.6MB) the query response times were 0.41s and 3.19s respectively. It is not possible to provide a general statement on whether or not these result are acceptable as it depends on the individual use cases, however, for the WLCG infrastructure, these values are acceptable as they are less than the 15s timeout limit that has been adopted by the query clients.

The inclusion of the (α, β)-currency as a quality metric gave additional insight. Unsurprisingly, for mutable information, it is almost certain that a randomly selected value from the information system is current. However for dynamic information such as the CE, the probability that the information is current is 0.008 and 0.681 for the MDS GIIS and the BDII, respectively. This means that the MDS GIIS would mainly be returning values that were no longer current. When a query load was placed on the BDII, the cache update time increased to 1100s which reduced the probability that the information is current to 0.12. This means that the BDII would also mainly be returning values that were no longer current and hence both implementations, one of which is used in the WLCG infrastructure, are not suitable for handling dynamic information. It can clearly be seen that the inclusion of the (α, β)-currency into the benchmarking method provides additional insight over using the query response time alone.

As the measurement of (α, β)-currency is highly dependent on the information model, the value of α can therefore be improved by considering changes to the information model to eliminate dynamic information and such issues should be of consideration during the design of the information model. However, the impact of such changes on the use case needs to be understood and such an approach may not be possible if fine-grained dynamic information is required.

7.2.4 Processing Queries in a Grid Environment

The processing of queries in a Grid environment was considered with reference to the field of distributed query processing and in particular the approaches of query shipping and data shipping. Not surprisingly, with scalability demands of the Grid environment, the latencies in a global network, the number of information sources and queries, the query response time is lower for the data shipping approach than for the query shipping approach although the actual performance difference is implementation-specific. As a common query is the global section query, and due to scale there will be multiple queries within a period of time, the data shipping approach should be used. The data shipping approach should be based on either an asynchronous cache update process or replication in order to give a predictable query response time for all queries. As the MDS experience shows, the fault-in approach using a synchronous update mechanism will block queries while a query shipping approach is used to perform the update. However, it must be understood if the data shipping approach used does not decrease the quality of information returned with respect to the value of α in the (α, β)-currency metric. This suggests that the fundamental problem for processing queries in a Grid environment is in fact to replicate the information from the information source at the information service. In the existing Grid information system used in WLCG, this replication is a simple copy, which means that for some object types redundant data is being shipped, and for others data is not being shipped frequently enough resulting in a low value for α. The efficiency of this replication mechanism can be improved by introducing an active information source which identifies the changes and ships them to the information service. A simulation of this approach based on real information from the WLCG infrastructure showed that it would generate nearly 6 million changes a day (69 per second) and have a throughput of 7.6 GB a day (88KB per second) and hence is technically feasible due to the low data rate.

7.2.5 A Novel Approach

The importance of a catalogue in the form of a service registry as a prerequisite for any Grid information system was highlighted. The key aspect of a service registry is the management policies that it implements and hence can be abstracted as a policy engine. It is used to discover the information sources to which queries can be sent or from which to accept information if using a messaging-based approach.

Two related architectures were presented; one is an incremental improvement of the current implementation used in the WLCG infrastructure, and the other is a simplification that uses a data shipping approach which maintains a replication of the information source at the information service. Both make use of a data shipping approach that sends changes as soon as they occur at the information source to improve the currency of highly-mutable (dynamic) information.

The existing Grid information system implementation used in WLCG is adapted to make use of a service registry and queries the services directly. Ignoring the dynamic information, the optimal frequency to re-check the services for changes was found for each service type. As the overhead for obtaining all information is small, the information from a service should be considered as an atomic unit rather than handling each attribute individually. A computer service should be queried every 1.36 hours, a storage service every 1.48 hours and any other service every 3 hours. For WLCG there are essentially two service registries, the GOC DB and the OIM. Although they contain information about the existence of the Grid services, these records do not contain the URL required for contacting the information source for that service. The addition of this attribute, and indeed all static information that does not change throughout the lifetime of the service, could be considered. In the short term, the site-BDII can still be queried but the additional latency from using this extra caching layer needs to be deducted from the optimal frequencies. The end result is that the time taken to update the information service can increase to 1 hour while maintaining a value of $\alpha > 0.99999$. Assuming that the volume of information from each information source is constant, the update time is related to the number of information sources in the Grid infrastructure, and hence allowing a longer update time will increase scalability with respect to the number of information sources.

For the dynamic information, an experiment evaluated the best- and worse-case scenarios for shipping the changes from within a LAN and over a high-latency WAN respectively. It was found that for the worst-case scenario (the CE object type published from The University of Melbourne) α has value greater than 0.99999. In these scenarios the value for β represents the period between when a change at the information source is reflected in the information system and is dominated by the network latency.

As the number of messages sent for the dynamic information is much greater than the number of changes for mutable information by definition, shipping the mutable attributes in addition would also be possible. This would avoid the need for two different approaches and hence make the existing Grid information system implementation used in WLCG redundant. However, for the initial population and consistency-checking a heartbeat is needed to periodically publish all the information. An advantage with this approach is that new services would be discovered quickly, it would send a heartbeat immediately on starting, and prompt updates would be received about changes with mutable information.

7.3 Future Work

The thesis can be extended as follows.

- This thesis focused on the GLUE 1.3 information model as it was the model used in the WLCG production infrastructure at the time. With the development and deployment of the GLUE 2.0 information model, the investigation into the nature of Grid information in Chapter 3 should be repeated. As the concepts from the GLUE 1.3 information model can be mapped to the GLUE 2.0 information model, no surprising result is expected, however due to changes in the composition, the frequencies of change of the object types may differ. Similarly, the benchmarking method in Chapter 4 should also be updated with a new set of the top ten queries.

- Throughout this thesis it was assumed that the information at the information source was always correct. In a real Grid infrastructure, incorrect information occurs for many reasons including, misconfiguration, misunderstanding, software defects (bugs) and software errors (operational). In addition there are limitations on the accuracy, for example with calculating the number of running jobs in a cluster containing thousands of machines at any one moment. While checking the syntax of values is trivial, it is more difficult to verify that a sensible value (one that is possible) is in fact correct. Methods which verify the correctness of information at the information source are therefore required to ensure that the assumption is valid.

- The catalogue in terms of a service registry was identified as a fundamental component of a Grid infrastructure. More research is required to understand the policies that underpin Grid computing and how the concept of a policy engine can be realised to provide a service registry.

- To use a messaging-based approach in the WLCG production infrastructure, the security aspects relating to authenticated messages and the service registry need to be understood.

- This thesis focused on local queries to a single information service. With an infrastructure such as the WLCG, queries can originate from anywhere around the globe. Having a single information service may result in information being transferred unnecessarily between a remote client on another continent and the information service. An investigation is required to understand if there is an advantage to have multiple information services and if so, their location. Taking this further, the most aggressive use case is queries that originate from jobs in the computer cluster. It would therefore be useful to understand what information is required and if there is an advantage to ship the data to the cluster or host itself.

- This thesis proposed replicating the information at the information source at the information service and suggested a method based on heartbeats and shipping updates. Other methods for replicating could be investigated that may improve upon this.

- The Grid information system only provides information about services that originate from the service themselves. This may not be the only information that is required by the virtual organisation's experiment frameworks.

- The rise of cloud computing and as-a-service utility providers has the potential to provide additional resources to virtual organisations that use Grid infrastructures. Understanding how to integrate these metered services into the Grid information system would be an important step, not only to integrate those resources but also to provide seamless interoperability between the cloud computing providers themselves.

Bibliography

[1] M. Flechl and L. Field, “Grid interoperability: joining grid information systems,” *Journal of Physics: Conference Series*, vol. 119, no. 6, Jul. 2008. DOI:10.1088/1742-6596/119/6/062030

[2] R. Stevens, P. Woodward, T. DeFanti, and C. Catlett, “From the I-WAY to the National Technology Grid,” *Communications of the ACM*, vol. 40, no. 11, pp. 50–60, Nov. 1997. DOI:10.1145/265684.265692

[3] I. Foster, “The anatomy of the grid: enabling scalable virtual organizations.” IEEE Comput. Soc, pp. 6–7. DOI:10.1109/CCGRID.2001.923162

[4] M. Atkinson *et al.*, “Web Service Grids: an evolutionary approach,” *Concurrency and Computation: Practice and Experience*, vol. 17, no. 2-4, pp. 377–389, Feb. 2005. DOI:10.1002/cpe.936

[5] H. Stockinger, “Defining the grid: a snapshot on the current view,” *The Journal of Supercomputing*, vol. 42, no. 1, pp. 3–17, Mar. 2007. DOI:10.1007/s11227-006-0037-9

[6] L. Evans, “The Large Hadron Collider,” *New Journal of Physics*, vol. 9, no. 9, pp. 335–335, Sep. 2007. DOI:10.1088/1367-2630/9/9/335

[7] B. Segal, L. Robertson, F. Gagliardi, and F. Carminati, “Grid computing: the European DataGrid project,” in *IEEE Nuclear Science Symposium Conference Record*, vol. 1, Lyon, France, Oct. 2000. DOI:10.1109/NSSMIC.2000.948988

[8] F. Gagliardi, B. Jones, F. Grey, and M. Heikkurinen, “Building an infrastructure for scientific Grid computing: status and goals of the EGEE project,” *Philosophical Transactions. Series A, Mathematical, Physical, and Engineering Sciences*, vol. 363, no. 1833, pp. 1729–1742, Aug. 2005, PMID: 16099744. DOI:10.1098/rsta.2005.1603

[9] I. Bird, “Computing for the Large Hadron Collider,” *Annual Review of Nuclear and Particle Science*, vol. 61, no. 1, pp. 99–118, Nov. 2011. DOI:10.1146/annurev-nucl-102010-130059

[10] R. Pordes *et al.*, “The open science grid,” *Journal of Physics: Conference Series*, vol. 78, Jul. 2007. DOI:10.1088/1742-6596/78/1/012057

[11] C. Catlett, “The philosophy of TeraGrid: building an open, extensible, distributed tera-scale facility,” in *In the Proceedings of 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, Berlin, Germany, May 2002, pp. 8–8. DOI:10.1109/CCGRID.2002.1017101

[12] W. Gentzsch *et al.*, “DEISA: distributed european infrastructure for Supercomputing Applications,” *Journal of Grid Computing*, vol. 9, no. 2, pp. 259–277, Mar. 2011. DOI:10.1007/s10723-011-9183-2

[13] M. Ellert *et al.*, “The NorduGrid project: using Globus toolkit for building grid infrastructure,” in *Proceedings of the VIII International Workshop on Advanced Computing and Analysis Techniques in Physics Research*, vol. 502, Moscow, Jun. 2002, pp. 407–410. DOI:10.1016/S0168-9002(03)00453-4

[14] U. Schwiegelshohn *et al.*, “Perspectives on grid computing,” *Future Generation Computer Systems*, vol. 26, no. 8, pp. 1104–1115, Oct. 2010. DOI:10.1016/j.future.2010.05.010

[15] “The Ganglia Monitoring System.” [Online]. Available: http://ganglia.sourceforge.net

[16] “Nagios.” [Online]. Available: http://www.nagios.org

[17] I. Bird, B. Jones, and K. Kee, “The organization and management of Grid infrastructures,” *Computer*, vol. 42, no. 1, pp. 36–46, Jan. 2009. DOI:10.1109/MC.2009.28

[18] L. Field and R. Sakellariou, “How dynamic is the Grid? Towards a quality metric for Grid information systems,” in *The proceedings of the 11th IEEE/ACM International Conference on Grid Computing*, Brussels, Belgium, Oct. 2010, pp. 113–120. DOI:10.1109/GRID.2010.5697957

[19] ——, “Benchmarking Grid Information Systems,” in *The proceedings of the 17th International Conference on Parallel Processing*, vol. 6852, Bordeaux, France, Sep. 2011, pp. 479–490. DOI:0.1007/978-3-642-23400-2

[20] “Argonne Workshop Explores Construction of a National Computational Grid.” [Online]. Available: http://www.crpc.rice.edu/newsletters/fal97/news_grid.html

[21] I. Foster and C. Kesselman, Eds., *The grid: blueprint for a new computing infrastructure*. San Francisco: Morgan Kaufmann Publishers, 1999.

[22] L. Smarr and C. E. Catlett, “Metacomputing,” *Communications of the ACM*, vol. 35, no. 6, pp. 44–52, Jun. 1992. DOI:10.1145/129888.129890

[23] T. A. DeFanti *et al.*, “Overview of the I-Way: Wide-Area Visual Supercomputing,” *International Journal of High Performance Computing Applications*, vol. 10, no. 2-3, pp. 123–131, Jun. 1996. DOI:10.1177/109434209601000201

[24] I. Foster and C. Kesselman, “Globus: a Metacomputing Infrastructure Toolkit,” *International Journal of High Performance Computing Applications*, vol. 11, no. 2, pp. 115–128, Jun. 1997. DOI:10.1177/109434209701100205

[25] I. Foster *et al.*, “Software infrastructure for the I-WAY high-performance distributed computing experiment,” in *Proceedings of 5th IEEE International Symposium on High Performance Distributed Computing*, Syracuse, NY, USA, Aug. 1996, pp. 562–571. DOI:10.1109/HPDC.1996.546227

[26] I. Foster and C. Kesselman, “Computational Grids,” in *Vector and Parallel Processing*, vol. 1981. Portugal: Springer Berlin Heidelberg, 2001, pp. 3–37. DOI:10.1007/3-540-44942-6

[27] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke, “A security architecture for computational grids,” in *Proceedings of the 5th ACM conference on Computer and communications security*. San Francisco, CA, USA: ACM Press, Nov. 1998, pp. 83–92. DOI:10.1145/288090.288111

[28] B. Morris, “Realizing the Information Future: The Internet and Beyond,” *Prometheus*, vol. 14, no. 2, pp. 308–311, Oct. 2008. DOI:10.1080/08109029608629240

[29] I. Foster, C. Kesselman, J. Nick, and S. Tuecke, “Grid services for distributed system integration,” *Computer*, vol. 35, no. 6, pp. 37–46, Jun. 2002. DOI:10.1109/MC.2002.1009167

[30] F. Berman, G. C. Fox, and A. J. G. Hey, Eds., *Grid computing: making the global infrastructure a reality*. New York: J. Wiley, 2003.

[31] LHC Experiments Committee, “LHC computing Grid : Technical Design Report,” CERN, Tech. Rep. LCG-TDR-001, Jun. 2005. [Online]. Available: http://cds.cern.ch/record/840543/

[32] T. Critchlow and K. K. Van Dam, Eds., *Data-intensive science*. Boca Raton: CRC Press, Taylor & Francis Group, 2013.

[33] O. Martin *et al.*, “The DataTAG transatlantic testbed,” *Future Generation Computer Systems*, vol. 21, no. 4, pp. 443–456, Apr. 2005. DOI:10.1016/j.future.2004.10.011

[34] Richard P. Mount, “US grid projects: PPDG and iVDGL,” in *In the Proceedings of 2001 Conference for Computing in High-Energy and Nuclear Physics*, Beijing,China, Sep. 2001.

[35] F. Gagliardi, B. Jones, M. Reale, and S. Burke, “European DataGrid project: experiences of deploying a large scale testbed for e-science applications,” in *Performance Evaluation of Complex Systems: Techniques and Tools*. Berlin, Heidelberg: Springer Berlin Heidelberg, vol. 2459, pp. 480–499.

[36] “The International Grid Trust Federation.” [Online]. Available: http://www.igtf.net

[37] “LHC switches off for two-year break.” [Online]. Available: http://www.bbc.co.uk/news/science-environment-21421460

[38] The ATLAS Collaboration, “A particle consistent with the Higgs boson observed with the ATLAS detector at the Large Hadron Collider,” *Science*, vol. 338, no. 6114, pp. 1576—1582, Dec. 2012. DOI:10.1126/science.1232005

[39] The CMS Collaboration, “Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC,” *Physics Letters B*, vol. 716, no. 1, pp. 30—61, Sep. 2012. DOI:10.1016/j.physletb.2012.08.021

[40] J. Mocicki *et al.*, “Ganga: a tool for computational-task management and easy access to Grid resources,” *Computer Physics Communications*, vol. 180, no. 11, pp. 2303–2316, Nov. 2009. DOI:10.1016/j.cpc.2009.06.016

[41] Morris Riedel, “Interoperation of world-wide production e-Science infrastructures,” *Concurrency and Computation: Practice and Experience*, vol. 21, no. 8, pp. 961–990, 2009. DOI:10.1002/cpe.v21:8

[42] I. Foster, Y. Zhao, I. Raicu, and S. Lu, “Cloud computing and grid computing 360-degree compared,” in *Proceedings of the 2008 Grid Computing Environments Workshop*, Austin, Texas, USA, Nov. 2008, pp. 1–10. DOI:10.1109/GCE.2008.4738445

[43] W. Vogels, “Beyond server consolidation,” *Queue*, vol. 6, no. 1, p. 20, Jan. 2008. DOI:10.1145/1348583.1348590

[44] N. Loutas, E. Kamateri, F. Bosi, and K. Tarabanis, “Cloud computing interoperability: the state of play,” in *Proceedings of the Third International Conference on Cloud Computing Technology and Science*, Athens, Greece, Nov. 2011, pp. 752–757. DOI:10.1109/CloudCom.2011.116

[45] L. Field, E. Laure, and M. W. Schulz, “Grid deployment experiences: grid interoperation,” *Journal of Grid Computing*, vol. 7, no. 3, pp. 287–296, Aug. 2009. DOI:10.1007/s10723-009-9128-1

[46] “VDT Software Distribution.” [Online]. Available: http://vdt.cs.wisc.edu

[47] I. Foster *et al.*, “The grid2003 production grid: principles and practice,” in *Proceedings of the 13th IEEE International Symposium on High performance Distributed Computing*, Jun. 2004, pp. 236–245. DOI:10.1109/HPDC.2004.1323544

[48] F. Donno *et al.*, “Storage resource manager version 2.2: design, implementation, and testing experience,” *Journal of Physics: Conference Series*, vol. 119, no. 6, Jul. 2008. DOI:10.1088/1742-6596/119/6/062028

[49] S. Matsuoka *et al.*, “Japanese computational Grid research project: NAREGI,” *Proceedings of the IEEE*, vol. 93, no. 3, pp. 522–533, Mar. 2005. DOI:10.1109/JPROC.2004.842748

[50] C. Zheng *et al.*, “The PRAGMA testbed - building a multi-application international Grid,” in *Proceedings of the Sixth International Symposium on Cluster Computing and the Grid*, Singapore, May 2006, pp. 57–57. DOI:10.1109/CCGRID.2006.1630948

[51] N. Geddes, “The national Grid service of the UK,” in *Proceedings of the Second IEEE International Conference on e-Science and Grid Computing*, Amsterdam, The Netherlands, Dec. 2006, pp. 94–94. DOI:10.1109/E-SCIENCE.2006.261178

[52] D. Bannon and B. Appelbe, “eResearch - paradigm shift or propaganda?” *Journal of Research and Practice in Information Technology*, vol. 39, no. 2, pp. 83–90, 2007.

[53] “Grid Computing ’Mappa Mundi’ Unveiled.” [Online]. Available: http://www.sciencedaily.com/releases/2006/11/061116084126.htm

[54] “The Common Information Model.” [Online]. Available: http://dmtf.org/standards/cim

[55] “The Distributed Management Task Force.” [Online]. Available: http://dmtf.org

[56] D. Thain, T. Tannenbaum, and M. Livny, “Distributed computing in practice: the Condor experience,” *Concurrency and Computation: Practice and Experience*, vol. 17, no. 2-4, pp. 323–356, 2005. DOI:10.1002/cpe.938

[57] J. J. Ordille and B. P. Miller, “Database challenges in global information systems,” in *Proceedings of the 193 ACM SIGMOD international conference on Management of data*, Washington, D.C., United States, 1993, pp. 403–407. DOI:10.1145/170035.170099

[58] R. Baeza-Yates and B. Ribeiro-Neto, *Modern information retrieval.* Harlow, England: Addison Wesley, 1999.

[59] S. Brin and L. Page, “The anatomy of a large-scale hypertextual Web search engine,” *Computer Networks and ISDN Systems*, vol. 30, no. 1-7, pp. 107–117, Apr. 1998. DOI:10.1016/S0169-7552(98)00110-X

[60] F. Ehm, L. Field, and M. W. Schulz, “Scalability and performance analysis of the EGEE information system,” *Journal of Physics: Conference Series*, vol. 119, no. 6, Jul. 2008. DOI:10.1088/1742-6596/119/6/062029

[61] O. McBryan, "GENVL and WWWW: Tools for taming the Web," *Computer Networks and ISDN Systems*, vol. 27, no. 2, p. 308, Nov. 1994. DOI:10.1016/S0169-7552(94)90149-X

[62] D. Eichmann, "The RBSE spider: balancing effective search against Web load," *Computer Networks and ISDN Systems*, vol. 27, no. 2, p. 308, Nov. 1994. DOI:10.1016/S0169-7552(94)90151-1

[63] G. Mathieu *et al.*, "GOCDB, a topology repository for a worldwide grid infrastructure," *Journal of Physics: Conference Series*, vol. 219, no. 6, p. 062021, Apr. 2010. DOI:10.1088/1742-6596/219/6/062021

[64] "European Middleware Initiative Registry, a solution for federated online services." [Online]. Available: http://www.isgtw.org/announcement/european-middleware-initiative-registry-solution-federated-online-services

[65] S. Fitzgerald *et al.*, "A directory service for configuring high-performance distributed computations," in *Proceedings of the Sixth IEEE International Symposium on High Performance Distributed Computing*, Portland, OR, USA, pp. 365–375. DOI:10.1109/HPDC.1997.626445

[66] M. J. Franklin, B. T. Jnsson, and D. Kossmann, "Performance tradeoffs for client-server query processing," in *Proceedings of the 1996 ACM SIGMOD international conference on Management of data*, vol. 25, Montreal, Quebec, Canada, Jun. 1996, pp. 149–160. DOI:10.1145/235968.233328

[67] O. Wolfson, S. Jajodia, and Y. Huang, "An adaptive data replication algorithm," *ACM Transactions on Database Systems*, vol. 22, no. 2, pp. 255–314, Jun. 1997. DOI:10.1145/249978.249982

[68] "Resource ReSerVation Protocol (RSVP)," Sep. 1997. [Online]. Available: http://www.rfc-editor.org/rfc/rfc2205.txt

[69] "Lightweight Directory Access Protocol," Mar. 2003. [Online]. Available: http://www.rfc-editor.org/rfc/rfc3494.txt

[70] "The LDAP Data Interchange Format," Jun. 2000. [Online]. Available: http://www.rfc-editor.org/rfc/rfc2849.txt

[71] C. Loomis, “Final evaluation of testbed operation,” EU Deliverable DataGrid-06-D6.8-414712-3-0, 2003. [Online]. Available: https://edms.cern.ch/file/414712/3.0/D6.8-3.0.pdf

[72] B. Konya *et al.*, “The NorduGrid architecture and middleware for scientific applications,” in *Proceedings of theInternational Conference on Computational Science*, vol. 2657/2003, Melbourne, Australia, Jun. 2003, p. 665. DOI:10.1007/3-540-44860-8

[73] L. Field and M. W. Schulz, “An investigation into the mutability of information in production grid information systems,” *Journal of Physics: Conference Series*, vol. 219, no. 6, Apr. 2010. DOI:10.1088/1742-6596/219/6/062046

[74] S. M. Fisher *et al.*, “The relational grid monitoring architecture: mediating information about the grid,” *Journal of Grid Computing*, vol. 2, no. 4, pp. 323–339, Apr. 2005. DOI:10.1007/s10723-005-0151-6

[75] B. Tierney *et al.*, “A grid monitoring architecture,” Open Grid Forum, Tech. Rep. GFD.7, 2002. [Online]. Available: http://www.ggf.org/documents/GFD.7.pdf

[76] T. Haerder and A. Reuter, “Principles of transaction-oriented database recovery,” *ACM Computing Surveys*, vol. 15, no. 4, pp. 287–317, Dec. 1983. DOI:10.1145/289.291

[77] Laurence Field and Markus Schulz, “Grid deployment experiences: evolution of the information and monitoring system,” in *Proceedings of the Conference for Computing in High-Energy and Nuclear Physics*, Mumbai, India, 2006.

[78] S. Fisher *et al.*, “Building a robust distributed system: some lessons from R-GMA,” *Journal of Physics: Conference Series*, vol. 119, no. 6, p. 062016, Jul. 2008. DOI:10.1088/1742-6596/119/6/062016

[79] J. M. Schopf *et al.*, “Monitoring the grid with the Globus Toolkit MDS4,” *Journal of Physics: Conference Series*, vol. 46, pp. 521–525, Sep. 2006. DOI:10.1088/1742-6596/46/1/072

[80] B. Plale *et al.*, “Understanding grid resource information management through a synthetic database benchmark/workload,” in *Proceedings of the 4th IEEE/ACM International Symposium on Cluster Computing and the Grid*, Chicago, IL, USA, Apr. 2004, pp. 277–284. DOI:10.1109/CCGrid.2004.1336578

[81] S. Wang *et al.*, "Developing the modular information provider (MIP) to support interoperable grid information services," in *Proceedings of the Fifth International Conference Grid and Cooperative Computing*, Hunan, China, Oct. 2006, pp. 448–453. DOI:10.1109/GCC.2006.38

[82] A. Anisenkov *et al.*, "AGIS: the ATLAS grid information system," *Journal of Physics: Conference Series*, vol. 396, no. 3, Dec. 2012. DOI:10.1088/1742-6596/396/3/032006

[83] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman, "Grid information services for distributed resource sharing," in *Proceedings 10th IEEE International Symposium on High Performance Distributed Computing*, San Francisco, CA, USA, pp. 181–194. DOI:10.1109/HPDC.2001.945188

[84] B. Brewington, "How dynamic is the Web?" *Computer Networks*, vol. 33, no. 1-6, pp. 257–276, Jun. 2000. DOI:10.1016/S1389-1286(00)00045-1

[85] "GLUE Schema version 1.3." [Online]. Available: http://glueschema.forge.cnaf.infn.it/Spec/V13

[86] C. Germain-Renaud *et al.*, "The grid observatory," in *Proceedings of the 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, Newport Beach, CA, USA, May 2011, pp. 114–123. DOI:10.1109/CCGrid.2011.68

[87] A. Papoulis, *Probability, random variables, and stochastic processes*, 4th ed. Boston, MA, USA: McGraw-Hill, 2007.

[88] Laurence Field and Markus W. Schulz, "Grid deployment experiences: The path to a production quality LDAP based grid information system," in *In the Proceedings of 2004 Conference for Computing in High-Energy and Nuclear Physics*, Interlaken, Switzerland, Oct. 2004, pp. 723–726.

[89] X. Zhang, J. Freschl, and J. Schopf, "A performance study of monitoring and information services for distributed systems," in *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing*, Seattle, WA, USA, Jun. 2003, pp. 270–281. DOI:10.1109/HPDC.2003.1210036

[90] X. Zhang, J. Freschl, and J. M. Schopf, "Scalability analysis of three monitoring and information systems: MDS2, R-GMA, and Hawkeye," *Journal*

of Parallel and Distributed Computing, vol. 67, no. 8, pp. 883–902, Aug. 2007. DOI:10.1016/j.jpdc.2007.03.006

[91] X. Zhang and J. Schopf, "Performance analysis of the Globus toolkit monitoring and discovery service, MDS2," in *IEEE International Conference on Performance, Computing, and Communications, 2004*, Phoenix, AZ, USA, 2004, pp. 843–849. DOI:10.1109/PCCC.2004.1395199

[92] S. Zanikolas and R. Sakellariou, "An importance-aware architecture for large-scale grid information services," *Parallel Processing Letters*, vol. 18, no. 03, p. 347, 2008. DOI:10.1142/S0129626408003442

[93] S. Paterson, R. Graciani, A. Tsaregorodtsev, and A. Casajus, "DIRAC pilot framework and the DIRAC workload management system," *Journal of Physics: Conference Series*, vol. 219, no. 6, p. 062049, Apr. 2010. DOI:10.1088/1742-6596/219/6/062049

[94] D. Kossmann, "The state of the art in distributed query processing," *ACM Computing Surveys*, vol. 32, no. 4, pp. 422–469, Dec. 2000. DOI:10.1145/371578.371598

[95] A. Tanenbaum, *Distributed systems.* Upper Saddle River: Pearson Prentice Hall, 2007.

[96] R. Alonso, D. Barbara, and H. Garcia-Molina, "Data caching issues in an information retrieval system," *ACM Transactions on Database Systems*, vol. 15, no. 3, pp. 359–384, Sep. 1990. DOI:10.1145/88636.87848

[97] D. Kossmann, M. J. Franklin, G. Drasch, and W. Ag, "Cache investment: integrating query optimization and distributed data placement," *ACM Transactions on Database Systems*, vol. 25, no. 4, pp. 517–558, Dec. 2000. DOI:10.1145/377674.377677

[98] M. Franklin and S. Zdonik, "Data in your face," *ACM SIGMOD Record*, vol. 27, no. 2, pp. 516–519, Jun. 1998. DOI:10.1145/276305.276360

[99] "Uniform Resource Locators (URL)," Dec. 1994. [Online]. Available: http://www.rfc-editor.org/rfc/rfc1738.txt

[100] J. Casey *et al.*, “A messaging infrastructure for WLCG,” *Journal of Physics: Conference Series*, vol. 331, no. 6, p. 062015, Dec. 2011. DOI:10.1088/1742-6596/331/6/062015

[101] D. Collados, J. Shade, S. Traylen, and E. Imamagic, “Evolution of SAM in an enhanced model for monitoring WLCG services,” *Journal of Physics: Conference Series*, vol. 219, no. 6, Apr. 2010. DOI:10.1088/1742-6596/219/6/062008

[102] L. Cons and M. Paladin, “The WLCG messaging service and its future,” *Journal of Physics: Conference Series*, vol. 396, no. 3, Dec. 2012. DOI:10.1088/1742-6596/396/3/032084

[103] “The Simple Text Oriented Messaging Protocol.” [Online]. Available: http://stomp.github.io

[104] R. Alfieri *et al.*, “VOMS, an authorization system for virtual organizations,” in *Grid Computing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, vol. 2970, pp. 33–40.

[105] C. Aiftimiei *et al.*, “Towards next generations of software for distributed infrastructures: the European middleware initiative,” in *Proceedings of the 8th International Conference on E-Science*, Chicago, IL, USA, Oct. 2012, pp. 1–10. DOI:10.1109/eScience.2012.6404415