Coordinated Resource Provisioning in Federated Grids

by

Rajiv Ranjan

Submitted in total fulfilment of
the requirements for the degree of

Doctor of Philosophy

Department of Computer Science and Software Engineering
The University of Melbourne, Australia

July 2007

Coordinated Resource Provisioning in Federated Grids

Rajiv Ranjan

Co-advisors: Dr. Aaron Harwood, A/Professor Rajkumar Buyya

Abstract

A fundamental problem in building large scale Grid resource sharing system is the need for efficient and scalable techniques for discovery and provisioning of resources for delivering expected Quality of Service (QoS) to users' applications. The current approaches to Grid resource sharing based on resource brokers are non-coordinated since these brokers make scheduling related decisions independent of the others in the system. Clearly, this worsens the load-sharing and utilisation problems of distributed Grid resources as sub-optimal schedules are likely to occur. Further, existing brokering systems rely on centralised information services for resource discovery. Centralised or hierarchical resource discovery systems are prone to single-point failure, lack scalability and fault-tolerance ability. In the centralised model, the network links leading to the server are very critical to the overall functionality of the system, as their failure might halt the entire distributed system operation.

In this thesis, I propose a new federated Grid system, called Grid-Federation that aims towards decentralised and coordinated coupling of distributed Grid resources as a part of single cooperative system. The Peer-to-Peer (P2P) network model forms the basis for the design of decentralised protocols for scheduling and resource discovery in the Grid-Federation. The decentralised organisation enhances the scalability and reliability of the system. Two levels of decentralised coordination is presented in this thesis: (i) a Service Level Agreement (SLA) based broker-to-broker coordination protocol that inhibits the brokers from over-provisioning the resources and also gives every site the admission control capability; and (ii) a P2P tuple space based coordination protocol that coordinates the scheduling process among the distributed resource brokers. The thesis demonstrates the effectiveness of the proposed Grid-Federation based decentralised protocols in coordinating scalable and robust resource management through extensive simulation studies.

Thesis Contributions: To support the thesis that Grid-Federation model along with its decentralised protocols for scheduling and coordination is the best possible way to implement new generation Grid resource sharing system I have:

- outlined key taxonomies for decentralised Grid resource sharing systems,
- proposed, modeled, and evaluated a decentralised resource sharing system called Grid-Federation, which aims toward policy based cooperative and coordinated coupling of distributed cluster resources,
- proposed, modeled, and evaluated an SLA based broker-to-broker service contract negotiation protocol,

- proposed, modeled, and evaluated the extension of Distributed Hash Tables (DHTs) such as Chord, Pastry using a spatial publish/subscribe index to support decentralised resource discovery protocols in the Grid-Federation,
- proposed, modeled, and evaluated a DHT-based tuple space for coordinating the application schedules among distributed brokers,
- designed and implemented the Alchemi-Federation software system, that supports a coordinated federation of Alchemi cluster over a DHT-based resource discovery system.

This is to certify that

(i) the thesis comprises only my original work,

(ii) due acknowledgement has been made in the text to all other material used,

(iii) the thesis is less than 100,000 words in length, exclusive of table, maps, bibliographies, appendices and footnotes.

Signature_____

Date_____

ACKNOWLEDGMENTS

I am heartily indebted to my thesis co-advisors, Dr. Aaron Harwood and A/Professor Rajkumar Buyya, for believing in my abilities, for their innovative ideas, timely suggestions, feedbacks and unflinching support throughout my Ph.D candidature. I feel highly privileged to have worked with my co-advisors, Aaron and Raj. To both of them I owe a great debt of gratitude for their patience, inspiration and friendship. Raj has taught me a great deal about Grid resource management and Grid Economics idea, which opened the doors to several exciting research areas explored in this thesis. Aaron, on the other hand, taught me peer-to-peer networks and distributed indexing that motivated me into exploring the domain of decentralised Grid system design.

I would also like to thank the people with whom I have collaborated over the course of this thesis including Lipo Chan. I sincerely appreciate Lipo's timely clarification of my doubts and questions about her publish/subscribe index on most occasions. I am also thankful to Jordi Pujol Ahull (developer of PlanetSim peer-to-peer simulator toolkit) and Jeff Hoye (developer of FreePastry toolkit) for their timely responses. A special mention of Carlos Alexandre Queiroz and Xingchen Chu for collaborating while implementing the software system Alchemi-Federation.

I thank Gridbus Lab group members including James Andrew Broberg, Srikumar Venugopal, Chee Sin Yeo, Jia Yu, Anthony Solisitio, Marcos Assuncao, Al-Mukaddim Khan Pathan, Pramod Kumar Konugurthi, Marco A. S. Netto, and Md Mustafizur Rahman for their excellent research advice. Regular feedbacks provided during weekly group meetings of GRIDS lab and P2P group have contributed significantly towards enhancement of the ideas presented in this thesis. Special thanks to Krishna Nandiminti and Hussein Gibbins for proof-reading my thesis work on many occasions.

I would also like to thank Adriana Iamnitchi (Assistant Professor, Computer Science and Engineering, University of South Florida), David Spence (CCLRC Rutherford Appleton Laboratory, e-Science Center, Oxford, UK), Jon Crowcroft (Professor of Communications Systems in the Computer Lab, at the University of Cambridge), and Manish Parashar (Professor of Electrical and Computer Engineering, Rutgers University) for their valuable feedbacks on some of the works presented in this thesis.

I wish to thank all the people whose works have been utilized for developing and implementing ideas outlined in the thesis. I am also grateful to the DEST (Department of Education, Science and Training, Australian Government) and University of Melbourne for funding my Ph.D candidature. This work was also partially supported through Australian Research Council discovery project grant.

I would like to thank Professor Rao Kotagiri and Professor Alistair Moffat for being my Ph.D committee member and chairperson of the examination committee respectively. I would like to thank administrative staff members in the Computer Science and Software Engineering Department including Binh Phan, Pinoo Bharucha, and Julien Reid for their support and help during last four years. I am thankful to the School of Graduate Stud-

ies and Faculty of Engineering for providing an extraordinary Ph.D research program at the University. I would also like to thank the anonymous thesis reviewers for their constructive comments that has helped me in improving the quality of the final manuscript substantially.

This thesis would never have been possible if it were not for the support of family and friends, especially my father, BrajKishore Singh, mother, Lal Muni Devi, brother, Sanjiv Kumar and girl friend Dr. Tejal Shah. Special mention for Tejal who took the arduous step of proof-reading the drafts of this thesis. Finally, I would like to thank AmitKumar Ramavatar Varma for his friendship and support during the last year of my Ph.D candidature. Above all, I am indebted to the All Mighty for making my life meaningful even outside the confines of this Ph.D candidature.

Rajiv Ranjan
Melbourne, Australia
July 2007.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

This chapter introduces the context of the research themes explored in this thesis. It starts with a high-level overview of Grid computing and analyses the current system design approaches being practiced for Grid resource management and application scheduling. Then, it puts forward the fundamental motivations behind decentralised and coordinated organisation of Grid systems; including resource brokers and resource discovery systems. The chapter thereafter provides discussion on the thesis outline and contributions. It ends with a summary of the published materials that were partially or fully utilised for compiling the thesis.

1.1 Grid Computing

The last few years have seen the emergence of a new generation of distributed systems that scale over the Internet, operate under decentralised settings and are dynamic in their behavior, where participants can leave or join the system at any time. One such system is referred to as Grid Computing and other similar systems include P2P Computing [122], Semantic Web [127], Pervasive Computing [149] and Mobile Computing [17, 68]. Grid Computing [71] provides the basic infrastructure required for sharing diverse sets of resources including desktops, computational clusters, supercomputers, storage, data, sensors, applications and online scientific instruments. Grid Computing offers its vast computational power to solve highly challenging problems in science and engineering such as protein folding, high energy physics, financial modeling, earthquake simulation, cli-

mate/weather modeling, aircraft engine diagnostics, earthquake engineering, virtual observatory, bio-informatics, drug discovery, digital image analysis, astrophysics and multiplayer gaming.

The notion of Grid Computing goes well beyond the traditional Parallel and Distributed Computing Systems (PDCS) as it involves various resources that belong to different administrative domains and are controlled by domain specific resource management policies. Furthermore, grids in general have evolved around complex business and service models where various small sites (resource owners) collaborate for computational and economic benefits. The task of resource management and application scheduling over a grid is a complex undertaking due to resource heterogeneity, domain specific policies, dynamic environment, and various socio-economic and political factors.

Grids can be primarily classified [180] into various types, depending on the nature of their emphasis:- computation, data, application service, interaction, knowledge and utility. Accordingly, grids are proposed as the emerging cyber infrastructure to power utility computing applications. Computational grids aggregate computational power of globally distributed computers (e.g., TeraGrid, ChinaGrid, and APACGrid). Data grids emphasize on a global-scale management of data to provide data access, integration and processing through distributed data repositories (e.g. LHCGrid, GriPhyN). Application Service (provisioning) grids focus on providing access to remote applications, modules and libraries hosted on data centers or computational grids (e.g. NetSolve and GridSolve). Interaction grids focus on interaction and collaborative visualization between participants (e.g. AccessGrid). Knowledge grids aim towards knowledge acquisition, processing, management and provide business analytic services driven by integrated data mining services. Utility grids focus on providing all the Grid services including compute power, data and service to end users as IT utilities on subscription basis. Furthermore, they provide the infrastructure necessary for negotiation of required QoS and handle the establishment and management of contracts and allocation of resources to meet competing demands. To summarize, these grids follow a layered design with computational grid being the bottom most layer while the utility grid is the top most layer. A grid at a higher-level utilises the services of grids that operate at lower layers in the design. For example, a data grid utilises the services of computational grid for data processing and hence builds on it. In

addition, lower-level grids focus heavily on infrastructure aspects whereas higher-level ones focus on users and quality of service delivery.

In this work, we focus on designing efficient techniques of discovery and provisioning of resources in computational grids. Computational grids enable aggregation of different types of compute resources including clusters, supercomputers and desktops. In general, compute resources have two types of attributes: (i) static attributes such as the type of operating system installed, network bandwidth (both Local Area Network and Wide Area Network interconnection), processor speed and storage capacity (including physical and secondary memory); and (ii) dynamic attributes such as processor utilisation, physical memory utilisation, free secondary memory size, current usage price and network bandwidth utilization.

1.2 Project Motivation

The fundamental objective behind the emergence of Grid computing systems is to facilitate a coordinated resource and problem sharing [71] environment among collaborative administrative domains. Grid resource brokering or superscheduling [151] activity is defined as scheduling of jobs across resources that belong to distinct administrative domains. Brokering in computational grids is facilitated by specialised resource brokers such as NASA-Scheduler [152], Nimrod-G [3] and Condor-G [75]. The main challenges involved with Grid brokering include: (i) querying Grid resource information services (GRIS) [10, 33, 54, 97, 150] for locating resources that match the job requirements; (ii) coordinating and negotiating SLAs; and (iii) job scheduling. The Grid resources are managed by their respective Local Resource Management System (LRMS) such as Condor [113], PBS [26], SGE [84], Legion [39], Alchemi [117] and LSF [185]. The LRMS manages job queues, initiating and monitoring their execution. A key consideration involved with the Grid brokering is the *distributed ownership*. As a result, brokers do not have any control over the resources. Further, there is *incomplete system-wide state information* available at brokers, in particular about the arrival pattern, service pattern and composition of jobs across the system.

However, existing techniques to scheduling in a Grid environment are non-coordinated.

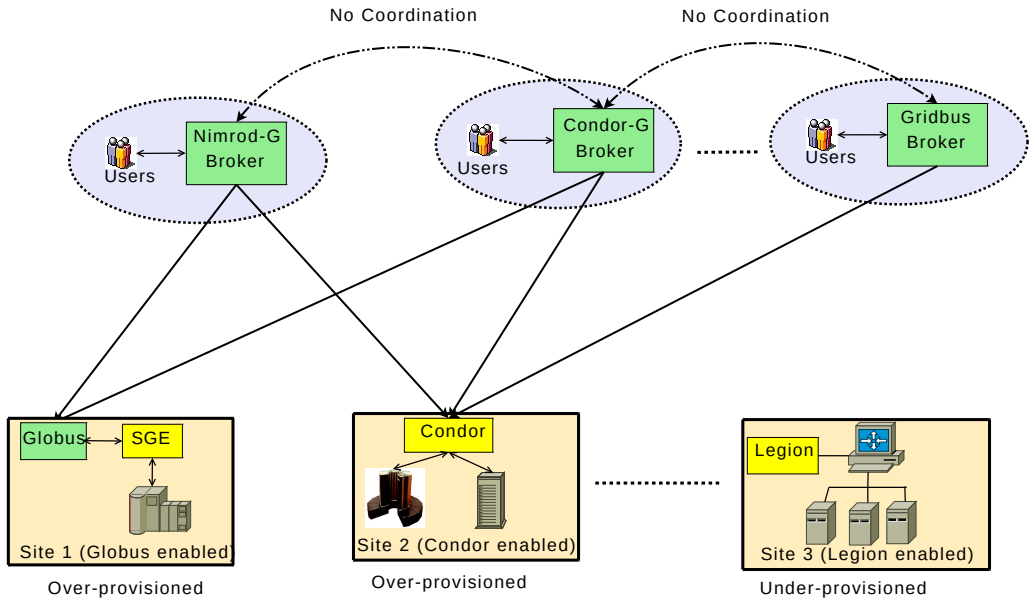


Figure 1.1: Non-Coordinated resource brokering in Grids.

In a non-coordinated system, application schedulers perform scheduling related activities independent of the other schedulers in the system. They directly submit their applications to the responsible LRMS managing the resources *without* taking into account the current load, priorities and utilisation scenarios of other application level schedulers. Clearly, this can lead to over-utilisation or bottleneck of some valuable resources while leaving others largely underutilised. Furthermore, these brokering systems do not have a coordination (or cooperative) mechanism and this exacerbates the load sharing and utilisation problems of distributed resources because sub-optimal schedules are likely to occur. Fig. 1.1 shows such a scenario in which the Nimrod-G, Condor-G and Gridbus [171] resource brokers are over-provisioning the resources at the Site-1 and Site-2 due to lack of coordination. At the same time, resources at Site-3 are left under-provisioned.

Another competing approach to the ad-hoc Grid resource assembling is the creation of a distributed Virtual Organisation (VO) [72] environment that includes scientific research groups working on collaborative scientific projects. Fig. 1.2 shows a VO based Grid architecture involving three research institutes. Membership to a specific VO is subject to the particular problem solving or project domain. An individual research institute can organise its own resources using a centralised broker which connects to the global VO wide broker. Various VO brokers connect in hierarchy to form a distributed scheduling

architecture. A VO user can submit jobs to either its centralised local resource broker or to the VO broker. This solution is fault-tolerant, but a hierarchic connection between VO's can impose serious performance limitations as the number of VOs increases. Further, there is no defined topology in which various institutes can connect with each other. The autonomy of an institute is dependent on the VO broker to which it connects. Any VO specific job migration and resource allocation admission control decision is also taken care of by the VO broker.

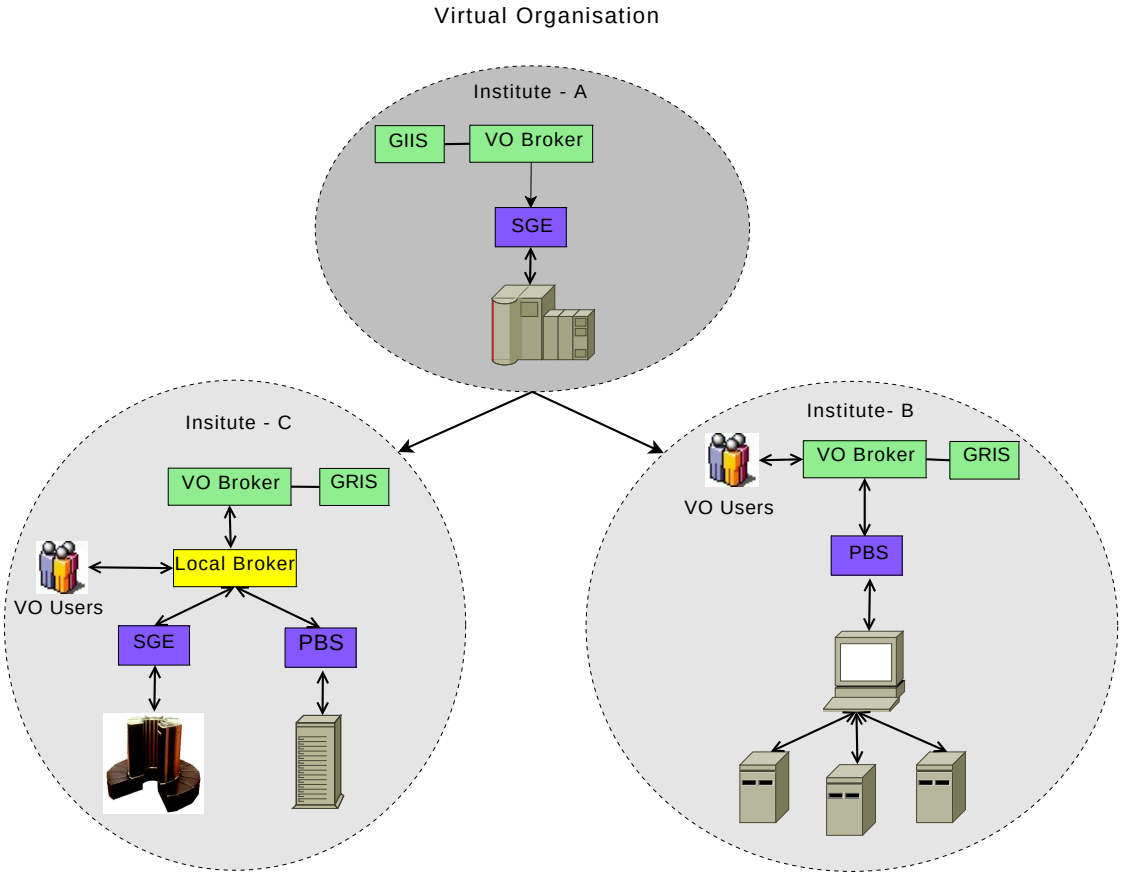


Figure 1.2: Virtual Organisation based Grid organisation.

Furthermore, end-users or their application-level schedulers submit jobs to a LRMS without having the knowledge about response time or service utility. The main reason for this being the lack of admission control decision making support between brokers and LRMSes. Sometimes these jobs are queued for relatively excessive times before being actually processed, leading to degraded QoS. To mitigate such long processing delays and enhance the value of computation, a scheduling strategy can use priorities from competing

user jobs that indicate varying levels of importance. To be effective, the schedulers require knowledge of how users value their computations in terms of QoS requirements, which usually varies from job to job. LRMS schedulers can provide a feedback signal that prevents the user from submitting unbounded amounts of work.

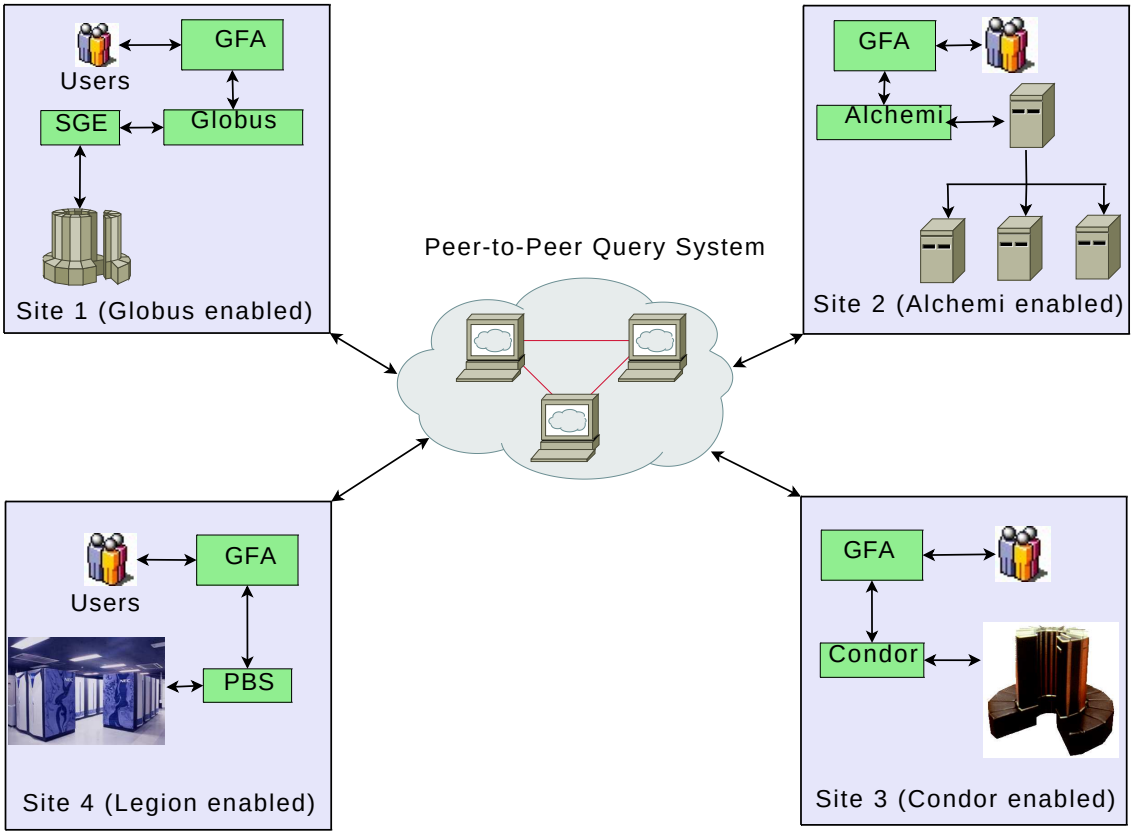


Figure 1.3: Proposed solution: Grid-Federation resource sharing system.

To address the lack of coordination between the broker-to-broker and broker-to-LRMS, we have proposed a mechanism for coordinated sharing of distributed Grid resources (in particular compute clusters and supercomputers) based on a scalable P2P query system. The resulting environment, called *Grid-Federation* [136, 137], allows the transparent use of resources from the federation when local resources are insufficient to meet its users' requirements. Fig. 1.3 shows an abstract model of our proposed Grid-Federation. Grid-Federation consists of cluster resources that are distributed over multiple organisations and control domains. Each site in the federation instantiates a Grid-Federation Agent (GFA) resource management system that exports the local resources to the federation. P2P network model forms the basis for organising the distributed GFA network.

In this case the P2P system provides a decentralised database with efficient updates and range query capabilities. This decentralised organisation of the system gives the provider more autonomy and additionally makes the system highly scalable. Further, to establish accountability between application schedulers and resource providers on service guarantee and delivery we have proposed an SLA-based scheduling and resource allocation algorithms [138]. A computational economy metaphor [3, 63, 162, 172] is utilised in negotiating and establishing the SLA contracts between the distributed brokers.

Traditionally, Grid computing systems such as resource brokers have evolved around a centralised Client-Server (CS) model. The responsibilities of the key functionalities such as resource discovery [67, 183] and resource provisioning coordination [120] in current Grid scheduling systems are delegated to the centralised server machines. In the centralised CS model, the network links leading to the server are very critical to the overall functionality of the system, as their failure might halt the entire distributed system operation. Current Grid systems have started showing the bad symptoms of centralised organisation in terms of bandwidth capacity and scalability as the number of brokers, users and providers increase in the system. Recent studies conducted by Zhang et al. [183] verified that existing systems including RGMA [183], Hawkeye [182] and MDS-2,3,4 [67]) fail to scale beyond 300 concurrent users, after which the throughput begins to decline below acceptable levels. With regards to the response time performance metric, MDS-2 performs the worst, superseded by R-GMA and Hawkeye.

Grids including APACGrid[1], LCGGrid, ChinaGrid are organised based on VO [71] hierarchical resource sharing model (refer to Fig. 1.2). The APAC (Australian Partnership for Advanced Computing) Grid interconnects various Grid sites distributed across Australian Institutions and Universities. The APACGrid uses a hierarchical information service, MDS-2. VPAC (Victorian Partnership for Advance Computing), which is a part of the APACGrid, hosts the centralised GIIS (Grid Index Information Service:-a component MDS-2), while the remaining Grid sites run the GRIS (Grid Resource Information Service) that connects to the VPAC GIIS. A Grid resource broker who wishes to access the APACGrid has to contact the VPAC GIIS, as contacting one of the other Grid sites running a GRIS would only allow access to information about that particular resource.

[1] http://grid.apac.edu.au/

This isolation in resource information organisation in grids and among grids leads to the resource fragmentation problem. In this case, Grid users get access to only small pool of resources. Further, the institution hosting the root GIIS service are central point of contact for the overall system. Failure of the root GIIS can partition the system, and can lead to significant performance bottlenecks.

To overcome the limitations of centralised and hierarchical information services, we have proposed a decentralised Grid resource information service based on a spatial publish/subscribe index. It utilises a Distributed Hash Table (DHT) routing substrate for delegation of d-dimensional service messages. DHTs have been proven to be scalable, self-organising, robust and fault-tolerant. The proposed Grid resource discovery service organises data by maintaining a logical d-dimensional publish/subscribe index over a network of distributed Grid brokers/Grid sites. The spatial nature of the publish/subscribe index has the capability to respond to complex Grid resource queries such as range queries involving various attribute types including, those that have a spatial component.

Further, the resource discovery system is extended to provide the abstraction/facility of a P2P tuple space for realising a decentralised coordination network. The P2P tuple space [111] can transparently support a decentralised coordination network for distributed brokering services. The P2P tuple space provides a global virtual shared space that can be concurrently and associatively accessed by all participants in the system and the access is independent of the actual physical or topological proximity of the tuples or hosts. The Grid peers maintaining the tuple space undertake activity related to job load-balancing across the Grid-Federation resources.

1.3 Outline and Contributions

In this section, the outline and contributions of the various chapters of the thesis are set out.

- Chapter 2 discusses the core state of art relevant to the work in this thesis. Comprehensive taxonomies related to decentralised scheduling, objective functions, coordination and security are presented and are later utilised for classifying the current state of the art. The key motivation behind Chapter 2 is to analyse the effectiveness

of the current solutions in facilitating a decentralised and coordinated Grid resource provisioning environment.

- Chapter 3 reviews and provides taxonomies for the second main area P2P Networks, which is utilised as a model for decentralising the proposed Grid resource management system. Specifically, Chapter 3 explores the area of complex Grid resource queries over DHTs. This chapter contributes towards providing comprehensive survey and taxonomy of DHT based indexing approaches that can support the d-dimensional Grid resource queries. Chapter 3 contributes by providing a qualitative comparison of the existing indices with respect to scalability and load-balancing. The presented comparison can be utilised by the Grid system developers with respect to deciding the kind of indexing system they should follow.

- Chapter 4 presents a novel model for coordinated coupling of distributed computational resources as a part generalised Grid resource sharing environment. The key contributions of Chapter 4 include the proposed new distributed resource management model called Grid-Federation, which provides: (i) a market-based coordination protocols for Grid scheduling; (ii) decentralisation via a P2P query system that gives site autonomy and scalability; (iii) ability to provide admission control facility at each site in the federation; (iv) incentives for resource owners to share their resources as part of the federation; and (v) access to a larger pool of resources for all users. The proposed approach provides better autonomy to the resource providers in the system as compared to existing VO based systems.

- Chapter 5 contributes in providing Service-Level Agreements (SLAs) based GFA-to-GFA and GFA-to-LRMS service negotiation algorithms. The brokers in the Grid-Federation system form a *contract net*, where the job migration in the net is facilitated through the SLA contracts. The main contributions of this work are: (i) an SLA bid based Grid scheduling approach; (ii) a Greedy back-filling cluster scheduling approach for LRMS that focuses on maximising the resource owners' payoff function; and (iii) allowing resource owners to have a finer degree of control over resource allocation decisions.

- Chapter 6 proposes a decentralised Grid resource discovery system based on a spatial publish/subscribe index. The proposed Grid resource discovery system organises data by maintaining a logical d-dimensional publish/subscribe index over a network of distributed Grid brokers. The main contributions of this work include: (i) extension of the DHTs with Grid resource discovery capability; (ii) a decentralised Grid resource discovery system based on a spatial and P2P publish/subscribe index; and (iii) overcoming the design limitations of current centralised and hierarchical Grid information services. Extensive simulation based study is conducted in order to prove the feasibility of the proposed resource discovery technique.

- Chapter 7 presents a new generation P2P tuple space that helps distributed Grid application schedulers and resource providers with coordinating application scheduling and resource allocation. The resource discovery system proposed in Chapter 6 forms the basis for the P2P tuple space. The main contributions of this work are: (i) a proposal for facilitating a P2P coordination space among administratively and topologically distributed Grid application schedulers and resource providers; and (ii) a decentralised load-distribution algorithm with focus on curbing the over-provisioning of resources and enhancing the overall utility of the system. Extensive simulations are conducted for evaluating the feasibility and performance of the proposed approach.

- Chapter 8 presents the design and implementation of a software system which partially prototypes the ideas presented in this thesis. The Grid-Federation scheduling framework has been developed using the Alchemi [117] desktop resource assembling system. The P2P resource discovery service utilises the Application Programming Interfaces (APIs) of the FreePastry software with respect to the Internet based message routing. The software was implemented strictly considering the Object Oriented Design (OOD) methodology. The developed framework is flexible enough to support other wide-area application services such as distributed auction network, storage trading environment.

- Chapter 9 presents the conclusion reached in this thesis, along with possible future research directions that can be built on this research work.

1.4 Publication Record

Portions of work presented in this thesis have been partially or completely derived from the following set of publications.
Chapter 3 is partially derived from the following publications.

- **R. Ranjan**, A. Harwood, and R. Buyya, Peer-to-Peer Based Resource Discovery in Global Grids: A Tutorial. To appear in *The IEEE Communication Surveys and Tutorials* (COMST), IEEE Communications Society Press, 2008.

Chapter 4 is partially derived from the following publications.

- **R. Ranjan**, R. Buyya, and A. Harwood, A Model for Cooperative Federation of Distributed Clusters, 2 Page Poster Paper, In Proceedings of the 14th IEEE International Symposium on High Performance Distributed Computing (HPDC-14), July 24-27, 2005, Research Triangle Park, North Carolina, USA.

- **R. Ranjan**, R. Buyya, and A. Harwood, A Case for Cooperative and Incentive Based Coupling of Distributed Clusters, In Proceedings of the 7th IEEE International Conference on Cluster Computing (Cluster 2005), IEEE Computer Society Press, September 27 - 30, 2005, Boston, Massachusetts, USA. An extended version to appear in *The Future Generation Computer Systems,* Elsevier Press, 2007.

Chapter 5 is partially derived from the following publication.

- **R. Ranjan**, A. Harwood, and R. Buyya, A SLA-Based Coordinated Superscheduling Scheme and Performance for Computational Grids , In Proceedings of the 8th IEEE International Conference on Cluster Computing (Cluster 2006), IEEE Computer Society Press, September 27 - 30, 2006, Barcelona, Spain.

Chapter 6 is partially derived from the following joint publication.

- **R. Ranjan**, L. Chan, A. Harwood, S. Karunasekera, and R. Buyya, Decentralised Resource Discovery Service for Large Scale Federated Grids, In Proceedings of the 3rd IEEE International Conference on e-Science and Grid Computing (e-Science 2007), IEEE Computer Society Press, December 10-13, Bangalore, India.

Comments: Lipo Chan has contributed towards implementation of spatial publish/subscribe index. I have extended her work to facilitate the Grid resource information service. Further, my contributions have been towards integrating the simulators, namely GridSim and PlanetSim to enable the decentralised resource discovery experiments. I do not claim the spatial publish/subscribe index to be the contribution of this thesis.

Chapter 7 is partially derived from the following joint publication.

- **R. Ranjan**, A. Harwood, and R. Buyya, Peer-to-Peer Tuple Space: A Novel Protocol for Coordinating Resource Provisioning, Technical Report, GRIDS-TR-2007-14, Grid Computing and Distributed Systems Laboratory, The University of Melbourne, Australia, July 30, 2007.

Chapter 8 is partially derived from the following joint publication.

- **R. Ranjan**, X. Chu, C. A. Queiroz, A. Harwood, and R.Buyya, Alchemi-Federation: A Self Organising Coordinated Desktop Grid Resource Sharing Environment, Technical Report, GRIDS-TR-2007-15, Grid Computing and Distributed Systems Laboratory, The University of Melbourne, Australia, July 30, 2007.

Chapter 2

An Overview of Decentralised Grid Systems

This chapter presents an overview of decentralised Grid brokering approaches with major focus on coordinated resource management. In other words, we focus on the brokering systems that aim towards coupling distributed Grid resources as part of a single cooperative resource sharing system such as a *federation* or virtual organisation. A distributed system configuration is considered as decentralised "if none of the participants in the system are more important than the others, in case that one of the participants fails, then it is neither more nor less harmful to the system than caused by the failure of any other participant in the system".

Decentralisation of Grid computing systems based on P2P network model can certainly overcome the current limitations of centralised and hierarchical model in scalability, single point failure, autonomy and trustworthiness. However, complete decentralised nature of the system raises other serious challenges in domains of application scheduling, resource allocation, coordination, resource discovery, security, trust and reputation management between participants.

Comprehensive taxonomy related to decentralised scheduling, objective function, coordination and security are presented and are later utilised for classifying the current state-of-the-art. This study contributes by providing better understanding of existing Grid resource management systems with respect to the degree of decentralisation and coordina-

tion that they can support. Further, we have briefly looked at the current security solutions available for building such decentralised Grid systems. We also looked at some of the representative systems that are based on the decentralised network models and aim towards cooperative resource management. Finally, in Table 2.1 we present the classification of the surveyed systems based on the taxonomy presented in this chapter.

2.1 Taxonomy

2.1.1 Scheduling

In this taxonomy (refer to Fig. 2.2), we categorise Grid superscheduling into online and offline approaches. We further consider the centralised and decentralised decision control with either online or offline settings. In the centralised superscheduling organisation [13], all the system-wide decision making is coordinated by a central controller. This scheduler organisation is conceptually able to produce very efficient schedules, because the central instance has all the necessary information about every job currently in the queue and the status of resources. Centralised organisation is simple to implement, easy to deploy and presents few management hassles. However, this scheme raises serious concerns when subjected to larger system size. Note that, throughout the thesis I have used the terms Grid superscheduling (superscheduler) and resource brokering (broker) inter-changeably.

The decentralised scheduler organisation negates the obvious limitation of the centralised organisation with respect to fault-tolerance, scalability and autonomy (facilitating domain specific resource allocation policies). This approach scales well for both, a small scale resource sharing environment (e.g. resource sharing under same administrative domain) to a large scale environment (e.g. the Internet). However, this approach raises serious challenges in the domain of distributed information management, enforcing system wide coordination, security, resource consumer authenticity and resource provider's policy heterogeneity. Systems including [29, 61, 76, 174] use this scheme.

The decentralised scheduling is further classified into two categories namely, coordinated decentralised [16, 29, 152, 174] and non-coordinated decentralised [3, 75]. In the decentralised non-coordinated scheme, application schedulers perform scheduling related

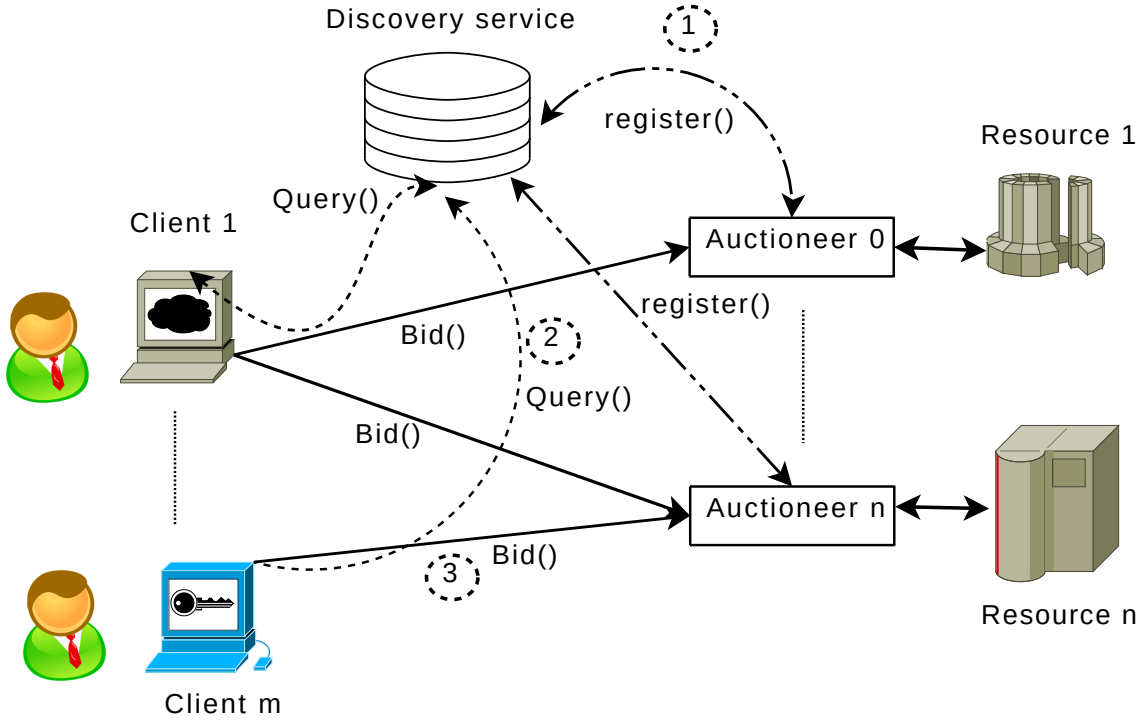


Figure 2.1: Decentralised non-coordinated scheduling in Tycoon.

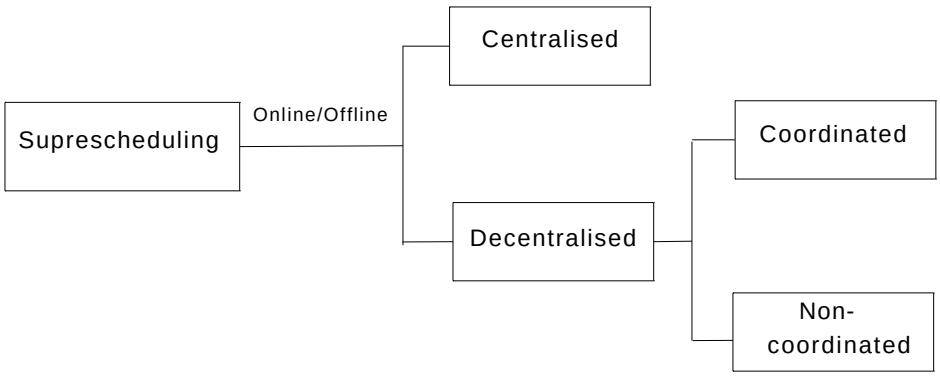


Figure 2.2: Distributed superscheduling taxonomy.

activities independent of the other schedulers in the system. Condor-G resource brokering system performs non-coordinated or non-cooperative scheduling by directly submitting jobs to the condor pools without taking into account their load and utilization status. Non-coordinated approach followed by these brokers exacerbates the load sharing and utilization problems of distributed resources since sub-optimal schedules are likely to occur. Fig. 2.1 shows the decentralised non-coordinated scheduling approach in Tycoon resource sharing system. Auctioneers advertise the resource availability and configuration to the

discovery service. Client agents query the discovery service to gather information about available auctioneers in the system. In Fig. 2.1, both the Client agents end up bidding to the auctioneer n because of lack of coordination among them.

On the other hand, decentralised coordinated scheduling schemes negotiates resource conditions with the local site managers in the system if not with the other application level schedulers. Legion-Federation system coordinates scheduling decision with other sites in the distributed environment through job query mechanism. A job query request (containing job type and composition) is sent to k remote sites for bidding. Each remote site Grid scheduler then contacts its LRMS to obtain job completion time on their local resource and sends this information back to the initiator's site. Finally, the site who bids with the least projected job completion time is selected for job scheduling.

2.1.2 Objective Function

Grid resources are dynamic in nature whose state can change in very small interval of time, hence it warrants scheduling and resource allocation policies that can adapt to changing conditions. Resources belong to different domains and are controlled by diverse resource management policies. Further, the Grid Participants (GPs) including resource providers and resource consumers associate diverse objective functions with resource allocation and scheduling processes. The resource owners in a grid form a group of participants who make rational choices independently or based on the strategic analysis of what others in the group might do [27]. They like to dictate the access privilege for their resources through diverse sharing policies. Thus, a resource owner enforces the pricing policy, admission control policy and domain specific resource allocation strategy.

Similarly, the resource consumers in a grid associate QoS-based utility constraints to their applications and expect that the constraints are satisfied within the acceptable limits. Every resource owner makes the policy related decision independently that best optimizes his objective function. Likewise, resource consumers have diverse QoS-based utility constraints, priority and demand patterns. Exact composition of a GP's objective function is determined by the mechanism design principles [58], [64]. A Grid system based on system centric mechanism defines relatively simple objective function. A system

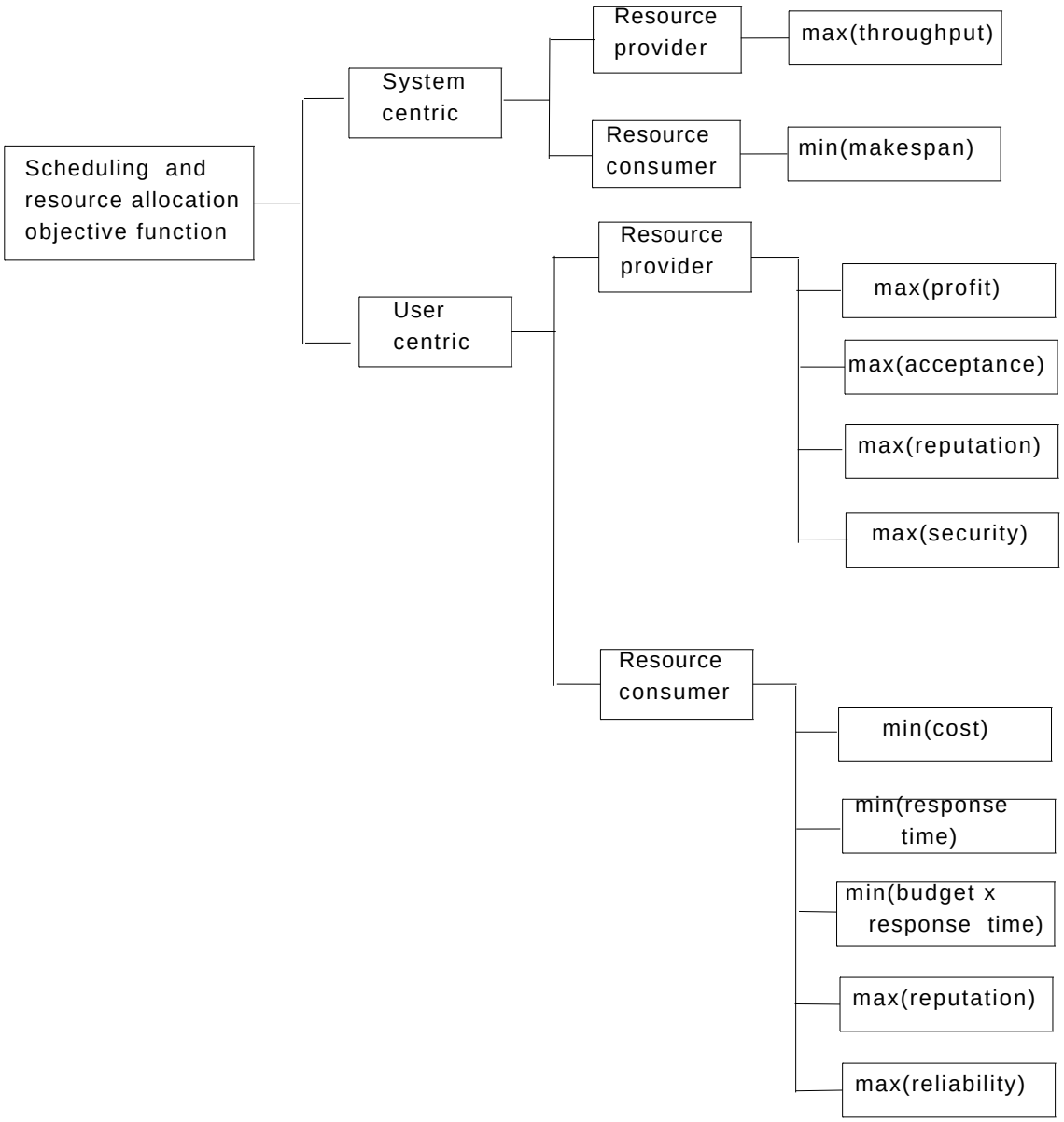


Figure 2.3: Grid scheduling and resource allocation objective function taxonomy.

centric scheduler focuses on maximising resource throughput on the owner side while minimizing overall user's application makespan.

Grid and PlanetLab systems including Tycoon, Bellagio, OurGrid, Sharp, and Nimrod-G apply market-based economic mechanism for resource management and application scheduling. Market driven scheduling mechanisms define user's objective functions based on QoS parameters. These QoS parameters include reputation, budget spent, response time or combination of all. Exact combination of QoS parameters is determined by the applied economic model. Some of the commonly used economic models [31] in resource

allocation includes the commodity market model [176], the posted price model, the bargaining model, the tendering/contract-net model, the auction model, the bid-based proportional resource sharing model, the bartering model and the monopoly model. Systems including OurGrid and Sharp are based on bartering of resources among cooperative domains. In this case, the focus of each participant is on maximizing its bartering reputation in the system. In bartered system, a participant is the consumer as well as the provider at the same time. In cooperative market model, such as the bartered economy, there is singleton objective function shared by both consumer and provider.

Competitive market models including commodities market, bid-based proportional sharing, and auction warrants separate objective functions for providers and consumers (refer to Fig. 2.3). Resource owners define objective function with focus on maximizing profit. For this purpose, they can adjust the resource prices [155] dynamically based on supply and demand pattern.

2.1.3 Coordination

Decentralised design of Grid system can effectively overcome the limitations of centralised and hierarchical VO-based traditional approaches. But the effectiveness of the resulting decentralised system depends on the level of coordination and cooperation [4] among the participants. Decentralised participants are pools of diverse peers or brokers sharing and controlling resources and which have agreed to co-operate for enhancing the overall utility of the system. Realising such a co-operation among dynamic and selfish participants warrants robust mechanism for coordination and negotiation policies. Coordinated application scheduling and resource management involves dynamic information exchange between various Grid schedulers and LRMSes in the system. In general, a coordination process can include a sequence of QoS inquiry and QoS guarantee negotiation message exchange between Grid schedulers and LRMSes. In Fig. 2.4 we present the coordination methods taxonomy.

Negotiation among all the participants can be based on well-known agent coordination mechanism called contract net protocol [157]. Contract net partitions the coordination space into two distinct domains including a *manager* and a *contractor*. Domain

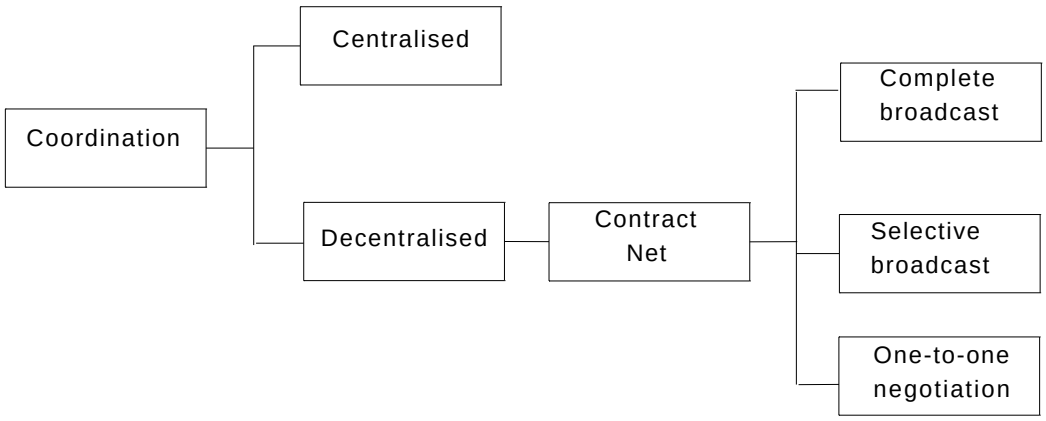


Figure 2.4: Coordination taxonomy.

membership in the contract net is governed by the role played by a participant. A resource broker in a decentralised Grid system can adhere to the role of a contractor that negotiates SLAs with resource providers. Effectively, resource providers work as a manager that export its local resources to the outside contractors and is responsible for decision relating to the admission control based on negotiated SLAs. Contract net based approaches to Grid scheduling have been widely explored in some of the recent works including [55, 130, 138]. The work in [102] studies the effect of contract net communication overhead on job execution time in a multi-agent Grid computing environment.

Distributed negotiation has substantial message overhead and it can worsen as system scales to a large number of participants. Traditionally, contractors in contract net broadcast call-for proposal (CFP) request to all managers in the system. Following this, managers reply with bid offers to the contractor. The contractor selects a winner based on its requirement and acknowledges with accept message to winner manager, while a bid reject message is sent to remaining managers. Communication protocols based on one-to-all broadcast are very expensive in terms of number of messages and network bandwidth usage. Similar negotiation protocol has been proposed in the work NASA-Scheduler and Legion-Federation for decentralised Grid scheduling. Condor-Flock P2P Grid system proposed selective broadcast to the flocks currently indexed by the Pastry routing table.

The SLA-based Grid superscheduling approach proposed in [138] advocates one-to-one negotiation among contractors and managers. Some approaches including Bellagio

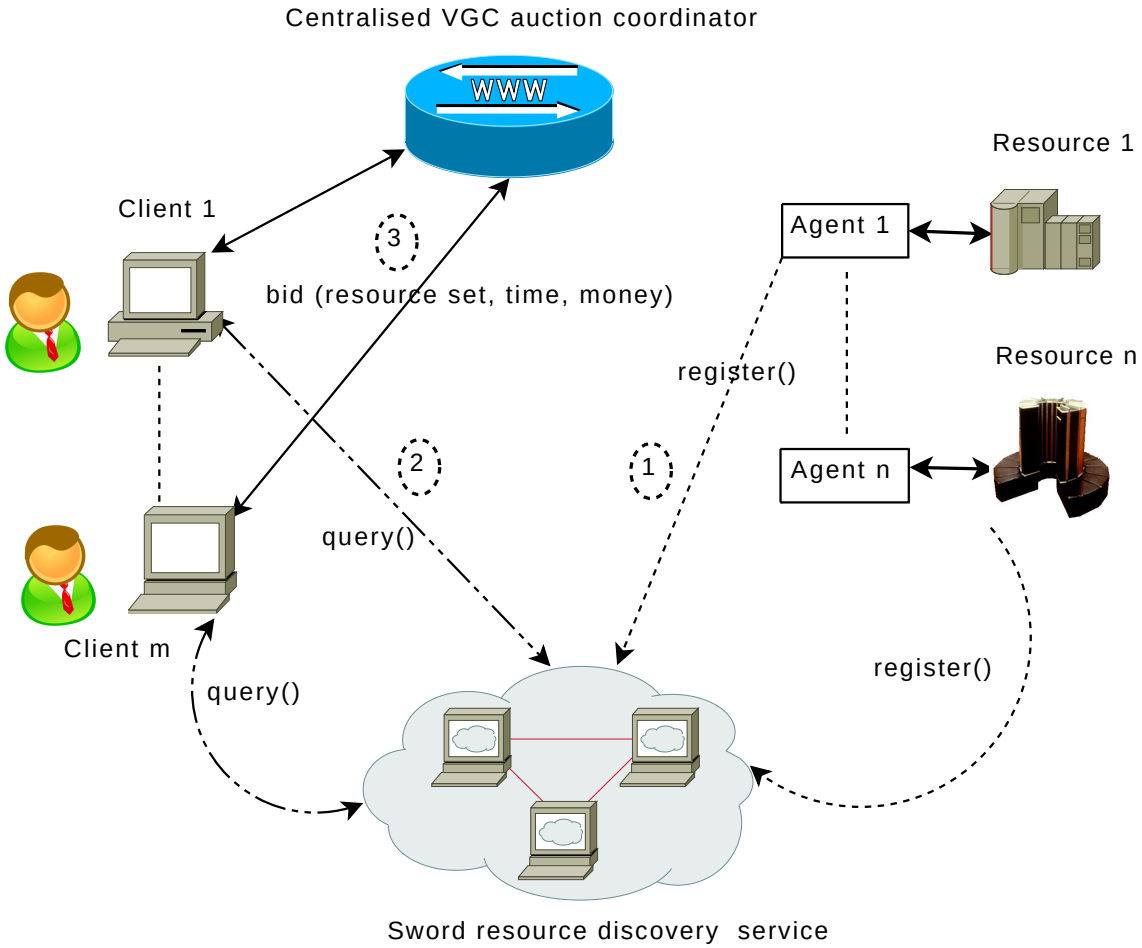


Figure 2.5: Centralised coordination in Bellagio.

and AMDLoad [86] advocate coordinating resource activity among decentralised participants based on centralised coordinators. Fig. 2.5 shows centralised coordination methodology applied by Bellagio system. Resource agents register the resource configuration with the Sword [128] resource discovery service. Client agents query the Sword to locate available resources in the system. Once the resource lists are obtained, Client agents bid for resources with the centralised auction coordinator. The bid parameters include the sets of resources desired, a time for which application would be deployed on resources, and the amount of virtual money clients are ready to spend.

2.1.4 Security and Trust

The decentralised organisation of Grid systems raises serious challenges in the domains of security and trust management. Implementing a secure decentralised Grid system requires solutions that can efficiently facilitate the following [37]: preserve the privacy of participants, ensure authenticity of the participants, robust authorization, securely route messages between distributed services, and minimise loss to the system due to malicious participants. In Fig. 2.6, we present the taxonomy for security and trust management in Grid and P2P systems.

The privacy of the participants can be ensured through secret key-based symmetric cryptographic algorithms such as 3DES, RC4, etc. These secret keys must be securely generated and distributed in the system. Existing key management systems such as public key algorithms (including DH, RSA, elliptic), Kerberos (trusted third party) can be utilised for this purpose. Authentication of the participants can be achieved through trust enforcement mechanisms such as X.509 certificates (Public Key Infrastructure) [93], and Kerberos (third party authentication), distributed trust and SSH. Authentication based on X.509 certificates warrants a trusted Certifying Authority (CA) in the system.

A Grid participant presents a X.509 certificate along with an associated private key (the combination of these entities forms a system wide unique credential) in order to authenticate itself with a remote service. A system can have a single CA, which is trusted by all the participants. However, single CA approach has limited scalability. An alternative to this is to have multiple CAs combining together to form a trust chain. In this case, a certificate signed by any CA in the system has global validity. The GSI [175] implementation of PKI supports dynamic trust chain creation through the Community Authorization Service (CAS) [132]. This is based on the policy that two participants bearing proxy certificates, signed by the same user, will inherently trust each other. Kerberos based implementation has significant shortcomings as it requires synchronous communication with the ticket granting server in order to setup communication between a client and server. If the ticket granting server goes offline or has a security breach then there is no way the system can operate. In case of X.509 based implementation, a CA can certify the credentials offline.

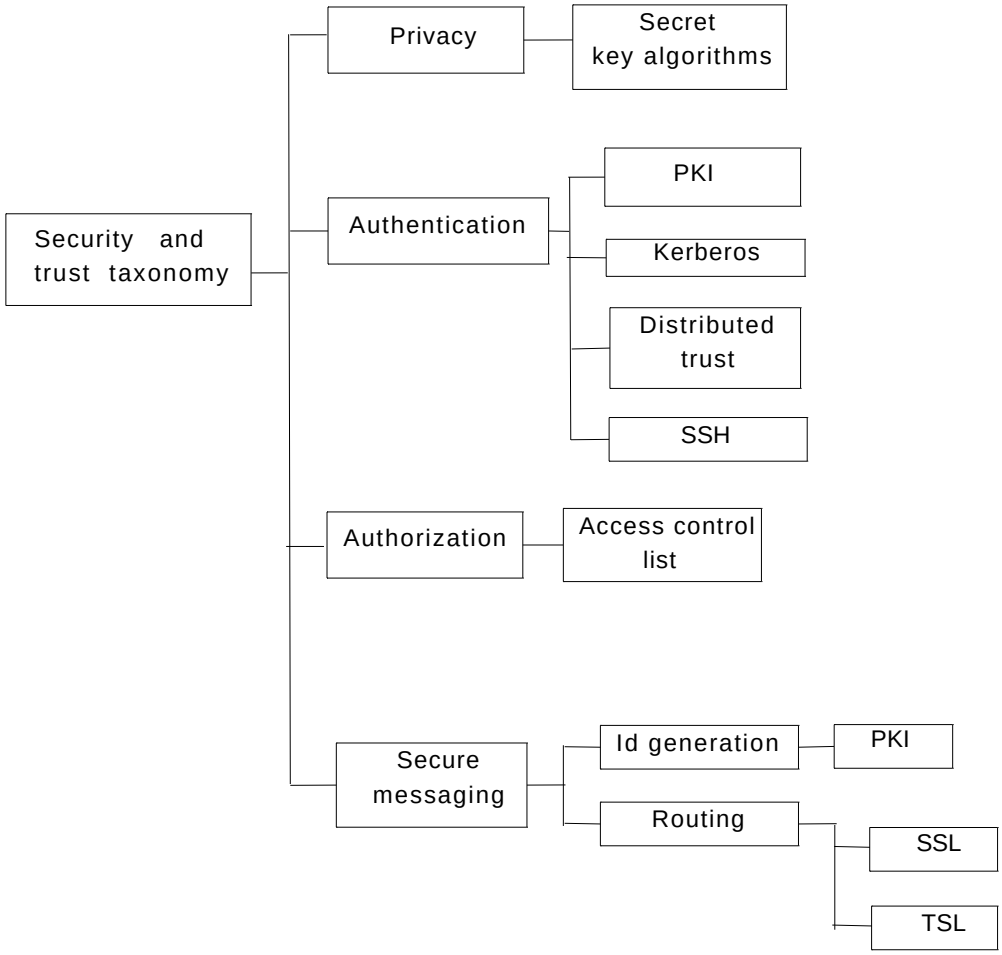


Figure 2.6: Security and trust taxonomy.

Having said that, a majority of implementations do rely on centralised trust enforcement entities such as a CA or a ticket granting authority. The JXTA [178] system provides a completely decentralised X.509 based PKI. Each JXTA peer is its own CA and issues a certificate for each service it offers. Peer CA certificates are distributed as part of the service advertisement process. Each of the CA certificate is verified via the *Poblano: "web of trust"* [178], a distributed reputation management system. A similar distributed trust mechanism called PeerReview [60] has also been proposed. These distributed trust management systems deter malicious participants through behavioral auditing. An auditor node A checks if it agrees with the past actions of an auditee node B. In case of disagreement, A broadcasts an accusation of B. Interested third party nodes verify evidence, and take punitive action against the auditor or the auditee.

The SSH based authentication scheme is comparatively easier to implement as it does

not require trusted third party certification. However, it does not allow the creation of a dynamic trust chain and in case a participant's private key is compromised, it requires every public key holder to be informed about this event. PlanetLab utilises SSH based authentication wherein the centralised PlanetLab Central service is responsible for distribution or copying of the keys. Unlike X.509 and Kerberos implementation, SSH does not support certificate translation mechanism (i.e. from X.509 to Kerberos or vice versa). Transport layer security protocols such as TLS [45], and SSL [44] are used for message encryption and integrity checking as they are transported from one host to the other on the Internet.

Table 2.1: Classification based on taxonomies.

System Name	**Scheduling Model**	**Objective Function**	**Coordination Model**	**Security Model**
Bellagio	Centralised	User centric, Bid-based proportional sharing	Centralised	SSH
Tycoon	Decentralised non-coordinated	User centric, Auction	One-to-All broadcast	N.A.
AMDload	Centralised	User centric, Auction	Centralised	N.A.
VO-Ranka	Centralised (VO)	User centric	Centralised	PKI
NASA-Scheduler	Decentralised coordinated	System centric	One-to-All broadcast	N.A.
Legion-Federation	Decentralised coordinated	System centric	One-to-All broadcast	N.A.
Trader-Federation	Decentralised coordinated	User centric, Commodity market	One-to-All broadcast	N.A.
Agent-Federation	Decentralised coordinated	User centric, Commodity market	One-to-All broadcast	N.A.
CondorFlock P2P	Decentralised coordinated	System centric	Selective broadcast	N.A.
MOSIX-Fed	Centralised	System centric	Centralised	N.A.
Multi-Request	Decentralised coordinated	System centric	Selective broadcast	N.A.
Sharp	Decentralised coordinated	User centric, bartering	One-to-one negotiation	PKI
Nimrod-G	Decentralised non-coordinated	User centric, Commodity market	N.A.	PKI
Condor-G	Decentralised non-coordinated	System centric	N.A.	PKI

Authorization deals with the verification of an action that a participant is allowed to undertake after a successful authentication. In a Grid, site owners have the privilege to control how their resources are shared among the participants. The resource sharing policy

takes into account the participant's identity and membership to groups or virtual organisations. Globus based Grid installation defines the access control list using a Gridmap file. This file simply maintains a list of the distinguished names of the Grid users and the equivalent local user account names that they are to be mapped to. Access control to a resource is then left up to the local operating system and application access control mechanisms.

Implementing a secure and trusted routing [37] primitive requires a solution to the following problems: secure generation and assignment of nodeIds, securely maintaining the integrity of routing tables, and secure message transmission between peers. Secure nodeId assignment ensures that an attacker or a malicious peer cannot choose the value of nodeIds that can give it membership of the overlay. If the node assignment process is not secure, then an attacker could sniff into the overlay with a chosen nodeId and get control over the local objects, or influence all traffic to and from the victim node. The nodeId assignment process is secured by delegating this capability to a central, trusted authority. A set of trusted certification authorities (CAs) are given the capability to assign nodeIds to peers and to sign nodeId certificates, which bind a random nodeId to the public key that uniquely identifies a peer and an IP address. The CAs ensure that nodeIds are chosen randomly from the id space, and prevent nodes from forging nodeIds. Furthermore, these certificates give the overlay a public key infrastructure, suitable for establishing encrypted and authenticated channels between nodes. Secure message forwarding on the Internet can be achieved through secure transport layer connections such as TLS and SSL.

2.1.5 Resource Discovery

Chapter 3 presents a comprehensive taxonomy and survey on decentralised resource discovery in computational Grid environment.

2.2 Related Systems

2.2.1 Legion-Federation

The work [174] proposes a federated model for distributed cooperative resource management. The model proposes federated resource sharing using Legion [39] LRMS. It considers two levels of application schedulers in the system namely, Local Site (LS) Scheduler and Wide-Area (WA) scheduler. Every member site has to instantiate these scheduling services. LSes or LRMSes are responsible for managing and controlling the set of resources assigned to them (domain specific). Various WA schedulers in conduction with respective LSes coordinate and enable WA application scheduling. WA scheduler has two functional components including a Scheduling Manager (SM) which is an interface to LS, and a Grid Scheduler (GS) which connects to other SMes in the federated system. The connection topology between GSes is a fully connected graph structure. The distributed information between various SMes is managed though a static file (containing addresses of other SMes). When a job arrives at a SM, a set of k sites out of total m sites are selected as future candidates. Following this, a job query request (containing job type and composition) is sent to k remote sites to initiate bidding on completion time. Each remote GS then contacts its LS through components to obtain job completion time on the local resource. The projected job completion time is returned to the initiator SM. The initiator SM then schedules the job on the remote site which had bided with the least projected job completion time.

2.2.2 AMDLoad

The work [86] proposes the load balancing framework based on the mechanism design theory. The Algorithmic Mechanism Design (AMD) theory is specification of outputs and payments to agents that lead them to behave in a way that results in system-wide equilibrium [64]. This work considers the AMD problem for one parameter agents in which, (i) a finite set of outputs is given, (ii) each agent has a privately known parameter t_i called true value, (iii) each agent's goal is to maximize its profit, and (iv) the goal of the mechanism is to select an output that optimizes a given cost function. So, the work

designs a truthful mechanism for solving the static load balancing problem in heterogeneous distributed system. Each computer in the distributed system is characterized by its processing rate μ_i, and only computer i knows the true value of its processing rate. The cost in each computer is proportional to its utilization and the payment is given for the provided utilization. The profit is defined as the difference between the payment and the cost. Under this static environment, they analyse and design the optimal algorithm to find the fraction of load λ_i that is allocated to each computer i such that the expected execution time is minimized. The proposed framework is based on a centralized model so that a dispatcher decides the allocation and payment of each computer. For a given static job arrival rate, the dispatcher sends a request for bid messages to each computer. When a computer receives the bid request, it replies with its bid ($1/\mu_i$) to the dispatcher. After the dispatcher collects all the bids, it computes the optional allocation and payment for each computer i. The dispatcher executes this protocol periodically or when the total job arrival rate is changed.

2.2.3 Trader-Federation

The work [73, 74] highlights the necessity of coordinated resource management in distributed systems. It presents a scheme called federation of distributed resource traders, which couples various autonomous resources or resource providers. A resource trader entity acts as an intermediary between consumers and providers. Every trader has local users, clients and resources who are members of the local resource domain. Federation of traders enables the participants to trade resources at both local and the Internet levels. Various traders cooperate within the federation to maximise a trading function. The trader presents two interfaces, local interface for its local users, and local resource providers, while remote interface to other traders in the federation. The federation works as a market place where various traders can negotiate for QoS parameter (response time, accuracy and details for a request) requested by the local users. The request routing within the federation is based on a trading graph. The trading graph defines different communication paths between traders. A resource request along with its QoS parameters is exchanged among traders. If a trader on the routing path cannot satisfy the request locally, then the request

is forwarded to other traders in the federation. Every trader maintains a service offer on behalf of its local resource provider. Every incoming request is matched against the local service offer. In case the match occurs, the message is removed from the system and the initiator trader is informed about the deal.

2.2.4 Sharp

Sharp [76] is a framework for secure distributed resource management. Participant sites can trade their resources with peering partners or contribute them to a peer federation according to the local site sharing policies. Sharp framework relies on the bartering economy as the basis to exchange resources between resource domains. A cryptographically signed object called Resource Tickets (RTs) is issued by each participating site. These RTs are exchanged between the participating sites for facilitating coordinated resource management. In Sharp framework, every participating site is completely autonomous and holds all the rights over the local resources. Sharp framework considers collection of logical sites or domains, each running local schedulers for physical resources (e.g. processors, memory, storage, network links, sensors) under its control. The fundamental resource management software entities in Sharp include site authority, service manager and agents. These entities connect to each other based on a peer-to-peer network model. The resources at each site are managed by a site authority, which maintains the hard state. The site authority accepts the resource claims presented by remote sites in the system. Resource agents mediate between site authorities and resource consumers (service managers). A resource is allocated to the service manager at a Sharp site using the two-phase negotiation process. Initially, a service manager obtains a resource claim in the form of a ticket from an agent. In the second phase, the service manager presents the ticket to the appropriate site authority to redeem it. The authority may reject the ticket or it may honor the request by issuing a lease for any subset of resources or terms specified in the ticket. A lease is hard claim over concrete resources, and is guaranteed valid for its term unless a failure occurs.

2.2.5 Agent-Federation

The work in [130] proposes a multi-agent infrastructure that applies an SLA protocol for solving the Grid superscheduling problem. The SLA negotiation protocol is based on the Contract Net Protocol [156]. The system models three types of agents: User agent (UA), Local Scheduler agent (LSA) and Superscheduler agent (SSA). Every active site in the system instantiates these agents. The UAs are the resource consumers who submit jobs to SSA for execution on the Grid platform. The UA also specifies SLA based QoS parameters such as expected response time, budget and preferred hosts associated with the job. The LSA functionality is similar to LRMS, managing job execution within an administrative domain. LSA obtains jobs from the SSA that are submitted by local and remote UAs. The SSA agents are responsible for coordinating job superscheduling across different sites in the system. The SSA agents negotiate SLA parameters with the local LSA and remote SSA before scheduling the job. The model defines two kinds of SLAs: Meta-SLA and Sub-SLA. Meta-SLA refers to the initial SLA parameters submitted by the UA to its SSA. A Meta-SLA presents high-level job requirements and it can be refined during negotiation process with the SSA, while the Sub-SLA refers to the SLA parameters that are negotiated between SSA and remote site SSA. The SSA decomposes the Meta-SLAs to form Sub-SLAs. The Sub-SLA can contain much low-level resource description such as the amount of physical memory required and the number of processors required.

2.2.6 Multi-Request

The work in [163] presents a Grid superscheduling protocol based on multiple job SLA negotiation scheme. The key factor motivating this work is redundantly distributing the job execution requests to multiple sites in a Grid instead of just sending to most lightly loaded one. The authors argue that placing job in the queue at multiple sites increases the probability that the back-filling strategy will be more effective in optimizing the scheduling parameters. The superscheduling parameters include resource utilization and the job average turn around time. In other words, the scheduling parameters are system centric. The LRMSes at various Grid sites apply First Come First Serve (FCFS) policy with Easy Back-filling [65] approach for resource allocation. Further, the system proposes dual

queuing systems at each site. One queue for local jobs, while other queue for remote jobs. The easy back-filling resource allocation scheme gives higher priority to the local job queue than the remote job queue.

2.2.7 VO-Ranka

The work [98] proposes and investigates a framework that enables policy based resource management in Grid computing. It considers a scheduling strategy that controls the request assignment to resources by adjusting resource usage accounts or by assigning varying request priorities. Proposed approach also supports reservation based Grid resource allocation. The system can provide different levels of QOS by assigning varying levels of privileges to users, groups and requests. The work considers Grid environment as a collection of virtual organizations (VOs), which is a group of consumers and producers collaborating together to facilitate usage of high-end computational resources. Further, these organizations can be distributed nationwide or worldwide, may participate in one or more virtual organizations by sharing some or all of their resources. Resource providers and resource consumers who are part of VO, share resources by defining how resource usage takes place in terms of where, what, who and when. It represents the policies in a three dimensional space consisting of resource provider, resource consumer and time. A resource provider is considered as an entity, which shares some particular physical resources within the context of VO and grid. Further, each resource provider is augmented by a list of resource configuration such as CPU, memory, storage space, bandwidth, etc. A resource consumer is defined as an entity that consumes a resource.

2.2.8 NASA-Scheduler

The work [152] models a Grid superscheduler architecture and studies three different distributed job migration algorithms. Each computational resource site has a Grid superscheduler (GS) and a local scheduler (LRMS). Scheduling in the Grid environment is facilitated through cooperation between site specific LRMS and the GS. Resource management and job scheduling activities related to the local resources is handled by LRMS. While the GS is responsible for resource discovery, monitoring system status (utilization,

network condition), coordinating job migration related information with other GS in the system. The distributed scheduling parameters influencing a job migration decision in the system include approximate wait time (AWT), expected run time (ERT) and resource utilization status (RUS). The GS considers three job migration algorithms including sender-initiated, receiver-initiated and symmetrically-initiated. Details about these algorithms can be found in [152]. A job migration process includes a series of job specific and resource specific coordination inflammation exchange between GSes. A job is selected for migration if the local GS determines that AWT for a given job on the local resource is above some prefigured threshold value ϕ. A GS initiates a job migration process by querying all of its partner resources about ERT, AWT of the job and the resulting RUS. ERT varies between various computational resources depending on the hardware and software types. Based on the query response, a GS calculates the potential turnaround cost (TC) of itself and each partner. Further optimal TC is computed by summing up AWT and ERT. If the minimum TC is within a small tolerance limit for multiple machines, then the site with the lowest RUS is chosen for job migration.

2.2.9 MOSIX-Fed

MOSIX is a cluster management system that applies process migration to enable a loosely coupled Linux cluster to work like a shared memory parallel computer. Recently, it has been extended to support a Grid of clusters to form a single cooperative system [16]. Basic feature of the federated environment includes automatic load balancing among participant clusters (owned by different owners) while preserving the complete autonomy. Proposed resource coupling scheme can be applied to form a campus or an enterprise Grid. MOSIX federation aims toward hierarchical coupling of cluster resources under same administrative domain. Resource discovery in such an arrangement is facilitated by hierarchical information dissemination scheme, that enables each node to be aware of the latest system wide state. Key features of MOSIX federation include dynamic, Grid-wide preemptive process migration. Scheduling decisions are based on adaptive on-line scheduling algorithm that tries to optimise the system centric parameters (throughput, utilization). Additionally, scheduling decision also considers the information gathered as

a result of hierarchical dissemination scheme. Such information helps in allocating the processes to appropriate nodes in the grid. The MOSIX federation supports dynamic partitioning of cluster nodes based on ownership. This means that the ownership of nodes in a cluster changes over a period of time reflecting the resource demand pattern. Each user is allowed to create his processes on the nodes belonging to his partition. However, to support dynamic load balancing a user, specifically a node owner, can configure to host remote processes thus contributing to the federated Grid. In this case, an owner can make two sets of machines, one for home users, while other for remote users. Thus, this allows a resource owner to clearly define what is shared and what is not.

2.2.10 CondorFlock P2P

The work [29] presents a scheme for connecting existing Condor work pools using P2P routing substrate Pastry [143]. Inherently, P2P substrate aids in automating the resource discovery in the Condor Flock Grid. Resource discovery in the flock is facilitated through resource information broadcast to the pools whose ids appear in the Pastry node's routing table. The contacted pools reply with the message confirming their willingness about resource sharing. This information is saved for future scheduling decisions. However, it implies that a Condor pool in the flock is only aware of the subset of resources available in the system. Note that, the proposed P2P-based overlay network facilitates only resource discovery, while other decisions such as resource sharing policy is controlled by the pool managers. The proposed scheme periodically compares the metrics such as queue lengths, average pool utilization and resource availability scenario, and based on these statistics a sorted list of pools from most suitable to least suitable is formulated. Using this list, a Condor pool chooses appropriate pools for flocking. Core Condor LRMS has also been extended to work with Globus [70], the new version is called Condor-G resource broker, which enables creation of global Grids and is designed to run jobs across different administrative domains. This system basically interacts with the resources managed by Globus system. This enables general purpose Globus toolkit to solve a particular problem (i.e., high-throughput computing) on the Grid. Its features include a full-featured queuing service, credential management and enhanced fault-tolerance mechanism (local crash,

network failure, remote crash).

2.2.11 OurGrid

OurGrid [9] provides a Grid superscheduling middleware infrastructure based on the JXTA routing overlay. The OurGrid community is a collection of a number of OurGrid Peers (OGPeers) that communicate using JXTA protocols. Every site in the system hosts OG Peer service. A resource consumer (user) runs a brokering system called OGBroker (an application-level scheduler). Every OGBroker connects to OurGrid community through its local OGPeer. A resource provider runs the software system called Swan, that facilitates access to his resource for any user in the OurGrid community. To summarise, a site in the OurGrid system has following software components Swan, OGBroker and OGPeer. The resource sharing in OurGrid is based on P2P file-sharing model such that every participant contributes as well as consumes resources to/from the community. To negate free-riding in a computational Grid environment, the model defines a new trust and reputation management scheme called *Network of Favors* [8]. A user submits his application to his OGBroker. Depending on the user's application requirement, OGBroker sends the request for Grid machines to other OGPeers though the JXTA overlay. Depending on the resource availability pattern and initiator site's reputation, the OG Peers reply to the resource query. In other words, superscheduling in OurGrid is primarily driven by the site's reputation in the community.

2.2.12 Bellagio

Bellagio [13] is a market-based resource allocation system for federated distributed computing infrastructures. Users specify resources of interest in the form of combinatorial auction bids. Thereafter, a centralised auctioneer allocates resources and decides payments for users. The Bellagio architecture consists of *resource discovery* and *resource market*. For resource discovery of heterogeneous resources, Bellagio uses SWORD [128]. For resource market, Bellagio uses a centralised auction system, in which users express resource preferences using a bidding language, and a periodic auction allocates resources to users. A bid for resource includes sets of resources desired, processing duration, and

the amount of virtual currency which a user is willing to spend. The centralised auctioneer clears the bid every hour. The resource exchange in the current system is done through virtual currency. Virtual currency is the amount of credit a site has, which is directly determined by the site's overall resource contribution to the federated system. Bellagio employs Share [48] for resource allocation in order to support a combinatorial auction for heterogeneous resources. Share uses the *threshold rule* [106] to determine payments. Once the payment amount of each winning bid has been determined by the threshold rule, the winning bidders receive resource capabilities after charging the appropriate amount.

2.2.13 Tycoon

Tycoon [109] is a distributed market-based resource allocation system. Application scheduling and resource allocation in Tycoon is based on decentralised isolated auction mechanism. Every resource owner in the system runs its own auction for his local resources. In addition to this, auctions are held independently, thus clearly lacking any coordination. Tycoon system relies on centralised Service Location Services (SLS) for indexing resource auctioneers' information. Auctioneers register their status with the SLS every 30 seconds. In case an auctioneer fails to update its information within 120 seconds then SLS deletes its entry. Application level superschedulers contact the SLS to gather information about various auctioneers in the system. Once this information is available, the superschedulers (on behalf of users) issue bids for different resources (controlled by different auctions) constraint to resource requirement and available budget. In this setting, various superschedulers might end up bidding for small subset of resources while leaving the rest underutilized. In other words, superscheduling mechanism clearly lacks coordination.

2.2.14 Nimrod-G

Nimrod-G [3, 30] is a resource management system (RMS) that serves as a resource broker and supports deadline and budget constrained algorithms for scheduling task-farming applications on the platform. It allows the users to lease and aggregate resources depending on their availability, capability, performance, cost and users QoS constraints. Application scheduling is based on user-centric parameters. The broker is capable of dy-

namically leasing Grid services/resource at runtime depending on their cost, capabilities, availability and users' requirements. Nimrod-G gathers resource information by querying the MDS-1/2/3 information services of the Globus system. The Nimrod-G resource brokering suffers from a lack of coordination, i.e., it does not consider the presence of other Nimrod-G brokers in the system while formulating application schedules.

2.3 Summary and Conclusion

There are different projects that aim toward assembling Grid resources as part of a large scale resource sharing environment. Every approach has adopted different network models for resource information and broker organisation. Brokering approaches including Nimrod-G, and Condor-G can be combined together to form a non-coordinated decentralised broker network that utilises the resource indexing services of centrally or hierarchically organised information services such as RGMA [183], MDS-2/3, and Hawkeye [182]. These brokering systems do not focus towards enabling a large scale cooperative environment.

VO-based approaches such as LHC Data Grid network, and VO-Ranka connect both resource brokers and information services in hierarchy. VO-based approaches are superior to the resource brokering approaches with respect to the system-wide coordination. Mosix-Fed connects the departmental clusters in hierarchy and performs load-adaptive process migration. The scalability of the Mosix-Fed environment to Grid scale has not been explored yet.

Other approaches to Grid resource assembling including Sharp, Tycoon, Bellagio, OurGrid, NASA-Scheduler, CondorFlock P2P, Multi-Request, Trader-Federation, AMD-Load, Agent-Federation, Legion-Federation focuses on enabling a single and cooperative resource sharing environment. These systems utilised different types of methodology in coordinating application scheduling and resource allocation. The finer details on scheduling and coordination protocols are discussed in the relevant taxonomy section. The next chapter presents a comprehensive study on P2P based complex Grid resource queries. It discusses the taxonomies related to the Grid resource and P2P network organisation.

Chapter 3

Peer-to-Peer Grid Resource Discovery: State of the Art

Efficient Resource discovery mechanism is one of the fundamental requirement for Grid computing systems, as it aids in resource management and scheduling of applications. Resource discovery activity involve searching for the appropriate resource types that match the user's application requirements. Various kinds of solutions to Grid resource discovery have been suggested, including the centralised and hierarchical information server approach. However, both of these approaches have serious limitations in regards to *scalability, fault-tolerance and network congestion*. To overcome these limitations, indexing resource information using a decentralised (such as P2P) network model has been actively proposed in the past few years.

This chapter investigates various decentralised resource discovery techniques primarily driven by P2P network model. To summarise, this chapter presents a: (i) resource taxonomy with focus on computational Grid paradigm; (ii) P2P taxonomy with focus on extending the current structured systems (such as Distributed Hash Tables) for indexing d-*dimensional* Grid resource queries; (iii) detailed survey of existing works that can support d-dimensional Grid resource queries; and (iv) classification of the surveyed approaches based on the proposed P2P taxonomy.

3.1 Introduction

Traditionally, resource brokers [105] including Nimrod-G, Condor-G and Tycoon [109] used services of centralised information services (such as R-GMA [183], Hawkeye [182], GMD [181], MDS-1 [67]) to index resource information. Under centralised organisation, the schedulers send resource queries to a centralised resource indexing service. Similarly, the resource providers update the resource status at periodic intervals using resource update messages. This approach has several design issues including: (i) highly prone to a single point of failure; (ii) lacks scalability; (iii) high network communication cost at links leading to the information server (i.e. network bottleneck, congestion); and (iv) the machine running the information services might lack the required computational power required to serve a large number of resource queries and updates.

To overcome the above shortcomings of centralised approaches, a hierarchical organisation [182] of information services has been proposed in systems such as MDS-3 [54] and Ganglia [145]. MDS-3 organises VO [71] specific information directories in a hierarchy. A VO includes a set of GPs that agree on common resource sharing policies. Every VO in a grid designates a machine that hosts the information services. A similar approach has been followed in the Ganglia system, which is designed for monitoring resources status within a federation of clusters. Each cluster designates a node as a representative to the federated monitoring system. This node is responsible for reporting cluster status to the federation. However, this approach also has similar problems as the centralised approach such as one-point of failure, and does not scale well for a large number of users/providers.

3.1.1 Decentralised Resource Indexing

Recently, proposals for decentralising a GRIS have gained significant momentum. The decentralisation of GRIS can overcome the issues related to current centralised and hierarchical organisations. An early proposal for decentralising Grid information services was made by Iamnitchi and Foster [97]. The work proposed a P2P based approach for organising the MDS directories in a flat, dynamic P2P network. It envisages that every VO maintains its information services and makes it available as part of a P2P based network. In other words, information services are the peers in a P2P network based coupling

of VOs. Application schedulers in various VOs initiate a resource look-up query which is forwarded in the P2P network using flooding (an approach similar to one applied in the unstructured P2P network Gnutella [41], [119]). However, this approach has a large volume of network messages generated due to flooding. To avoid this, a Time to Live (TTL) field is associated with every message, i.e. the peers stop forwarding a query message once the TTL expires. To an extent, this approach can limit the network message traffic, but the search query results may not be deterministic in all cases. Thus, the proposed approach can not guarantee to find the desired resource even though it exists in the network.

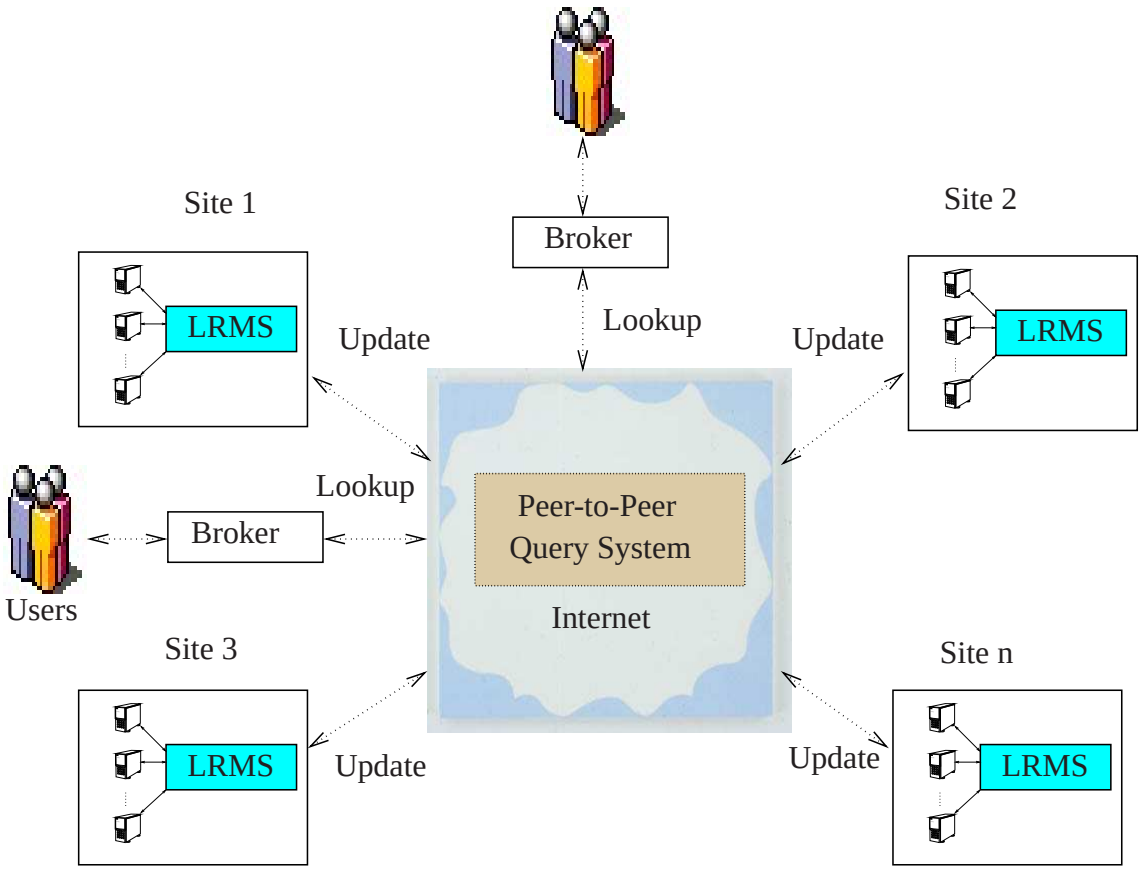


Figure 3.1: Brokering and resource queries.

Recently, organising a GRIS over structured P2P networks has been widely explored. Structured P2P networks offer deterministic search query results with logarithmic bounds on network message complexity. Structured P2P look-up systems including Chord [161], CAN [140], Pastry [143] and Tapestry [184] are primarily based on Distributed Hash Tables (DHTs). DHTs provide hash table like functionality at the Internet scale. A DHT is

a data structure that associates a key with a data. Entries in the distributed hashtable are stored as a (key,data) pair. A data can be looked up within a logarithmic overlay routing hops if the corresponding key is known. Fig. 3.1 shows an abstract model for organising resource brokering systems over a P2P query system. The brokers access the resource information by issuing lookup queries. The resource providers register the resource information through update queries.

It is widely accepted that DHTs are the building blocks for next-generation large scale decentralised systems. Some of the example distributed systems that utilizes DHT routing substrate include distributed databases [95], group communication [38], E-mail services [123], resource discovery systems [15, 42, 128, 150, 168] and distributed storage systems [56]. Current implementations of DHTs are known to be efficient for 1-dimensional queries [95] such as "find all resources that match the given search point". In this case, distinct attribute values are specified for resource attributes. Extending DHTs to support d-dimensional range queries such as finding all resources that overlap a given search space is a complex problem. Range queries are based on range of values for attributes rather than on a specific value. Current works including [10, 24, 33, 42, 52, 128, 134, 150, 159, 168] have studied and proposed different solutions to this problem.

3.1.2 Conceptual Design of a Distributed Resource Indexing System

A layered architecture to build a distributed resource indexing system is shown in Fig. 3.2. The key components of a Internet-based resource indexing system includes:

- **Resource layer:** This layer consists of all globally distributed resources that are directly connected to the Internet. The range of resources include desktop machines, files, supercomputers, computational clusters, storage devices, databases, scientific instruments and sensor networks. A computational resource can run variants of operating systems (such as UNIX or Windows) and queuing systems (such as Condor, Alchemi, SGE, PBS,LSF).

- **Lookup layer:** This layer offers core services for indexing resources at the Internet scale. The main components at this layer are the middlewares that support

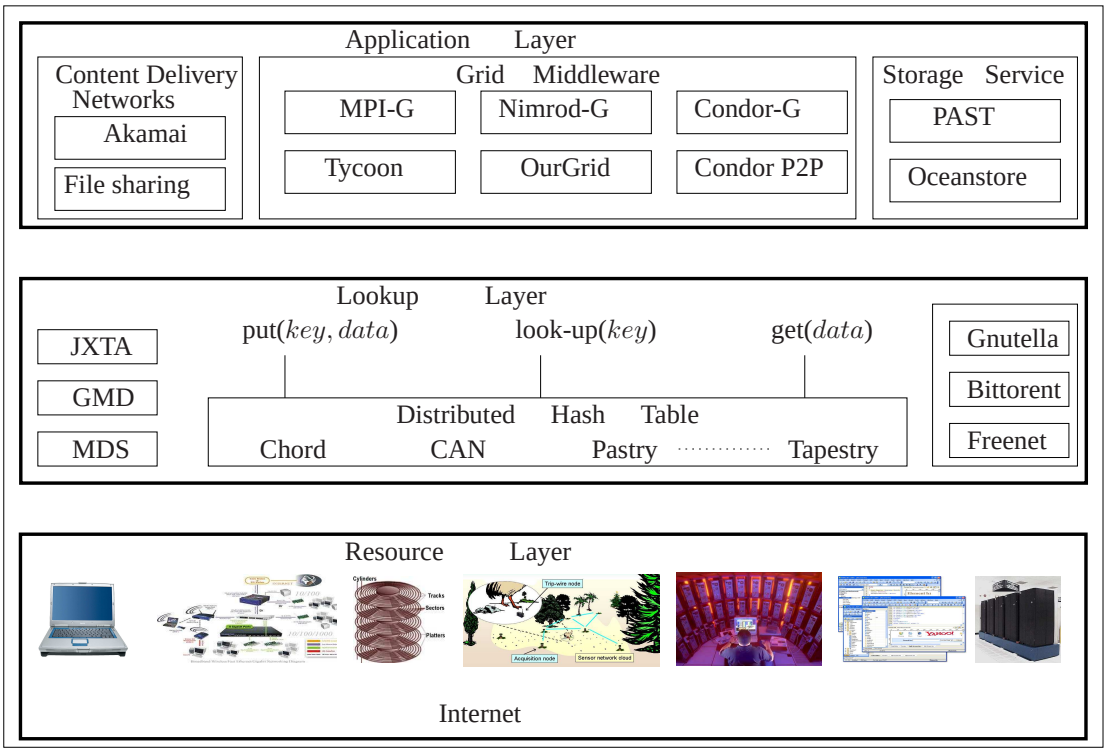


Figure 3.2: Distributed resource indexing: a layered approach.

Internet-wide resource look-ups. Recent proposals at this layer have been utilizing structured P2P protocols such as Chord, CAN, Pastry and Tapestry. DHTs offer deterministic search query performance while guaranteeing logarithmic bounds on the network message complexity. Other, middlewares at this layer includes JXTA [173], Grid Market Directory (GMD) [181] and unstructured P2P substrates such as Gnutella [41] and Freenet [49].

- **Application layer:** This layer includes the application services in various domains including: (i) Grid computing; (ii) distributed storage; (iii) P2P networks; and (iv) Content Delivery Networks (CDNs) [148], [131]. Grid computing systems including Condor-Flock P2P [29] uses services of Pastry DHT to index condor pools distributed over the Internet. Grid brokering system such as the Nimrod-G utilizes directory services of Globus [70] for resource indexing and superscheduling. The OurGrid superscheduling framework incorporates JXTA for enabling communication between OGPeers in the network. Distributed storage systems including PAST [59] and OceanStore [107] utilizes services of DHTs such as Pastry and

Tapestry for resource indexing.

The rest of this chapter is organised as follows. Section 3.2 presents taxonomies related to general computational resources' attributes, look-up queries and organisation model. Section 3.3 discusses taxonomies for P2P network organisation, d-dimensional data distribution mechanism and query routing mechanism. Section 3.4 summarizes various algorithms that model GRIS over a P2P network. Section 3.5 compares the surveyed algorithms based on their scalability and index load-balancing capability. Section 3.6 provides recommendation on utilising the surveyed approaches in implementing a resource discovery system. Finally, the chapter with discussion on open issues in Section 3.7 and conclusion in Section 3.8.

3.2 Resource Taxonomy

The taxonomy for a computational Grid resource is divided into the following (refer to Fig. 3.3): (i) resource organisation; (ii) resource attribute; and (iii) resource query.

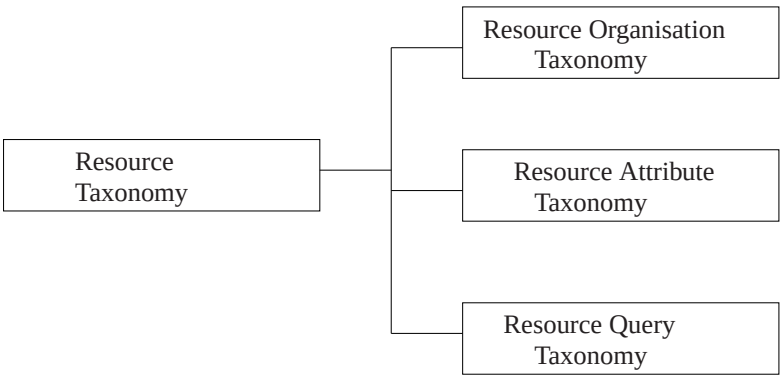


Figure 3.3: Resource taxonomy.

3.2.1 Resource/GRIS organisation

The taxonomy defines GRIS organisation as (refer to Fig. 3.4) :

- Centralised: Centralisation refers to the allocation of all query processing capability to single resource. The main characteristics of a centralised approach include control and efficiency. All look-up and update queries are sent to a single entity in the

system. GRISes including RGMA [183] and GMD [181] are based on centralised organisation.

- Hierarchical: A hierarchical approach links GRIS's either directly or indirectly, and either vertically or horizontally. The only direct links in a hierarchy are from the parent nodes to their child nodes. A hierarchy usually forms a tree like structure. GRIS system including MDS-3 [54] and Ganglia [145] are based on this network model.

- Decentralised: No centralised control, complete autonomy, authority and query processing capability is distributed over all resources in the system. The GRIS organised under this model is fault-tolerant, self-organising and is scalable to large number of resources. More details on this organisation can be found in Section 3.3.

  There are four fundamental challenges related to different organisation models including: (i) scalability; (ii) adaptability; (iii) availability; and (iv) manageability. Centralised models are easy to manage but do not scale well. When network links leading to the central server get congested or fail, then the performance suffers. Hence, this approach may not adapt well to dynamic network conditions. Further, it presents a single point of failure, so overall availability of the system degrades considerably. Hierarchical organisation overcomes some of these limitations including scalability, adaptability and availability. However, these advantages over a centralised model comes at the cost of overall system manageability. In this case, every site specific administrator has to periodically ensure the functionality of their local daemons. Further, the root node in the system may present a single point failure similar to the centralised model. Decentralised systems, including P2P, are coined as highly scalable, adaptable to network conditions and highly available. But manageability is a complex task in P2P networks as it incurs a lot of network traffic.

3.2.2 Resource Attribute

A compute Grid resource is described by a set of attributes which is globally known to the application superschedulers. The superscheduler which is interested in finding a resource

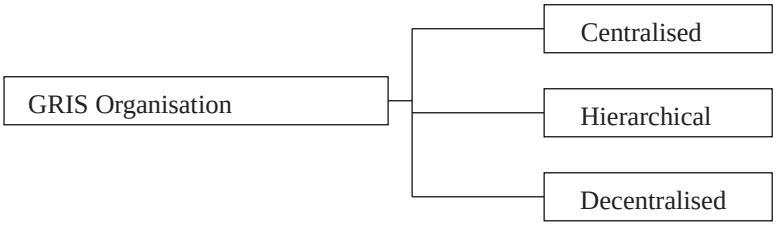


Figure 3.4: Resource organisation taxonomy.

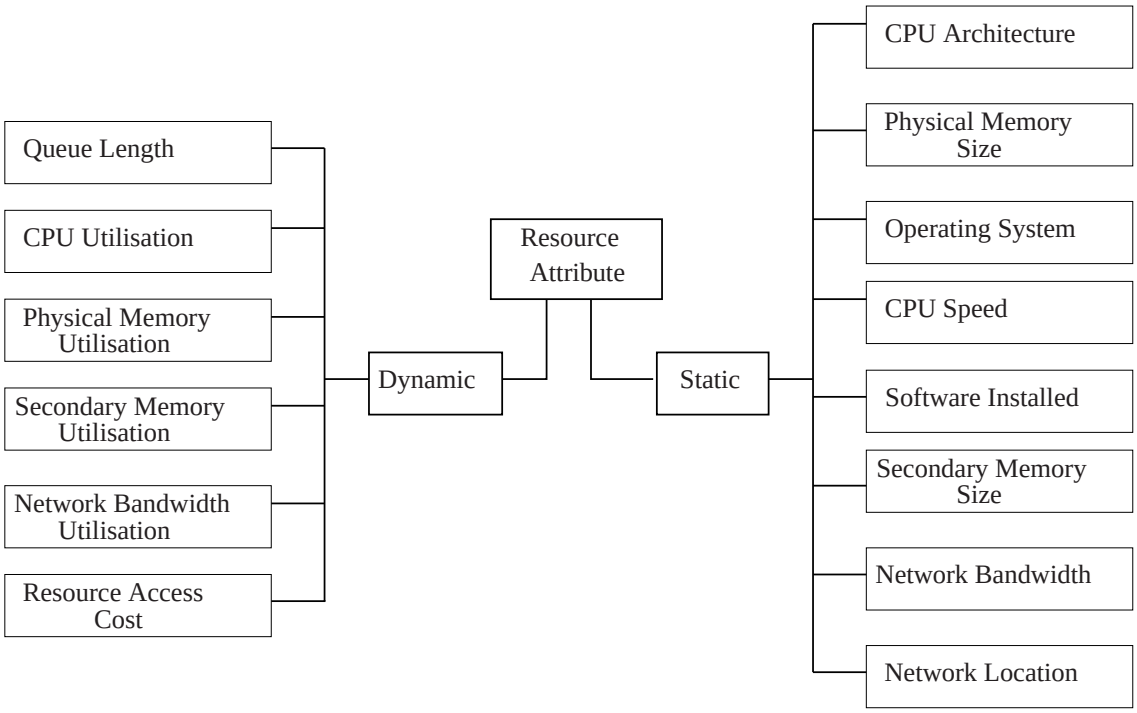


Figure 3.5: Resource attribute taxonomy.

to execute a user's job issues queries to GRIS. The queries are a combination of desired attribute values or their ranges, depending on the user's job composition. In general, compute resources have two types of attributes: (i) static or fixed value attributes such as: type of operating system installed, network bandwidth (both LAN and WAN interconnection), network location, CPU speed, CPU architecture, software library installed and storage capacity (including physical and secondary memory); and (ii) dynamic or range valued attributes such as CPU utilisation, physical memory utilisation, free secondary memory size, current usage price and network bandwidth utilisation. Figure 3.5 depicts the resource attribute taxonomy.

3.2.3 Resource Query

The ability of superschedulers such as MyGrid, Grid-Federation Agent, Nimrod-G, NASA-Scheduler, Condor-Flock P2P to make effective application scheduling decision is directly governed by the efficiency of GRIS. Superschedulers need to query a GRIS to compile information about resource's utilisation, load and current access price for formulating the efficient schedules. Further, a superscheduler can also query a GRIS for resources based on selected attributes such as nodes with large amounts of physical and secondary memory, inter-resource attributes such as network latency, number of routing hops or physical attributes such as geographic location. Similarly, the resource owners query a GRIS to determine supply and demand pattern and accordingly set the price. The actual semantics of the resource query depends on the underlying Grid superscheduling model or Grid system model.

Resource Query Type

Superscheduling or brokering systems require two basic types of queries: (i) resource look-up query (RLQ); and (ii) resource update query (RUQ). An RLQ is issued by a superscheduler to locate resources matching a user's job requirements, while an RUQ is an update message sent to a GRIS by a resource owner about the underlying resource conditions. In Condor-flock P2P system, flocking requires sending RLQs to remote pools for resource status and the willingness to accept remote jobs. Willingness to accept remote jobs is a policy specific issue. After receiving an RLQ message, the contacted pool manager replies with an RUQ that includes the job queue length, average pool utilization and number of resources available. The distributed flocking is based on the P2P query mechanism. Once the job is migrated to the remote pool, basic matchmaking [135] mechanism is applied for resource allocation. In Table 4.1, we present RLQ and RUQ queries in some well-known superscheduling systems.

An Example Superscheduling Resource Query

In this section we briefly analyse the superscheduling query composition in the superscheduling system called Tycoon [109]. The Tycoon system applies market-based principles, in particular an auction mechanism, for resource management. Auctions are completely independent without any centralised control. Every resource owner in the system coordinates its own auction for local resources. The Tycoon system provides a centralised Service Location Service (SLS) for superschedulers to index resource auctioneers' information. Auctioneers register their status with the SLS every 30 seconds. If an auctioneer fails to update its information within 120 seconds then the SLS deletes its entry. Application level superschedulers contact the SLS to gather information about various auctioneers in the system. Once this information is available, the superschedulers (on behalf of users) issue bids for different resources (controlled by different auctions), constrained by resource requirement and available budget. A resource bid is defined by the tuple (h, r, b, t) where h is the host to bid on, r is the resource type, b is the number of credits to bid, and t is the time interval over which to bid. Auctioneers determine the outcome by using a bid-based proportional resource sharing economy model.

Auctioneers in the Tycoon superscheduling system send an RUQ to the centralised GRIS (referred to as service local services). The update message consists of the total number of bids currently active for each resource type and the total amount of each resource type available (such as CPU speed, memory size, disk space). An auctioneers RUQ has the following semantics:

$$total\ \ bids = 10\ \ \&\&\ \ CPU\ \ Arch = \text{“}pentium\text{”}\ \ \&\&\ \ CPU\ \ Speed = 2\ GHz\ \ \&\&\ \ Memory = 512$$

Similarly, the superscheduler, on behalf of the Tycoon users, issues an RLQ to the GRIS to acquire information about active resource auctioneers in the system. A user resource look-up query has the following semantics:

$$return\ \ auctioneers\ \ whose\ \ CPU\ \ Arch = \text{“}i686\text{”}\ \ \&\&\ \ CPU\ \ Speed \geq$$

1 GHz && $Memory \geq 256$

Table 3.1: Resource query in superscheduling or brokering systems.

System Name	Resource Lookup Query	Resource Update Query	GRIS Model
Condor-Flock P2P	Query remote pools in the routing table for resource status and resource sharing policy	Queue length, average pool utilization and number of resources available	Decentralised
Grid-Federation	Query decentralised federation directory for resources that matches user's job QoS requirement (CPU architecture, no. of processors, available memory, CPU speed)	Update resource access price and resource conditions (CPU utilisation, memory, disk space, no. of free processors)	Decentralised
Nimrod-G	Query GMD or MDS for resources that matches jobs resource and QoS requirement	Update resource service price and resource type available	Centralised
Condor-G	Query for available resource using Grid Resource Information Protocol (GRIP), then individual resources are queried for current status depending on superscheduling method	Update resource information to MDS using GRRP	Centralised
Our-Grid	MyPeer queries OGPeer for resources that match user's job requirements	Update network of favors credit for OurGrid sites in the community	Decentralised
Gridbus Broker	Query GMD or MDS for resources that matches jobs resource and QoS requirement	Update resource service price and resource type available	Centralised
Tycoon	Query for auctioneers that are currently accepting bids and matches user's resource requirement	Update number of bids currently active and current resource availability condition	Centralised
Bellagio	Query for resources based on CPU load, available memory, inter-node latency, physical and logical proximity	Update resource conditions including CPU , memory and network usage status	Decentralised
Mosix-Grid	Information available at each node through *gossiping algorithm*	Update CPU usage, current load, memory status and network status	Hierarchical

In Fig. 3.6, we present the taxonomy for GRIS RLQ and RUQ. In general, the resource queries [142] can be abstracted as lookups for objects based on a single dimension or multiple dimensions. Since, a Grid resource is identified by more than one attribute, an RLQ or RUQ is always d-dimensional. Further, both the 1-dimensional and d-dimensional query can specify different kinds of constraints on the attribute values. If

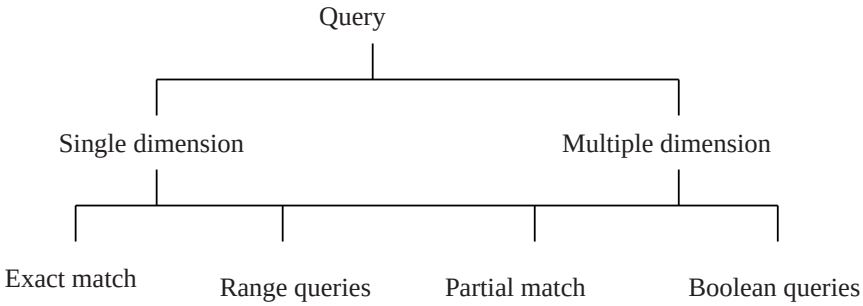


Figure 3.6: Resource query taxonomy.

the query specifies a fixed value for each attribute then it is referred to as a *d-dimensional Point Query* (DPQ). However, in case the query specifies a range of values for attributes, then it is referred to as a *d-dimensional Window Query* (DWQ) or *d-dimensional Range Query* (DRQ). Depending on how values are constrained and searched for, these queries are classified as:

- Exact match query: The query specifies the desired values for all resource attributes sought. For example, Architecture='x86' and CPU-Speed='3 Ghz' and type='SMP' and price='2 Grid dollars per second' and RAM='256 MB' and No. of processors=10 and Secondary free space='100 MB' and Interconnect bandwidth='1 GB/s' and OS='linux'. (Multiple Dimension Exact Match Query).

- Partial match query: Only selected attribute values are specified. For example, Architecture='sparc' and type='SMP' and No. of processors=10. (Multiple Dimension Partial Match Query).

- Range queries: Range values for all or some attributes are specified. For example, Architecture='Macintosh' and type='Cluster' and '1 GHz' $\leq$ CPU-Speed $\leq$ '3 GHz' and '512MB' $\leq$ RAM $\leq$ '1 GB'. (Multiple Dimension Range Query).

- Boolean queries: All or some attribute values satisfying certain boolean conditions. Such as, ((not RAM $\leq$ '256 MB') and not No. of processors $\leq$ 5). (Multiple Dimension Boolean Query).

3.3 P2P Taxonomy

The taxonomy for P2P based GRIS is divided into the following (refer to Fig. 3.7): (i) P2P network organisation; (ii) data organisation; and (iii) d-dimensional query routing organisation.

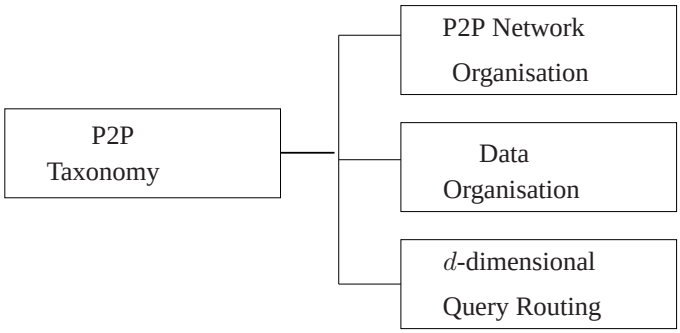


Figure 3.7: Peer-to-Peer network taxonomy.

3.3.1 P2P Network Organisation

The network organisation refers to how peers are logically structured from the topological perspective. Fig. 3.8 shows the network organisation taxonomy of general P2P systems. Two categories are proposed in P2P literature [122]: unstructured and structured. An unstructured system is typically described by a power law random graph model [23, 50], as peer connections are based on the popularity of content. These systems do not put any constraints on placement of data items on peers and how peers maintain their network connections. Detailed evaluation and analysis of network models [34, 101] for unstructured systems can be found in [118]. Unstructured systems including Napster, Gnutella and Kazaa offer differing degrees of decentralisation. The degree of decentralisation refers to the extent peers can function independently with respect to efficient object look-up and query routing. Our taxonomy classifies unstructured systems as *deterministic* or *non-deterministic* [118].

Deterministic system means that a look-up operation will be successful within predefined bounds. Systems including Napster, BitTorrent fall into this category. In these systems, the object lookup operation is centralised while download is decentralised. Under

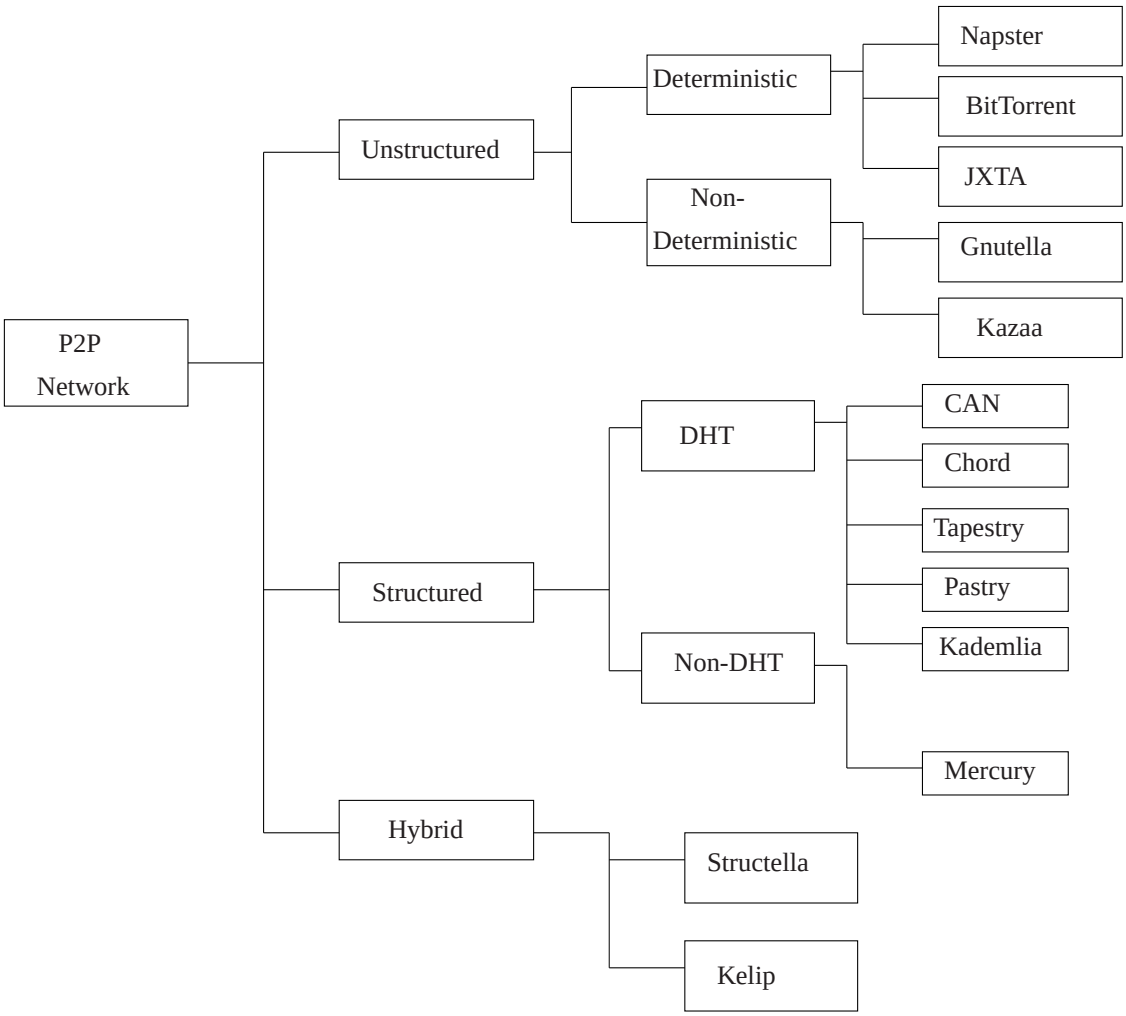


Figure 3.8: Peer-to-Peer network organisation taxonomy.

centralised organisation, a specialised (index) server maintains the indexes of all objects in the system (e.g Napster, BitTorrent). The resource queries are routed to index servers to identify the peers currently responsible for storing the desired object. The index server can obtain the indexes from peers in one of the following ways: (i) peers directly inform the server about the files they are currently holding (e.g. Napster); or (ii) by crawling the P2P network (an approach similar to a web search engine). The look up operations in these systems is deterministic and is resolved with a complexity of $O(1)$. We classify JXTA as an unstructured P2P system that offers deterministic search performance. At the lowest level JXTA is a routing overlay, not unlike routers that interconnect to form a network. Hence there is no structure, but there is a routing algorithm that allows any router to router communication. In JXTA both object look-up and download operations

are completely decentralised.

Other unstructured systems including Gnutella, Freenet, FastTrack and Kazaa offer non-deterministic query performance. Unlike Napster or BitTorrent, both object lookup and download operation in these systems are decentralised. Each peer maintains indexes for the objects it is currently holding. In other words, indexes are completely distributed. The Gnutella system employs a query flooding model for routing object queries. Every request for an object is flooded (broadcasted) to the directly connected peers, which in turn flood their neighboring peers. This approach is used in the GRIS model proposed by [97]. Every RLQ message has a TTL field associated with it (i.e. maximum number of flooding hops/steps allowed). Drawbacks for flood-based routing include high network communication overhead and non-scalability. This issue is addressed to an extent in FastTrack and Kazaa by introducing the notion of super-peers. This approach reduces network overhead but still uses a flooding protocol to contact super-peers.

Structured systems such as DHTs offer deterministic query search results within logarithmic bounds on network message complexity. Peers in DHTs such as Chord, CAN, Pastry and Tapestry maintain an index for $O(\log(n))$ peers where n is the total number of peers in the system. Inherent to the design of a DHT are the following issues [14]: (i) generation of node-ids and object-ids, called keys, using cryptographic/randomizing hash functions such as SHA-1 [103, 133]. The objects and nodes are mapped on the overlay network depending on their key value. Each node is assigned responsibility for managing a small number of objects; (ii) building up routing information (routing tables) at various nodes in the network. Each node maintains the network location information of a few other nodes in the network; and (iii) an efficient look-up query resolution scheme. Whenever a node in the overlay receives a look-up request, it must be able to resolve it within acceptable bounds such as in $O(\log(n))$ time. This is achieved by routing the look-up request to the nodes in the network that are most likely to store the information about the desired object. Such probable nodes are identified by using the routing table entries. Though at the core various DHTs (Chord, CAN, Pastry etc.) are similar, still there exists substantial differences in the actual implementation of algorithms including the overlay network construction (network graph structure), routing table maintenance and node join/leave handling. The performance metrics for evaluating a DHT include

fault-tolerance, load-balancing, efficiency of lookups and inserts and proximity awareness [115]. In Table-3.2, we present the comparative analysis of Chord, Pastry, CAN and Tapestry based on basic performance and organisation parameters. Comprehensive details about the performance of some common DHTs under churn can be found in [110].

Table 3.2: Summary of the complexity of structured P2P systems.

P2P System	**Overlay Structure**	**Lookup Protocol**	**Network parameter**	**Routing table size**	**Routing complexity**	**join/leave overhead**
Chord	1-dimensional, circular-ID space	Matching key and NodeID	n= number of nodes in the network	$O(\log(n))$	$O(\log n)$	$O((\log n)^2)$
Pastry	Plaxton-style mesh structure	Matching key and prefix in NodeID	n= number of nodes in the network, b=base of the identifier	$O(\log_b(n))$	$O(b\ \log_b(n)+b)$	$O(\log n)$
CAN	d-dimensional ID space	key,value pairs map to a point P in the d-dimensional space	n= number of nodes in the network, d=number of dimensions	$O(2\ d)$	$O(d\ n^{1/d})$	$O(2\ d)$
Tapestry	Plaxton-style mesh structure	Matching suffix in NodeID	n= number of nodes in the network, b=base of the identifier	$O(\log_b(n))$	$O(b\ \log_b(n)+b)$	$O(\log n)$

Other classes of structured systems such as Mercury do not apply randomising hash functions for organising data items and nodes. The Mercury system organises nodes into a circular overlay and places data contiguously on this ring. As Mercury does not apply hash functions, data partitioning among nodes is non-uniform. Hence it requires an explicit load-balancing scheme. In recent developments, new generation P2P systems have evolved to combine both unstructured and structured P2P networks. We refer to this class of systems as hybrid. Structella [36] is one such P2P system that replaces the random graph model of an unstructured overlay (Gnutella) with a structured overlay, while still adopting the search and content placement mechanism of unstructured overlays to support complex queries. Other hybrid P2P design includes Kelips [89] and its variants. Nodes in Kelips overlay periodically gossip to discover new members of the network, and during

this process nodes may also learn about other nodes as a result of lookup communication. Other variant of Kelips [87] allows routing table entries to store information for every other node in the system. However, this approach is based on assumption that system experiences low churn rate [110]. Gossiping and one-hop routing approach has been used for maintaining the routing overlay in the work [158]. In Table 3.3, we summarize the different P2P routing substrate that are utilized by the existing algorithms for organising a GRIS.

3.3.2 Data Organisation

Traditionally, DHTs have been efficient for 1-dimensional queries such as finding all resources that match the given attribute value. Extending DHTs to support DRQs, to index all resources whose attribute value overlap a given search space, is a complex problem. DRQs are based on ranges of values for attributes rather than on specific values. Compared to 1-dimensional queries, resolving DRQs is far more complicated, as there is no obvious total ordering of the points in the attribute space. Further, the query interval has varying size, aspect ratio and position such as a window query. The main challenges involved in enabling DRQs in a DHT network [80] include efficient: (i) data distribution mechanisms; and (ii) data indexing or query routing techniques. In this section, we discuss various data distribution mechanisms while we analyse data indexing techniques in the next section.

A data distribution mechanism partitions the d-dimensional [20, 77] attribute space over the set of peers in a DHT network. Efficiency of the distribution mechanism directly governs how the query processing load is distributed among the peers. A good distribution mechanism should possess the following characteristics [80]:

- Locality: tuples or data points nearby in the attribute space should be mapped to the same node, hence limiting the lookup complexity.
- Load balance: the number of data points indexed by each peer should be approximately the same to ensure uniform distribution of query processing [25, 139].
- Minimal metadata: prior information required for mapping the attribute space to the peer space should be minimal.

- Minimal management overhead: during peer join and leave operation, update policies such as the transfer of data points to a newly joined peer should cause minimal network traffic.

In the current P2P indexing literature (refer to section 3.4), d-dimensional data distribution mechanisms based on the following structures have been proposed (refer to Fig. 3.10): (i) space filling curves; (ii) tree-based structures; and (iii) variant of SHA-1/2 hashing. In Table 3.4, we summarise various data structures used in different algorithms for d-dimensional data distribution. Further, in Table 3.5, we present a classification of the existing algorithms based on the number of routing overlays utilized for managing d-dimensional data.

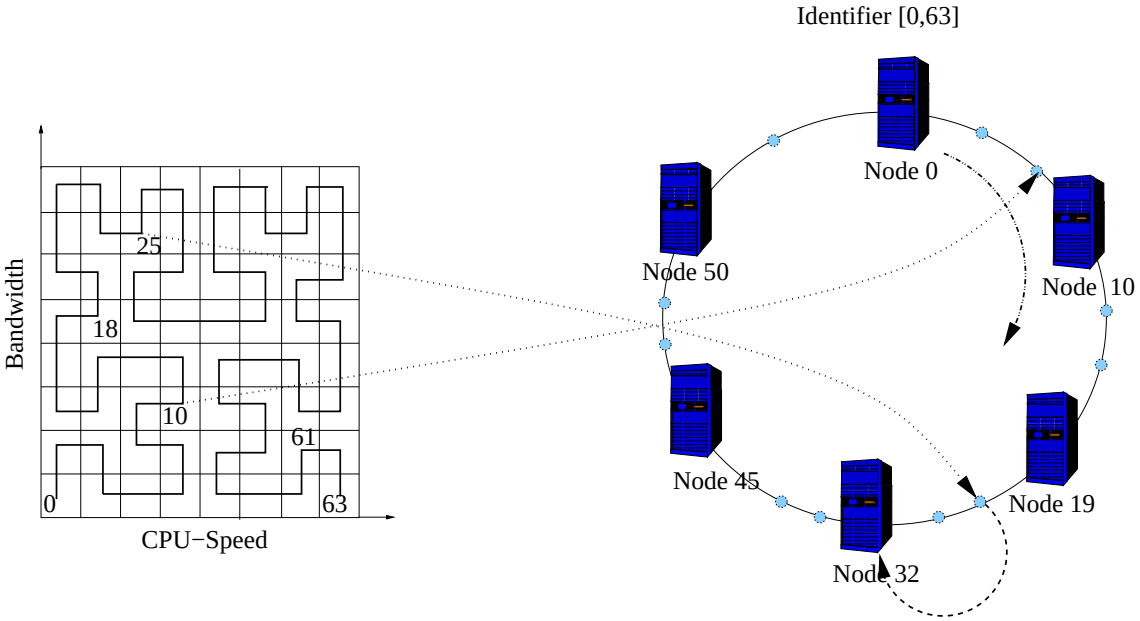


Figure 3.9: An example 2-dimensional data organisation in Squid based on Hilbert SFC.

The Space Filling Curves data structure (*SFCs*) [11, 99] includes the Z-curve [129] and Hilbert's curve [100]. SFCs map the given d-dimensional attribute space into a 1-dimensional space. The work in [10] utilises space-filling curves (SFC), in particular the reverse Hilbert SFC for mapping a 1-dimensional attribute space to a two-dimensional CAN P2P space. Similarly, the work in [150] uses the Hilbert SFC to map a d-dimensional index space into a 1-dimensional space. The resulting 1-dimensional indexes are contiguously mapped on a Chord P2P network. The approach proposed in [80] utilises Z-curves

for mapping d-dimensional space to 1-dimensional space. SFCs exhibit the locality property by mapping the points that are close in d-dimensional space to adjacent spaces in the 1-dimensional space. However, as the number of dimensions increases, locality becomes worse since SFCs suffer from "curse of dimensionality" [104]. Further, SFC based mapping fails to uniformly distribute the load among peers if the data distribution is skewed. Hence, this leads to a non-uniform query processing load for peers in the network. In Fig. 3.9 2-dimensional attribute space is contiguously mapped to the 1-dimensional Chord space. Hilbert SFC index forms the basis for transforming a d-dimensional attribute space to a 1-dimensional key space. The attribute point with Hilbert SFC index value 25 is mapped to the Node 32 in the Chord space.

Some of the recent works [52, 81, 134, 166] utilize tree-based data structures for organising the data. The approach proposed in [166] adopts the MX-CIF quadtree [147] index for P2P networks. A distributed quadtree index assigns regions of space (a quadtree block) to the peers. If the extent of a spatial object goes beyond a quadtree block, then recursive subdivision of the that block can be performed. With a good base hash function one can achieve a uniform random mapping of the quadtree blocks to the peers in the network. This approach will map two quadtree blocks that are close to each other to totally different locations on the Chord space. Another recent work called DragonFly [108], uses the same base algorithm with an enhanced load balancing technique called recursive bisection [21]. Recursive bisection works by dividing a cell/block recursively into two halves until a certain load condition is met. The load condition is defined based on two load parameters known as the load limit and the load threshold. Hence, this approach has better load balancing properties as compared to the SFC-based approaches in the case of a skewed data set.

DragonFly builds a d-dimensional Cartesian space based on the Grid resource attributes, where each attribute represents a single dimension. The logical d-dimensional index assigns regions of space to the peers. If a peer is assigned a region (index cell) in the d-dimensional space, then it is responsible for handling all the activities related to the subscription and publication associated with the region. Each cell is uniquely identified by its centroid, termed as the *control point*.

Other approaches including [33, 168] manipulate existing SHA-1/2 hashing for map-

ping d-dimensional data to the peers. MAAN addresses the 1-dimensional range query problem by mapping attribute values to the Chord identifier space via a uniform locality preserving hashing scheme. A similar approach is also utilized in [170]. However, this approach shows poor load balancing characteristics when the attribute values are skewed.

To conclude, the choice of data structure is directly governed by the data distribution pattern. A data structure that performs well for a particular data-set may not do the same in case the distribution changes. Additional techniques such as peer virtualization (as proposed in Chord) or multiple realities (as proposed in CAN) may be utilized to improve the query processing load.

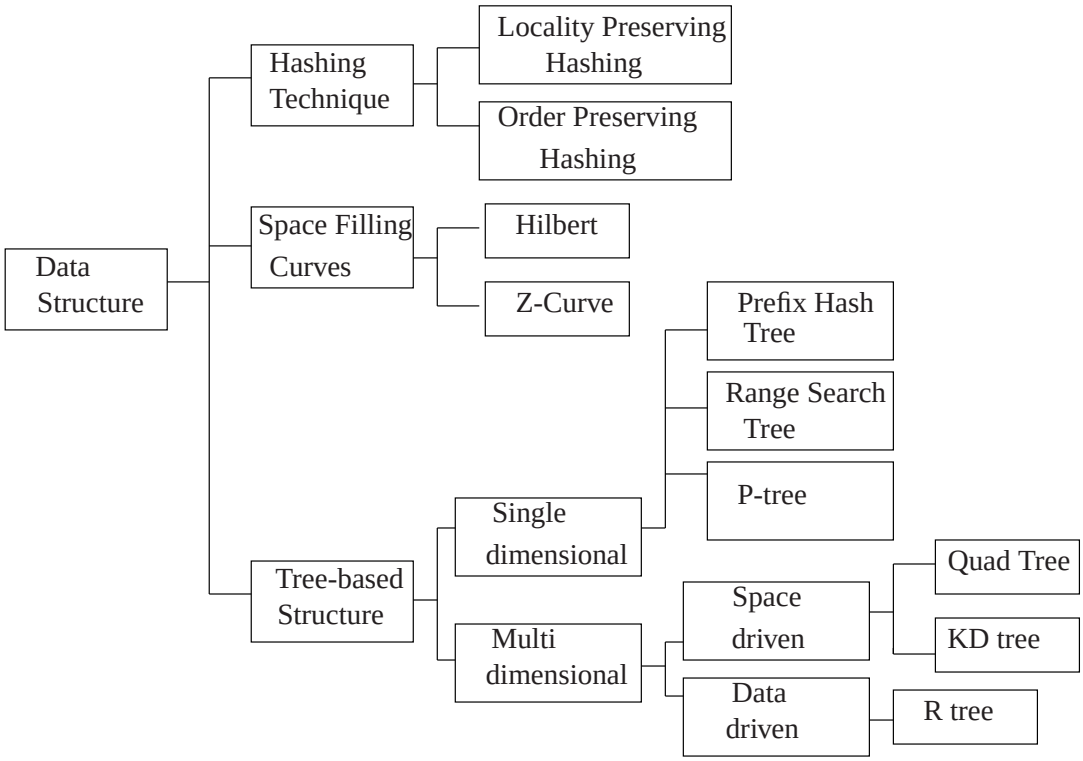


Figure 3.10: Data structure taxonomy.

3.3.3 D-dimensional Query Routing

DHTs guarantee deterministic query lookup with logarithmic bounds on network message cost for 1-dimensional queries. However, Grid RLQs are normally DPQ or DRQ. Hence, existing routing techniques need to be augmented in order to efficiently resolve a DRQ.

Various data structures that we discussed in previous section effectively create a logical d-dimensional index space over a DHT network. A look-up operation involves searching for a index or set of indexes in a d-dimensional space. However, the exact query routing path in the d-dimensional logical space is directly governed by the data distribution mechanism (i.e. based on the data structure that maintains the indexes).

In this context, various approaches have proposed different routing/indexing heuristics. Efficient query routing algorithm should exhibit the following characteristics [80]:

- Routing load balance: every peer in the network on the average should route forward/route approximately same number of query messages.

- Low per-node state: each peer should maintain a small number of routing links hence limiting new peer join and peer state update cost. In Table 3.4, we summarize the query look-up complexity involved with the existing algorithms.

Resolving a DRQ over a DHT network that utilises SFCs for data distribution consists of two basic steps [150]: (i) mapping the DRQ onto the set of relevant clusters of SFC-based index space; and (ii) routing the message to all peers that fall under the computed SFC-based index space. The simulation based study proposed in [80] has shown that SFCs (Z-curves) incur constant routing costs irrespective of the dimensionality of the attribute space. Routing using this approach is based on a skip graph, where each peer maintains $O(\log(n))$ additional routing links in the list. However, this approach has serious load balancing problems that need to be fixed using external techniques [79].

Routing DRQs in DHT networks that employ tree-based structures for data distribution requires routing to start from the root node. However, the root peer presents a single point of failure and load imbalance. To overcome this, the authors in [166] introduced the concept of fundamental minimum level. This means that all the query processing and the data storage should start at that minimal level of the tree rather than at the root. Another approach [80] utilises a P2P version of a Kd-tree [19] for mapping d-dimensional data onto a CAN P2P space. The routing utilises the neighboring cells of the data structure. The nodes in this network that manage a dense region of space are likely to have large number of neighbors, hence leading to an unbalanced routing load.

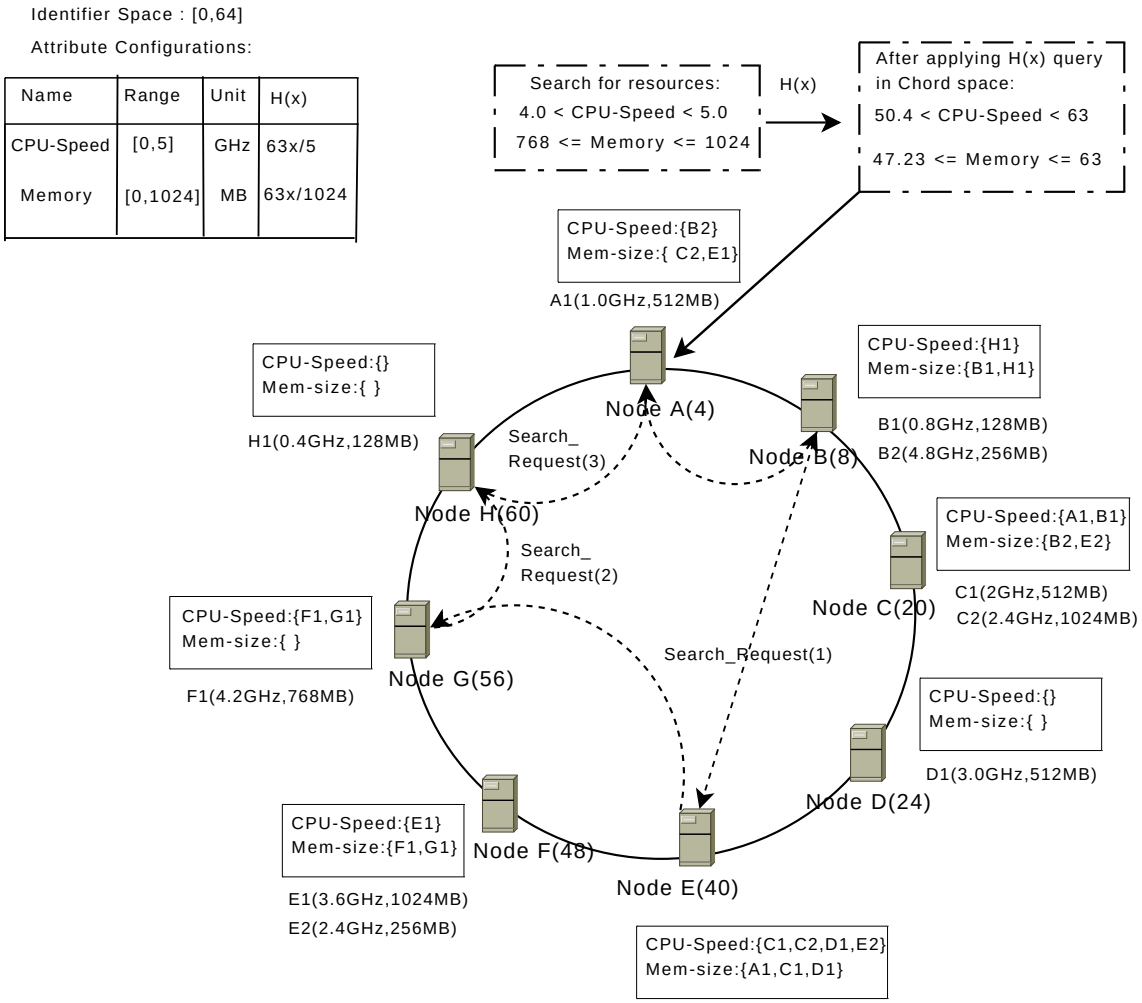


Figure 3.11: An example for single-attribute-dominated query resolution in MAAN approach.

Other approaches based on variants of standard hashing schemes (such as MAAN) apply different heuristics for resolving range queries. The single-attribute dominated query routing (SAQDR) heuristic abstracts resource attributes into two categories: (i) dominant attribute; and (ii) non-dominant attribute. The underlying system queries for the node that maintains the index information for the dominant attribute. Once such a node is found, the node searches its local index information looking at satisfying the values for other non-dominant attributes in the DRQ. The request is then forwarded to the next node which indexes the subsequent range value for the dominant attribute. This approach comprehensively reduces the number of routing steps needed to resolve a DRQ. However, this approach suffers from routing load-imbalance in the case of a skewed attribute space.

Fig. 3.11 shows an example of single attribute dominated routing in MAAN. This MAAN network has the identifier space in the range [0, 2^6]. The attribute ranges and corresponding locality preserving hash functions are also shown in the figure. The node A initiates a $search\_request(1)$ with hash value of lower bound of the dominated attribute (CPU-Speed). The $search\_request(1)$ has following semantics in the Chord key space: $look\_up$ (50.4, CPU-Speed, (4.0GHz, 5.0GHz), Memory-Size $\in$ [768MB, 1024MB], $\{$ EMPTY $\}$). The look-up request is routed to the current successor node G using the standard Chord method. Node G currently owns the key related to both the attribute values CPU-Speed and Memory, it augments the corresponding value to the result set X and forwards the query in the network to look-up for the upper bound on the CPU-Speed. The query finally terminates at Node A which happens to be initiator node as well. In Table 3.6, we present the classification of the existing algorithms based on query resolution heuristic, and data locality preserving characteristics.

Table 3.3: Classification based on P2P routing substrate.

Routing Substrate	**Network Organisation**	**Distributed Indexing Algorithm Name**
Chord	Structured	PHT [134], MAAN [33], Dgrid [168], Adaptive [81], DragonFly [108], QuadTree [166], Pub/Sub-2 [170], P-tree [52], Squid [150]
Pastry	Structured	XenoSearch [159], AdeepGrid [42], Pub/Sub-1 [165]
CAN	Structured	HP-protocol [10], Kd-tree [80],Meghdoot [88], Z-curve [80], Super-P2P R*-Tree [114]
Bamboo	Structured	SWORD [128]
Epidemic-DHT [87]	Hybrid	XenoSearch-II [158]
Others	Unstructured	Mercury [24], JXTA search [85], P2PR-tree [124]

Table 3.4: Classification based on data structure applied for enabling ranged search and look-up complexity.

Algorithm Name	**Data Structure**	**Lookup Complexity**
PHT [134]	Trie	$O(\log \|D\|)$; D is the total number of bits in the binary string representation, for 1-dimensional range query
MAAN [33]	Locality preserving hashing	$O(n \times \log n + n \times s_{min})$, s_{min} is the minimum range selectivity per dimension; n total peers
Dgrid [168]	SHA-1 hashing	$O(\log_2 Y)$ for each dimension, Y is the total resource type in the system
SWORD [128]	N.A.	N.A.
JXTA search [85]	RDBMS	N.A.
DragonFly [108]	QuadTree	$O(E[K] \times (log_2 n + f_{max} - f_{min}))$; n is the total peers in the network; f_{max} is the maximum allowed depth of the tree, f_{min} is the fundamental minimum level, $E[K]$ is the mean number disjoint path traversed for a window query, its distribution is function of the query size
QuadTree [166]	QuadTree	$O(E[K] \times (log_2 n + f_{max} - f_{min}))$; n is the total peers in the network; f_{max} is the maximum allowed depth of the tree, f_{min} is the fundamental minimum level, $E[K]$ is the mean number disjoint path traversed for a window query, its distribution is function of the query size
Pub/Sub-2 [170]	Order preserving hashing	$1/2 \times O(\log n)$; Equality query, n is total peers, $1/2 \times O(n_s \log n)$, n_s is step factor; for ranged query, in a 1-dimensional search space
P-tree [52]	Distributed B-+ tree	$O(m + \log_d n)$; n is total peers, m is number of peers in selected range, d is order of the 1-dimensional distributed B-tree
Pub/Sub-1 [165]	SHA-1 hashing	$O(n_r \log n)$; n is total peers, n_r is the number of range intervals searched in a 1-dimensional search space
XenoSearch [159]	SHA-1 hashing	N.A.
XenoSearch-II [158]	Hilbert space filling curve	N.A.
AdeepGrid [42]	SHA-1 hashing	N.A.
HP-protocol [10]	Reverse hilbert space filling curve	N.A.
Squid [150]	Hilbert space filling curve	$n_c \times O(\log n)$; n_c is the total no. of isolated index clusters in the SFC based search index space, n is the total number of peers
Mercury [24]	N.A.	$O((\log n)/k)$; k Long distance links; n is total peers, in a 1-dimensional search space
Adaptive [81]	Range search tree	$O(\log R_q)$; R_q is range selectivity, in a 1-dimensional search space
Kd-tree [80]	Kd-tree, skip pointer based on skip graphs	N.A.
Meghdoot [88]	SHA-1 hashing	$O(dn^{\frac{1}{d}})$, n is the total peers in the network, d is the dimensionality of CAN space
Z-curve [80]	Z-curves, skip pointer based on skip graphs	N.A.
P2PR-tree [124]	Distributed R-tree	N.A.
Super-P2P R*-Tree [114]	Distributed R*-tree	$O(E[k] \times (d/4)(n^{1/d}))$; $E[k]$ is the mean number of MBRs indexed per range query or NN query, d is the dimensionality of the indexed/CAN space, n is the number of peers in the system.

Table 3.5: Classification based on No. of routing overlays for d-dimensional search space.

Single	**Multiple**
JXTA search [85], Dragon-Fly [108], XenoSearch-II [158], SWORD [128], Squid [150], Kd-tree [80], Meghdoot [88], Z-curve [80], QuadTree [166], Dgrid [168], P2PR-tree [124], AdeepGrid [42], Super-P2P R*-Tree [114]	PHT [134], Adaptive [81], Pub/Sub-2 [170], P-tree [52], XenoSearch [159], Pub/Sub-1 [165], MAAN [33], Mercury [24], HPPROTOCOL [10]

Table 3.6: Classification based on query resolution heuristic, data distribution efficiency and data locality preserving characteristic.

Algorithm Name	**Heuristic Name**	**Preserves Data Locality (Yes/No)**
PHT [134]	Chord routing	N.A.
MAAN [33]	Iterative resolution, single attribute dominated routing based on Chord	N.A.
Dgrid [168]	Chord routing	N.A.
SWORD [128]	Bamboo routing	No
JXTA search [85]	Broadcast .	N.A.
DragonFly [108]	Generic DHT routing	No
QuadTree [166]	Generic DHT routing	No
Pub/Sub-2 [170]	Chord routing	N.A.
P-tree [52]	Generic DHT routing	N.A.
Pub/Sub-1 [165]	Pastry routing	N.A.
XenoSearch [159]	Generic DHT routing	N.A.
XenoSearch-II [158]	Generic DHT routing	N.A.
AdeepGrid [42]	Single shot, recursive and parallel searching based on Pastry	No
HP-protocol [10]	Brute force, controlled flooding, directed controlled flooding based on CAN	N.A.
Squid [150]	Generic DHT routing	Yes
Mercury [24]	Range-selectivity based routing	N.A.
Adaptive [81]	Generic DHT routing	N.A.
Kd-tree [80]	Skip pointer based routing	Yes
Meghdoot [88]	CAN based routing	Yes
Z-curve [80]	Skip pointer based routing	Yes
P2PR-tree [124]	Block/group/subgroup pointer based routing	Yes
Super-P2P R*-Tree [114]	CAN based routing	Yes

3.4 Survey of P2P based Grid Information Indexing

3.4.1 Pastry Based Approaches

Pub/Sub-1: Building Content-Based Publish/Subscribe Systems with Distributed Hash Tables

The Publish/Subscribe system [165] is implemented on top of the topic-based Scribe [38] system. The system defines different schema for publication and subscription messages for each application domain (such as a stock market or an auction market). When a request (publication or subscription) is submitted to the system, it is parsed for various index digests. An index digest is a string of characters that is formed by concatenating the attribute type, name, and value of each attribute in the index. An example index digest is $[USD : Price : 100 : Inch : Monitor : 19 : String : Quality : Used]$. The system can support both point and range queries through different query resolution heuristics. The system handles range values by building a separate index hash key for every attribute value in the specified range. This method has serious scalability issues. The proposed approach to overcome this limitation is to divide the range of values into intervals and a separate hash key is built for each such index digest representing that interval. However, this approach can only handle range values of single attribute in a index digest (does not support multi-attribute range value in a single index digest).

XenoSearch: Distributed Resource Discovery in the XenoServer Open Platform

XenoSearch [159] is a resource discovery system built for the XenoServer [90] execution platform. The XenoSearch indexes the resource information that are advertised periodically by the XenoServers. An advertisement contains information about the identity, ownership, location, resource availability, and access prices of a XenoServer. The XenoSearch system converts these advertisements to points in a d-dimensional space, wherein different dimensions represent different attributes (such as topological location, QoS attributes etc). The XenoSearch system is built over the Pastry [143] overlay routing protocol. A separate Pastry ring operates for each dimension with XenoSearch nodes registering separately in each ring. A XenoServer registers for each dimension and derives the overlay

key by hashing its co-ordinate position in that dimension. Effectively, in different dimensions a XenoServer is indexed by different $keys$. The d-dimensional range searches are performed by making a series of search requests in each dimension and finally computing their intersection. Recently, XenoSearch has been enhanced with new search and data placement technique [158]. The new approach puts emphasis upon both the location and resource constraints associated with a search entity.

AdeepGrid: Peer-to-Peer Discovery of Computational Resources for Grid Applications

The proposed [42] GRIS model hashes the d-dimensional static and dynamic resource attributes to the Pastry ID space. The system augments additional 32-bits to the ID or key size (hence resulting key is 160-bit long) as compared to 128-bit in the standard Pastry ring. In this case, the first 128-bits are used to encode the static attributes while the remaining 32-bits for the dynamic attributes. The static part of the Resource ID is mapped to a fixed point while the dynamic part is represented by potentially overlapping arcs on the overlay. Resolving RLQ involves locating the node that currently hosts the desired resource attributes (Resource ID). This is accomplished by utilizing standard Pastry routing. Three different heuristics for resolving the RLQs are proposed: (i) single-shot searching; (ii) recursive searching; and (iii) parallel searching.

3.4.2 Chord Based Approaches

DGRID: A DHT-Based Grid Resource Indexing and Discovery Scheme

Work by Teo et al. [168] extends Chord DHT with the GRIS capability. The unique characteristic about this approach is that the resource information is maintained in the originating domain. Every domain in DGRID designates an index server to the Chord based GRIS network. The index server maintains state and attribute information for the local resource set. The proposed approach intelligently manipulates the existing Chord ID generation scheme to enable a GRIS network. The search or look-up operation in the DGRID is based on Chord look-up primitives. Given a key p, is mapped to a particular virtual index server on the overlay network using the query $get(p)$. The DGRID indexing

approach also supports domain specific resource type search. Overall, the look-up cost is bounded by the underlying Chord protocol i.e. $O(\log N)$. In general the look-up cost for a particular resource type t is $O(\log Y)$, Y is the total number of resource types available in the network.

Adaptive: An Adaptive Protocol for Efficient Support of Range Queries in DHT-based Systems

The work in [81] presents an algorithm to support range queries based on a distributed logical Range Search Tree (RST). Inherently, the RST is a complete and balanced binary tree with each level corresponding to a different data partitioning granularity. The system abstracts the data being registered and searched in the network as a set of attribute-value pairs (AV-pairs). It utilizes the Chord for distributed routing and network management issues. A typical range query with length R_q is resolved by decomposing it into $O(\log(R_q))$ sub-queries. These sub-queries are then sent to the nodes that index the corresponding data. The system supports updates and queries for both static and dynamic resource attributes.

Pub/Sub-2: Content-based Publish-Subscribe Over Structured P2P Networks

The work in [170] presents a content-based publish-subscribe indexing system based on the Chord DHT. The system is capable of indexing d-dimensional index space by having a separate overlay for each dimension. Every i-th dimension i.e. a resource attribute has a distinct data-type, name and value. A attribute name is normally a string, whereas the value can be a string or numeric in any range constrained by the minimum and maximum value along with the attribute's precision. An attribute in a subscription is placed on a node obtained by hashing its value based on the Chord method. A subscription can declare a range of values in the attribute's range. The query is resolved in set of steps, where a step is computing using the maximum, minimum and precision values for the attribute. In the subsequent steps the previous attribute value is incremented by the precision value and mapped to the corresponding Chord node. Updating the range values is done by following the same procedure for all Chord nodes that store the given range of values. The overall message routing complexity depends on the type of constraints defined over the attributes.

In case of equality constraints, the average number of routing hops is $O(1/2\log(n))$. When the constraint is a range then the complexity involved is $O(n_s \times 1/2\log(n))$, where n_s is the step factor.

QuadTree: Using a Distributed Quadtree Index in the Peer-to-Peer Networks

The work in [166] proposes a distributed quad-tree that adopts an MX-CIF quadtree index [147] for accessing spatial data or objects in P2P networks. The work builds upon the region quad-tree data structure. In this case, by applying the fundamental quad-tree decomposition property the underlying two-dimensional square space is recursively decomposed into four congruent blocks until each block is contained in one of the objects in its entirety or is not contained in any of the objects. The distributed quad-tree index assigns regions of d-dimensional space to the peers in a P2P system. Every quad-tree block is uniquely identified by its centroid, termed as the control point. Using the control point, a quad-tree block is hashed to a peer in the network. The Chord method is used for hashing the blocks to the peers in the network. If a peer is assigned a quad-tree block, then it is responsible for processing all query computations that intersects the block. Multiple control points (i.e. quad-tree blocks) can be hashed to the same peer in the network. To avoid a single point of failure at the root level of the quad-tree the authors incorporate a technique called *fundamental minimum level,* f_{min}. This technique means that objects are only allowed to be stored at levels $l \geq f_{min}$ and therefore all the query processing starts at levels $l \geq f_{min}$. The scheme also proposes the concept of a *fundamental maximum level,* f_{max}, which limits the maximum depth of the quad-tree at which objects are inserted.

DragonFly: A Publish-Subscribe Scheme with Load Adaptability

The work in [108] proposes a content-based publish-subscribe system with load adaptability. They apply a spatial hashing technique for assigning data to the peers in the network. The system supports multi-attribute point and range queries. Each distinct attribute is assigned a dimension in a d-dimensional Cartesian space. The d-dimensional Cartesian space is arranged as a tree structure with the domain space mapped to the root node of the tree. In particular, the tree structure is based on a quad tree [147]. To negate a single point of failure at the root node, system adopts a technique called the *fundamental minimum*

level. More details about this technique can be found in [166]. This technique recursively divides the logical space into four quadrants. With each recursion step on a existing quadrant, four new quadrants are generated. Hence, multiple recursion steps basically create a mutli-level tree data structure. Note that, the quad-tree decomposition method is followed only at the f_{min} level, beyond this the index cell is divided depending on its corresponding publish/subscribe load. The tree based organisation of DragonFly introduces parent-child relationships between tree cells. Another important feature of DragonFly is the diagonal hyperplane. In 2-d space, the diagonal hyperplane is a line spanning from the north-west to the south-east vertices of the rectangular space. The hyperplane forms the basis for mapping the subscription and publication objects.

MAAN: A Multi-Attribute Addressable Network for Grid Information Services

Cai et al. [33] present a multi-attribute addressable network (MAAN) approach for enabling a GRIS. They extend the Chord [161] protocol to support DRQs. MAAN addresses the d-dimensional range query problem by mapping the attribute values to the Chord identifier space via a uniform locality preserving hashing. Note that, for every attribute dimension a separate Chord overlay is maintained. For attributes with the numerical values, MAAN applies locality preserving hashing functions to assign an identifier in the m-bit identifier space. The total routing complexity involved in resolving a 1-dimensional range query is $O(\log N + K)$, where $O(\log N)$ is the underlying Chord routing complexity and K is the number of nodes that store values in attribute's range. MAAN also supports multi-attribute query resolution by extending the single-attribute range query routing algorithm.

Squid: Flexible Information Discovery in Decentralised Distributed Systems

Schmidt et al. [150] proposed a GRIS model that utilizes SFCs for mapping d-dimensional attribute space to a 1-dimensional search space. All data elements are described using a sequence of attributes such as memory, CPU speed and network bandwidth. The attributes form the coordinates of a d-dimensional space, while the data elements are the points. This mapping is accomplished using a locality-preserving mapping called Space Filling Curves (*SFC*) [11], [99]. SFCs are used to generate a 1-dimensional index space

from the d-dimensional attribute space, where d is the number of different attribute types. Any range query or query composed of attributes, partial attributes, or wild-cards, can be mapped to regions of the attribute space and subsequently to the corresponding clusters in the SFC. The Chord protocol is utilized to form the overlay network of peers. Each data element is mapped, based on its SFC-based index or key, to the first node whose identifier is equal to or follows the key in the identifier space. The look-up operation involving partial queries and range queries typically requires interrogating more than one node, since the desired information is distributed across multiple nodes. The look-up queries can consist of combination of a attributes, partial attributes or wildcards. The result of the query is a complete set of data elements that matches the user's query.

P-tree: Querying Peer-to-Peer Networks Using P-trees

Crainniceanu et al. [52] propose a distributed, fault-tolerant P2P index structure called P-tree. The main idea behind the proposed scheme is to maintain parts of semi-independent B^+-trees at each peer. The Chord protocol is utilized as a P2P routing substrate. Every peer in the P2P network believes that the search key values are organised in a ring, with the highest value wrapping around to the lowest value. Whenever a peer constructs its search tree, the peer pretends that its search key value is the smallest value in the ring. Each peer stores and maintains only the *left-most root-to-leaf path* of its corresponding B^+- tree. The remaining part of the sub-tree information is stored at a subset of other peers in the overlay network. Furthermore, each peer only stores tree nodes on the root-to-leaf path, and each node has at most $2d$ entries. In this case, the total storage requirement per peer is $O(d\ log_d N)$. The proposed approach guarantees $O(\log_d N)$ search performance for equality queries in a consistent state. Here d is the order of the sub-tree and N is the total number of peers in the network. Overall, in a stable system when no inserts or deletes operation is being carried out, the system provides $O(m + log_d N)$ search cost for range queries, where m is the number of peers in the selected range in 1-dimensional space.

3.4.3 CAN Based Approaches

One torus to rule them all (Kd-tree and Z-curve based indexing)

The work in [80] proposes two approaches for enabling DRQs over the CAN DHT. The d-dimensional data is indexed using the well known spatial data structures: (i) z-curves; and (ii) Kd-tree. First scheme is referred to as SCRAP: Space Filling Curves with Range Partitioning. Resolving DRQs in SCRAP network involves two basic steps: (i) mapping DRQ into SRQ using the SFCs; and (ii) routing the 1-dimensional range queries to the peers that indexes the desired look-up value. For routing query in 1-dimensional space the work proposes a scheme based on skip graph [12]. Other approach referred to as d-dimensional Rectangulation with Kd-trees (MURK). In this scheme, d-dimensional space (for instance a 2-d space) is represented as "rectangles" i.e. (hypercuboids in high dimensions), with each node maintaining one rectangle. In this case, these rectangles are used to construct a distributed Kd–tree. The leaf node in the tree are stored by the peers in the network. Routing in the network is based on the following schemes: (i) CAN DHT is used as basis for routing the DRQs; (ii) random pointers–each peer has to maintain skip pointers to random peers in the network. This scheme provides similar query and routing efficiency as multiple realities in CAN; and (iii) space–filling skip graph-each peer maintain skip pointers to $O(\log(n))$ other peers at exponentially increasing distances from itself in the network.

Meghdoot: Content-Based Publish/Subscribe over P2P Networks

The work in [88] proposes a content-based Pub/Sub system based on CAN routing substrate. Basic models and definitions are based on the scheme proposed in the work [146]. The model is capable of indexing a d-dimensional attribute space in a CAN routing space. An indexing space consisting of d attributes is always mapped to a CAN space of $2d$ dimensions. An attribute A_i with domain value $[L_i, H_i]$ corresponds to dimensions $2i-1$ and $2i$ in a $2d$-dimensional Cartesian space. The $2d$ dimensional logical space is partitioned among the peers in the system. A subscription S for d attributes is mapped to the point $< l_1, h_1, l_2, h_2, \ldots, l_d, h_d >$ in the $2d$ dimensional space which is referred to as the subscription point. Pub/Sub applications submit their subscription to a randomly cho-

sen peer P_0. A origin peer P_0 routes the subscription request to the target peer P_t using the basic CAN routing scheme. The peer P_t owns a point in the d-dimensional space to which a subscription S maps. The overall complexity involved in routing a subscription is $O(d\ n^{1/d})$, where n is the number of peers in the system and d is the dimensionality of the Cartesian space. Similarly every publish event is mapped to a particular point in the d-dimensional space, also referred to as the event point/event zone. The event is then routed to the P_t from the origin peer using the standard CAN routing. All the peers that own the region affected by a event are notified accordingly. Following this, all the peers in the affected region matches the new event against the previously stored subscriptions. Finally, the event is delivered to applications that have subscribed for the event.

HP-protocol: Scalable, Efficient Range Queries for Grid Information Services

Andrejak et al. [10] extend the CAN routing substrate to support 1-dimensional range queries. They apply the SFC in particular the Hilbert Curves for mapping a 1-dimensional attribute space (such as no. of processors) to a d-dimensional CAN space. For each resource attribute/dimension a separate CAN space is required. To locate a resource based on multiple attributes, the proposed system iteratively queries for each attribute in different CAN space. Finally, the result for different attributes are concatenated similar to "join" operation in the database. Given a range query r with lower and upper bounds $\in[l, u]$, a query message is routed to an information server which is responsible for the point $\frac{l+u}{2}$. Once such a server is located, then the request is recursively flooded to all its neighbors until all the IKs are located. Three different kinds of message flooding scheme are presented including the brute force, controlled flooding and directed control flooding. Each of these scheme has different search strategy and hence have different message routing complexities.

Super-P2P R*-Tree: Supporting Multi-dimensional Queries in P2P Systems

The authors in the work [114] extend the d-dimensional index R*-tree [18], for supporting range and k-Nearest Neighbour (kNN) queries in a super-peer [177] based P2P system. The resulting distributed R*-tree is referred to as a NR-tree. Routing in the distributed d-dimensional space is accomplished through the CAN protocol. The d-dimensional

distributed space is partitioned among the super-peer networks based on the Minimum Bounding Rectangle (MBR) of objects/points. Each partition (super-peer network) refers to a index-cluster (i.e. a MBR), and can be controlled by one or more super-peer. Effectively, a index-cluster includes a set of passive peers and super-peers. Evey index-cluster maps to a zone in the CAN based P2P space. The functionality of a super-peer is similar to a router, it keep tracks of other index-clusters, performs inter-cluster routing, indexes data in other super-peer partition and maintains cluster-specific NR-tree. Every passive peer joins the network by contacting any available super-peer. The contacted super-peer routes the join request to other super-peer, which is responsible for the zone indexed by the passive peer. Every passive peer maintains a part of the cluster-specific NR-tree. The bulk of query processing load is coordinated by super-peers. Super-peers can forward query to its passive-peers, in case the indexed data is managed by them.

3.4.4 Miscellaneous

SWORD: Distributed Resource Discovery on PlanetLab

SWORD [128] is a decentralised resource discovery service that supports multi-attribute queries. This system is currently deployed and tested over PlanetLab [47] resource sharing infrastructure. It supports different kind of query composition including per-node characteristics such as load, physical memory, disk space and inter-node network connectivity attributes such as network latency. For each resource attribute A_i, a corresponding DHT key k_i is computed using the standard SHA-1 scheme. A key k_i is computed based on the corresponding value of A_i at the time attribute value is sent. Each attribute is hashed to a 160-bit DHT key. The mapping function convert attribute values from their native data-type (String) and range (numeric) to a range of DHT keys. On receiving the attribute value tuple, the server node stores the tuple in the local table. In case, these values are not updated within timeout interval then are deleted (assuming node has probably left the network or owner of the key has changed due to change in attribute values). SWORD resolves multi-attribute range query similar to [24].

Mercury: Supporting Scalable Multi-Attribute Range Queries

Mercury [24] is a distributed resource discovery system that supports multi-attribute based information search. Mercury handles multi-attribute lookups by creating a separate routing hub for every resource dimension. Each routing hub represents a logical collection of nodes in the system and is responsible for maintaining range values for a particular dimension. Note that, while the notion of a circular overlay is similar to DHTs, Mercury do not use any randomizing cryptographic hash functions for placing the nodes and data on the overlay. In contrast, Mercury overlay network is organised based on set of links. These links include the: i) successor and predecessor links within the local attribute hub; ii) k links to other nodes in the local attribute hub (intra-hub links); and iii) one link per hub (inter-hub link) that aids in communicating with other attribute hubs and resolving multi-attribute range queries. Note that, k intra-hubs links is a configurable parameter and could be different for different nodes in the attribute overlay. In this case, the total routing table size at a node is $k + 2$. When a node n_k is presented with message to find a node that maintains a range value $[l_i, r_i]$, it chooses the neighbor n_i such that the clockwise distance $d(l_i, v)$ is minimized, in this case the node n_i maintains the attribute range value $[l_i, r_i]$. Key to message routing performance of Mercury is the choice of k intra-hub links. To set up each link i, a node draws a number $x \in \mathcal{I}$ using the harmonic probability distribution function: $p_n(x) = \frac{1}{n \log x}$. Following this, a node n_i attempts to add the node $n^{`}$ in its routing table which manages the attribute range value $r + (M_a - m_a) \times x$; where m_a and M_a are the minimum and maximum values for attribute a.

PHT: Prefix Hash Tree

The work in [134] presents a mechanism for implementing range queries over DHT based system via a trie-based[1] scheme. The bucket in the trie is stored at the DHT node obtained by hashing its corresponding prefixes. In the PHT, every vertex corresponds to a distinct prefix of the data domain being indexed. The prefixes of the nodes in the PHT form a

[1] A trie is a multi-way retrieval tree used for storing strings over an alphabet in which there is one node for every common prefix and all nodes that share a common prefix hang off the node corresponding to the common prefix.

universal prefix set[2]. The scheme associates a prefix label with each vertex of the tree. Given a vertex with label l, its left and right child vertices's's are labeled as l_0 and l_1 respectively. The root of the tree is always labeled with the attribute name and all the subsequent vertexes are labeled recursively. This logical PHT is distributed across nodes in the DHT-based network. Using the DHT look-up operation, a PHT node with label l is thus assigned to a node with identifier closest to HASH(l). Look-up for a range query in PHT network is performed by locating the node corresponding to the longest common prefix in the range. When such a node is found, then parallel traversal of its sub-tree is done to retrieve all the desired items.

JXTA: JXTA Search

JXTA Search [173] is an open framework based on the JXTA [85] routing substrate. JXTA search network consists of search hubs, information providers and information consumers. The network message communication protocol is based on the XML format. In the JXTA network, search hubs are organised into N distinct groups. These groups are referred to as *advertisement groups*. These search hubs act as point of contact for providers and consumers. Further each search hub is a member of a network of hubs which has at least one representative of hubs from every advertisement group. These groups are termed as *query groups*. Hence, in this case there is 100% reachability to all stored information in the network. Every information provider in the network registers its resource information with its local search hub. Each hub periodically sends update message (new additions and deletions of registrations) to all the hub in its advertisement group. Whenever an information consumer wishes to look for data on the search network, it issues an information request query to the hub it knows or has membership. The hub that receives this query first searches its local index and then other hubs in its advertisement group. If a match is found in the same advertisement group, then the query is forwarded to that hub. In case the query cant be resolved in the local advertisement group then it is broadcasted to all remaining advertisement groups using a query group membership information.

[2] A set of prefix is a universal prefix set if and only if for any infinite binary sequence b there is exactly one element in the set which is a prefix of b.

P2PR-Tree: An R-Tree Based Spatial Index for P2P Environments

The work in [124] presents a scheme for adopting the R-tree [77] in a P2P setting. P2PR-tree statically divides the d-dimensional attribute space (universe) into a set of blocks (rectangular tiles). The blocks formed as a result of initial division of the space forms level 0 of the distributed tree. Further, each block is statically divided into a set of groups, which constitute level 1 in the tree. Any further division on the group level (and subsequently on the subgroup) is done dynamically and are designated as subgroups at level i ($i \geq 2$). When a new peer joins the system, it contacts one of the existing peers which informs it about the Minimum Bounding Rectangle (MBR) of the blocks. Using this overall block structure information, a peer decides which block(s) it belongs to. A query Q_L for a object is propagated recursively top down starting from level 0. When a query arrives at any peer P_i in the system, P_i checks whether its MBR covers the region indexed by the query. If so, then P_i searches its own R-tree and returns the results and the search is terminated at that point. Otherwise the peer forwards the query to the relevant block, group, subgroup or peer using its routing table pointers. This process is repeated untill the query block is located or the query reaches dead end of the tree.

3.5 Comparision of surveyed techniques: scalability and load-balancing

A majority of the surveyed approaches utilise a logical index structure that distributes the data among peers in a decentralised GRIS. The logical structure maintains a d-dimensional (where $d \geq 1$) index space over the DHT key space and forms the basis for the routing and indexing of data objects. Some approaches (refer to Table 3.3) support only 1-dimensional queries for every distinct routing space. MAAN, Pub/Sub-1 and Pub/Sub-2 utilise variants of the SHA-1 hashing scheme for range partitioning 1-dimensional data over the DHT key space. We call these approaches variants of SHA-1, as they create a logical index space over the DHT key space which is utilised by the query routing heuristics. These algorithms did not consider the case of data skewness that can lead to routing load imbalance among the peers.

P-tree and Adaptive proposed a distributed version of B-+ tree index as the basis for range partitioning 1-dimensional data. The PHT approach uses a Trie based structure for enabling 1-dimensional range queries in a peer-to-peer network. XenoSearch organises resource information in the form of a logical tree where the leaves are the individual XenoServers. Query routing in XenoSearch is based on aggregation points (APs). An AP is managed by a XenoServer node in the system and is responsible for all the query computation for ranges of values covered by the AP. The Pastry Ids for the XenoServer responsible for an AP can be computed algorithmically. An AP owner in the system is similar to a super-peer which is responsible for handling all query computation intersecting its region of ownership. The Adaptive approach considered the case of data skewness and proposed a solution based on Load Balancing Matrix (LBM) while PHT, P-tree and XenoSearch did not propose any solution to this problem.

HPProtocol uses the inverse Hilbert mapping to map 1-dimensional index space to CAN's d-dimensional key space. Mercury directly operates on the attribute space along with random sampling technique utilised for facilitating query routing and load-balancing. A serious limitation of all the above approaches is the message overhead involved in maintaining a separate routing space for each attribute dimension. Further, searching in a d-dimensional space requires querying every dimension separately and then finding an intersection. This leads to high message communication overhead for lookup and update queries. Clearly, these are not scalable ways to organise a Grid resource attribute dataset that has many dimensions.

The JXTA system does not create a logical index space over the distributed search network:- instead, search is based on query broadcast among the advertisement group. This might prove costly in terms of number of messages generated. The Sword and Dgrid systems use a variant of SHA-1 hashing that partitions the DHT key space among different attribute types. Both Sword and Dgrid systems store all the attribute values in a single DHT ring. The Sword query resolution scheme is similar to MAAN, and so it is also costly in terms of routing hops and messages generated. The AdeepGrid approach encodes all the resource attributes into a single object and then performs SHA-1 hashing to generate a Pastry ring identifier. However, in this case the authors do not address the issue of data skewness. Further, the proposed search techniques are not capable of returning

deterministic results in all cases.

There are also some approaches that have utilised spatial indices for distributing the data among the peers (refer to Table 3.4). Spatial indices including Hilbert curves [150], Z-curves [80], k-d tree [80], MX-CIF Quad-tree [166], R-tree [124] and R*-tree [114] have the capability to logically organise a d-dimensional index space over a single DHT key space. SFC based indices including Hilbert curves and Z-curves have issues with routing load-balance in case of a skewed index distribution. However, as the authors point out, SFC index load can be balanced through external techniques. In the case of Hilbert curves, dynamic techniques such as node virtualisation, load-partitioning with neighbor peers etc. are utilised for this purpose. In XenoSearch-II system, Hilbert curves are utilised for mapping the d-dimensional index space to the 1-dimensional key space of Chord. However, XenoSearch-II does not propose any technique to counter load-imbalance among peers.

Indexing approach based on Z-curves required an external load-balancing technique. In the same work, they introduced a P2P version of a k-d tree. This approach also has routing load-balance issues that need to be addressed. In another recent work, a MX-CIF Quad tree based spatial index has been proposed. DragonFly utilises an index similar to the MX-CIF Quad tree with the difference that it does not allow recursive decomposition of index space. Instead, the index cells are split as they exceed the pre-configured load threshold value (similar to Meghdoot). The authors argue that their approach does not require explicit load-balancing algorithms in contrast to that of the others. The P2P based R*-tree index in [114] uses CAN as the routing space. The index space is partitioned among super peers and passive peers. The bulk of the query load is handled by the super peers in the network similar to the Gnutella [41] system.

Meghdoot does not utilise any spatial index for organising a d-dimensional data set. Instead, it utilises a basic $2d$ CAN space for indexing a d-dimensional data set. Further, Meghdoot incorporates dynamic technique to counter the data skewness issue. The load-balancing technique in Meghdoot splits an overloaded index cell (zone) among the lightly loaded peers. The P2P R-tree index divides the d-dimensional attribute space into a set of blocks (similar to MX-CIF Quad tree index), these blocks form the root of the distributed index tree. The work also includes a dynamic load division technique in case a peer index cell gets overloaded. However, this is an early work and it does not provide any bounds

on messages and routing hops required in a d-dimensional index search.

To summarise, spatial indices are better suited for handling the complexity of Grid resource queries compared to 1-dimensional data indices (as proposed in P-tree, MAAN, XenoSearch etc.). However, even spatial indices have routing load-balance issues in case of skewed data set. Nevertheless, they are more scalable in terms of the number of hops and messages generated while searching in a d-dimensional space.

3.6 Recommendations

The surveyed DHT-based index services provides the basic platform for organising and maintaining a decentralised Grid resource discovery system. A Grid system designer should follow a layered approach such as OPeN [166] in architecting and implementing a resource discovery system. The OPeN architecture consists of three layers: the *Application* layer, *Core Services* layer and *Connectivity* layer. The application layer implements all the logic that encapsulates the query requirements of the underlying Grid computing environment such as the computational grids, the data grids etc. The Core services layer undertakes the tasks related to consistency management of virtual d-dimensional indices. The Connectivity layer provides services related to Key-based routing, overlay management and replica placement. The Application service, in conjunction with the Core services, undertakes the resource discovery tasks including distributed information updates, lookups and virtual index consistency management. The management of Application and Core services layer can be delegated to a component of broker software. We refer to this broker component as a *Grid peer* service. While the maintenance of connectivity layer can be left to the basic DHT implementations such as FreePastry[3] and OpenDHT [141].

We recommend to the Grid system developers that for implementing the Core services layer they utilise the spatial indices surveyed in this article. Overall, spatial indices are superior to 1-dimensional indices as they incur lesser number of messages for d-dimensional object lookups and updates. However, there are different trade offs involved with each of the spatial indices, but basically they can all support scalability and Grid resource indexing. Some spatial index would perform optimally in one scenario but the performance

[3]FreePastry is an open source implementation of Pastry. http://freepastry.rice.edu/FreePastry.

could degrade if the data distribution changed significantly.

3.7 Open Issues

P2P based organisation of the Grid resource discovery services promises an attractive and efficient solution to overcome the current limitations associated with the centralised and hierarchical model. However, the P2P nature of the system raises other serious challenges including, security [154], trust, reputation and inter-operational ability between distributed services. Enforcing trust among the peers (a component of Grid broker service) that host the indexing services warrants robust models for: (i) managing a peer's reputation; and (ii) secure communication. A majority of the current solutions for security and trust management rely on centralised trust management entities such as CAs and ticket granting authorities. Achieving a completely decentralised security infrastructure is certainly a challenging future research direction. Recent efforts in this direction include emergence of distributed trust management systems such as PeerReview and Poblano. However, these trust management systems rely on behavioural auditing of the participant and the distributed auditing process can take a while untill a malicious participant is identified and shunted out of the system. This delay can allow ample opportunity to the malicious participant to effect significant harm to the system.

3.8 Summary and Conclusion

In the recent past, we have observed an increase in the complexity involved with Grid resources including their management policies, organisation and scale. Key elements that differentiate a computational Grid system from a PDCS include: (i) autonomy; (ii) decentralised ownership; (iii) heterogeneity in management policies, resource types and network inter-connect; and (iv) dynamicity in resource conditions and availability. Traditional Grid systems [3, 13, 75] based on centralised information services are proving to be bottleneck with regard to scalability, fault-tolerance and mechanism design issues. To address this, P2P based resource organisation is being advocated. P2P organisation is scalable, adaptable to dynamic network conditions and highly available.

In this work, we presented a detailed taxonomy that characterizes issues involved in designing a P2P GRIS. We classified the taxonomies into two sections: (i) resource taxonomy; and (ii) P2P taxonomy. Our resource taxonomy highlighted the attributes related to a computational Grid resource. Further, we summarized different kinds of queries that are being used in current computational Grid systems. In general, Grid superscheduling query falls under the category of d-dimensional point or window query. However, it still remains to be seen whether a universal Grid resource query composition language is required to express different kinds of Grid RLQs and RUQs.

We presented classification of P2P approaches based on three dimensions including: (i) P2P network organisation; (ii) approaches to distribution of the data among the peers; and (iii) routing of d-dimensional queries. In principle, data distribution mechanism directly dictates how a query is routed among the relevant peers. D-dimensional resource index is distributed among peers by utilizing the data structures such as SFCs, quad-trees, R-trees and Kd-trees. Some of the approaches have also modified existing hashing schemes to facilitate the 1-dimensional range queries in a DHT network. Every approach has its own merits and limitations. Some of these issues were highlighted in the resource and P2P network organisation taxonomy section.

Chapter 4

Grid-Federation

To overcome the limitations of current non-coordinated Grid brokering approaches, this chapter presents a mechanism for coordinated sharing of distributed clusters based on computational economy. The resulting environment, called *Grid-Federation*, allows the transparent use of resources from the federation when local resources are insufficient to meet its users' requirements. The use of computational economy methodology in coordinating resource allocation not only facilitates the QoS based scheduling, but also enhances utility delivered by resources. We show by simulation, while some users that are local to popular resources can experience higher cost and/or longer delays, the overall users' QoS demands across the federation are better met. Also, the federation's average case message passing complexity is seen to be scalable, though some jobs in the system may lead to large numbers of messages before being scheduled.

4.1 Introduction

Clusters of computers have emerged as mainstream parallel and distributed platforms for high-performance, high-throughput and high-availability computing. Grid computing [71] extends the cluster computing idea to wide-area networks. A grid consists of cluster resources that are usually distributed over multiple administrative domains, managed and owned by different organisations having different resource management policies. With the large scale growth of networks and their connectivity, it is possible to couple these cluster resources as a part of one large Grid system. Such large scale resource cou-

pling and application management is a complex undertaking, as it introduces a number of challenges in the domain of security, resource/policy heterogeneity, resource discovery, fault tolerance, dynamic resource availability and underlying network conditions.

The resources on a Grid (e.g. clusters and supercomputers) are managed by local resource management systems (LRMSes) such as Condor [113] and PBS [26]. These resources can also be loosely coupled to form campus Grids using multi-clustering [2] systems such as SGE [84] and LSF [185] that allow sharing of clusters owned by the same organisation. In other words, these systems do not allow their combination similar to autonomous systems, to create an environment for *cooperative federation* of clusters, which we refer as Grid-Federation.

Other related concept called Virtual Organisation (VO) [71] based grid resource sharing has been proposed in the literature. Effectively, a VO is formed to solve specific scientific problem. All the participants follow the same resource management policies defined by a VO. Hence, a VO represents a socialist world, wherein the participants have to adhere to community-wide agreed policies and priorities. In contrast, proposed Grid-Federation is a democratic world with complete autonomy for each participant. Further, a participant in the federation can behave rationally as we propose the use of economic model for resource management. Grid-Federation users submit their job to the local scheduler. In case local resources are not available or are not able to meet the requirement then job is transparently migrated to a remote resource (site) in the federation, although this job migration is constraint to user's QoS requirements. In a VO, user jobs are managed by a global scheduler which enforces resource allocation based on VO-wide policies.

Recall that, majority of existing approaches to resource brokering or superscheduling [151] in a Grid environment are non-coordinated. Superschedulers such as Nimrod-G [3], Gridbus broker, and Condor-G [75] perform scheduling related activities independent of the other superschedulers in the system. They directly submit their applications to the underlying resources *without* taking into account the current load, priorities, utilization scenarios of other application level schedulers. Clearly, this can lead to over-utilization or a bottleneck on some valuable resources while leaving others largely underutilized. Furthermore, these superschedulers do not have a coordination [1] mechanism and this exacerbates the load sharing and utilization problems of distributed resources because

sub-optimal schedules are likely to occur.

Currently, system-centric approaches such as NASA-Scheduler [152], Legion [39, 174], Condor, Condor-Flock [29], AppLes [22], PBS and SGE provide limited support for QoS driven resource sharing. These system-centric schedulers, allocate resources based on parameters that enhance system utilization or throughput. The scheduler either focuses on minimizing the response time (sum of queue time and actual execution time) or maximizing overall resource utilization of the system and these are not specifically applied on a per-user basis (user oblivious). System centric schedulers treat all resources with the same scale, as if they are worth the same and the results of different applications have the same value; while in reality the resource provider may value his resources differently and has a different objective function. Similarly, a resource consumer may value various resources differently and may want to negotiate a particular price for using a resource. Hence, resource consumers are unable to express their valuation of resources and QoS parameters. Furthermore, the system-centric schedulers do not provide any mechanism for resource owners to define what is shared, who is given the access and the conditions under which sharing occurs [72].

4.1.1 Grid-Federation

To overcome these shortcomings of non-coordinated, system-centric scheduling systems, we propose a new distributed resource management model, called Grid-Federation. Our Grid-Federation system is defined as a large scale resource sharing system that consists of a coordinated federation (the term is also used in the Legion system and should not be confused with our definition), of distributed clusters based on policies defined by their owners (shown in Fig. 4.1). Fig. 4.1 shows an abstract model of our Grid-Federation over a shared federation directory. To enable policy based transparent resource sharing between these clusters, we define and model a new RMS system, which we call Grid Federation Agent (GFA). In this chapter, we assume that the directory information is shared using some efficient protocol (e.g. a P2P protocol [96, 125]). In this case the P2P system provides a decentralised database with efficient updates and range query capabilities. Individual GFAs access the directory information using the interface shown in Fig. 4.1, i.e.

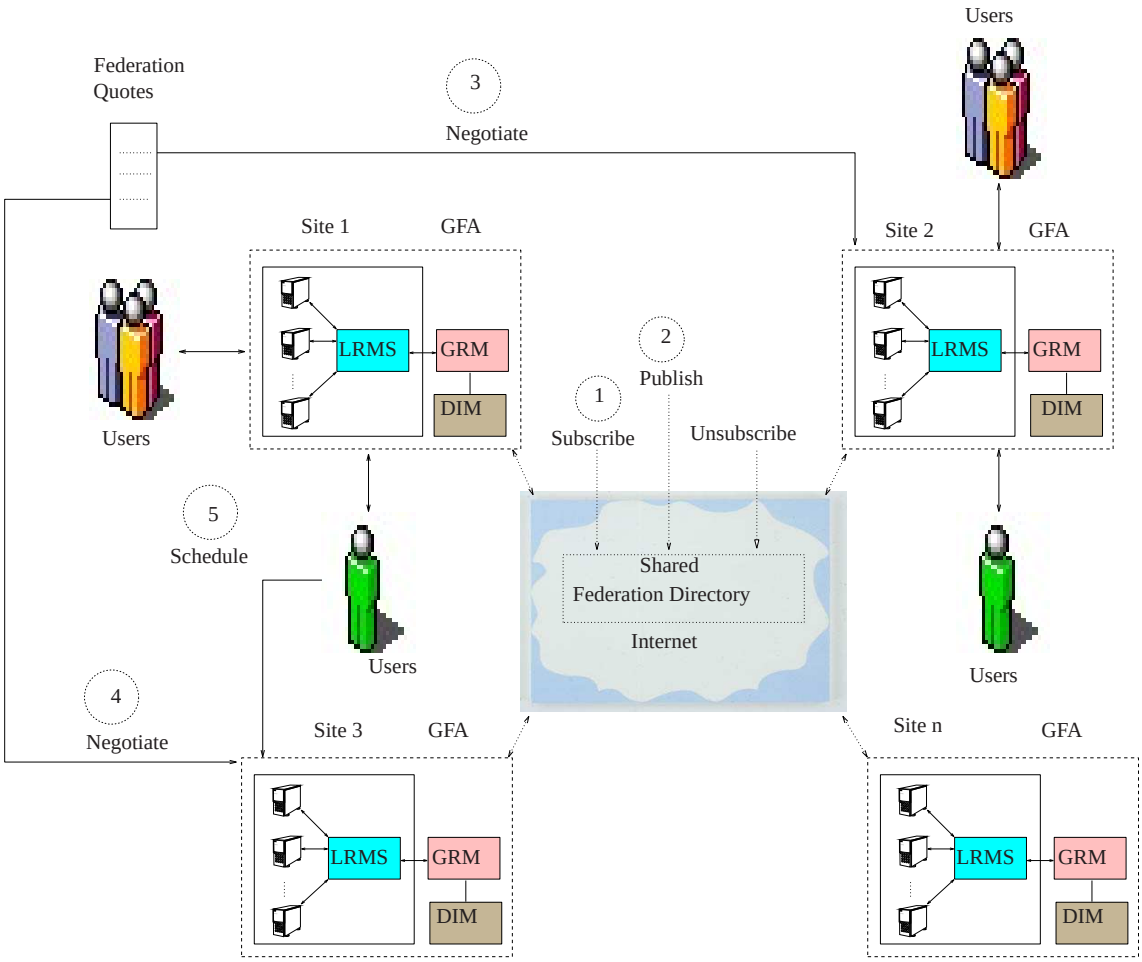


Figure 4.1: Grid-Federation resource sharing system.

subscribe, publish, unsubscribe. In this chapter, we are not concerned with the specifics of the interface although we do consider the implications of the required message-passing, i.e. the messages sent between GFAs to undertake the scheduling work. In Chapter 6, we present design, modeling and evaluation of a P2P publish/subscribe based Grid resource discovery service.

Our approach considers the emerging computational economy metaphor [3, 63, 162, 172] for Grid-Federation. In this case resource owners: can clearly define what is shared in the Grid-Federation while maintaining a complete autonomy; can dictate who is given access; and receive incentives for leasing their resources to federation users. We adopt the market based economic model from [3] for resource allocation in our proposed framework. Some of the commonly used economic models [31] in resource allocation includes the commodity market model, the posted price model, the bargaining model, the

Table 4.1: Superscheduling technique comparison.

Index	System Name	Network Model	Scheduling Parameters	Scheduling Mechanism
1	NASA-Scheduler	Random	System-centric	Partially coordinated
2	Condor-Flock P2P	P2P (Pastry)	System-centric	Partially coordinated
3	Grid-Federation	P2P (Decentralised directory)	User-centric	Coordinated
4	Legion-Federation	Random	System-centric	Coordinated
5	Nimrod-G	Centralised	User-centric	Non-coordinated
6	Condor-G	Centralised	System-centric	Non-coordinated
7	Our-Grid	P2P	System-centric	Coordinated
8	Tycoon	Centralised	User-centric	Non-coordinated
9	Bellagio	Centralised	User-centric	Coordinated

tendering/contract-net model, the auction model, the bid-based proportional resource sharing model, the community/coalition model and the monopoly model. We focus on the commodity market model [176]. In this model every resource has a price, which is based on the demand, supply and value in the Grid-Federation. Our economy model driven resource allocation methodology focuses on: (i) optimising resource provider's objective functions, and (ii) increasing end-user's perceived QoS value based on QoS level indicators and QoS constraints.

The rest of this chapter is organised as follows. In Section 4.2 we summarise our Grid-Federation and Section 4.3 deals with various experiments that we conducted to demonstrate the utility of our work. Section 4.4 explores various related projects. We end this chapter with some concluding remarks in Section 4.5.

4.2 Grid-Federation

Grid Federation Agent

We define our Grid-Federation (shown in Fig. 4.1) as a mechanism that enables logical coupling of cluster resources. The Grid-Federation supports policy based [40] transparent sharing of resources and QoS [98] based job scheduling. We also propose a new computational economy metaphor for cooperative federation of clusters. Computational

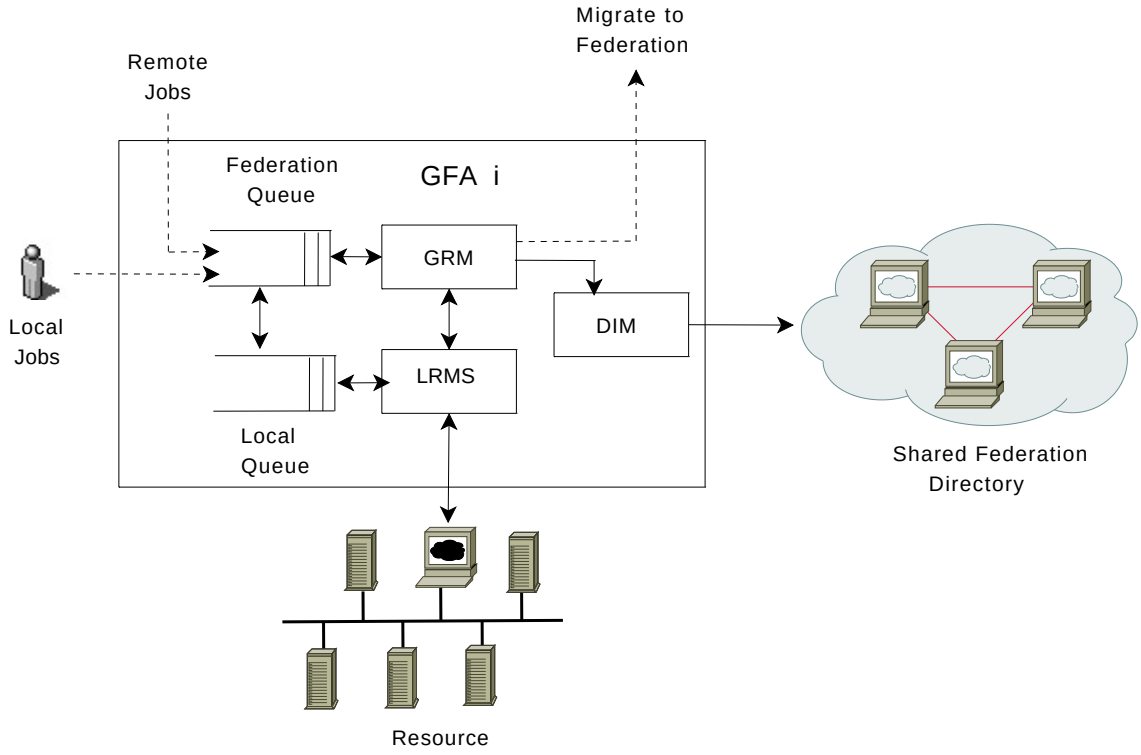


Figure 4.2: Grid-Federation superscheduling architecture.

economy [3, 162, 172] enables the regulation of supply and demand of resources, offers incentive to the resource owners for leasing, and promotes QoS based resource allocation. The Grid-Federation consists of the cluster owners as resource providers and the end-users as resource consumers. End-users are also likely to be topologically distributed, having different performance goals, objectives, strategies and demand patterns. We focus on optimising the resource provider's objective and resource consumer's utility functions by using a quoting mechanism. The Grid-Federation consists of cluster resources distributed across multiple organisations and administrative domains. To enable policy based coordinated resource sharing between these clusters, we define and model a new RMS system, which we call Grid Federation Agent (GFA). A cluster can become a member of the federation by instantiating a GFA component. GFA acts as a resource co-coordinator in the federated space, spanning over all the clusters. These GFAs in the federation inter-operate using an agreed communication primitive over the shared federation directory.

This section provides comprehensive details about our proposed Grid-Federation, in-

cluding models used for budget and deadline calculations in the simulations of the next section. The model defines the following functional modules of a GFA:

Grid Resource Manager (**GRM**)

The Grid resource manager is responsible for superscheduling the locally submitted jobs in the federation. Further, it also manages the execution of remote jobs in conjunction with the LRMS on the local resource. *Local jobs* refers to the jobs submitted by the local population of users, while *remote jobs* refers to the incoming jobs from remote Grid resource managers. A Grid resource manager provides admission control facility at each site in the federation. Fig. 4.2 shows the Grid-Federation superscheduling architecture that we propose. In Fig. 4.2, a GFA i in the federation with modules GRM, LRMS and DIM is shown. The GRM component of GFA is connected to the federation queue which accepts the incoming remote jobs (from the federation) as well as local jobs. All the remote jobs are transferred to the local queue which is controlled by the GFA's LRMS module. A GRM can also export the locally submitted jobs to other sites in the federation depending on the user specified QoS requirements. The job submission and migration process is represented by a dashed arrow in the Fig. 4.2.

A local user submits his job to the GRM which then places it in the federation queue. GRM analyses the user's QoS specification and then sends a query message to the DIM. The DIM returns the I-st fastest or I-st cheapest machine as specified in the QoS requirements. If the returned machine is the local resource then the job is transferred to the local queue. Otherwise, the job is transferred to a remote site in the federation. GRMs undertake one-to-one negotiation before submitting a job to a remote site. The GRM local to the submitted job sends admission control negotiate message to the remote GRM requesting a guarantee on the total job completion time. Following this, the contacted GRM queries its LRMS. If the LRMS reports that the job can be completed within the specified deadline, then the admission control acceptance message is sent to the requesting GRM. On receiving the acceptance, the GRM sends the job. The inter-site GRM-to-GRM negotiation scheme prevents the GRMs from submitting unlimited amount of jobs to the resources. Further, this approach allows autonomy for every resource domain, as they

Table 4.2: Resource and workload notations.

Symbol	Meaning
n	number of Grid Federation Agents (GFAs) in the Grid network.
n_u	number of users over all clusters ($\sum_{k=1}^{n} n_k$, n_k number of users at GFA k).
R_i	configuration of the i-th resource in the system.
I_k	incentive earned by resource owner k over simulation period.
ρ_i	resource utilisation for resource at GFA i.
x_i	processor architecture for resource at GFA i.
c_i	resource access cost for resource at GFA i.
p_i	number of processors for reosurce at GFA i.
ϕ_i	operating system type for resource at GFA i.
μ_i	processor speed at GFA i.
$u_{i,j}$	i^{th} user from j^{th} GFA/resource.
$J_{i,j,k}$	i-th job from the j-th user of k-th GFA.
$p_{i,j,k}$	number of processor required by $J_{i,j,k}$.
$b_{i,j,k}$	assigned budget to $J_{i,j,k}$.
$d_{i,j,k}$	assigned deadline to $J_{i,j,k}$.
$\phi_{i,j,k}$	operating system type required by $J_{i,j,k}$.
$D(J_{i,j,k}, R_k)$	time function (expected response time for $J_{i,j,k}$ at resource k).
$B(J_{i,j,k}, R_k)$	cost function (expected budget spent for $J_{i,j,k}$ at resource k).
$d^e_{i,j,k}$	effective deadline for $J_{i,j,k}$.
$l_{i,j,k}$	job length for $J_{i,j,k}$ (in terms of million instructions)
$\alpha_{i,j,k}$	communication overhead for $J_{i,j,k}$
$\tau(J_{i,j,k})$	returns next SLA bid interval $\Delta t_{neg_{i,j,k,p}}$ for $J_{i,j,k}$.
$t_{neg_{i,j,k}}$	total SLA bid interval/delay for $J_{i,j,k}$.
$\Delta t_{neg_{i,j,k,p}}$	total delay for p-th SLA bid for $J_{i,j,k}$.
n_j	total jobs in the federation ($\sum_{(k,u_i)=1}^{n} n_k, u_i$).
$t_{s_{i,j,k}}$	job submission delay (user to GFA).
$t_{r_{i,j,k}}$	finished job return delay (GFA to user) .

Table 4.3: Queuing and resource discovery service notations.

Symbol	Meaning
λ_{SLA_i}	SLA arrival rate at GFA i.
μ_{SLA_i}	SLA satisfaction rate at GFA i.
λ^{in}	total incoming RLQ/RUQ arrival rate at a network queue i.
λ^{out}	outgoing RLQ/RUQ rate at a network queue i.
μ_{n_i}	average network queue service rate at a Grid peer i.
$Q_{m,t}$	set of jobs that have been assigned but not accepted at GFA m at time t.
$Q^a_{m,t}$	set of jobs that have been accepted at GFA m at time t.
$Q^s_{m,t}$	set of jobs sorted in decreasing order of incentive it provides to the resource owner at GFA m at time t.
μ_r	average query reply rate for index service at GFA/peer i.
λ^{in}_u	incoming RUQ (publish) rate at a application service i.
λ^{in}_l	incoming RLQ (subscribe) rate at a application service i.
λ^{in}_a	incoming query rate at a Chord routing service i from the local application service.
λ^{in}_{index}	incoming index query rate at a application service i from its local Chord routing service.
$r_{i,j,k}$	an RLQ for $J_{i,j,k}$.
U_i	an RUQ for the i-th GFA/peer/resource.
dim	dimensionality or number of attributes in the Cartesian space.
f_{min}	minimum division level of d-dimensional resource attribute space.
f_{max}	maximum allowed depth of the d-dimensional index tree.
d	number of dimensions for the CAN.
b	base of the identifier space for Pastry.
K	network queue size.
$gindex_i$	object encapsulating details on a GFA's IP address, service port number etc.
M	random variable denoting number of of messages generated in mapping an RLQ or RUQ.
T	random variable denoting number of disjoint query path undertaken in mapping an RLQ or RUQ.

have capability to perform per job-basis admission control decision. All migrated jobs are queued in the federation queue, then subsequently transferred to the local queue for final execution process.

The proposed Grid-Federation mechanism can leverage services of Grid-Bank [6] for credit management. The participants in the system can use Grid-Bank to exchange Grid Dollars.

Local Resource Management System (LRMS)

In our proposed Grid-Federation distributed resource sharing system, we assume that every cluster has a generalized RMS, such as a SGE or PBS that manages cluster wide resource allocation and application scheduling. Most of the available RMS packages have a centralised organisation similar to the master-worker pool model. In the centralised organisation, there is only one scheduling controller (master node) which coordinates system-wide decisions. Grid resource manager queries LRMS to acquire information about local job queue size, expected response time for a job, and resource utilisation status.

Distributed Information Manager (DIM)

The DIM performs tasks like resource discovery and advertisement through well defined primitives. It interacts with an underlying shared federation directory (shown in Fig. 4.1). Recall that we assume the directory information is shared using some efficient protocol (e.g. a P2P protocol). In this case, the P2P system provides a decentralised database with efficient updates and range query capabilities. Individual GFAs access the directory information using the interface shown in Fig. 4.1, i.e. subscribe, quote, unsubscribe and query. In this chapter, we are not concerned with the specifics of the interface (which can be found in Chapter 6). The resource discovery function includes searching for suitable cluster resources while resource advertisement is concerned with advertising resource capability (with pricing policy) to other clusters in the federation. The federation directory maintains quotes or advertised costs from each GFA in the federation.

In Table 4.2 and 4.3, we present the various notations and model parameters that are

utilised in this thesis. Each quote consists of a resource description R_i, for cluster i, and a cost c_i for using that resource configured by respective cluster owners. Using R_i and c_i, a GFA can determine the cost of executing a job on cluster i and the time taken, assuming that the cluster i has no load. The actual load of the cluster needs to be determined dynamically and the load can lead to changes in time taken for job completion. In this work, we assume that c_i remains static throughout the simulations. Each GFA can query the federation directory to find the k-th fastest cluster or the k-th cheapest cluster. We assume the query process is optimal, i.e. that it takes $O(\log n)$ messages [33] to query the directory, when there are n GFAs in the system. Although, we consider the number of additional messages that are used to satisfy our Grid-Federation scheduling process.

4.2.1 Decentralised Market Place and Grid-Federation

Grid computing assembles resources that are well managed, powerful and well connected to the Internet. Grids present a platform for Grid Participants (GPs) to collaborate and coordinate resource management activities. Key GPs include the *producers* (Grid resource-owners) and *consumers* (Grid users). GPs have different goals, objectives, strategies, and supply and demand functions. GPs are topologically distributed and belong to different administrative domains. Controlled administration of Grid resources gives an ability to provide a desired QoS in terms of computational and storage efficiency, software or library upgrades. However, such controlled administration of resources gives rise to various social and political issues on which these resources are made available to the outside world.

A resource owner invests a significant amount of money in establishing the resource such as, initial cost of buying, setting up, maintenance cost including hiring the administrator and expense of timely software and the hardware upgrades. There is a complex diversity in terms of resources' usage policies, loads and availability. Resource owners in a grid behave as rational participants having distinct stake holdings with potentially conflicting and diverse utility functions. In this case, resource owners apply resource sharing policies that tend to maximize their utility functions [64, 86]. Similarly, the resource consumers in a grid associate QoS based utility constraints to their applications and ex-

pect that the constraints are satisfied within the acceptable limits. Every resource owner makes the policy related decision independently that best optimizes his objective function. Likewise, resource consumers have diverse QoS based utility constraints, priorities and demand patterns.

To capture the above dynamics and complexity of Grid resource sharing environment, Grid-Federation applies market based economy principles for resource allocation and application scheduling. In particular, we adopt commodity market model. In this model, every resource owner sets up a fixed price based on the demand for his resources in the decentralised market place. Resource owner advertises its resource access cost through its local GFA service. Analysing different pricing algorithm based on supply and demand function is a vast research area. Investigating how the cluster owners determine the price [43, 155, 176] of their commodity is subject of future work.

4.2.2 General Grid-Federation Superscheduling Technique

In this section we describe our general Grid-Federation scheduling technique. In Fig. 4.1 a user who is local to GFA 3 is submitting a job. If the user's job QoS can not be satisfied locally then GFA 3 queries the federation directory to obtain the quote of the 1-st fastest or 1-st cheapest cluster. In this case, the federation directory returns the quote advertised by GFA 2. Following this, GFA 3 sends a negotiate message (enquiry about QoS guarantee in terms of response time) to GFA 2. If GFA has too much load and cannot complete the job within the deadline then GFA 3 queries the federation directory for the 2-nd cheapest/fastest GFA and so on. The query-negotiate process is repeated until GFA 3 finds a GFA that can schedule the job (in this example the job is finally scheduled on site 3).

Every federation user must express how much he is willing to pay, called a *budget*, and required response time, called a *deadline*, for his job number j. In this work, we say that a job's QoS has been satisfied if the job is completed within budget and deadline, otherwise it is not satisfied. Every cluster in the federation has its own resource set R_i which contains the definition of all resources owned by the cluster and ready to be offered. R_i can include information about the CPU architecture, number of processors, RAM size, secondary storage size, operating system type, etc. In this work, $R_i = (p_i, \mu_i, \gamma_i)$ which includes the

number of processors, p_i, their speed, μ_i and underlying interconnect network bandwidth γ_i. We assume that there is always enough RAM and correct operating system conditions, etc. The cluster owner charges c_i per unit time or per unit of million instructions (MI) executed, e.g. per 1000 MI.

We write $J_{i,j,k}$ to represent the i-th job from the j-th user of the k-th resource. A job consists of the number of processors required, $p_{i,j,k}$, the job length, $l_{i,j,k}$ (in terms of instructions), the budget, $b_{i,j,k}$, the deadline or maximum delay, $d_{i,j,k}$ and the communication overhead, $\alpha_{i,j,k}$.

To capture the nature of parallel execution with message passing overhead involved in the real application, we considered a part of total execution time as the communication overhead and remaining as the computational time. In this work, we consider the network communication overhead $\alpha_{i,j,k}$ for a parallel job $J_{i,j,k}$ to be randomly distributed over the processes. In other words, we don't consider the case e.g. when a parallel program written for a hypercube is mapped to a mesh architecture. We assume that the communication overhead parameter $\alpha_{i,j,k}$ would scale the same way over all the clusters depending on γ_i. The total data transfer involved during a parallel job execution is given by

$$\Gamma(J_{i,j,k}, R_k) = \alpha_{i,j,k}\,\gamma_k \tag{4.1}$$

The time for job $J_{i,j,k} = (p_{i,j,k}, l_{i,j,k}, b_{i,j,k}, d_{i,j,k}, \alpha_{i,j,k})$ to execute on resource R_m is

$$D(J_{i,j,k}, R_m) = \frac{l_{i,j,k}}{\mu_m\, p_{i,j,k}} + \frac{\Gamma(J_{i,j,k}, R_k)}{\gamma_m} \tag{4.2}$$

$$D(J_{i,j,k}, R_m) = \frac{l_{i,j,k}}{\mu_m\, p_{i,j,k}} + \frac{\alpha_{i,j,k}\,\gamma_k}{\gamma_m} \tag{4.3}$$

and the associated cost is

$$B(J_{i,j,k}, R_m) = c_m \frac{l_{i,j,k}}{\mu_m\, p_{i,j,k}}. \tag{4.4}$$

If $s_{i,j,k}$ is the time that $J_{i,j,k}$ is submitted to the system then the job must be completed by time $s_{i,j,k} + d_{i,j,k}$.

4.2.3 QoS Driven Resource Allocation Algorithm for Grid-Federation

We consider a deadline and budget constrained (DBC) scheduling algorithm, or cost-time optimisation scheduling. The federation user can specify any one of the following optimisation strategies for their jobs:

- optimisation for time (OFT) – give minimum possible response time within the budget limit;
- optimisation for cost (OFC) – give minimum possible cost within the deadline.

For each job that arrives at a GFA, called the local GFA, the following is done:

1. Set $r = 1$.
2. If OFT is required for the job then query the federation directory for the r-th fastest GFA; otherwise OFC is required and the query is made for the r-th cheapest GFA. Refer to the result of the query as the remote GFA.
3. The local GFA sends a message to the remote GFA, requesting a guarantee on the time to complete the job.
4. If the remote GFA confirms the guarantee then the job is sent, otherwise $r := r + 1$ and the process iterates through step 2.

Recall that we assume each query takes $O(\log n)$ messages and hence in this work we use simulation to study how many times the iteration is undertaken, on a per job basis and on a per GFA basis. The remote GFA makes a decision immediately upon receiving a request as to whether it can accept the job or not. If the job's QoS parameters cannot be satisfied (after iterating up to the greatest r such that GFA could feasibly complete the job) then the job is dropped.

Effectively, for job $J_{i,j,k}$ that requires OFC then GFA m with R_m is chosen such that $B(J_{i,j,k}, R_m) = \min_{1<m'\leq n}\{B(J_{i,j,k}, R_{m'})\}$, and $D(J_{i,j,k}, R_m) \leq s_{i,j,k} + d_{i,j,k}$. Similarly, for OFT then GFA m is chosen such that $D(J_{i,j,k}, R_m) = \min_{1<m'\leq n}\{D(J_{i,j,k}, R_{m'})\}$, and $B(J_{i,j,k}, R_m) \leq b_{i,j,k}$.

4.2.4 Quote Value

We assume c_i remains static throughout the simulations. In this work, we are only interested in studying the effectiveness of our Grid-Federation superscheduling algorithm based on the static access charge c_i. In simulations, we configure c_i using the function:

$$c_i = f(\mu_i) \tag{4.5}$$

where,

$$f(\mu_i) = \frac{c}{\mu}\ \mu_i \tag{4.6}$$

c is the access price and μ is the speed of the fastest resource in the Grid-Federation.

4.2.5 User Budget and Deadline

While our simulations in the next section use trace data for job characteristics, the trace data does not include user specified budgets and deadlines on a per job basis. In this case we are forced to fabricate these quantities and we include the models here.

For a user, j, we allow each job from that user to be given a budget (using Eq. 4.4),

$$b_{i,j,k} = 2\,B(J_{i,j,k}, R_k). \tag{4.7}$$

In other words, the total budget of a user over simulation is unbounded and we are interested in computing the budget that is required to schedule all of the jobs.

Also, we let the deadline for job i (using Eq. 4.2) be

$$d_{i,j,k} = 2\,D(J_{i,j,k}, R_k). \tag{4.8}$$

we assign two times the value of total budget and deadline for the given job, as compared to the expected budget spent and response time on the originating resource.

Table 4.4: Workload and resource configuration.

Index	Resource / Cluster Name	Trace Date	Processors	MIPS (rating)	Total Jobs in Trace	Quote	NIC to Network Bandwidth (Gb/Sec)
1	CTC SP2	June96-May97	512	850	79,302	4.84	2
2	KTH SP2	Sep96-Aug97	100	900	28,490	5.12	1.6
3	LANL CM5	Oct94-Sep96	1024	700	201,387	3.98	1
4	LANL Origin	Nov99-Apr2000	2048	630	121,989	3.59	1.6
5	NASA iPSC	Oct93-Dec93	128	930	42,264	5.3	4
6	SDSC Par96	Dec95-Dec96	416	710	38,719	4.04	1
7	SDSC Blue	Apr2000-Jan2003	1152	730	250,440	4.16	2
8	SDSC SP2	Apr98-Apr2000	128	920	73,496	5.24	4

4.3 Performance Evaluation

4.3.1 Workload and Resource Methodology

We used trace based simulation to evaluate the effectiveness of the proposed system and the QoS provided by the proposed superscheduling algorithm. The workload trace data was obtained from [66]. The trace contains real time workload of various supercomputers/resources that are deployed at the Cornell Theory Center (CTC SP2), Swedish Royal Institute of Technology (KTH SP2), Los Alamos National Lab (LANL CM5), LANL Origin 2000 Cluster (Nirvana) (LANL Origin), NASA Ames (NASA iPSC) and San-Diego Supercomputer Center (SDSC Par96, SDSC Blue, SDSC SP2) (See Table 4.4). The workload trace is a record of usage data for parallel jobs that were submitted to various resource facilities. Every job arrives, is allocated one or more processors for a period of time, and then leaves the system. Furthermore, every job in the workload has an associated arrival time, indicating when it was submitted to the scheduler for consideration. As the experimental trace data does not include details about the network communication overhead involved for different jobs, we artificially introduced the communication overhead element as 10% of the total parallel job execution time.

The simulator was implemented using GridSim [32] toolkit that allows modeling and simulation of distributed system entities for evaluation of scheduling algorithms. GridSim offers a concrete base framework for simulation of different kinds of heterogeneous resources, brokering systems and application types. This toolkit can simulate resource brokering for resources that belong to a single administrative domain (such as a cluster)

or multiple administrative domain (such as a grid). The core of simulation is based on *simjava* [94], a discrete event simulation package implemented in java. The main classes of GridSim includes GridResource, GridSim, Gridlet, AllocPolicy and GridInformation-Service. These classes communicate using discrete message passing events. To enable parallel workload simulation with GridSim, we extended the existing AllocPolicy and SpaceShared entities.

Our simulation environment models the following basic entities in addition to existing entities in GridSim:

- local user population – models the workload obtained from trace data;
- GFA – generalized RMS system;
- GFA queues (federation and local) – placeholder for incoming jobs from local user population and the federation;
- GFA shared federation directory – simulates an efficient distributed query process such as P2P.

For evaluating the QoS driven resource allocation algorithm, we assigned a synthetic QoS specification to each resource including the Quote value (price that a cluster owner charges for service), having varying MIPS rating and underlying network communication bandwidth. The simulation experiments were conducted by utilizing workload trace data over the total period of 2 days (in simulation units) at all the resources. Hence, effectively our simulation considers only a fraction of jobs per computing site as compared to the total number of jobs that were submitted. For example, originally 79302 jobs were submitted to CTC SP2 over a period of 1 year, while our simulation considers only 417 jobs (no. of jobs submitted over 2 days). We consider the following resource sharing environment for our experiments:

- independent resource:- Experiment 1;
- federation without economy:- Experiment 2;
- federation with economy:- Experiments 3, 4, and 5.

Table 4.5: Workload processing statistics (Without Federation).

Index	Resource / Cluster Name	Average Resource Utilization (%)	Total Jobs	Total Job Accepted(%)	Total Job Rejected(%)
1	CTC SP2	53.492	417	96.642	3.357
2	KTH SP2	50.06438	163	93.865	6.134
3	LANL CM5	47.103	215	83.72	16.27
4	LANL Origin	44.55013	817	93.757	6.24
5	NASA iPSC	62.347	535	100	0
6	SDSC Par96	48.17991	189	98.941	1.058
7	SDSC Blue	82.08857	215	57.67	42.3255
8	SDSC SP2	79.49243	111	50.45	49.54

4.3.2 Experiment 1 – independent resources

In this experiment the resources were modeled as an independent entity (without federation). All the workload submitted to a resource is processed and executed locally (if possible). In Experiment 1 (and 2) we consider, if the user request can not be served within requested deadline, then it is rejected otherwise it is accepted. In original trace, as jobs were supposed to be scheduled on the local resource so they were queued in untill the required number of processors became available. Effectively, no job was rejected in the original trace. During Experiment 1 (and 2), we evaluate the performance of a resource in terms of average resource utilization (amount of real work that a resource does over the simulation period excluding the queue processing and idle time), job acceptance rate (total percentage of jobs accepted) and conversely the job rejection rate (total percentage of jobs rejected). The result of this experiment can be found in Table 4.5 and Fig. 4.3. Experiment 1 is essentially the control experiment that is used as a benchmark for examining the affects of using federated (with and without economy) sharing of resources.

4.3.3 Experiment 2 – with federation

In this experiment, we analyzed the workload processing statistics of various resources when part of the Grid-Federation but not using an economic model. In this case the workload assigned to a resource can be processed locally. In case a local resource is not available then online scheduling is performed that considers the resources in the federation in decreasing order of their computational speed. We also quantify the jobs depending on

Table 4.6: Workload processing statistics (With Federation).

Index	Resource / Cluster Name	Average Resource Utilization (%)	Total Job	Total Job Accepted(%)	Total Job Rejected(%)	No. of Jobs Processed Locally	No. of Jobs Migrated to Federation	No. of Remote jobs processed
1	CTC SP2	87.15	417	100	0	324	93	72
2	KTH SP2	68.69	163	99.38	0.61	110	52	35
3	LANL CM5	67.20	215	90.69	9.30	145	50	70
4	LANL Origin	77.62	817	98.89	1.10	733	75	81
5	NASA iPSC	78.73	535	99.81	0.18	428	106	129
6	SDSC Par96	79.17	189	100	0	143	46	30
7	SDSC Blue	90.009	215	98.60	1.39	105	107	77
8	SDSC SP2	87.285	111	97.29	2.70	54	54	89

whether they are processed locally or migrated to the federation. Table 4.6 and Fig. 4.3 describes the result of this experiment.

4.3.4 Experiment 3 – with federation and economy

In this experiment, we study the computational economy metaphor in the Grid-Federation. In order to study economy based resource allocation mechanism, it was necessary to fabricate user budgets and job deadlines. As the trace data does not indicate these QoS parameters, so we assigned them using Eqs. 5.8 and 5.9 to all the jobs across the resources. We performed the experiment under 11 different combination of user population profile: $OFT = i$ and $OFC = 100 - i$ for $i = 0, 10, 20, \ldots, 100$.

Fig. 4.4, 4.5, 4.6, 4.7 and 4.8 describes the result of this experiment.

4.3.5 Experiment 4 – message complexity with respect to jobs

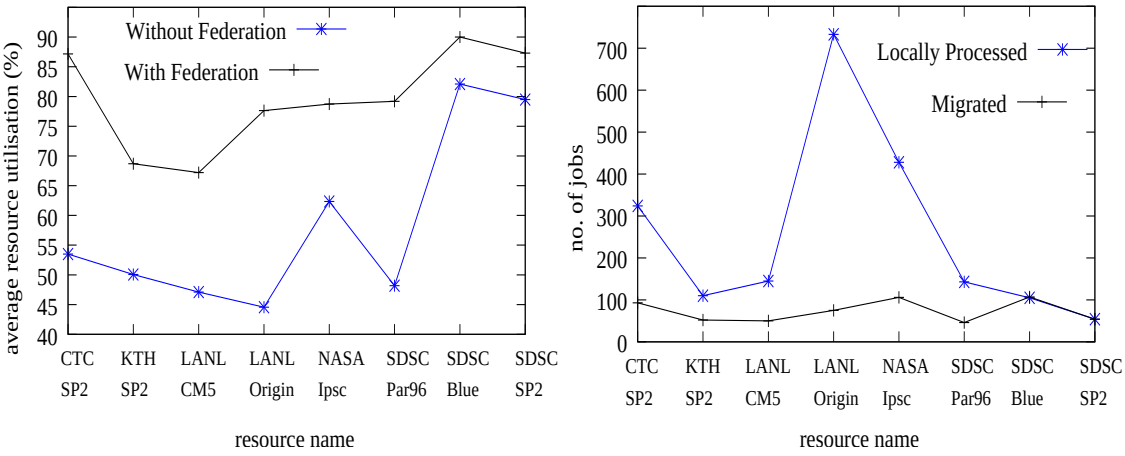
In this experiment, we consider total incoming and outgoing messages at all GFA's. The various message type includes negotiate, reply, job-submission (messages containing actual job) and job-completion (message containing job output). We quantify the number of local messages (sent from a GFA to undertake a local job scheduling) and remote messages (received at a GFA to schedule a job belonging to a remote GFA in the federation). The experiment was conducted for the same user populations as explained in experiment 3. Fig. 4.9 describes the result of this experiment.

4.3.6 Experiment 5 – message complexity with with respect to system size

This experiment measures the system's performance in terms of the total message complexity involved as the system size grows from 10 to 50. In this case, we consider the average, max and min number of messages (sent/recv) per GFA/per Job basis. Note that, in case n messages are undertaken to schedule a job then it involves traversing (if $n > 2$ then $(n-2)/2$, else $n/2$) entries of the GFA list. To accomplish larger system size, we replicated our existing resources accordingly (shown in Table 4.4). The experiment was conducted for the same user populations as explained in experiment 3. Fig. 4.10 and 4.11 describe the result of this experiment.

4.3.7 Results and observations

Justifying Grid-Federation based resource sharing



(a) Average resource utilization (%) vs. resource name

(b) No. of jobs vs. resource name

Figure 4.3: Resource utilization and job migration plot.

During experiment 1 we observed that 5 out of 8 resources remained underutilized (less than 60%). During experiment 2, we observed that overall resource utilization of most of the resources increased as compared to experiment 1 (when they were not part of the federation), for instance resource utilization of CTC SP2 increased from 53.49% to 87.15%. The same trends can be observed for other resources too (refer to Fig. 4.3(a)).

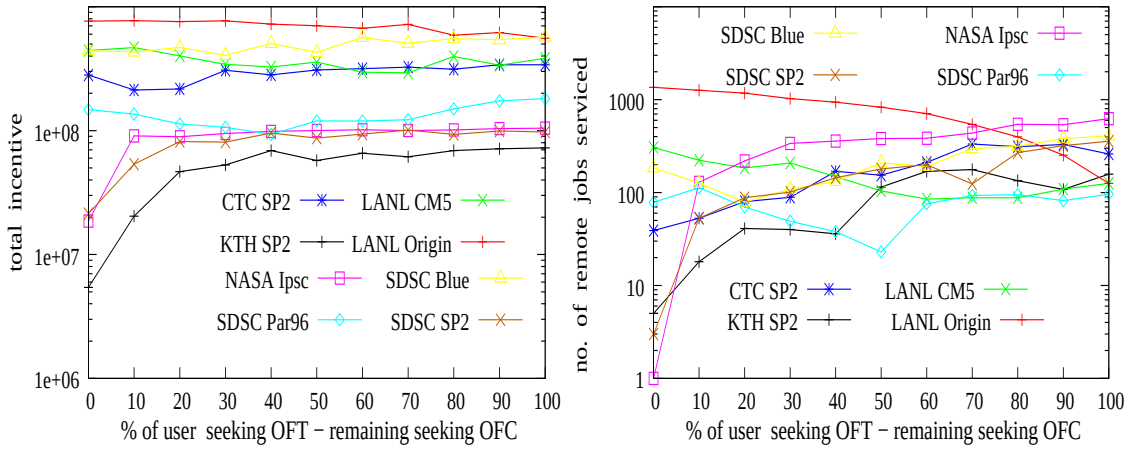
There was an interesting observation regarding migration of the jobs between the resources in the federation (load-sharing). This characteristic was evident at all the resources including CTC SP2, KTH SP2, NASA iPSC etc. At CTC, which had total 417 jobs to schedule, we observed that 324 (refer to Table 4.6 or Fig. 4.3(b)) of them were executed locally while the remaining 93 jobs migrated and executed at some remote resource in the federation. Further, CTC executed 72 remote jobs, which migrated from other resources in the federation.

The federation based load-sharing also lead to a decrease in the total job rejection rate, this can be observed in case of resource SDSC Blue where the job rejection rate decreased from 42.32% to 1.39% (refer to Table 4.5 and Table 4.6). Note that, the average job acceptance rate, over all resources in the federation, increased from 90.30% (without federation) to 98.61% (with federation). Thus, for the given job trace, it is preferable to make use of more resources, i.e. to migrate jobs. In other words, the job trace shows the potential for resource sharing to increase utilization of the system.

Resource Owner Perspective

In experiment 3, we measured the computational economy related behavior of the system in terms of its supply-demand pattern, resource owner's incentive (earnings) and end-user's QoS constraint satisfaction (average response time and average budget spent) with varying user population distribution profiles. We study the relationship between resource owner's total incentive and end-user's population profile.

The total incentive earned by different resource owners with varying user population profile can be seen in Fig. 4.4(a). The result shows as expected that the owners (across all the resources) earned more incentive when users sought OFT (Total Incentive 2.30×10^9 Grid Dollars) as compared to OFC (Total Incentive 2.12×10^9 Grid Dollars) . During OFT, we observed that there was a uniform distribution of the jobs across all the resources (refer to Fig. 4.5(a)) and every resource owner earned some incentive. During OFC, we observed a non-uniform distribution of the jobs in the federation (refer to Fig. 4.5(a)). We observed that the resources including CTC SP2, LANL CM5, LANL Origin, SDSC par96 and SDSC Blue earned significant incentives. This can also be observed in their resource utilization statistics (refer to Fig. 4.5(a)). However, the faster resources (e.g. KTH SP2,

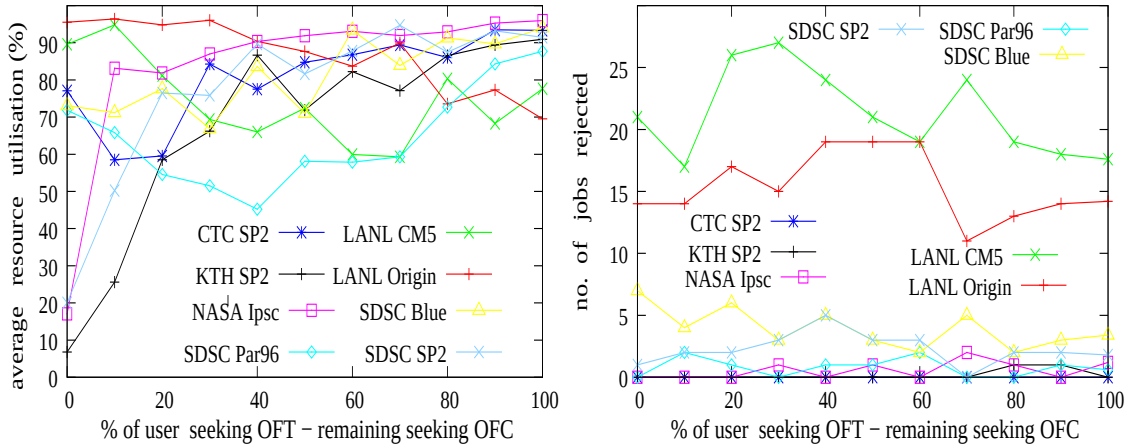


(a) Total incentive (Grid Dollars) vs. user population profile

(b) No. of remote job serviced vs. user population profile

Figure 4.4: Resource owner perspective.

NASA iPSC and SDSC SP2) remained largely underutilized and did not get significant incentives.



(a) Average resource utilization (%) vs. user population profile

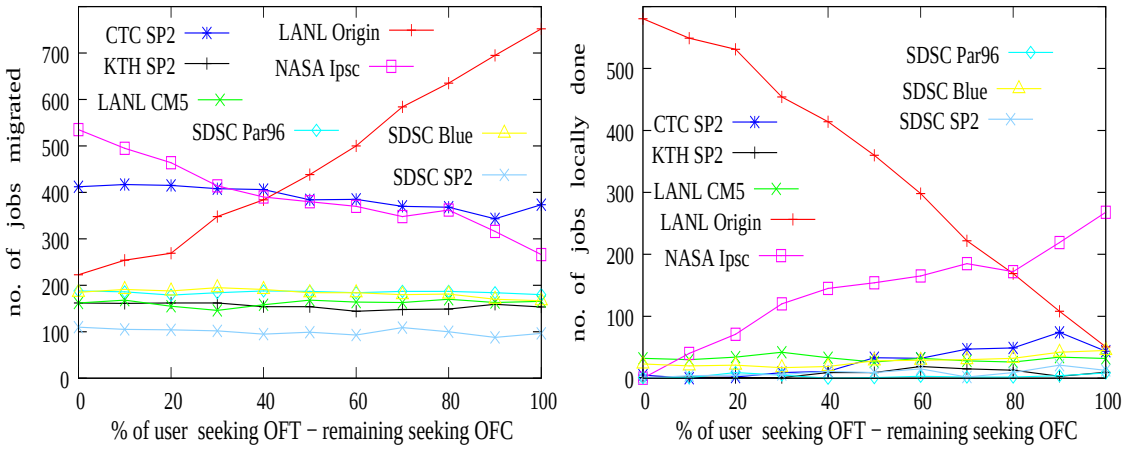
(b) No. of jobs rejected vs. user population profile

Figure 4.5: Resource owner perspective.

Furthermore, the results indicate an imbalance between the resource supply and demand pattern. As the demand was high for the cost-effective resources compared to the time-effective resources, these time-effective resources remained largely underutilized. In this case, the majority of jobs were scheduled on the cost-effective computational resources (LANL CM5, LANL Origin, SDSC Par96 and SDSC Blue). This is the worst

case scenario in terms of resource owner's incentive across all the resources in the federation. Although, when the majority of end-users sought OFT (more than 50%), we observed uniform distribution of jobs across resources in the federation. Every resource owner across the federation received significant incentive (refer to Fig. 4.4(a)) and had improved resource utilization (refer to Fig. 4.5(a)). These scenarios show balance in the resource supply and demand pattern.

Further, in this case (the majority of users sought OFT (more than 50 percent)), the average resources in terms of cost/time effectiveness (SDSC Par96 and SDSC Blue) made significant incentive (which can also be seen in their average utilization) as compared to when OFC users constituted the majority population. Probably, this is due to computational strength of cost-effective resources (Since LANL Origin and LANL CM5 offered 2048 and 1024 nodes, therefore collectively they satisfied the majority of end-users). So, when OFT users formed the majority it resulted in increased inflow of the remote jobs to these average resources. Similar trends can be identified in their respective total remote job service count (refer to Fig. 4.4(b)). Note that, total remote job service count for cost-effective computational resources (LANL Origin, LANL CM5) decreased considerably as the majority of end-users sought OFT(refer to Fig. 4.4(b)).



(a) No. of jobs migrated vs. user population profile (b) No. of jobs locally done vs. user population profile

Figure 4.6: Resource owner perspective.

Fig. 4.6 shows job migration characteristics at various resources with different population profile. We observed that the most cost-efficient resource (LANL Origin) experi-

enced increased job migration rate in the federation as the majority of its users opted for OFT. Conversely, for the most time-efficient resource (NASA iPSC) we observed slight reduction in the job migration rate.

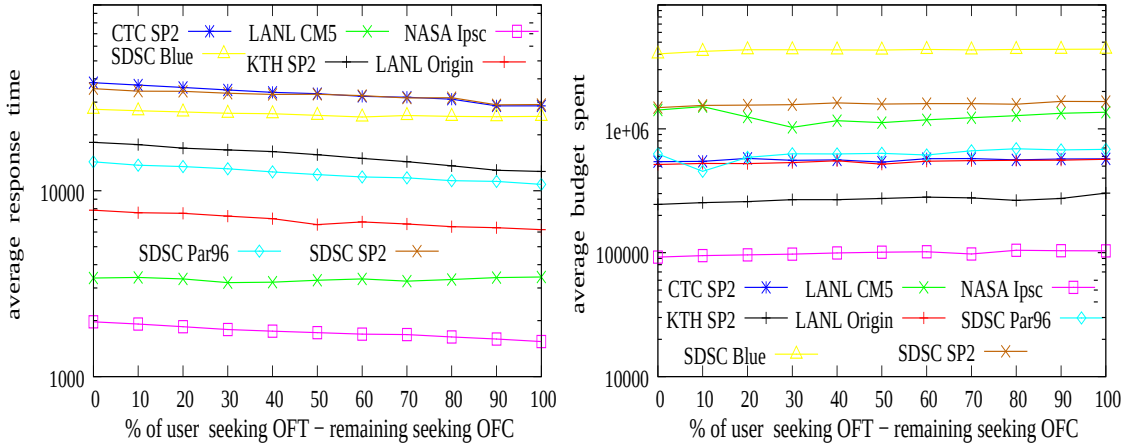
Thus, we conclude that resource supply (number of resource providers) and demand (number of resource consumers and QoS constraint preference) pattern can determine resource owner's overall incentive and his resource usage scenario.

End Users Perspective

We measured end-users QoS satisfaction in terms of the average response time and the average budget spent under OFC and OFT. We observed that the end-users experienced better average response times (excluding rejected jobs) when they sought OFT for their jobs as compared to OFC (100% users seek OFC). At LANL Origin (excluding rejected jobs) the average response time for users was 7.865×10^3 simulation seconds which reduced to 6.176×10^3 for OFT (100% users seek OFT) (refer to Fig. 4.7(a)). The end-users spent more budget in the case of OFT as compared OFC (refer to Fig. 4.7(b)). This shows that users get more utility for their QoS constraint parameter response time, if they are willing to spend more budget. Overall, the end-users across all the resources in the federation experienced improved response time when the majority constituted OFT population. Although, the end-users belonging to resource LANL CM5 did not had significant change in their response time even with OFT preference. It may be due to their job arrival pattern, that may have inhibited them from being scheduled on the time-efficient resources (though we need to do more investigation including job arrival pattern and service pattern at various resources in order to understand this).

Note that, Fig. 4.8(a) and Fig. 4.8(b) include the expected budget spent and response time for the rejected jobs assuming they are executed on the originating resource. Fig. 4.5(b) depicts the number of jobs rejected across various resources during economy scheduling. During this experiment, we also quantified the average response time and the average budget spent at the fastest (NASA iPSC) and the cheapest resource (LANL Origin) when they are not part of the Grid-Federation (without federation). We observed that the average response time at NASA iPSC was 1.268×10^3 (without federation) simulation seconds as compared to 1.550×10^3 (refer to Fig. 4.8(a)) simulation seconds during OFT (100% users

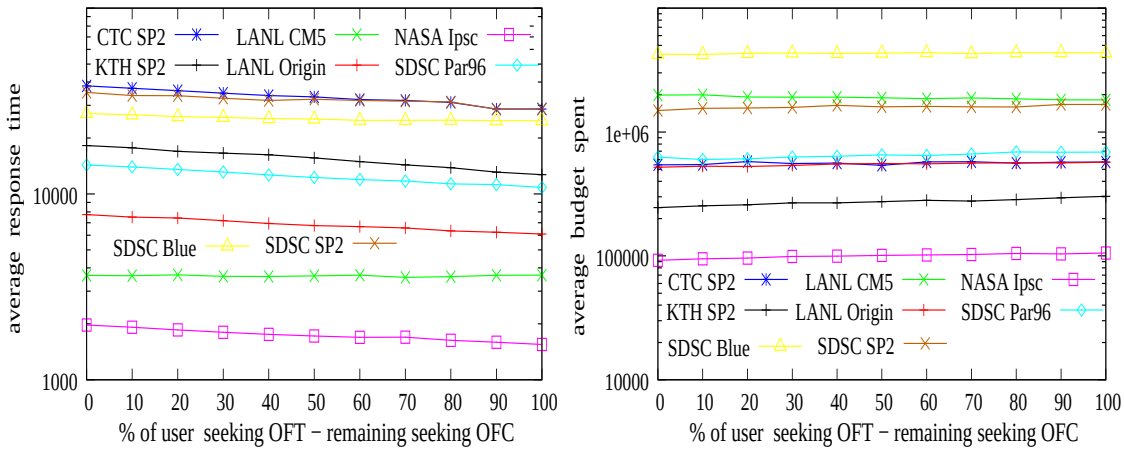
seek OFT) (as part of federation). Accordingly, at LANL Origin the average budget spent was 4.851×10^5 (without federation) Grid Dollars as compared to 5.189×10^5 (refer to Fig. 4.8(b)) Grid Dollars during OFC (100% users seek OFC) (as part of the federation). Note that, the plots Fig. 4.8(a) and Fig. 4.8(b) do not include the average response time and budget spent for without federation case.



(a) Average response time (Sim Units) vs. user population profile

(b) Average budget spent (Grid Dollars) vs. user population profile

Figure 4.7: Federation user perspective: excluding rejected jobs.



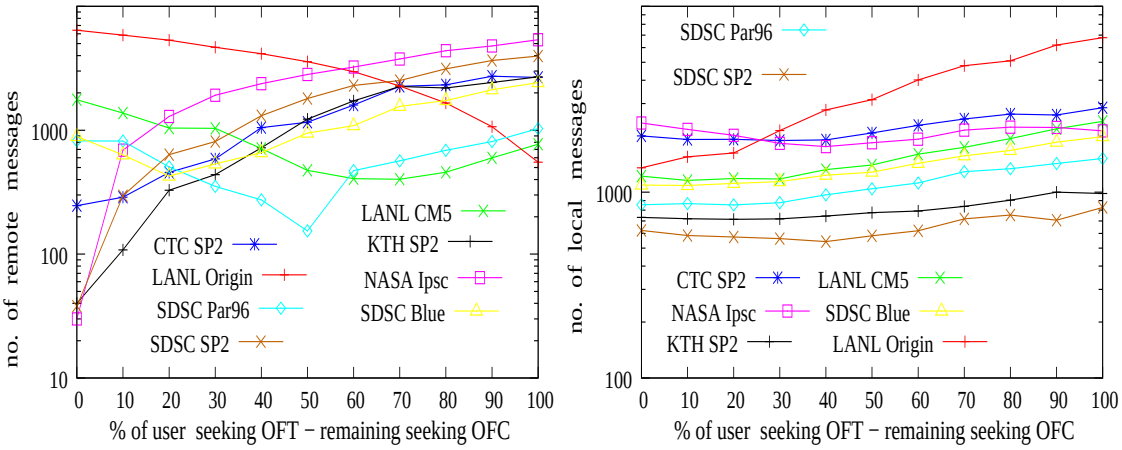
(a) Average response time (Sim Units) vs. user population profile

(b) Average budget spent (Grid Dollars) vs. user population profile

Figure 4.8: Federation user perspective: including rejected jobs.

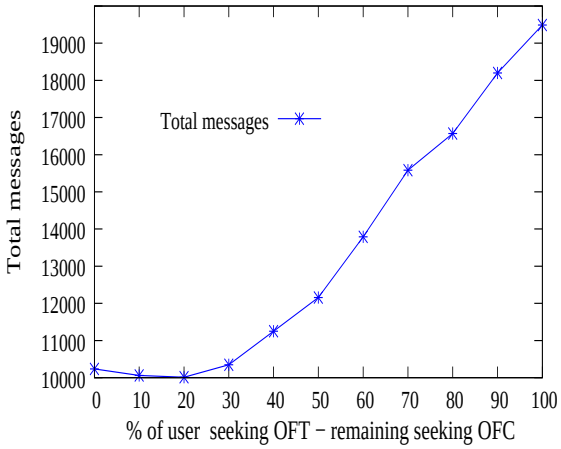
Clearly, this suggests that although federation-based resource sharing leads to better optimisation of objective functions for the end-users across all the resources in the feder-

ation, sometimes it may be a disadvantage to the users who belong to the most efficient resources (in terms of time or cost).



(a) No. of remote messages vs. user population profile

(b) No. of local messages vs. user population profile



(c) Total messages vs. user population profile

Figure 4.9: Remote-Local message complexity.

Remote and Local Message Complexity

In experiment 4, we measured the total number of messages sent and received at various GFA's in the federation with varying user population profiles. Fig. 4.9 shows the plot of the local and remote message count at various GFAs in the federation during economy scheduling. When 100% users seek OFC, we observed that resource LANL Origin received maximum remote messages (6.407×10^3 messages) (refer to Fig. 4.9(a)) followed with LANL CM5 (the second cheapest). LANL Origin offers the least cost, so in this

case every GFA in the federation attempted to migrate their jobs to LANL Origin, hence leading to increased inflow of the remote messages. While when 100% users seek OFT, we observed maximum number of remote messages at the resource NASA iPSC (refer to Fig. 4.9(a)) followed by SDSC SP2 (the second fastest). Since, these resources were time-efficient, therefore all the GFAs attempted to transfer their jobs to them. The total messages involved during this case was 1.948×10^4 as compared to 1.024×10^4 during OFC. This happened because the resources LANL Origin and LANL CM5 had 2048 and 1024 computational nodes and a fewer number of negotiation messages were undertaken between GFA's for the job scheduling.

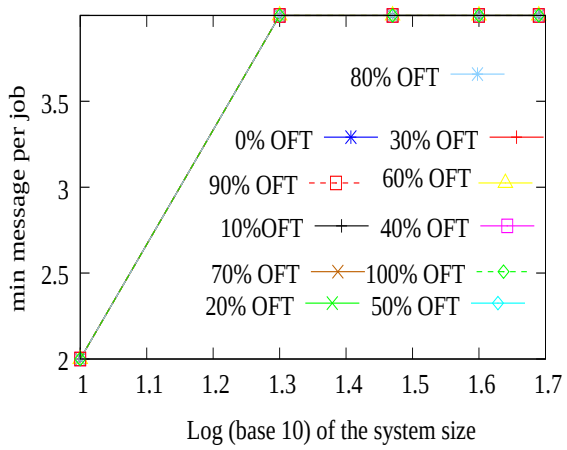
Fig. 4.9(b) shows total number of local messages undertaken at a resource for scheduling work. The results shows, as more users sought OFT, it resulted in increased local message count at cost-effective resources (LANL Origin, LANL CM5). Conversely, faster resources experienced greater remote message count. With 50% seek OFC and 50% seek OFT, we observed uniform distribution of local and remote messages across the federation (refer to Fig.4.9(a)).

To summarise, we observed linear increase in the total message count with increasing number of the end-users seeking OFT for their jobs (refer to Fig. 4.9(c)). Hence, this suggests that the resource supply and demand pattern directly determines the total number of messages undertaken for the job scheduling in the computational economy based Grid system.

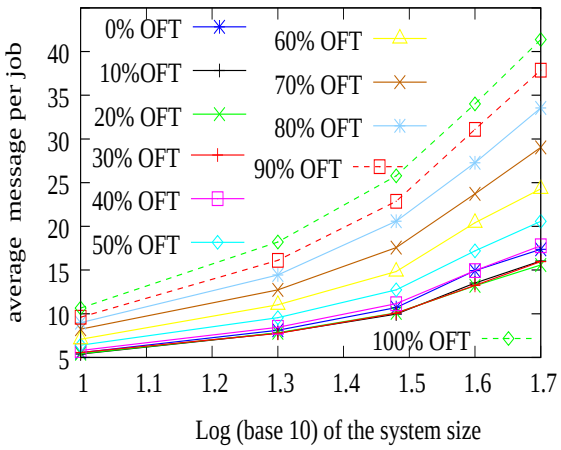
Overall, it can be concluded that the population mix of users in which 70% seek OFC and 30% seek OFT seems most suitable from the system and a resource owner perspective. In this case, we observed uniform distribution of jobs, incentives across the resources. Further, this population mix does not lead to excessive message count as compared to other population mix having greater percentage of users seeking OFT.

System's Scalability Perspective

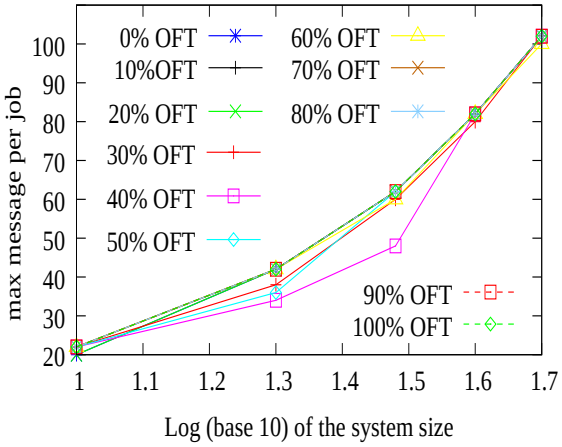
In experiment 5, we measured the proposed system's scalability with increasing numbers of resource consumers and resource providers. The first part of this experiment is concerned with measuring the average number of messages required to schedule a job in the federation as the system scales. We observed that at a system size of 10, OFC scheduling



(a) Min message per job vs. system size



(b) Average message per job vs. system size

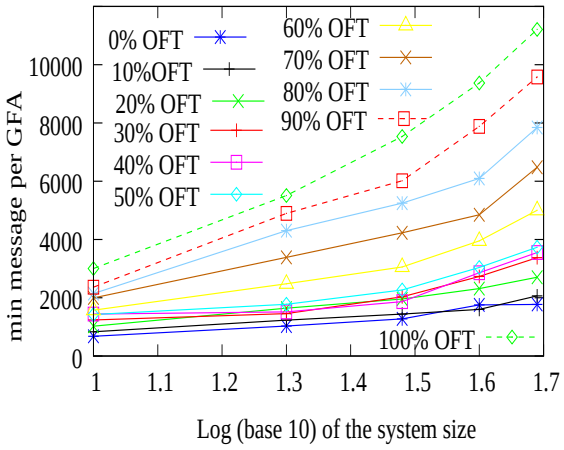


(c) Max message per job vs. system size

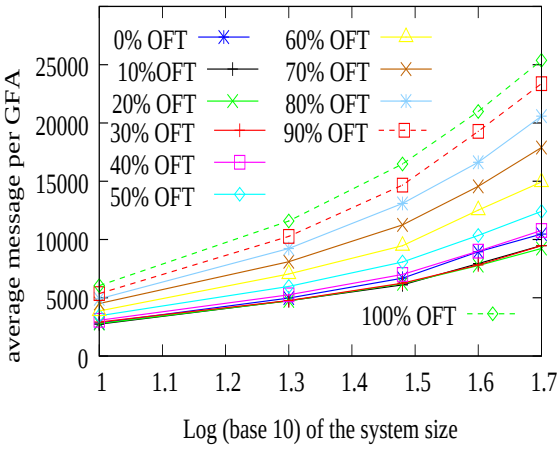
Figure 4.10: System's scalability perspective: message complexity per job with increasing system size.

required an average 5.55 (refer to Fig. 4.10(b)) messages as compared to 10.65 for OFT (Fig. 4.10(b)). As the system scaled to 50 resources, the average message complexity per job increased to 17.38 for OFC as compared to 41.37 during OFT. This suggests that OFC job scheduling required less number of messages than OFT job scheduling, though we need to do more work to determine whether this is due to other factors such as budgets/deadlines assigned to jobs. We also measured the average number of (sent/received) messages at a GFA while scaling the system size (refer to Fig. 4.11). During OFC with 10 resources, a GFA sent/received an average 2.836×10^3 (refer to Fig. 4.11(b)) messages to undertake scheduling work in the federation as compared to 6.039×10^3 (refer to Fig. 4.11(b)) messages during OFT. With 40 resources in the federation, the average mes-

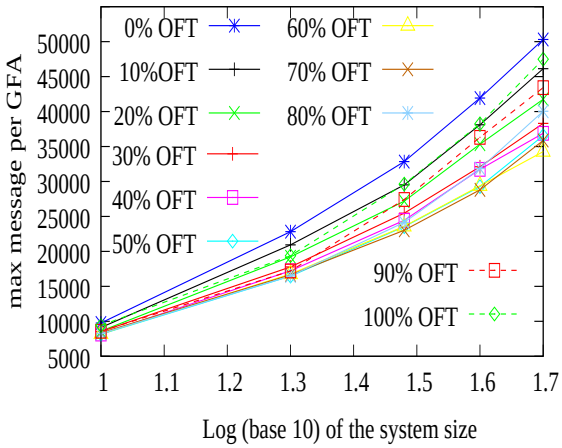
sage count per GFA increased to 8.943×10^3 for OFC as regards to 2.099×10^4 messages for OFT.



(a) Min message per GFA vs. system size



(b) Average message per GFA vs. system size



(c) Max message per GFA vs. system size

Figure 4.11: System's scalability perspective: message complexity per GFA with increasing system size.

Figures 4.10(b) and 4.11(b) suggests that the user population including 10%, 20% or 30% OFT seekers involves less number of messages per job/per GFA basis in comparison to 0% OFT seekers. However, with further increase in OFT seekers generates more messages per job/per GFA basis.

From Fig. 4.10(b) and 4.11(b), note that the average message count grows relatively slowly to an exponential growth in the system size. Thus, we can expect that the average message complexity of the system is scalable to a large system size. More analysis is required to understand the message complexity in this case. However, the maximum

message count suggests that some parts of the system are not scalable and we need to do more work to avoid these worst cases, e.g. by incorporating more intelligence into the shared federation directory.

Overall, we averaged the budget spent for all the users in the federation during OFC and without federation (independent resources). We observed that during OFC, the average budget spent was 8.874×10^5 Grid Dollars (we included the expected budget spent of rejected jobs on the originating resource) as compared to 9.359×10^5 during without federation. However, at most popular resource (LANL Origin) the average budget spent for local users during OFC was 5.189×10^5 as compared to 4.851×10^5 during without federation. Similarly, we averaged the response time for all the users in the federation during OFT and without federation. We observed that during OFT, the average response time was 1.171×10^4 simulation units (we included the expected response time of rejected jobs on the originating resource) as compared to 1.207×10^4 during without federation. But at the most popular resource (NASA iPSC) the average response time for local users during OFT was 1.550×10^3 as compared to 1.268×10^3 during without federation. Clearly, this suggests that while some users that are local to the popular resources can experience higher cost or longer delays during the federation based resource sharing but the overall users' QoS demands across the federation are better met.

4.4 Related Work

Resource management and scheduling for parallel and distributed systems has been investigated extensively in the recent past (AppLes, NetSolve [35], Condor, LSF, SGE, Legion, Condor-Flock, NASA-Superscheduler, Nimrod-G and Condor-G). In this chapter, we mainly focus on superscheduling systems that allow scheduling jobs across wide area distributed clusters. We highlight the current scheduling methodology followed by Grid superscheduling systems including NASA-superscheduler, Condor-Flock (based on P2P substrate Pastry [143]), Legion-based federation and Resource Brokers. Furthermore, we also discuss some computational economy based cluster and Grid systems.

The work in [152] models a Grid superscheduler architecture and presents three different distributed job migration algorithms. In contrast to this superscheduling system,

our approach differs in the following (i) the job-migration or the load-balancing in the Grid-Federation is driven by user specified QoS constraints and resource owners' sharing policies; (ii) our approach gives a resource owner complete autonomy over resource allocation decision; and (iii) our superscheduling mechanism utilizes decentralised shared federation directory for indexing and querying the resources.

The work in [29] presents a superscheduling system that consists of Internet-wide Condor work pools. They utilize Pastry routing substrate to organize and index the Condor work pool. The superscheduling mechanism is based on system-centric parameters. In comparison to this work, Grid-Federation is based on decentralised shared federation directory. Further, our superscheduling scheme considers user-centric parameters for job scheduling across the federation.

OurGrid [9] provides a Grid superscheduling middle-ware infrastructure based on the P2P network paradigm. The OurGrid community is basically a collection of a number of OurGrid Peer (OG Peer) that communicate using P2P protocols. Superscheduling in OurGrid is primarily driven by the site's reputation in the community. In contrast, we propose more generalized resource sharing system based on real-market models. Further, our superscheduling system focuses on optimising resource owners and consumers objective functions.

Bellagio [13] is a market-based resource allocation system for federated distributed computing infrastructures. Resource allocation in this system is based on bid-based proportional resource sharing model. Bids for resources are cleared by a centralised auctioneer. In contrast, we propose a decentralised superscheduling system based on commodities markets. Resource allocation decision in our proposed system is controlled by the concerned site , hence providing complete site autonomy.

Tycoon [109] is a distributed market-based resource allocation system. Application scheduling and resource allocation in Tycoon is based on decentralised isolated auction mechanism. Every resource owner in the system runs its own auction for his local resources. Furthermore, auctions are held independently, thus clearly lacking any coordination. In contrast, we propose a mechanism for cooperative and coordinated sharing of distributed clusters based on computational economy. We apply commodity market model for regulating supply and demand of resources in the Grid-Federation.

Nimrod-G [3] is an resource management system (RMS) that serves as a resource broker and supports deadline and budget constrained algorithms for scheduling task-farming applications on the platform. The superscheduling mechanism inside the Nimrod-G does not take into account other brokering systems currently present in the system. This can lead to over-utilization of some resources while under-utilization of others. To overcome this, we propose a set of distributed brokers having a transparent coordination mechanism.

Other systems including Libra [153] and REXEC [46] apply market methodologies for managing cluster resources within a single administrative domain. Finally in Table 4.1, we summarise various superscheduling systems based on underlying network model, scheduling parameter and scheduling mechanism.

4.5 Conclusion

We proposed a new computational economy based distributed cluster resource management system called Grid-Federation. The federation uses agents that maintain and access a shared federation directory of resource information. A cost-time scheduling algorithm was applied to simulate the scheduling of jobs using iterative queries to the federation directory. Our results show that, while the users from popular (fast/cheap) resources have increased competition and therefore a harder time to satisfy their QoS demands, in general the system provides an increased ability to satisfy QoS demands over all users. The result of the QoS based resource allocation algorithm indicates that the resource supply and demand pattern affects resource provider's overall incentive. Clearly, if all users are seeking either time/cost optimisation then the slowest/most expensive resource owners will not benefit as much. However if there is a mix of users, some seeking time and some seeking cost optimisation then all resource providers gain some benefit from the federation.

We analyzed how the resource supply and demand pattern affects the system scalability/performance in terms of total message complexity. In general, the cost-time scheduling heuristic does not lead to excessive messages, i.e. to excessive directory accesses and we expect the system to be scalable. Overall, the proposed Grid-Federation, in conjunction with a scalable, shared, federation directory, is a favourable model for building large scale Grid systems.

Chapter 5

SLA-driven Coordination Between Grid Brokers

This chapter presents an SLA based Grid superscheduling approach that promotes co-operative resource sharing. Superscheduling is facilitated between administratively and topologically distributed Grid sites via Grid schedulers such as resource brokers and workflow engines. The proposed market-based broker-to-broker SLA mechanism is based on a well-known agent coordination protocol, called *contract net*. The key advantages of this approach are that it allows: (i) enhanced one-to-one coordination among distributed Grid resource brokers; (ii) resource owners to have finer degree of control over the resource allocation, which is something that is not possible with traditional mechanisms; and (iii) brokers to bid for SLA contracts in the contract net, with focus on completing a job within a user specified deadline. Trace based simulation study is conducted in order to prove the feasibility of the proposed SLA-based coordination protocol.

5.1 Introduction

In this chapter, we propose an SLA [5, 51, 55, 130] based coordinated superscheduling scheme for federated Grid systems. An SLA is the agreement negotiated between a superscheduler (resource consumer) and LRMSes (resource provider) about acceptable job QoS constraints. These QoS constraints may include the job response time and budget

spent. Inherently, an SLA is the guarantee given by a resource provider to a remote site job superscheduler for completing the job within the specified deadline, within the agreed budget or satisfying both at the same time. A SLA-based coordinated job superscheduling approach has several advantages: (i) it inhibits superschedulers from submitting unbounded amounts of work to LRMSes; (ii) once an SLA is reached, users' are certain that agreed QoS shall be delivered by the system; (iii) job queuing or processing delay is significantly reduced, thus leading to enhanced QoS, otherwise a penalty model [179] is applied to compensate them; and (iv) gives LRMSes more autonomy and better control over resource allocation decisions.

Our SLA model incorporates an economic mechanism [13, 31, 109] for job superscheduling and resource allocation. The economic mechanism enables the regulation of supply and demand of resources, offers incentive to the resource owners for leasing, and promotes QoS based resource allocation. Recall from Chapter 4, we mainly focus on the decentralised commodity market model [176]. In this model every resource has a price, which is based on the demand, supply and value. An economy driven resource allocation methodology focuses on: (i) optimizing resource provider's payoff function; and (ii) increasing end-user's perceived QoS value; and (iii) guarantees that the advertised or negotiated resource behavior are delivered to the consumers else providers are penalized. Note that our proposed superscheduling approach is studied as part of the Grid-Federation scheduling system (refer to Chapter 4). Finer details about this system is presented in Chapter 4.

Our SLA model considers a collection of computational cluster resources as a contract net [157]. As jobs arrive, the Grid superschedulers undertake one-to-one contract negotiation with the LRMSes managing the concerned resource. The SLA contract negotiation message includes: (i) whether a job can be completed within the specified deadline; and (ii) SLA bid expiration time (maximum amount of time a superscheduler is willing to wait before finalizing the SLA). The SLA bid expiration time methodology we apply here is different from that adopted in the Tycoon system [109]. In Tycoon, the SLA bid expiration time at a resource is the same for all the jobs irrespective of their size or deadline. In this case, the total bid-processing delay is directly controlled by the local resource auctioneer. In our model, the superscheduler bids with an SLA bid expiration time proportional

to the job's deadline. The focus is on meeting the job's SLA requirements, particularly the job's deadline. The SLA contract negotiation in NASA-Scheduler and Condor-Flock P2P [29] is based on general broadcast and limited broadcast communication mechanism respectively. Hence, these approaches have the following limitations: (i) high network overhead; and (ii) scalability problems.

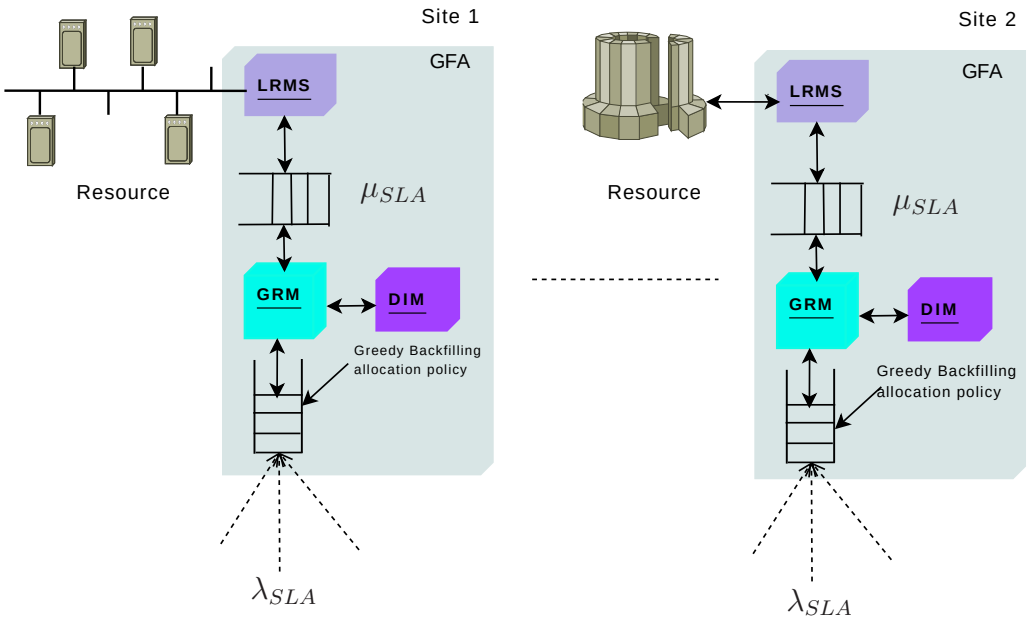


Figure 5.1: SLA bid queues in the Grid-Federation.

Our time constrained SLA bid-based contract negotiation approach gives LRMSes finer control over the resource allocation decision as compared to traditional First-Come-First-Serve (FCFS) approach. Existing superscheduling systems including Nimrod-G, NASA-Scheduler, Condor-Flock P2P, Condor-G and Legion-Federation [174] assume every LRMS allocates the resources using FCFS scheduling scheme. In this work, we propose a Greedy Back-filling LRMS scheduling that focuses on maximizing resource owner's payoff function. In this case, a LRMS maintains a queue of SLA bid requests generated by various superschedulers in the system at a time t. Every SLA bid has an associated expiry time. If the concerned LRMS does not reply within that expiry period, then the SLA request is considered to be expired. Greedy back-filling is based on well known Greedy or Knapsack method [23, 50, 78]. A LRMS periodically iterates through the local SLA bids and finalizes the contract with those that fit the resource owner's payoff

function.

Fig. 5.1 shows the queue of SLA bids at each site in the federation. Every incoming SLA bid is added to the GRM request queue, $Q_{m,t}$ and a bid expiration timeout event is scheduled after time interval $\tau(J_{i,j,k})$. Every resource i has a different SLA bid arrival rate, λ_{SLA_i} and SLA bid satisfaction rate, μ_{SLA_i}.

The rest of this chapter is organised as follows. Section 5.2.1 presents details about our proposed bid-based SLA contract negotiation model. In Section 5.2.2, we give details about our proposed Greedy backfilling LRMS scheduling approach. In Section 5.3, we present various experiments and discuss our results. Section 5.4 presents some of the related work in negotiation-based superscheduling. We end this chapter with concluding remarks in Section 5.5.

5.2 Models

5.2.1 SLA model

The SLA model we consider is that of a set of distributed cluster resources each offering a fixed amount of processing power. The resources form part of the federated Grid environment and are shared amongst the end-users, each having its own SLA parameters. SLAs are managed and coordinated through an admission control mechanism enforced by GFA at each resource site. Each user in the federation has a job $J_{i,j,k}$. We write $J_{i,j,k}$ to represent the i-th job from the j-th user of the k-th resource. A job consists of the number of processors required, $p_{i,j,k}$, the job length, $l_{i,j,k}$ (in terms of millions of instructions), the communication overhead, $\alpha_{i,j,k}$ and SLA parameters, i.e., the budget, $b_{i,j,k}$, the deadline or maximum delay, $d_{i,j,k}$. More details about the job model can be found in Chapter 4.

SLA bid with expiration time

The collection of GFAs in the federation are referred to as a contract net, and job-migration in the net is facilitated through the SLA contracts. Each GFA can take on two roles either a *manager* or *contractor*. The GFA to which a user submits a job for processing is referred to as the manager GFA. The manager GFA is responsible for superscheduling the job in

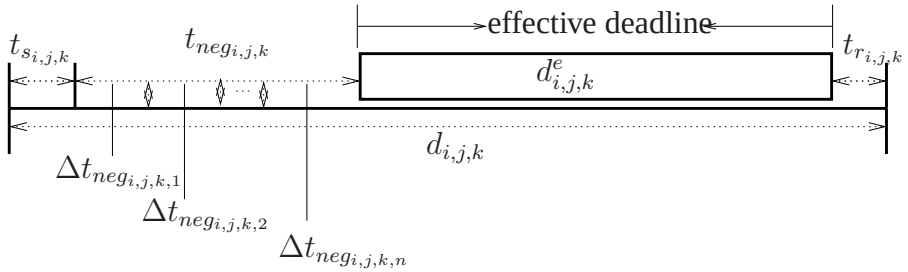


Figure 5.2: Job superscheduling timeline.

the net. The GFA which accepts the job from the manager GFA and overlooks its execution is referred to as the contractor GFA. Individual GFAs are not assigned these roles in advance. The role may change dynamically over time as per the user's job requirements. Thus, the GFA alternates between these two roles or adheres to both over the course of superscheduling.

As jobs arrive at a GFA, the GFA adopts the role of a manager. Following this, the manager GFA queries the shared federation directory to obtain the quote for the contractor GFA that matches the user specified SLA parameters. Note that, users can seek optimization for one of the SLA parameters i.e. either response time (OFT) or the budget spent (OFC). Once, the manager obtains the quote for the desired contractor, it undertakes one-to-one SLA contract negotiation with the contractor. The SLA contract negotiation message includes: (i) whether the job $J_{i,j,k}$ can be completed within the specified deadline; and (ii) SLA bid expiration time $\Delta t_{negi,j,k,l}$. The contractor GFA has to reply within the bid time $\Delta t_{negi,j,k,l}$, otherwise the manager GFA undertakes SLA contract negotiation with the next available contractor in the net. Algorithm SLA bidding mechanism (refer to Fig. 5.3) depicts various events and corresponding superscheduling actions undertaken by a GFA.

Our SLA contract model considers a part of the total job deadline as the SLA contract negotiation time (refer to Eq. 5.1). The manager GFA bids with a different SLA expiration interval given by Eq. 5.2. In Fig. 5.2 we show the job superscheduling timeline. The timeline includes the job submission delay, $t_{s_{i,j,k}}$, total SLA contract negotiation delay, $t_{neg_{i,j,k}}$, expected response time (computed using Eq. 5.1) and finished job return delay, $t_{r_{i,j,k}}$. The total SLA contract bidding delay available to the manager GFA for superscheduling job $J_{i,j,k}$ is given by:

```
1  PROCEDURE: SLA_BIDDING_MECHANISM
2  begin
3      begin
4          SUB-PROCEDURE: EVENT_USER_JOB_SUBMIT (J_{i,j,k})
5          call SLA_BID (J_{i,j,k}).
6      end
7      begin
8          SUB-PROCEDURE: SLA_BID (J_{i,j,k})
9          Send SLA bid for job J_{i,j,k} to the next available contractor GFA (obtained by querying
           the shared federation directory).
10     end
11     begin
12         SUB-PROCEDURE: EVENT_SLA_BID_REPLY (J_{i,j,k})
13         if SLA Contract Accepted then
14             Send the job J_{i,j,k} to accepting GFA.
15         end
16         else
17             call SLA_BID_TIMEOUT (J_{i,j,k}).
18         end
19     end
20     begin
21         SUB-PROCEDURE: SLA_BID_TIMEOUT(J_{i,j,k})
22         if τ (J_{i,j,k}) ≥ 0 then
23             call SLA_BID (J_{i,j,k}).
24         end
25         else
26             Drop the job J_{i,j,k}.
27         end
28     end
29 end
```

Figure 5.3: SLA bidding mechanism.

$$t_{neg_{i,j,k}} = d_{i,j,k} - t_{s_{i,j,k}} - d^{e}_{i,j,k} - t_{r_{i,j,k}} \tag{5.1}$$

The total SLA contract bid negotiation delay $t_{neg_{i,j,k}}$ assumes a finite number of values $\Delta t_{neg_{i,j,k,1}}$, $\Delta t_{neg_{i,j,k,2}}$,...,$\Delta t_{neg_{i,j,k,n}}$ in superscheduling a job $J_{i,j,k}$ (refer to Fig. 5.2). We define the value of $\Delta t_{neg_{i,j,k,l}}$ by

$$\Delta t_{neg_{i,j,k,l}} = \frac{t_{neg_{i,j,k}} - \sum_{p=1}^{l-1} \Delta t_{neg_{i,j,k,p}}}{2}, l > 0 \tag{5.2}$$

Note that, the value for $\Delta t_{neg_{i,j,k,l}}$ can be given by other distributions [7] such as uniform or random. We intend to analyze various distributions for an SLA bid interval and study its effect on our proposed superscheduling approach in our future work. For simplicity, in this work we use the distribution given by Eq. 5.2.

As the superscheduling iteration increases, the manager GFAs give less time to the contractor to decide on the SLA in order to meet the user's job deadline. This approach allows a large number of scheduling iterations to the manager GFA. However, if the user's SLA parameters cannot be satisfied (after iterating up to the greatest r such that GFA could feasibly complete the job), then the job is dropped. To summarise, an SLA bid for job $J_{i,j,k}$ includes:

- l-th SLA bid expiry interval $t_{neg_{i,j,k,l}}$ (computed using Eq. 5.2);
- expected response time ($d^e_{i,j,k}$) (computed using Eq. 5.1).

We consider the function:

$$\tau : J_{i,j,k} \longrightarrow \mathcal{Z}^+ \tag{5.3}$$

which returns the next allowed SLA bidding time interval $\Delta t_{neg_{i,j,k,p}}$ for a job $J_{i,j,k}$ using Eq.5.2.

5.2.2 Greedy backfilling: (LRMS scheduling model)

Most of the existing LRMSes apply system-centric policies for allocating jobs to resources. Some of the well-known system-centric policies include: (i) FCFS; (ii) Conservative back-filling [160]; and (ii) Easy back-filling [65]. Experiments [144] have shown that the job back-filling approach offers significant improvement in performance over the FCFS scheme. However, these system centric approaches allocate resources based on parameters that enhance system utilization or throughput. The LRMS either focuses on minimizing the response time (sum of queue time and actual execution time) or maximizing overall resource utilization of the system, and these are not specifically applied on a per-user basis (user oblivious). Further, the system centric LRMSes treat all resources with the same scale, thus neglecting the resource owner's payoff function. In this case, the resource owners do not have any control over resource allocation decisions. While in reality the resource owner would like to dictate how his resources are made available to the outside world and apply a resource allocation policy that suits his payoff function. To summarize, the system-centric approaches do not provide mechanisms for resource owners to dictate resource: (i) sharing; (ii) access and; (iii) allocation policies.

To address this, we propose a Greedy method based resource allocation heuristic for LRMSes. Our proposed heuristic focuses on maximizing the payoff function for the resource owners. The heuristic is based on the well-known Greedy method [23], [50]. The Greedy method for solving optimization problems considers greedily maximizing or minimizing the short-term goals and hoping for the best, without regard to the long-term effects. This method has been used to solve *the knapsack problem* [78]. Greedy method considers a set S, consisting of n items, with each item i having a positive benefit b_i, a positive weight w_i. Given the knapsack capacity W the Greedy heuristic focuses on maximizing the total benefit $\sum_{i \in S^a} b_i(x_i/w_i)$ with constraint $\sum_{i \in S^a} x_i \leq W$, such that $S^a \subseteq S$. In this case, x_i is the portion of each item i which the Greedy method selects.

The LRMS scheduler iterates through the SLA bid queue in case any of the following events occur: (i) new SLA bid arrives to the site; (ii) job completion; or (iii) an SLA bid reaches its expiration time. Procedure Greedy Back-filling (refer to Fig. 5.4) depicts various events and corresponding scheduling actions undertaken by the LRMS.

5.2.3 Integer linear programming (ILP) formulation of scheduling heuristic

Queue, $Q_{m,t}$, maintains the the set of job SLA bids currently negotiated with the LRMS at GFA m by time t. We consider the SLA bid acceptance variable $x_{i,j,k}$

- Definition of variable:

  $x_{i,j,k} = 1$ if the SLA request for job $J_{i,j,k}$ is accepted;

  $x_{i,j,k} = 0$ otherwise.

  The Greedy Back-filling heuristic accepts SLA requests constrained to the availability of number of processors requested for job $J_{i,j,k}$ and expected response time $d^e_{i,j,k}$.

- Definition of the constraints:

```
1  PROCEDURE: GREEDY_BACKFILLING
2  begin
3      r = p_m
4      c = 0
5      Q_{m,t} ← φ
6      Q^a_{m,t} ← φ
7      Q^s_{m,t} ← φ
8      begin
9          SUB-PROCEDURE:Event_SLA_Bid_ARRIVAL(J_{i,j,k})
10         A SLA request message for the job J_{i,j,k} that arrives at a GFA Q_{m,t} ← Q_{m,t} ∪ {J_{i,j,k}}
11         Schedule the SLA bid timeout event after τ(J_{i,j,k}) time units
12         call STRICT_GREEDY()
13     end
14     begin
15         SUB-PROCEDURE:Event_SLA_Bid_Timeout(J_{i,j,k})
16         A SLA bid for job J_{i,j,k} that reaches timeout period
17         if (r ≥ p_{i,j,k} and d^e_{i,j,k} ≥ D(J_{i,j,k}, R_m)) then
18             Call RESERVE(J_{i,j,k})
19         end
20         else
21             Reject the SLA bid for job J_{i,j,k}
22             Reset Q_{m,t} ← Q_{m,t} − {J_{i,j,k}}
23         end
24     end
25     begin
26         SUB-PROCEDURE:Event_Job_Finish(J_{i,j,k})
27         A job J_{i,j,k} that finishes at a GFA  Reset r = r + p_{i,j,k}
28         call STRICT_GREEDY()
29     end
30     begin
31         SUB-PROCEDURE: RESERVE(J_{i,j,k})
32         Reserve p_{i,j,k} processors for the job J_{i,j,k}
33         Reset r = r − p_{i,j,k}, Q_{m,t} ← Q_{m,t} − {J_{i,j,k}}, Q^a_{m,t} ← Q^a_{m,t} ∪ {J_{i,j,k}}
34     end
35     begin
36         SUB-PROCEDURE: STRICT_GREEDY()
37         Reset c = 0
38         Sort the SLA bids in Q_{m,t} in decreasing order of incentives and store in Q^s_{m,t}
39         Get the next SLA bid for job J_{i,j,k} from the list Q^s_{m,t}, c=c+1
40         if (r ≥ p_{i,j,k} and d^e_{i,j,k} ≥ D(J_{i,j,k}, R_m)) then
41             Call RESERVE(J_{i,j,k})
42         end
43         else
44             if c < sizeof(Q^s_{m,t}) then
45                 Iterate through step 1.39
46             end
47         end
48     end
49 end
```

Figure 5.4: Greedy-Backfilling resource allocation algorithm.

$$\sum_{\substack{1\leq i\leq n_j \\ 1\leq j\leq n_u \\ 1\leq k\leq n}} p_{i,j,k} \leq p_m \tag{5.4}$$

p_m total number of processors available at a LRMS (GFA) m. $p_{i,j,k}$ denotes number of processor requested during the SLA bid for job $J_{i,j,k}$. All the accepted SLA bids for jobs are maintained in the queue $Q^a_{m,t}$.

- Payoff or Objective function: The LRMS scheduler accepts SLA bids for the jobs such that it maximizes the resource owners' payoff functions by applying the Greedy backfilling heuristic

$$I_m = max(\sum_{\substack{1\leq i\leq n_j \\ 1\leq j\leq n_u \\ 1\leq k\leq n \\ 1\leq m\leq n}} B(J_{i,j,k}, R_m)) \tag{5.5}$$

5.2.4 Economic parameters

Setting price (c_i)

The resource owners configure the resource access cost c_i to reflect its demand in the federation. A resource owner can vary c_i depending on the resource demand λ_{SLA_i} and resource supply μ_{SLA_i} pattern. In case, $\lambda_{SLA_i} > \mu_{SLA_i}$, then the resource owner can increase c_i. However, λ_{SLA_i} depends on the user population profile. If the majority of users are seeking optimization for response time then time-efficient resources may increase c_i until $\lambda_{SLA_i} = \mu_{SLA_i}$. Furthermore, to find the bounds of c_i, it is mandatory to consider the amount of budget available to the users.

For simplicity, in this work we assume that c_i remains static throughout the simulations. We intend to analyze different pricing algorithms [43], [155], [176] based on the supply and demand function, as future work. Using the static price c_i, we quantify how varying the SLA bid time affects the federated superscheduling systems' performance. In simulations, we configure c_i using the function:

$$c_i = f(\mu_i) \tag{5.6}$$

where,

$$f(\mu_i) = \frac{c}{\mu}\,\mu_i \tag{5.7}$$

c is the access price and μ is the speed of the fastest resource in the Grid-Federation. Details about how users are charged on per job basis can be found in Chapter 4.

User budget and deadline

While our simulations in the next section use trace data for job characteristics, the trace data does not include user specified budgets and deadlines on a per job basis. In order to study our proposed SLA model and superscheduling approach, we are forced to fabricate these quantities and we include the models here.

For a user, j, we allow each job from that user to be given a budget,

$$b_{i,j,k} = 2\,B(J_{i,j,k}, R_k). \tag{5.8}$$

In other words, the total budget of a user over simulation is unbounded and we are interested in computing the budget that is required to schedule all of the jobs.

Also, we let the deadline for job $J_{i,j,k}$ be

$$d_{i,j,k} = 3\,D(J_{i,j,k}, R_k). \tag{5.9}$$

We assign three times the expected response time for the given job, as compared to expected response time on the originating resource. We use the multiplying constant as 3, for allowing the superschedulers ample time during SLA bidding. However, as a future work we intend to analyse how does the system performance change when multiplying constant approaches 1 and infinity. Details about the budget and time function can be found in Chapter 4.

Table 5.1: Workload and resource configuration.

Index	Resource / Cluster Name	Trace Date	Processors	MIPS (rating)	Total Jobs in Trace	Quote	NIC to Network Bandwidth (Gb/Sec)
1	CTC SP2	June96-May97	512	850	79,302	4.84	2
2	KTH SP2	Sep96-Aug97	100	900	28,490	5.12	1.6
3	LANL CM5	Oct94-Sep96	1024	700	201,387	3.98	1
4	LANL Origin	Nov99-Apr2000	2048	630	121,989	3.59	1.6
5	NASA iPSC	Oct93-Dec93	128	930	42,264	5.3	4
6	SDSC Par96	Dec95-Dec96	416	710	38,719	4.04	1
7	SDSC Blue	Apr2000-Jan2003	1152	730	250,440	4.16	2
8	SDSC SP2	Apr98-Apr2000	128	920	73,496	5.24	4

5.3 Performance Evaluation

5.3.1 Workload and resource methodology

We performed real workload trace driven simulation to evaluate the effectiveness of the proposed SLA based superscheduling approach. We utilised the same traces and resources as described in Chapter 4. For reference, in Table 5.1 we outline the resource characteristics utilised for modeling the simulation environment. Similar to the Chapter 4, our simulation environment models the following basic entities in addition to existing entities in GridSim:

- local user population – models the workload obtained from trace data;
- GFA – generalized RMS system;
- GFA queue – placeholder for incoming jobs from local user population and the federation;
- GFA shared federation directory – simulates an efficient distributed query process such as P2P.

For evaluating the SLA based superscheduling, we assigned a synthetic QoS specification to each resource including the Quote value (price that a cluster owner charges for service), with varying MIPS rating and underlying network communication bandwidth. The simulation experiments were conducted by utilizing workload trace data over the total period of four days (in simulation units) at all the resources. We consider federation

with computational economy mechanism as the resource sharing environment for our experiments.

5.3.2 Experiment 1 - Quantifying scheduling parameters related to resource owners and end-users with varying total SLA bid time

We quantify the following scheduling parameters related to resource owners and end-users:

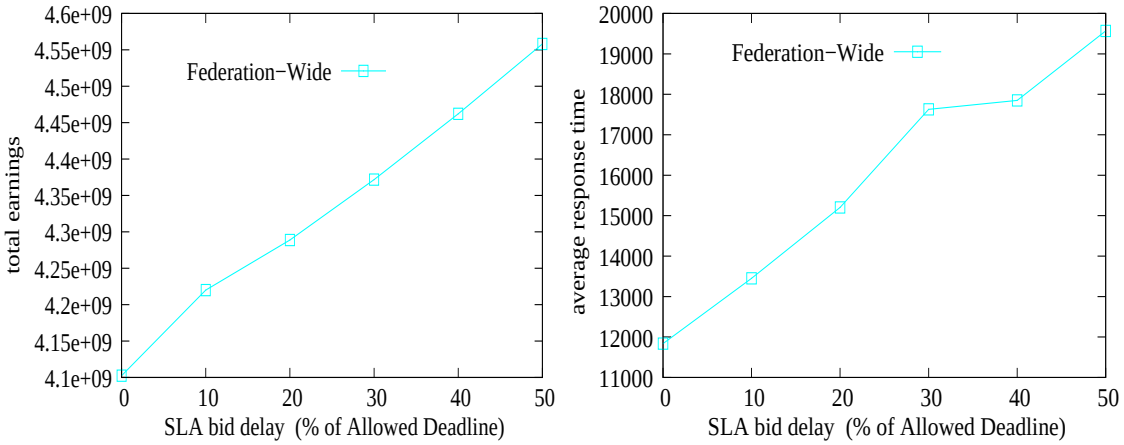
- resource owner: payoff function (total earnings, earnings per processor), resource utilization (in terms of total MI executed);
- end-users: QoS satisfaction (average response time, average budget spent), number of jobs accepted.

We performed the simulations which comprised of end-users seeking OFT for their jobs (i.e. 100% users seek OFT). We vary the total SLA bid from 0% to 50% of total allowed job deadline. In case, no SLA bid delay is allowed (i.e. 0% of total allowed deadline) then the contacted GFA has to immediately make the admission control decision. In this case, we simulate FCFS based strategy for finalizing the SLA. However, in other cases we consider a Greedy Back-filling SLA approach.

5.3.3 Experiment 2 - Quantifying message complexity involved with varying total SLA bid time

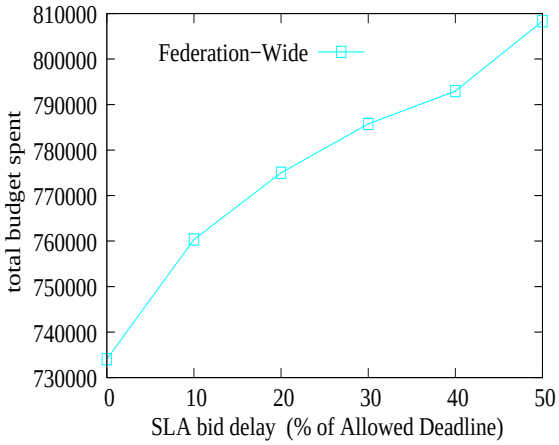
In this experiment we consider the message complexity involved in our proposed superscheduling approach, using the following superscheduling parameters:

- average number of messages per job at a resource i: the number of SLA bid requests undertaken at a resource on the average before the job was actually scheduled;
- local message count: number of SLA bid scheduling messages undertaken for local jobs at a resource i;
- remote message count: number of SLA bid scheduling message overhead for remote jobs at a resource i.



(a) total SLA bid delay vs. total federation earning (Grid dollars)

(b) total SLA bid delay vs. average response time (sim units)



(c) total SLA bid delay vs. average budget spent (Grid dollars)

Figure 5.5: Federation perspective.

5.3.4 Results and observations

Federation perspective

In experiment 1, we measure how varying of the total time for SLA bids coupled with Greedy backfilling resource allocation strategy affects the Grid participants across the federation. We quantify how the additional decision making time given to the LRMSes before finalizing the SLA contracts affects the overall system performance in terms of resource owner's and end-user's objective functions. We observed that when the LRMSes across the federation applied FCFS technique for finalizing the SLAs (i.e. no decision making time was given, so the LRMSes have to reply as soon as the SLA request was made), the

resource owner's made 4.102×10^9 Grid dollars as incentive (refer to Fig.5.5(a)).

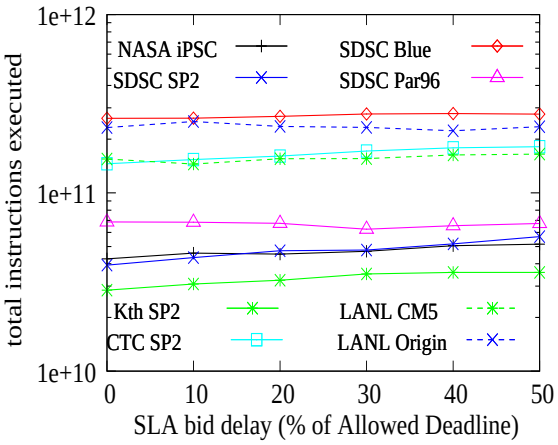
We observed that with an increase in the total SLA bidding time (i.e. as the LRMSes were allowed decision making time before finalizing the SLAs hence they applied Greedy Back-filling scheduling on the queue of SLA bids), the resource owners earned more incentive as compared to the FCFS case. When 10% of the total deadline was allowed for SLA bids, the total incentive earned across the federation increased to 4.219×10^9 Grid dollars. While, in case 50% of total job deadline was allowed for the SLA bids, the total incentive accounted to 4.558×10^9 Grid dollars. Hence, the resource owners across federation experienced an increase of approximately 10% in their incentive as compared to the FCFS case.

However, we observed that with an increase in the total SLA bid delay, the end-users across the federation experienced degraded QoS. During the FCFS case, the average response time across the federation was 1.183×10^4 sim units (refer to Fig.5.5(b)). However, in case of 10% SLA bid delay the average response time increased to 1.344×10^4 sim units. Finally, when 50% of the total job deadline was allowed as SLA bid delay the average response time further increased to 1.956×10^4 sim units. Furthermore, in this case the end-users end up spending more budget as compared to the FCFS case (refer to Fig.5.5(c)).

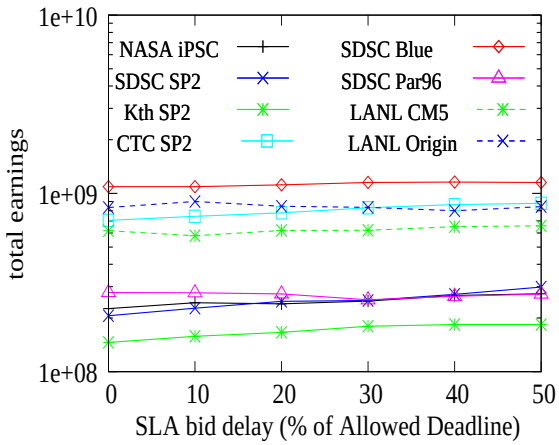
Hence, we can see that although the proposed approach leads to better optimization of resource owners' payoff function, it has degrading effect on the end-user's QoS satisfaction function across the federation.

Resource owner perspective

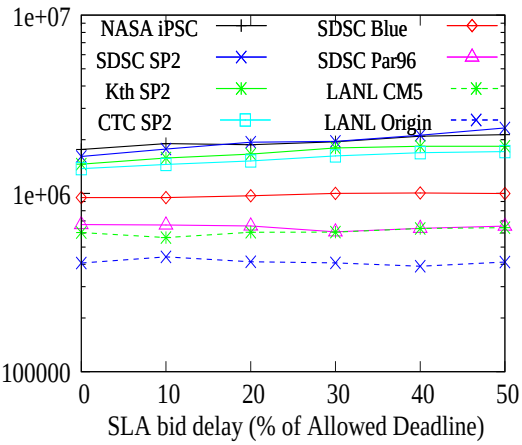
In experiment 1, we quantified how varying the total SLA bid time/delay affects the individual resource owners in the federation. We analyzed, how the proposed approach affects the superscheduling parameters related to the resource owner's payoff function. The most time-efficient resources in the federation i.e. NASA-iPSC, SDSC-SP2, Kth-SP2 and CTC-SP2 (refer to Table 5.1) experienced substantial increase in the total incentive earned with an increase in total decision making time. When no time was allowed for decision making (the FCFS case), these resources earned 1.764×10^6, 1.61×10^6, 1.458×10^6, 1.377×10^6 and 9.464×10^5 Grid dollars (refer to Fig. 5.6(c)) per processor.



(a) total SLA bid delay vs. total MI executed



(b) total SLA bid delay vs. total earnings (Grid dollars)



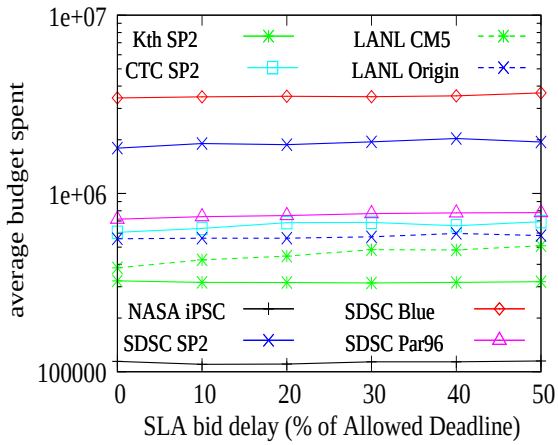
(c) total SLA bid delay vs. total earnings per processor

Figure 5.6: Resource owner perspective.

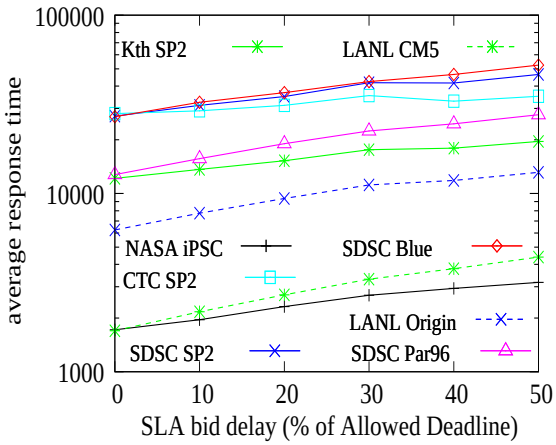
When the jobs in the system were allowed 30% of their total deadline as SLA bid time or admission control decision making time, these resources earned 1.946×10^6, 1.957×10^6, 1.799×10^6, 1.622×10^6 and 9.996×10^5 Grid dollars per processing unit. Same trends can be observed in the plots for total earnings (refer to Fig. 5.6(b)) and number of machine instructions executed during the simulation period (refer to Fig. 5.6(a)).

Thus, we can see that when LRMSes are given decision making time, they have better control over resource allocation/admission control decision. Furthermore, we can see that Greedy Back-filling approach leads to better optimization of owner's payoff function as compared to the FCFS approach.

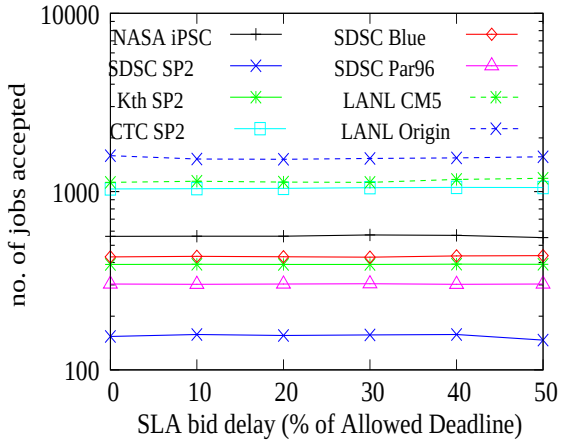
End-users perspective



(a) Average resource utilization (%) vs. user population profile



(b) total SLA bid delay vs. average budget spent (Grid dollars)



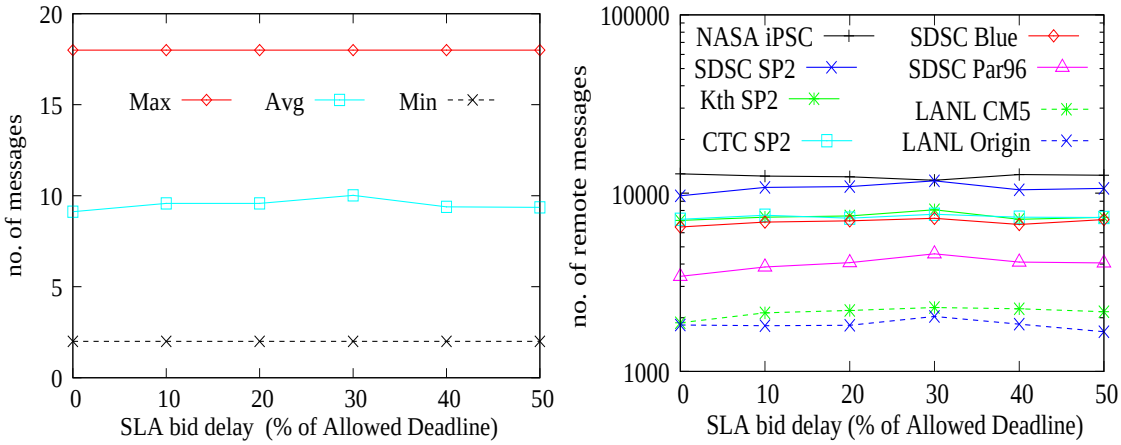
(c) total SLA bid delay vs. no. of jobs accepted

Figure 5.7: End-users perspective.

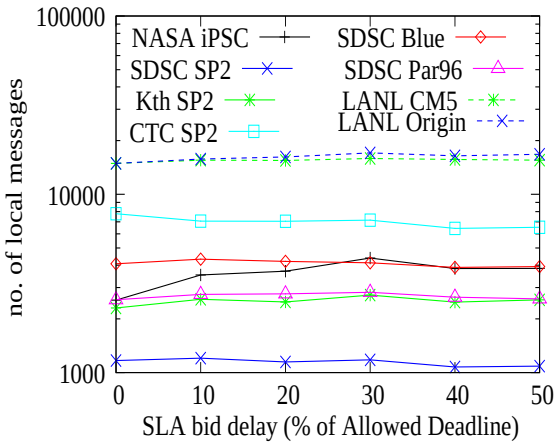
In experiment 1, we also quantified the QoS satisfaction parameters for end-users across all the resources in the Grid-Federation. When LRMSes across the federation applied FCFS scheduling, end-users at the resource NASA-iPSC experienced 1.719×10^3 sim units as average response time (refer to Fig.5.7(b)). They also spent 1.143×10^5 Grid dollars on the average to get their job done in the federation (refer to Fig.5.7(a)). However, when the user's allowed 50% of the total job deadline as SLA bid time, the average response time at NASA-iPSC increased to 3.170×10^3 sim units. In this case, end-users paid 1.14928×10^5 Grid dollars. Fig.5.7(c) depicts the plot for number of jobs accepted for users across resources in the federation with increasing SLA bid time.

Thus, we can see that FCFS based LRMS SLA contract allocation approach is better as far as end-user's QoS satisfaction is concerned as compared to Greedy Back-filling. However, such an approach is difficult to realize into today's Internet based system where resource owners have rational goals and focus on maximizing their payoff function, while delivering an acceptable level of QoS to the end-users.

System message complexity perspective



(a) total SLA bid delay vs. average no. of messages (b) total SLA bid delay vs. no. of remote messages per job



(c) total SLA bid delay vs. no. of local messages

Figure 5.8: System's scalability perspective.

In experiment 2, we quantified the message complexity involved with our proposed superscheduling approach. We measure the number of SLA bid messages required on average across the federation to schedule a job. This metric also includes the messages

for sending the executable and receiving the output. Fig.5.8(a), 5.8(b) and 5.8(c) depicts the plots for scheduling message complexity involved with our approach.

Our simulations show that when no SLA bid delay was allowed, the average SLA bid message per job across the federation was 9.12 (refer to Fig.5.8(a)). As the system allowed 40% of the total job deadline as SLA bid delay, the SLA message/job remained almost the same at about 9.32. Thus, we can see that our proposed superscheduling approach does not incur any additional communication overhead.

In Fig.5.8(b), we quantify the remote superscheduling message complexity at various resources in the Grid-Federation. We observed that the most time-efficient resource i.e. NASA-iPSC received the maximum number of remote messages followed by SDSC-SP2 and KTH-SP2. The same characteristic holds for all cases i.e. as the total SLA bid time increases from 0% to 50% of the allowed job deadline.

In Fig.5.8(c), we quantify the local superscheduling message complexity at various resources in the Grid-Federation. Results show that the resources LANL-Origin and LANL-CM5 were subjected to maximum local superscheduling messages. Both the resources were cost-efficient and all their local users were seeking OFT. Hence these resources undertook SLA bid negotiation with time-efficient resources, causing a large number of superscheduling messages (note that the number of jobs at LANL-Origin and LANL-CM5 were 1706 and 1287) respectively.

5.4 Related work

In this section, we briefly summarize Grid superscheduling approaches that apply a SLA-based or a negotiation-based job superscheduling process.

The work in [130] proposes a multi-agent infrastructure that applies a SLA protocol for solving the Grid superscheduling problem. The SLA negotiation protocol is based on the Contract Net Protocol [156]. In contrast: (i) we propose a SLA-based coordination scheme based on computational economy; and (ii) our work considers site autonomy issues, and propose a Greedy Back-filling resource allocation strategy for a LRMS to maximize resource provider payoff function.

The work in [163] presents a Grid superscheduling technique based on a multiple job

SLA negotiation scheme. The key factor motivating this work is redundantly distributing the job execution requests to multiple sites in the grid instead of just sending a request to least loaded one. The authors argue that placing a job in the queue at multiple sites increases the probability that the backfilling strategy will be more effective in optimizing the scheduling parameters, which includes resource utilization and job average turn around time. In other words, the scheduling parameters are system centric. In contrast to this superscheduling system, our approach differs in the following: (i) job-migration or SLA-based coordination is based on user centric scheduling parameters; (ii) our approach gives a LRMSes more flexibility over resource allocation decision; and (iii) our cluster resource allocation mechanism i.e. Greedy Back-filling algorithm focuses on maximizing resource owners payoff function.

The work in [152] models a Grid superscheduler architecture. Each Grid site has a Grid scheduler (GS), Grid middleware (GM) and a local scheduler (LRMS). Three different cooperative superscheduling schemes are presented for distributed load-balancing. Effectively, the information coordination in this approach is based on complete broadcast communication approach that may generate a large number of network messages. Such an approach has serious scalability concerns. Further, each GS in the system allocates resources to the remote and local jobs in a FCFS manner without considering any site-specific objective function. In contrast to this superscheduling system, our approach differs in the following: (i) the SLA coordination in Grid-Federation is based on one-to-one SLA negotiation mechanism hence effectively limiting the network communication overhead; and (ii) we apply a Greedy backfilling approach at Grid sites for maximizing resource owner payoff function.

The work in [29] presents a superscheduling system that consists of Internet-wide Condor work pools (often referred as flock). A superscheduling manager or pool manager in the flock periodically compares metrics such as queue lengths, average pool utilization and resource availability scenario, and formulates a sorted list of pools based on these statistics. Using this list, the pool manager chooses appropriate pools for flocking. Further, a condor work pool accepts a remote job if it has free resources. The issues related to site specific resource allocation policy is not considered. In contrast: (i) we consider the site autonomy issues through Greedy Back-filling LRMS scheduling approach;

(ii) our SLA bidding approach gives resource owner more control before finalizing the SLA agreements; (iii) we consider a one-to-one SLA coordination mechanism for superscheduling, hence largely limiting the network communication overhead; and (iv) our approach incorporates an economic mechanism for superscheduling.

Tycoon [109] is a distributed market-based resource allocation system. Job scheduling and resource allocation in Tycoon is based on a decentralised isolated auction mechanism. Every resource owner in the system runs its own auction for its local resources. In contrast: (i) our superscheduling approach is based on decentralised commodity markets; and (ii) we consider a Greedy Back-filling resource allocation heuristic for LRMSes.

The work in [179] proposes SLA based cluster resource allocation. The SLA acts as a contract between the end-user and the cluster provider whereby the provider pays the penalty amount if the negotiated SLA is not satisfied. In contrast: (i) our work is targeted for computational grids where different site with different resource management policies collaborate together; (ii) we are interested in quantifying the affect of SLA negotiation intervals on end-users and providers objective function; and (iii) our SLA parameter includes the users' deadline, budget and timeout (the total amount of time superscheduler is willing to wait before SLA agreement is reached). In this, work we assume once SLA agreement is reached it will be satisfied. We do not consider any compensation or penalty model.

5.5 Conclusion

In this chapter, we presented an SLA-based superscheduling approach based on the Contract Net Protocol. The proposed approach models a set of resource providers as a contract net while job superschedulers work as managers, responsible for negotiating SLA contracts and job superscheduling in the net. Superschedulers bid for SLA contracts in the net with a focus on completing the job within the user specified deadline. We analyzed how the varying degree of SLA bidding time (i.e. admission control decision making time for LRMSes) affects the resource providers' payoff function. The results show that the proposed approach gives resource owners finer control over resource allocation decisions. However, the results also indicate that the proposed approach has a degrading effect on

the user's QoS satisfaction. However, we need to do more research on abstracting the user's QoS requirement. We need to analyse how the deadline type for the user jobs can be abstracted into different types such as into urgent and relaxed deadline. In these cases, jobs with an urgent requirement can be given a preference while finalizing SLA contracts hence providing improved QoS satisfaction to users. We analysed how varying the bid time for SLA contracts affects the system scalability and performance in terms of total message complexity. In general, the proposed superscheduling heuristic does not incur excessive messages on a per job basis as compared to the FCFS case.

Chapter 6

Decentralised Resource Discovery Service for Federated Grids

This chapter presents a decentralised Grid resource discovery system based on a spatial publish/subscribe index. It utilises a Distributed Hash Table (DHT) routing substrate for delegation of d-dimensional service messages. Our approach has been validated using a simulated publish/subscribe index that assigns regions of a d-dimensional resource attribute space to the grid peers in the system. We generated the resource attribute distribution using the configurations obtained from the Top 500 Supercomputer list. The simulation study takes into account various parameters such as resource query rate, index load distribution, number of index messages generated, overlay routing hops and system size.

6.1 Introduction

Recently, Internet-scale services including distributed resource brokering [75], distributed gaming, content distribution networks, P2P storage, and distributed auctions have received significant research interest both from researchers and industry. Concurrently, resource sharing platform such as grids [71] and PlanetLab [47] have emerged as the defacto means for hosting these distributed services. One of the main challenges involving a planetary scale deployment of these services is locating the appropriate set of nodes that match the

service's requirement.

An efficient resource discovery mechanism is a mandatory requirement of Grid systems (such as desktop grids, resource brokers, work-flow engines), as it aids in resource management and scheduling of applications. Traditionally, Grid resource brokering services such as Nimrod-G, Condor-G, Tycoon, Grid workflow engine, Gridbus Broker used services of centralised and hierarchical information services (such as R-GMA [183], Hawkeye [182], MDS-2,3,4 [67]). The limitations of these existing approaches have already been discussed in Chapter 1 and Chapter 3.

Last few years have seen the rapid growth in the e-Science applications and design of custom schedulers such as workflow engines for successfully harnessing the computational Grid resources. In order to tackle this growth, we need to design scalable infrastructure solutions. We envisage decentralisation of grids [69, 91, 164] as a viable way to realise an efficient Grid computing infrastructure. Decentralisation can be accomplished through an Internet-wide Grid resource look-up system along the same lines as the Domain Name Service (DNS). In other words, there is a need to build a scalable Grid resource information service that will allow and promote all existing Grid resources to combine together into a single cooperative system. Such a system would solve the problems associated with centralised or hierarchical organisation, resource fragmentation and conflicting application schedules. Fig. 6.1 shows such a Grid computing environment organisation based on a decentralised resource discovery system.

One of the possible ways to overcome the limitation of a centralised or hierarchical approach, is to partition the resource index space across the set of dedicated database servers [128]. For achieving fault-tolerance these database servers can be replicated across multiple machines. Further, the index space can be partitioned across servers based on attribute types and values. However one of the major drawbacks of this scheme is that satisfying a range query would require sending simultaneous messages to set of servers. This might prove costly in terms of the number of messages generated in the system. Further, if the number of users increase rapidly then upgrading the hardware infrastructure can prove to be an expensive process.

An other possible way to tackle this problem is to organise the index space using a Distributed Hash Table (DHT) method [161]. In this case, commodity machines such as

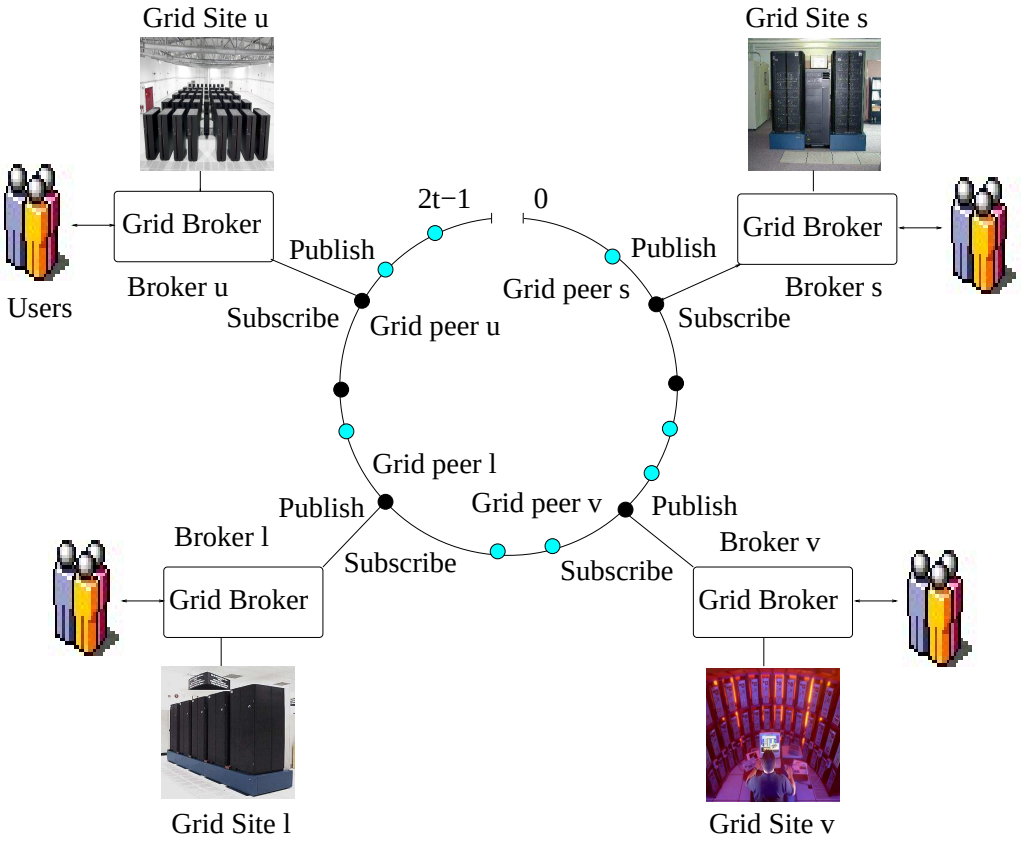


Figure 6.1: Grid brokers and Grid sites with their Grid peer service and some of the hashings to the Chord ring.

desktops can be used to host the DHT and indexing services. DHTs are inherently self-organising, fault-tolerant and scalable. Further, DHT services are light-weight and hence, do not warrant an expensive hardware infrastructure. A majority of Google's data center services are hosted by the commodity machines, and this is a case in point.

In this work, we present a decentralised Grid resource discovery service building on the DHT-based spatial publish/subscribe index reported in [108]. The publish/subscribe [62] way of communication adheres well to the needs of Grid computing. The completely decoupled nature of publish/subscribe communication adapts well to Grid participants who are dynamic and are separated in time and space. In general, a publish/subscribe system conveys published information from any provider to all interested information consumers who have previously subscribed for the same. In this setting the publisher or the subscriber do not use source/destination identifiers/addresses. Further, the spatial [147] nature of the publish/subscribe index has the capability to respond to complex Grid re-

source queries (refer to Chapter 3) such as range queries involving various attribute types including those that have a spatial component.

Table 6.1: Summary of the complexity of structured P2P systems.

DHT	**Routing table size**	**Routing complexity**	**join/leave overhead**
Chord	$O(\log n)$	$O(\log n)$	$O((\log n)^2)$
Pastry	$O(\log_b n)$	$O(b\ \log_b n + b)$	$O(\log n)$
CAN	$O(2\ d)$	$O(d\ n^{1/d})$	$O(2\ d)$
Tapestry	$O(\log_b n)$	$O(b\ \log_b n + b)$	$O(\log n)$

The proposed Grid resource discovery service organises data by maintaining a logical d-dimensional publish/subscribe index over a network of distributed Grid brokers/Grid sites. These brokers create a Chord overlay [161], which collectively maintain the logical publish/subscribe index to facilitate a decentralised resource discovery process. Note that, basically any DHT could be utilised for routing of d-dimensional index. Depending on the DHT (such as Pastry [143], CAN [140]) the complexity for routing table size, look-up, and peer join/leave would be different (refer to Table 6.1). But basically they can all support the proposed resource discovery service. We present more details about the publish/subscribe index in Section 6.3. Fig. 6.1 depicts the proposed resource discovery system involving Grid brokers and Grid Sites (shown as dark coloured blocks on the Chord ring). Resource brokering services such as a GFA, Condor-G etc. issue a Resource Lookup Query (RLQ) by subscribing for a publication object that matches a user's application requirement. Grid resource providers update their resource status by publishing information at periodic intervals through a Resource Update Query (RUQ).

The RLQs and RUQs are mapped as subscribe and publish objects in the system (shown as light coloured block in Fig. 6.1 on the Chord ring). Dark dots are the Grid peers that are currently part of Chord based Grid network. The index publication/subscription process is facilitated by a *Grid Peer* Service, which is a component of the broker service. The Grid Peer Service is responsible for distributed information publication, subscription and overlay management processes. More details about the Grid brokering service model that we consider can be found in Chapter 4.

The rest of this chapter is organised as follows. In Section 6.2, we present details

on the indexing requirements of Grid-Federation resource sharing model. Section 6.3 presents details about the underlying d-dimensional publish/subscribe index that we leverage for this work. In section 6.4, we summarise the average message and routing hop complexity involved with routing of RLQ/RUQ objects. Section 6.5 presents the simulation model that we utilise for evaluating the performance of Grid resource discovery system. In Section 6.6, we present various experiments and discuss our results. Section 6.7 summarises current state of the art in resource discovery system design. We end this chapter with conclusion in Section 6.8.

6.2 Grid Resource Brokering Service and Queries

In general, a GFA service requires two basic types of queries: (i) an RLQ, a query issued by a broker service to locate resources matching the user's application requirements; and (ii) an RUQ, is an update query sent to a resource discovery service by a Grid site owners about the underlying resource conditions. Since, a Grid resource is identified by more than one attribute, an RLQ or RUQ is always d-dimensional. Further, both of these queries can specify different kinds of constraints on the attribute values. If a query specifies a fixed value for each attribute then it is referred to as a *d-dimensional Point Query* (DPQ). However, in case the query specifies a range of values for attributes, then it is referred to as a *d-dimensional Window Query* (DWQ) or a *d-dimensional Range Query* (DRQ). In database literature, a DWQ or an DRQ is also referred to as a *spatial range query*.

Recall that, compute Grid resources have two types of attributes: (i) static attributes– such as the type of operating system installed, network bandwidth (both Local Area Network (LAN) and Wide Area Network (WAN) interconnection), processor speed and storage capacity (including physical and secondary memory); and (ii) dynamic attributes– such as processor utilization, physical memory utilization, free secondary memory size, current usage price and network bandwidth utilization.

Every GFA in the federation publishes its local resource information with the decentralised resource discovery system. An RUQ or a publish object consists of a resource description R_i, for a cluster i. R_i includes information about the CPU architecture, number of processors, RAM size, secondary storage size, operating system type, resource

usage cost etc. In this work, $R_i = (p_i, x_i, \mu_i, \emptyset_i, \rho_i, c_i)$ which includes the number of processors, p_i, processor architecture, x_i, their speed, μ_i, their utilization, ρ_i, installed operating system type, $\emptyset_i$, and a cost c_i for using that resource, configured by the site owner. A site owner charges c_i per unit time or per unit of million instructions (MI) executed, e.g. per 1000 MI. A GFA publishes the R_i into distributed resource discovery system by encapsulating it into an RUQ object, U_i.

A job in the Grid-Federation system is written as $J_{i,j,k}$, to represent the i-th job from the j-th user of the k-th resource. A job specification consists of the number of processors required, $p_{i,j,k}$, processor architecture, $x_{i,j,k}$, the job length, $l_{i,j,k}$ (in terms of instructions), the budget, $b_{i,j,k}$, the deadline or maximum delay, $d_{i,j,k}$, and operating system required, $\emptyset_{i,jk}$. A GFA aggregates these job characteristics including $p_{i,j,k}$, $x_{i,j,k}$, $\emptyset_{i,j,k}$ with a constraint on maximum speed, cost and resource utilization into an RLQ object, $r_{i,j,k}$ and sends it as a subscription object to the resource discovery system. More details about the job model can be found in Chapter 4.

6.2.1 An Example RUQ and RLQ

Every GFA periodically sends an RUQ to the distributed resource discovery system. The publish, or resource update object includes a resource description set R_i:

Publish: Total-Processors= 100 && Processor-Arch="pentium" && Processor-Speed= 2 GHz && Operating-System = Linux && Utilization= 80 && Access-Cost=1 Dollar/min.

Note that, the above RUQ is a DPQ. However, an RUQ can also be compiled as a DRQ depending on a Grid site configuration. As jobs arrive the GFAs (on behalf of the Grid-Federation users) issue an RLQ to the distributed resource discovery system to acquire information about active resource providers in the system. An RLQ has the following semantics:

Subscribe: Total-Processors $\geq$ *70 && Processor-Arch="pentium" && 2 GHz* $\leq$

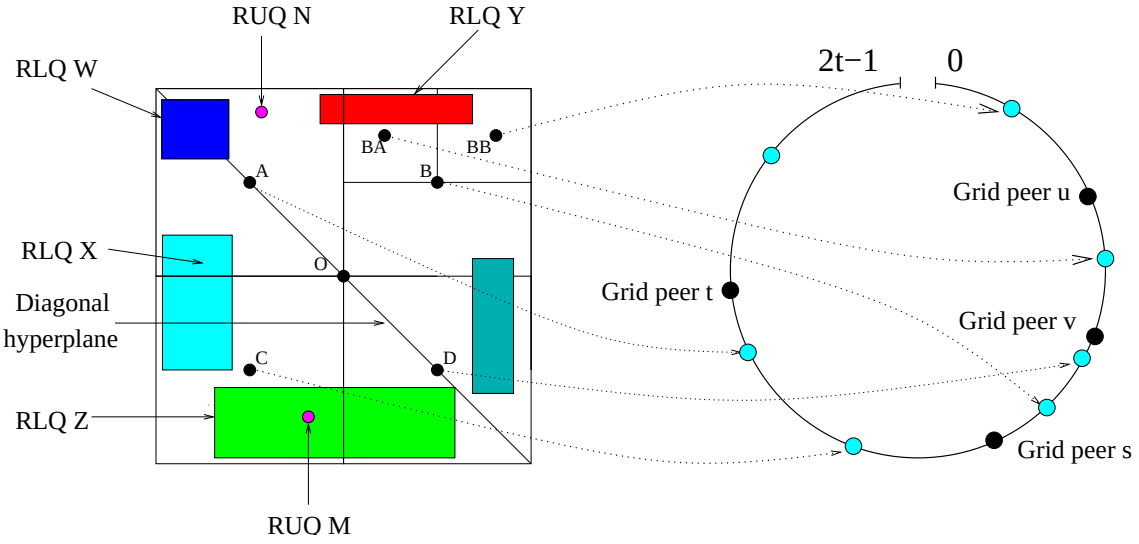


Figure 6.2: Spatial RLQs $\{W,X,Y,Z\}$, cell control points, point RUQs $\{M\}$ and some of the hashings to the Chord.

Processor-Speed $\leq$ *5GHz && Operating-System = Solaris && 0.0* $\leq$ *Utilization* $\leq$ *90 && 0 Dollar/min* $\leq$ *Access-Cost* $\leq$ *5 Dollar/min .*

6.2.2 Handling Dynamic Resource Information

The proposed resource discovery service handles dynamic information such as the number of available processors, memory utilisation etc. It allows GFA to tag these dynamic resource information in the RUQ objects. Next, the RUQ messages containing the objects are periodically sent in the decentralised DHT space. In this way, the dynamism of the resources are truly reflected to the decentralised look-up space. This methodology enables the dynamic and scalable resource discovery and selection in a distributed Grid resource sharing environment.

6.3 P2P-Based Spatial Publish/Subscribe Index

In this section, we describe the features of the P2P-based publish/subscribe index that we utilise for our Grid resource discovery system.

There are many different kinds of spatial indices such as Space Filling Curves (SFCs)

(including Hilbert curves [150], Z-cruves [80]), k-d tree [80], MX-CIF Quad-tree [167], and R*-tree [114] that could be utilised for organising a P2P publish/subscribe based Grid resource discovery system. SFC based indices including Hilbert curves and Z-curves have issues with routing load-balance in case of skewed index distribution. However, as authors point out SFC index load can be balanced through external techniques. In case of Hilbert curves dynamic techniques such as node virtualisation, load-partitioning with neighbor peers etc are utilised for this purpose. The authors in the work utilising Z-curves have also proposed an external load-balancing technique. In the same work they introduce a P2P version of a k-d tree. This approach also has routing load-balance issues that need to be addressed.

In other recent work, a MX-CIF Quad tree based spatial index has been proposed. The authors argue that their approach does not require explicit load-balancing algorithms in contrast to others. The P2P based R*-tree index in [114] uses CAN as the routing space. The index space is partitioned among super peers and passive peers. The bulk of query load is handled by the super peers in the network similar to the Gnutella [41] system. To summarise, there are different trade offs involved with each of the spatial indices, but basically they can all support scalability and Grid resource indexing functionality.

In this work, we utilise the spatial publish/subscribe index proposed in the work [108]. The publish/subscribe index uses a logical d-dimensional domain space for mapping subscription and publication objects. The MX-CIF Quad-tree spatial hashing technique [147] is used to hash the logical d-dimensional index onto a DHT network.

The index that organises the publish/subscribe events/objects is similar to the one proposed in the work [167], with the only difference being recursive subdivision of space does not follow the regular MX-CIF Quad-tree approach beyond the f_{min} level. Instead it is based on the relevant publish/subscribe load on the index cells. Further, no external load-balancing technique is required to balance the index routing load among the Grid peers. The message routing process uses a key-based routing (KBR) protocol, such as Chord, that supports the delegation of d-dimensional service messages. We have chosen this publish/subscribe index for simplicity, and our approach would work with other spatial indices but the analysis for message complexity, routing hops, index latency and finer points of load-balancing would be different.

The publish/subscribe index utilises a content-based approach. It builds a d-dimensional Cartesian space based on the Grid resource attributes, where each attribute represents a single dimension. The logical d-dimensional index assigns regions of space to the Grid peers in the resource discovery system. If a Grid peer is assigned a region (cell) in the d-dimensional space, then it is responsible for handling all the activities related to the RLQs and RUQs associated with the region. More details on this spatial hashing technique can be found in the article [166].

Each cell is uniquely identified by its centroid, termed as the *control point*. Fig. 6.2 depicts some control points and some example hashings using the Chord method. The d-dimensional coordinate values of a cell's control point is used as the key and hashed onto the Chord. Dark dots are the Grid peers that are currently part of the network. Light dots are the control points hashed on the Chord. For this figure, f_{min} = 1, dim=2. RLQ/RUQ objects are inserted into the distributed structure by mapping them to index cells and hashing the control points of these cells on to the Chord. In this example, the control point C is hashed to the Grid peer t and the RLQ object X is stored with that control point. The Cartesian space has a tree structure due to two types of division process, explained as follows:

6.3.1 Minimum Division (f_{min})

This process divides the Cartesian space into multiple index cells when the d-dimensional publish/subscribe index is first created. The cells resulted from this process remain constant throughout the life of the publish/subscribe domain and serve as entry points for subsequent RLQ (subscribe) and RUQ (publish) processes. The number of cells produced at the minimum division level is always equal to $(f_{min})^{dim}$, where dim is dimensionality of the Cartesian space. Every Grid peer in the network has basic information about the Cartesian space coordinate values, dimensions and minimum division level.

6.3.2 Load Division

This process is performed by the cells (at f_{min}) when their storage capacities are undermined by heavy RLQ workload. An overloaded cell subdivides itself to produce multiple

child cells, which collectively undertake the workload. This is a dynamic process that is repeated by the child cells, if they also become overloaded. This growing process introduces the parent-child relationship, where a cell at level *m* is always a child of a particular cell at level *m-1*. To minimise the amount of information that needs to be known by the cells for correct routing, the parent-child relationship is limited at one level. It means that every cell only has a direct relationship with its child cells. Note that, the maximum depth (f_{max}) of the distributed index tree is curbed by constraining the load division process after a certain number of executions. Although such a constraint provides controllable performance benefits, it may lead to query load-imbalance in some cases.

6.3.3 Query Mapping.

This action involves the identification of the cells in the Cartesian space to map an RLQ or RUQ. For mapping RLQs, the search strategy depends whether it is a DPQ or DRQ. For a DPQ type RLQ, the mapping is straight forward since every point is mapped to only one cell in the Cartesian space. For a DRQ type RLQ, mapping is not always singular because a range look-up can cross more than one cell. To avoid mapping a range RLQ to all the cells that it crosses (which can create many unnecessary duplicates) a mapping strategy based on diagonal hyperplane of the Cartesian space is utilised. This mapping involves feeding an RLQ candidate index cells as inputs into a mapping function, F_{map}. This function returns the IDs of index cells to which given RLQ should be mapped (refer to Fig. 6.3). Spatial hashing is performed on these IDs (which returns keys for Chord space) to identify the current Grid peers responsible for managing the given keys. A Grid peer service uses the index cell(s) currently assigned to it and a set of known base index cells obtained at the initialisation as the candidate index cells.

Similarly, the RUQ/publish process also involves the identification of the cell in the Cartesian space using the same algorithm. An RUQ is always associated with an event region and all cells that fall fully or partially within the event region will be selected to receive the corresponding RUQ. The calculation of an event region is also based upon the diagonal hyperplane of the Cartesian space.

```
1 begin
2     index cell(s) = F_map(candidate index cells)
3     if (index cell is not null) then
4         ID = spatial_hash(index cell)
5         Lookup Grid peer through Chord routing network based on ID,
6         to either store the subscription or match the publication to stored subscriptions.
7     end
8 end
```

Figure 6.3: Subscribing or publishing.

6.3.4 Query Routing.

Using the query mapping policies, the resource discovery service searches for a cell (from minimum division) in the Cartesian space that overlaps with area sought by an RLQ. When this cell is found, the service starts the RLQ mapping process by contacting the peer (in the network) that owns the cell. When the cell receives an RLQ, two cases are considered:

- In the first case, the cell has undergone a load division process and it routes the RLQ to the child cell that is responsible for the region in which the RLQ is mapped.
- In the second case, the cell has not undergone any load division process. Hence, there will be no further routing and the cell keeps the RLQ for future event notification.

6.4 Message Complexity and Routing Hop Analysis

In this section, the complexity analysis for message and routing hop is presented. We denote the number of messages generated in mapping a DRQ by a random variable M. The distribution of M is the function of the problem parameters including query size, dimensionality of search space, query rate, division threshold and data distribution. As the dimensionality increases, the order of the tree increases and each tree node has more children. If the height of the tree is kept constant, then increasing the Cartesian space dimensions does not increase the maximum hop length. However, constraining the maximum height of the tree, may lead to load imbalance at some Grid peers. Note that, the derivation presented in this chapter assumes that the Chord method is used for delegation

of service messages in the network.

Essentially, a control point at the f_{min} level of the logical d-dimensional Cartesian space can be reached in $O(\log\ n)$ routing hops with high probability (using the Chord method). Since each Grid peer at f_{min} level of the index tree controls its division with the child cells, therefore every control point owner can maintain a cache of IP address for its child cells. The child cells are created as a result of dynamic load division process. Hence, the number of routing hops required to delegate an index message beyond the f_{min} reduces to $O(1)$. However, under high churn conditions when the Grid peer membership changes, the Chord stabilisation process and transfer of index keys delays the caching of IPs. During such periods the cache miss can occur and in this case the routing may have to be done using the standard Chord method. Since, we consider Grid sites to be well provisioned and well connected to the Internet. Therefore, we do not expect a highly dynamic behaviour (high join, leave, and failure rate) in contrast to the traditional P2P file sharing systems.

Based on above discussion, in order to compute the worst case message lookup and routing complexity one additional random variable T is considered. T denotes the number of disjoint query path undertaken in mapping an RLQ or RUQ. In the worst case, every disjoint query ends up at the maximum allowed depth of the tree i.e. f_{max}. Hence every disjoint path would undertake $\Theta(log\ n + f_{max} - f_{min})$ routing hops with high probability. Hence, the expected value of M is given by:

$$E[M] = \Theta(E[T] \times (\log\ n + f_{max} - f_{min}))$$

6.5 Simulation Model

In this section, we present simulation model for evaluating the performance of our resource discovery system. The proposed model is applicable to large networks of the scale of the Internet. The simulation model considers the message queuing and processing delays at the intermediate peers in the network. In a centralised system, the index look-up latency is essentially zero, assuming the computation delay due to processing of local indices is negligible. For the P2P system, assuming negligible computation delay for index

processing logic at intermediate peers, the time to complete an RLQ or RUQ is time for the query to reach all the cells (including both parent and child cells) that intersect with the query region.

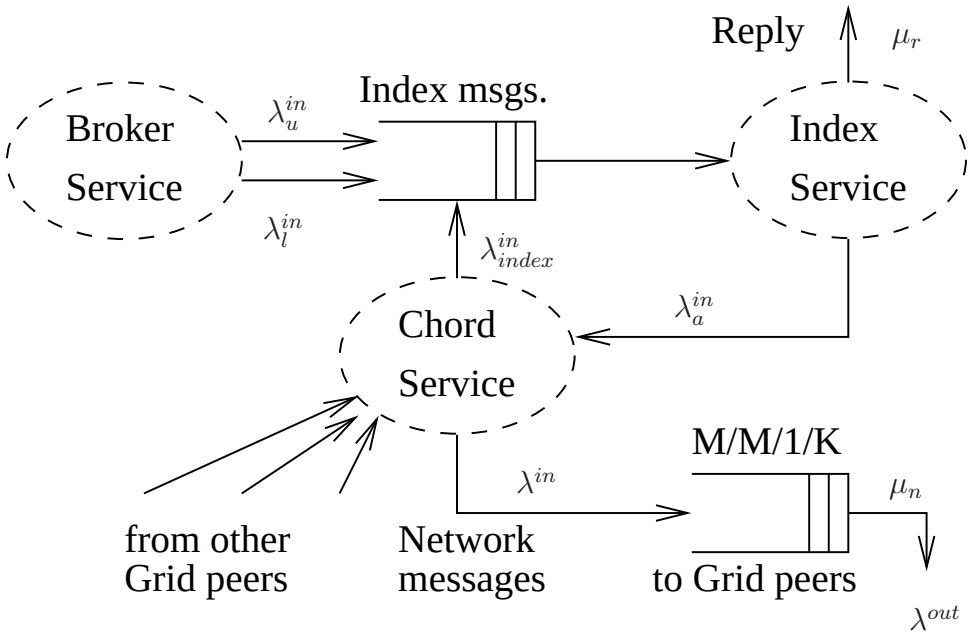


Figure 6.4: Network message queueing model at a Grid peer i.

In our message queueing model, a Grid peer node (through its Chord routing service) is connected to an outgoing message queue and an incoming link from the Internet (as shown in Fig. 6.4). The network messages delivered through the incoming link (effectively coming from other Grid peers in the overlay) are processed as soon as they arrive. Further, the Chord routing service receives messages from the local publish/subscribe index service. Similarly, these messages are processed as soon as they arrive at the Chord routing service. After processing, Chord routing service queues the message in the local outgoing queue. Basically, this queue models the network latencies that a message encounters as it is transferred from one Chord routing service to another on the overlay. Once a message leaves an outgoing queue it is directly delivered to a Chord routing service through the incoming link. The distributions for the delays (including queueing and processing) encountered in an outgoing queue are given by the M/M/1/K [7] queue steady state probabilities.

Our simulation model considers an interconnection network of n Grid peers whose

overlay topology can be considered as a graph in which each peer maintains connection to a $O(\log\ n)$ other Grid peers (i.e. the Chord overlay graph). As shown in Fig. 6.4, every Grid peer is connected to a broker service that initiates lookup and update queries on behalf of the users and site owner. We denote the rates for RLQ and RUQ by λ_l^{in} and λ_u^{in} respectively. The queries are directly sent to the local index service which first processes them and then forwards them to the local Chord routing service. Although, we consider a message queue for the index service but we do not take into account the queuing and processing delays as it is in microseconds. Index service also receives messages from the Chord routing service at a rate λ_{index}^{in}. The index messages include the RLQs and RUQs that map to the control area currently owned by the Grid peer, and the notification messages arriving from the the network.

6.6 Performance Evaluation

In this section, we perform simulations to capture the interplay among various Grid resource query and P2P network parameters and their contribution to the overall performance of Grid resource discovery system.

6.6.1 Experiment Setup

We start by describing the test environment setup.

Broker Network Simulation:

Our simulation infrastructure is modeled by combining two discrete event simulators namely *GridSim* [32], and *PlanetSim* [82]. GridSim offers a concrete base framework for simulation of different kinds of heterogeneous resources, services and application types. The core of GridSim is based on the *SimJava* [94], a discrete event simulation package.

PlanetSim is an event-based overlay network simulator. It can simulate both unstructured and structured overlays. However, in this work we utilise the services of the Chord implementation of the PlanetSim. To enable event time synchronisation between PlanetSim and GridSim, we modified the basic PlanetSim classes including *Node, Network and*

EndPoint to extend the core *GridSim* class. We model the resource brokering service (i.e. a GFA inside the GridSim) that initiates RLQs and RUQs on behalf of users and resource providers. Every GFA connects to a local publish/subscribe index service which runs on a Chord node in the PlanetSim. Every instance of the index service in the network is responsible for managing and indexing a region in the logical d-dimensional space. Our simulation considers message queueing delay, processing delay, and packet loss at the intermediate overlay Chord nodes.

Simulation Configuration

This section explains the distributions for simulation parameters.

Network configuration: The experiments were conducted using a 32 bit Chord overlay i.e. 32 bit node and key ids. The network size, n, was fixed at 128 broker nodes/Grid sites for experiment 1. In experiment 2, the system size is scaled from 100 to 500 in steps of 100. The network queue message processing rate, μ_n, at a Grid peer was fixed at 500 messages per second. We vary the value for network message queue size, K, as 10^2, 10^3, and 10^4 in experiment 1. While in experiment 2, we fixed K to 10^4. In experiment 2 we basically simulate a large message queue size such that no message is dropped by the resource discovery system.

Query rate configuration: We vary the RLQ rate, λ_l^{in}, and RUQ rate, λ_u^{in}, from 1 to 100 queries per simulation second. At every step the RLQ rate is always equal to the RUQ rate. In experiment 2, the RLQ and RUQ rate are fixed at 1 query per second for different system sizes.

Publish/subscribe index configuration: The minimum division, f_{min} of the logical d-dimensional publish/subscribe index was set to 3, while the maximum height of the index tree, f_{max}, was also limited to 3. This means we basically do not allow the partitioning of index space beyond the f_{max} level. In this case, a cell at a minimum division level does not undergo any further division. Hence, no RLQ/RUQ object is stored beyond the f_{max} level. The index space resembles a Grid-like structure where each index cell is

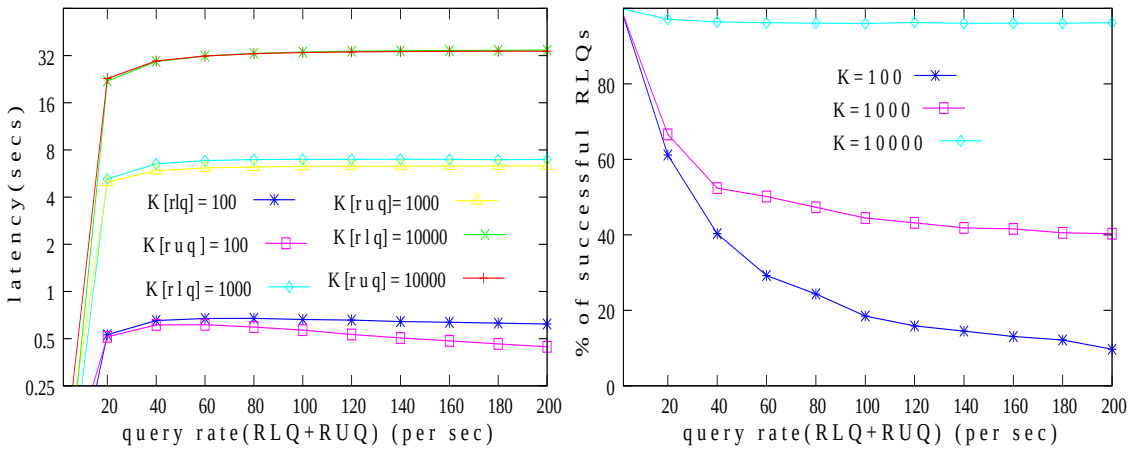
randomly hashed to a Grid peer based on its control point value. The publish/subscribe Cartesian space had 6 dimensions including number of processors, p_i, resource access cost, c_i, processor speed, m_i, processor utilisation, ρ_i, processor architecture, x_i, and operating system type, ϕ_i. Hence, this configuration resulted to 729 (3^6) Grid index cells at the f_{min} level. On an average, 7 index cells are hashed to a Grid peer in a network comprising of 128 Grid sites.

Indexed data distribution: We generated an uniform resource type distribution using the resource configuration obtained from the Top 500 Supercomputer list[1]. The list included 22 distinct processor types, so in our simulated Grid resource index space, the probability of occurrence of a particular processor type is $1/22$. We utilised the resource attributes including processor architecture, its number, its speed, and installed operating system from the Supercomputer list. The values for c_i and ρ_i were fabricated. The values for c_i and ρ_i were uniformly distributed over the interval $[0, 10]$ and $[5, 80]$ respectively. Every RLQ was constrained such that it always subscribed for the operating system type, processor architecture, maximum number of processors required which was also available on the local site. An RLQ is thrashed from the system, once it matches with an RUQ. Following this, a match event notification is sent to the concerned broker service. A load of 200 RLQ and 200 RUQ objects is injected into the resource discovery system by a broker service over the simulation period during experiment 1. While in case of experiment 2 we configured a broker service to inject only 50 RLQ and 50 RUQ objects.

6.6.2 Effect of Query Rate

The first set of experiments measured the RLQ/RUQ query performance with an increasing incoming query rate across the Grid peers in the broker network. We started from a RLQ/RUQ rate of 1 query per second and increased it till 100 queries per second. We configured the other input parameters as following n=128, f_{min}=3, $f_{max} = 3$, μ_n=500, and dim=6. All the broker nodes join the system at the same time, stabilise their finger tables and initialize their logical index space. Over the simulation period, we do not consider a Grid peer join or leave activity. We identified six metrics to measure the RLQ/RUQ query

[1]Top 500 Supercomputer List, http://www.top500.org/



(a) query rate (RLQ + RUQ) (per sec) vs. lookup latency (secs)

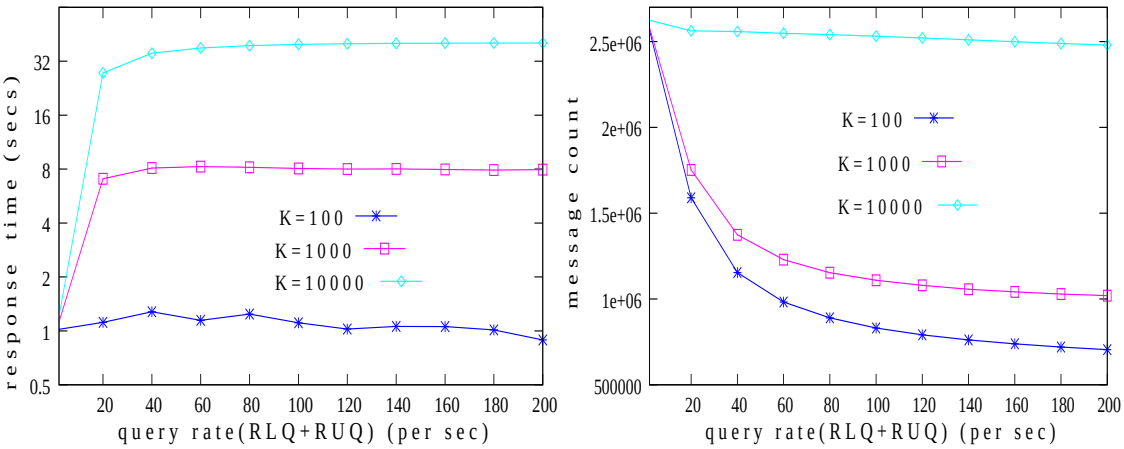
(b) query rate (RLQ + RUQ) (per sec) vs % of successful RLQs

Figure 6.5: Simulation: Effect of query rate.

performance including latency, % of successful RLQs, response time, routing hops, total number of messages generated for mapping RLQs/RUQs, and the total number of messages in the system over the simulation period. Measurements for parameters including latency, response time, routing hops is averaged over all the broker services in the system. While the measurements for the remaining parameters are computed by summing up their values across the broker services.

Fig. 6.5 and Fig. 6.6 show the plots for these parameters with an increasing query rate across the system. Fig. 6.5(a) shows results for the average RLQs/RUQs latency, Fig. 6.5(b) shows results for the % of successful RLQs and Fig. 6.6(a) shows the average response time for the RLQs across the system. These measurements were conducted for different values of network message buffer capacity i.e. K. Results show that for lower values of K (i.e. 10^2, 10^3) network drops significant number of messages (refer to Fig. 6.6(b)). Fig. 6.6(b) shows total number of messages generated in the system over the simulation period for different query rates and message queue sizes. Hence, for these message queue sizes successful RLQs/RUQs encounter comparatively lower traffic hence leading to almost same latency (refer to Fig. 6.5(a)) and response time (refer to Fig. 6.6(a)). But the downside of this is that at higher rates significantly larger number RLQs are dropped by the system (refer to Fig. 6.5(b)). However, this is not true for the case when the network has higher buffering capability (i.e. $K = 10^4$), in this case the

messages encounter significantly more traffic thus worsening the queuing and processing delays. Second, with a larger message queue size system experiences much higher query success rates (refer to Fig. 6.5(b)).



(a) query rate (RLQ + RUQ) (per sec) vs response time (secs)

(b) query rate (RLQ + RUQ) (per sec) vs message count

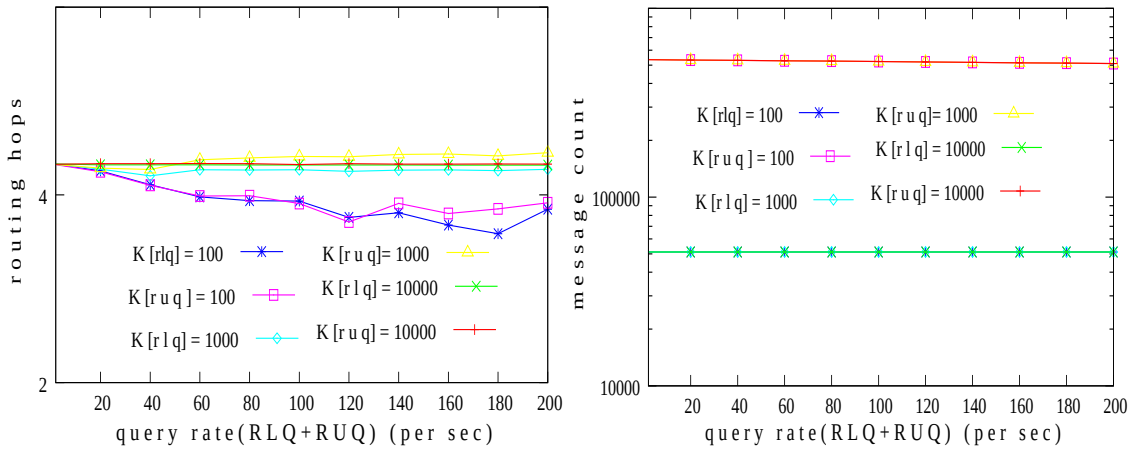
Figure 6.6: Simulation: Effect of query rate.

Our results state that the average number of routing hops for RLQs and RUQs remain constant irrespective of the query rate in the system (refer to Fig. 6.7(a)). The main reason for this is, having the same value for both the index tree depth parameters i.e. f_{min}=f_{max}. Thus, we do not allow the partition of index cell or load distribution between Grid peers beyond f_{min}. With different query rates the height of the distributed index tree remained constant, hence leading to a similar number of routing hops. Fig. 6.7(b) shows the results for the total number of messages generated in the system for all RLQs/RUQs. As expected the number of messages generated for the RLQs/RUQs remained constant, since the data distribution was same for all query rates.

Thus, it is evident that at higher query rates, the messages experience greater queuing and processing delays. This can be directly observed in the RLQ/RUQ latencies which have significantly larger values at moderately higher query rates.

6.6.3 Effect of System Size

In our second experiment, we examine the resource discovery system's scalability in terms of the number of participating Grid sites. We used the same resource distribution as

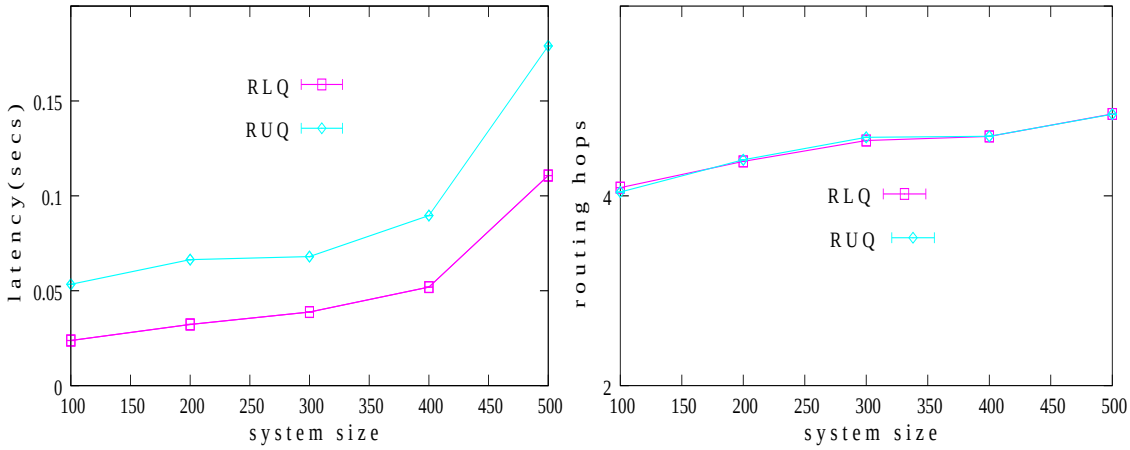


(a) query rate (RLQ + RUQ) (per sec) vs. routing hops

(b) query rate (RLQ + RUQ) (per sec) vs. message count

Figure 6.7: Simulation: Effect of query rate.

before, but scaled it such that the probability of occurrences of particular resource types remained constant. We started from a system size of 100 and increased it till 500. We fixed the RLQ/RUQ rate to 1 query per second, across the Grid peers in the broker network. We configured the other input parameters as following f_{min}=3, $f_{max} = 3$, μ_n=500, K=10^4 and dim=6. All the Grid peers join the system at the same time, stabilise their finger tables and initialize their logical index space. Over the simulation period, we do not consider a Grid peer join or leave activity.



(a) system size vs. lookup latency (secs)

(b) system size vs. routing hops

Figure 6.8: Simulation: Effect of system size.

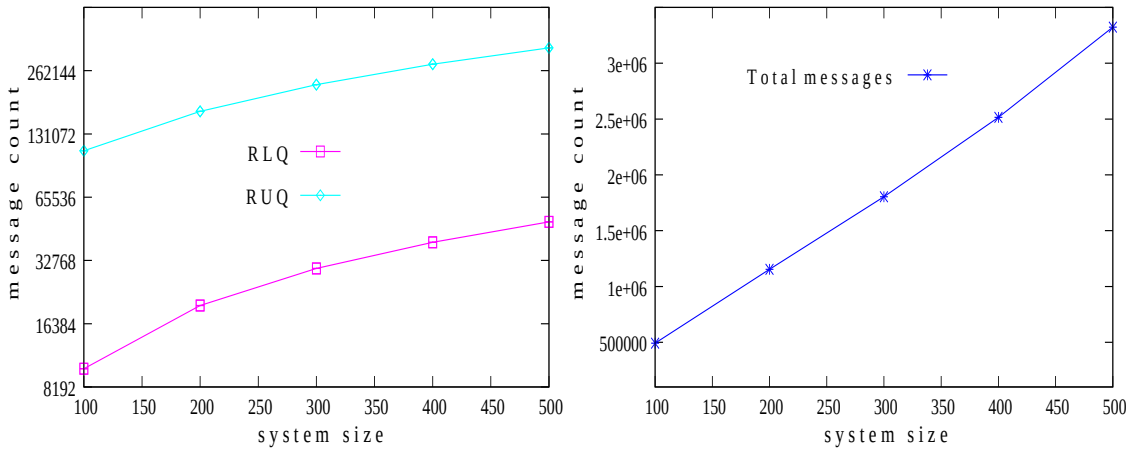
Fig. 6.8(a) shows the growth of the RLQ/RUQ latency as a function of increasing Grid

network size. As expected, the query latencies do not increase significantly, because the growth rate of latency is a logarithmic function of the Grid network size n. That is on average an RLQ or RUQ encounters $O(\log\ n)$ Grid peers before being finally mapped. Similarly, in Fig. 6.8(b) we observed that the number of routing hops undertaken RLQ/RUQ increased marginally with the system size. At the system size of 100, the RLQs/RUQs undertook 4.12 routing hops on an average. For a system size of 500, the average query path increased to 5.39 hops i.e. increased by about 30%.

Fig. 6.9(a) shows the results for the number of messages generated for RLQs/RUQs and Fig. 6.9(b) shows the results for the total number of messages generated as the system scaled from 100 to 500 sites. As expected the number of messages generated for RLQs/RUQs increased with system size. A system compromising of 100 Grid sites produced 109007 RUQ messages, which increased to 336579.4 messages when the system scaled to 500 Grid sites (refer to Fig.6.9(a)). We observed a similar growth for RLQ messages as well with an increase in the system size. The total messages generated (including RLQ and RUQ) increased significantly as the system scaled from 100 to 500 sites (refer to Fig. 6.9(b)). Further, in this case we observed 575% increase in the total number of messages generated in the system. The main reason for this being, as the broker network size increases the total number of messages generated in the system grows as $\Theta(n\ log\ n)$. In other words, the number of messages generated is the function of number of brokers, number of RLQ/RUQs sent and number of routing hops undertaken to map the queries. Hence, in this case we expect linear or close to linear growth in the total message count.

6.7 Related Work

The approach [97] involved a drawback of generating a large volume of network messages due to flooding. This system can not guarantee to find the desired resource even though it exists in the network due the Time to Live (TTL) field associated with query messages. SWORD [128] system creates a separate search segment for each attribute and hence the query routing needs to be augmented with external techniques for resolving d-dimensional queries. In contrast, our resource discovery system utilises a spatial publish/subscribe index that hashes a d-dimensional index space to a 1-dimensional key



(a) system size vs. message count (b) system size vs. message count

Figure 6.9: Simulation: Effect of system size.

space of Chord overlay. The publish/subscribe index does not require any additional query resolution and load-balancing heuristic. JXTA Search [173] does not apply any index for organising the distributed data. A cross-domain search operation in JXTA involves a query broadcast to all the advertisement groups using the query group membership information. The OurGrid system utilises JXTA for organising its brokering service. In contrast, our resource discovery is based on a deterministic routing substrate Chord. Our system does not require a broadcast primitive for data discovery in a Grid network, hence is more efficient in terms of number of messages generated in the system. Squid [150] system applies Hilbert space filling curves for mapping a d-dimensional index space to a 1-dimensional key space. Squid maps these contiguous d-dimensional indices to the overlay key space. The approach has issues with index load balance which is fixed using external technique. In contrast, our proposed resource discovery system utilises a spatial publish/subscribe index that does not need any external load-balancing.

6.8 Conclusion

In this chapter, we presented a decentralised Grid resource discovery system. It utilises a P2P spatial publish/subscribe index for organising d-dimensional Grid resource data. We analysed experimentally how the query arrival rate and Grid system size affects the system performance. We reached to the following conclusions in this chapter: (i) the

resource query rate i.e. RLQ and RUQ rate directly affects the performance of the decentralised resource discovery system. At higher rates, Grid resource queries can experience considerable latencies; and (ii) contrary to what one may expect, the Grid system size does not have a significant impact on the performance of the resource discovery system, in particular the query latency and the number of message routing hops. Encouraged, by the results obtained in this chapter, in the next chapter we propose a P2P tuple space model that builds on the resource discovery system. The resulting tuple space is utilised for enhanced coordination among GFAs and system-wide load-balancing.

Chapter 7

Peer-to-Peer Tuple Space based Coordinated Resource Provisioning

This chapter proposes a novel approach to facilitate coordination among distributed application schedulers in a wide-area resource leasing environment such as grids and PlanetLab. The resource types in these environments include computational resources (such as supercomputer, clusters, desktops) that offer processing power, storage resources, sensors and network links. The resources are: (i) highly dynamic in behaviour, where their status can change in a small time period, (ii) controlled and administered by different domains, and (iii) topologically separated over the Internet. The fundamental goal of our work is to develop decentralised coordination among users (in case of the PlanetLab) and among resource brokers (in case of the grids) to curb the over-provisioning of resources that leads to degraded resource performance and user QoS satisfaction.

7.1 Introduction

Several research projects including Bellagio, Tycoon, NASA-Scheduler, OurGrid, Sharp, Condor-Flock and Grid-Federation (refer to Chapter 4) have proposed federated sharing of topologically distributed networked computing resources to facilitate a cooperative and coordinated sharing environment. In a federated resource sharing environment, every participant gets access to a larger pool of resources, and resource providers get economic or

bartering benefits depending upon the resource leasing policy. Distributed resource sharing systems including Bellagio and Tycoon have been deployed and tested over PlanetLab environment, while the Grid-Federation, NASA-Scheduler, Condor-Flock and OurGrid are targeted towards computational grid environments.

However, the effectiveness of federated resource sharing environments can not be optimally achieved without a proper coordination mechanism between the schedulers; resource brokers in case of grids and slice initiators in case of PlanetLab. The coordination mechanisms in NASA-scheduler, OurGrid, and Condor-Flock P2P are based on general broadcast and limited broadcast communication mechanisms respectively. Hence, these approaches have the following limitations: (i) high network overhead; and (ii) scalability problems. Resource allocation coordination in Tycoon is based on a decentralised, isolated auction mechanism. Every resource owner in the system runs its own auction on behalf of their local resources. In this case, a scheduler might end-up bidding across a large number of auctions. On the other hand, resource allocation in Bellagio system is based on the bid-based proportional resource sharing model. Bids for resources are periodically cleared by a centralized auction coordinator. Clearly, the coordination mechanisms followed by Bellagio and Tycoon are neither efficient nor scalable. The Sharp architecture coordinates resource allocation among various competing schedulers through pair-wise peering arrangement. For example, site A may grant to site B a claim on its local resources in exchange for a claim that enables access to B resources. This pair-wise approach may work well for a small system size, but can prove to be serious bottleneck as the system scales out.

One of the possible ways to solve this problem is to host a coordinator service on a centralised machine [83, 120, 169]. Every application scheduler is required to submit his demands to the coordinator (similar to the Bellagio system). Similarly, resource providers update their resource usage status periodically with the coordinator. The centralised resource allocation coordinator performs system wide load-distribution primarily driven by resource demand and availability. However, this approach has several design limitations including: (i) single point of failure; (ii) lacks scalability; (iii) high network communication cost at links leading to the coordinator (i.e. network bottleneck, congestion); and (iv) computational power required to serve a large number of participants.

Another possible way to tackle this problem is to distribute the role of the centralised coordinator among a set of machines based on a P2P network model. New generation P2P routing substrates such as DHTs [143, 161] can be utilised for efficiently managing such decentralised coordination network. DHTs have been proven to be self-organising, fault-tolerant and scalable.

We advocate organising Grid schedulers (and users in case of PlanetLab) and Grid resources based on a DHT overlay. Application schedulers post their resource demands by injecting a *Resource Claim* object into the decentralised coordination space, while resource providers update the resource supply by injecting a *Resource Ticket* object (similar terminologies have been used by the Sharp system). These objects are mapped to the DHT-based coordination services using a spatial hashing technique. The details on spatial hashing technique and object composition are discussed in Section 7.3. A decentralised coordination space is managed by a software service (a component of the Grid peer service) known as a coordination service. It undertakes activities related to decentralised load-distribution, coordination space management etc.

A coordination service on a DHT overlay is made responsible for matching the published resource tickets to subscribed resource claims such that the resource ticket issuers are not overloaded. Resource tickets and resource claims are mapped to the coordination space based on distributed spatial hashing technique. Every coordination service in the system owns a part of the coordination space governed by the overlay's hashing function (such as SHA-1). In this way, the responsibility of load-distribution and coordination is delegated to a set machines instead of delegating it to one. The actual number of machines and their respective coordination load is governed by the spatial index's load-balancing capability. Note that, both resource claim and resource ticket objects have their extent in d-dimensional space.

1-dimensional hashing provided by current implementation of DHTs are insufficient to manage complex objects such as resource tickets and claims. DHTs generally hash a given unique value/identifier (e.g. a file name) to a peer key space and hence they cannot support mapping and lookups for complex objects. Management of those objects whose extents lie in d-dimensional space warrants embedding a logical index structure in place of the 1-dimensional DHT key space. Spatial indices such as Space Filling Curves (SFC) [150],

k-d Tree [80], R-Tree [114], and MX-CIF Quadtree [166] can be utilised for managing such complex objects over a DHT key space.

In this work, we utilise the P2P publish/subscribe based Grid resource discovery system, described in Chapter 6, for managing and indexing the resource claim and resource ticket objects. The decentralised resource discovery system utilises a d-dimensional spatial index to maintain complex Grid resource look-up (resource claim) and update queries (resource ticket). More details on the spatial index can found in Chapter 6, and details on how we utilise it for distributed load-distribution and coordination among application schedulers can be found in Section 7.3.2.

The rest of chapter is organised as follows: in Section 7.2, we present the background information on shared-spaces based coordinated communication. Section 7.3 discusses the P2P tuple space model that we propose in this chapter. In Section 7.4 and 7.5, we present the finer details on the application scheduling and resource provisioning algorithms. Section 7.7, we present various experiments and discuss our results. We end this chapter with concluding remarks in Section 7.8.

7.2 Background and State-of-the-Art

7.2.1 Shared-space Based Coordinated Communication

The idea of implementing globally accessible "data-space" or "coordination-space" for communication between distributed services goes back to the *blackboard systems* proposed by the Artificial Intelligence research community in early 1970s. The blackboard system was utilised as a global slate by experts to collaborate on solving the difficult problems. Experts would search the blackboard for problems of their expertise and post the solutions. The idea of global slate was implemented in the systems including JavaSpaces [121], TSpaces [120] and XMLSpaces [169]. These implementations were based on the centralised CS-model which has limited scalability. Initially, the slates were utilised for coordinating parallel application execution between a cluster of computers. Traditionally, these blackboard systems supported $Read()$ and $Write()$ primitive for information coordination between services.

The shared-space based coordination approach or model was proposed by the Linda [83] system, which defined a centralised tuple space that provided abstraction of a shared message store for supporting generative communication. Linda defines a tuple as an ordered sequence of typed fields and a tuple space as a shared repository that includes a set of tuples which can be accessed by several distributed processes synchronously. The Linda system also defines separate tuple access primitives for reading, writing and destroying. Tuples are written to the shared space through execution of $out(t)$ primitive, read using the non-destructive primitive $rd(\bar{t})$, and extracted using the destructive primitive $in(\bar{t})$.

7.2.2 State-of-the-Art

In recent times, there have been proposals for organising a coordination space based on a decentralised network model, the representative systems being Lime [126], PeerWare [53], PeerSpace [28] and Comet [111]. Systems including Lime and PeerWare support a global coordination space using a distributed index called Global Virtual Data Structure (GVDS). The focus of Lime system is to provide coordination among participants in mobile environments. The global data space is built by combining the local data spaces of participating peers. The changes made in the local data space are reflected in the global data space. The data structure managed by PeerWare is organised as a graph composed of nodes and documents which are collectively referred to as items. Every peer in the system maintains a local graph structure, which are superimposed on each other to form the GVDS. The management of such a global data structure in a highly dynamic and large distributed system is not scalable.

The most related state of the art to this research is Comet System, that utilises DHTs as the basis for organising the GVDS. The advantage of utilising the DHT is that updates, inserts and deletes on the local tuples (keys) are not required to be communicated to the global tuple space. The changes to the tuple space due to these operations (insert, delete, and update) are handled by the logical mapping structure that forms the basis for tuple space management. A Hilbert SFC index, proposed in Squid [150], is utilised as the mapping structure from the logical tuple space to the Chord identifier space. In contrast, our mapping structure is based on the spatial publish/subscribe index whose details can

be found in Chapter 6.

7.3 Peer-to-Peer Tuple Space Model

In this section we first describe the communication, coordination and indexing models which are utilised to facilitate the P2P tuple space. Then we look at the composition of tuples, access primitives that form the basis for coordinating the application schedules among the decentralised and distributed resource brokers.

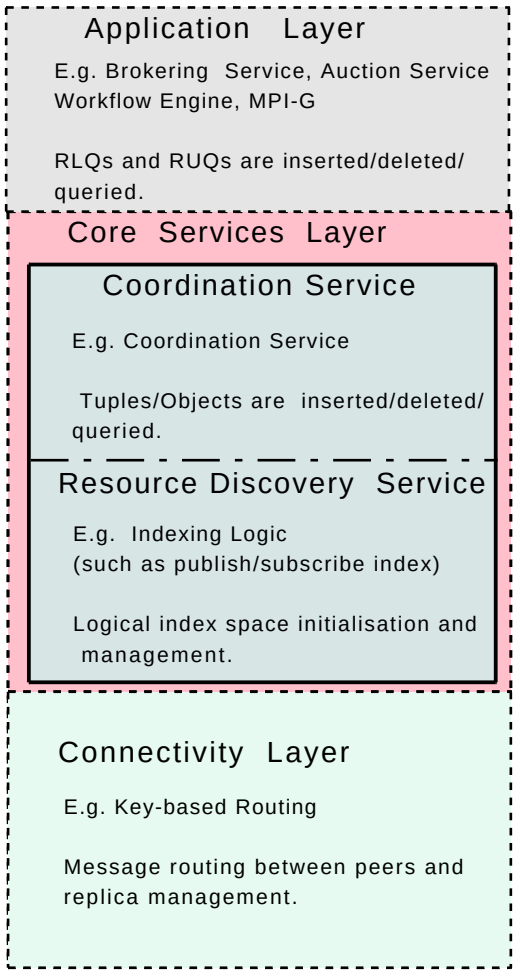


Figure 7.1: A schematic overview of the Coordination service architecture.

7.3.1 Layered Design of the Coordination Space

Fig. 7.1 shows the layered design of the proposed P2P tuple space based coordination service. The OPeN architecture proposed in prior work [166] is utilised as the base model in architecting and implementing the proposed service. The OPeN architecture consists of three layers: the *Application* layer, *Core Services* layer and *Connectivity* layer. Grid Services such as resource brokers work at Application layer and insert objects including Resource Lookup Query (RLQ) and Resource Update Query (RUQ) to the Core services layer.

We have implemented the Coordination service as a sub-layer of the Core services layer. The Coordination service accepts the application objects such as RLQs/ RUQs. These objects are then wrapped with some additional logic to form a coordination tuple or object. The coordination logic, in this case the resource provisioning logic, are executed by the Coordination service on these tuples or objects. However, the calls between the Coordination service and Resource discovery service are done through the standard publish/subscribe mechanism. The Resource discovery service is responsible for managing the logical index space and communicating with the Connectivity layer. The details on the workings of the Resource discovery service can be found in Chapter 6. Note that, the proposed tuple space does not strictly follow the standard Linda primitive, instead it exposes the APIs such as $publish(ticket)$, $subscribe(claim)$ and $unsubscribe(claim)$ that suites the requirements of the Application layer brokering service.

The Connectivity layer is responsible for undertaking key-Based routing in the DHT space such as Chord, CAN, Pastry etc. The actual implementation protocol at this layer does not directly affect the operations of the Core services layer. In principle, any DHT implementation at this layer could perform the desired task. However, in this chapter the simulation models the Chord substrate at the Connectivity layer. Chord hashes the peers and objects (such as fileIds, logical indices etc) to the circular identifier space and guarantees that an object in the network can be located in $O(\log\ n)$ steps with high probability. Each peer in the Chord network is required to maintain the routing state of only $O(\log\ n)$ other peers, where n is the total network size.

7.3.2 Coordination Tuples/Objects

This section gives details about the resource claim and ticket objects that form the basis for enabling decentralised coordination mechanism among the brokers/GFAs in a Grid system. These coordination objects include:- Resource Claim and Resource Ticket. We start with the description of the components that form the part of a Grid-Federation resource ticket object.

Resource Ticket

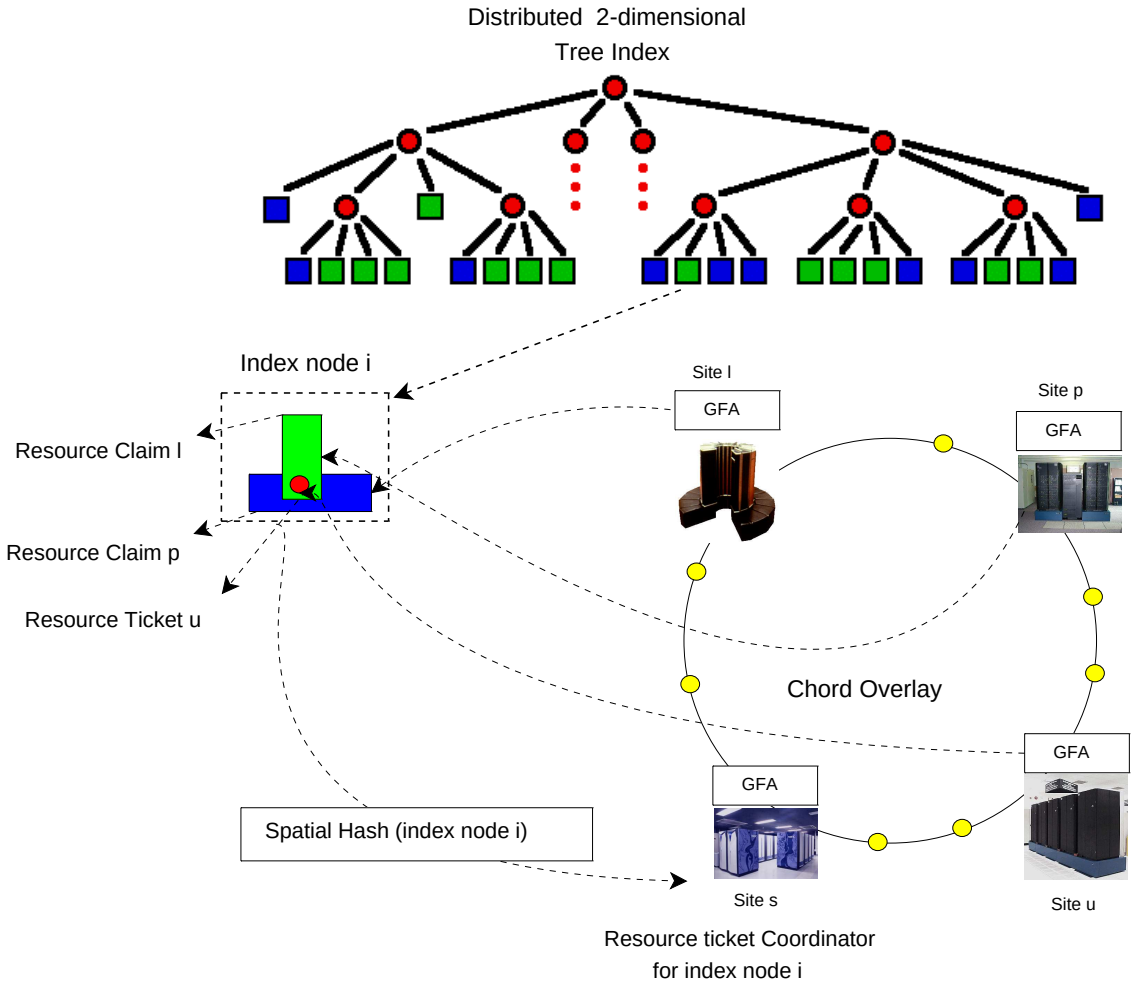


Figure 7.2: Resource allocation and application scheduling coordination across Grid sites.

Every GFA in the federation publishes its resource ticket with the local Coordination

service. A resource ticket object U_i consists of a resource description R_i, for a cluster i. A R_i can include information about the CPU architecture, number of processors, RAM size, secondary storage size, operating system type, resource usage cost etc. In this work $R_i = (p_i, x_i, \mu_i, \emptyset_i, \rho_i, c_i)$, which includes the number of processors, p_i, processor architecture, x_i, their speed, μ_i, their utilization, ρ_i, installed operating system type, $\emptyset_i$, and a cost c_i for using that resource. A site owner charges c_i per unit time or per unit of million instructions (MI) executed, e.g. per 1000 MI. The ticket publication process can be based on time intervals or resource load triggers. Recall from Chapter 6 that a resource ticket object has similar semantics to the RUQ object.

Resource Ticket: Total-Processors= 100 && Processor-Arch= Pentium && Processor-Speed= 2 GHz && Operating-System = Linux && Utilization=0.80 && Acess-Cost=1 Dollar/min.

Resource Claim

A resource claim object encapsulates the resource configuration needs of a user's job. In this work, we focus on the job types whose needs are confined to computational grid or PlanetLab resources. Users submit their application's resource requirements to the local GFA. The GFA service is responsible for searching the resources in the federated system. An user job in the Grid-Federation system is written as $J_{i,j,k}$, to represent the i-th job from the j-th user of the k-th resource. A job consists of the number of processors required, $p_{i,j,k}$, processor architecture, $x_{i,j,k}$, the job length, $l_{i,j,k}$ (in terms of instructions), the budget, $b_{i,j,k}$, the deadline or maximum delay, $d_{i,j,k}$ and operating system required, $\emptyset_{i,jk}$. A GFA aggregates these application characteristics including $p_{i,j,k}$, $x_{i,j,k}$, $\emptyset_{i,j,k}$ with constraint on maximum speed, cost and resource utilization into a resource claim object, $r_{i,j,k}$. Recall from Chapter 6 that a resource claim object has similar semantics as an RLQ object and is d-dimensional in composition.

Resource Claim: Total-Processors $\geq$ 70 && Processor-Arch= pentium && 2 GHz $\leq$ Processor-Speed $\leq$ 5GHz && Operating-System = Solaris && 0.0 $\leq$ Utilization $\leq$ 0.90

&& 0 Dollar/min $\leq$ *Access-Cost* $\leq$ *5 Dollar/min .*

The resource ticket and claim objects are spatially hashed to an index cell i in the d-dimensional coordination space. Similarly, coordination services in the Grid network hash themselves into the space using the overlay hashing function (SHA-1 in case of Chord and Pastry). The details on index cell mapping to the coordination services is described in Chapter 6. In Fig. 7.2, resource claim objects issued by site p and l are mapped to the index cell i, and are currently hashed to the site s. In this case, site s is responsible for coordinating the resource sharing among all the resource claims that are mapped to the cell i. Subsequently, site u issues a resource ticket (shown as dot in Fig. 7.2) which falls under the region of space currently required by users at site p and l. In this case, the coordinator service of site s has to decide which of the sites (i.e. either l or p or both) be allowed to claim the ticket issued by site u. This load-distribution decision is based on the fact that it should not lead to over-provisioning of resources at site u.

In case a resource ticket matches with one or more resource claims, then a coordinator service sends *notification* messages to the resource claimers such that it does not lead to the overloading of the concerned resource ticket issuer. Thus, this mechanism prevents the resource brokers from overloading the same resource. In case of PlanetLab environment, it can prevent the users from instantiating *slivers* on the same set of nodes. Once a scheduler receives notification that its resource claim has matched with an advertised resource ticket, the scheduler undertakes a Service Level Agreement (SLA) (described in Chapter 5) contract negotiation with the ticket issuer site. In case agreement is reached, the scheduler can go ahead and deploy its application/experiment. The GFAs have to reply as soon as the SLA enquiry arrives. In other words, we set the SLA timeout interval as 0. We do this in order to study the effectiveness of coordination space with respect to decentralised load-balancing. As excessive timeout interval can lead to a deadlock situation in the system, with coordination service sending the notifications while the ticket issuer is not accepting SLA contracts. In future we intend to study how varying the degree of SLA timeouts can affect the system performance in terms of load-balancing and provider's economic benefit.

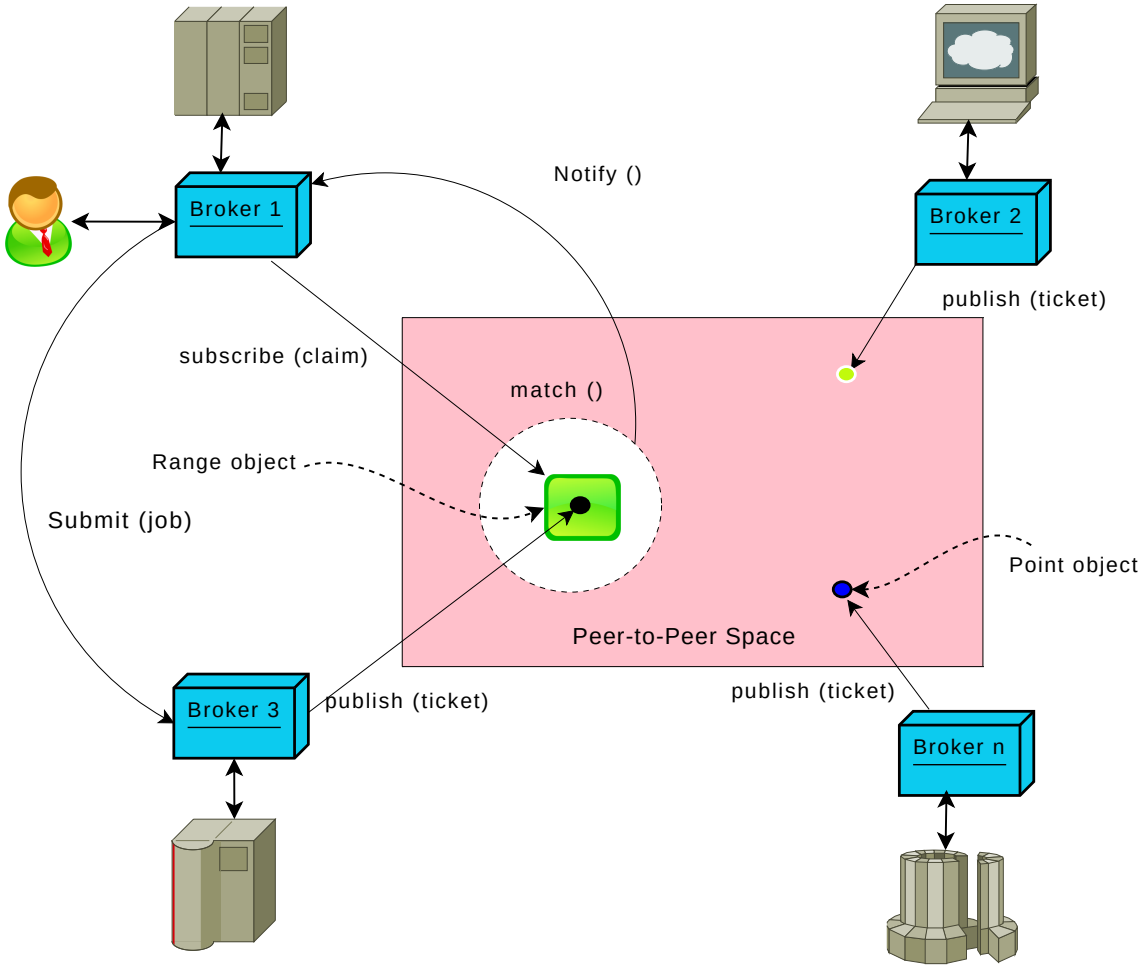


Figure 7.3: Scheduling and resource provisioning coordination through P2P tuple space.

7.4 Distributed Application Scheduling Algorithm

In this section we provide detailed descriptions of the scheduling algorithm that is undertaken by a GFA in the Grid-Federation system following the arrival of a job:

1. When a job arrives at a GFA, the GFA compiles a resource ticket object for that job. It then posts this resource ticket object with the P2P tuple space though the Core services layer. The complete pseudo code for this process is shown in Fig. 7.4. In Fig. 7.3 GFA 1 is posting a resource claim on behalf of its local user.

2. When a GFA receives a notification for a resource ticket and resource claim match from the P2P coordination space, it then undertakes SLA-based negotiation with the ticket

issuer GFA. After successful notification, the coordination service unsubscribes the resource claim for that job from the tuple space. In Fig. 7.3 the match event occurs and GFA 1 is notified that it can place the job with GFA 3. Following this, GFA 1 undertakes SLA negotiation with GFA 3, which is accepted and, finally GFA 1 migrates the locally submitted job to the GFA 3.

3. If SLA negotiation is successful then the GFA sends the job to the remote GFA, otherwise it again posts the resource claim object for that particular job to the coordination space.

7.5 Distributed Resource Provisioning Coordination Algorithm

In this section we present the details on the decentralised resource provisioning algorithm which is undertaken by the coordination services across the P2P tuple space.

1. When a resource claim object arrives at a coordination service for future consideration, the coordination service queues it in the existing claim list as shown in the Fig. 7.5.

2. When a resource ticket object arrives at a coordination service, the coordination service calls the auxiliary procedure match($Ticket\ U_i$) (as shown in Fig. 7.5) to gather the list of resource claims that overlaps with the submitted resource ticket object in the d-dimensional space. This initial resource claim match list is passed to another auxiliary procedure Load_Dist(matchList, ticket).

3. The Load_dist() procedure notifies the resource claimers about the resource ticket match until the ticket issuer is not over-provisioned. The Load_Dist() procedure can utilise the resource parameters such as number of available processors and the threshold queue length as the over-provision indicator. These over-provision indicators are encapsulated with the resource ticket object by the GFAs. The GFAs can post the resource ticket object

```
1  PROCEDURE: GFA_SCHEDULING
2  begin
3      begin
4          Sub-Procedure: Event_User_Job_Submit (Job  J_{i,j,k})
5          encapsulate the claim object r_{i,j,k} for job J_{i,j,k}
6          call Post_Resource_Claim (r_{i,j,k}).
7      end
8      begin
9          Sub-Procedure: Post_Resource_Claim (Claim r_{i,j,k})
10         call subscribe (r_{i,j,k}).
11     end
12     begin
13         Sub-Procedure: Event_Resource_Status_Changed(Resource R_i)
14         encapsulate the ticket object U_i for resource R_i
15         call publish (U_i).
16     end
17     begin
18         Sub-Procedure: Event_Coordinator_Reply (GFA  gindex_i)
19         call SLA_Bid (J_{i,j,k}, gindex_i).
20     end
21     begin
22         Sub-Procedure: SLA_Bid (Job  J_{i,j,k}, GFA  gindex_i)
23         Send SLA bid for job J_{i,j,k} to the decentralised coordinator adviced GFA gindex_i.
24     end
25     begin
26         Sub-Procedure: Event_SLA_Bid_Reply (J_{i,j,k})
27         if (SLA Contract Accepted) then
28             Send the job J_{i,j,k} to accepting GFA.
29         end
30         else
31             call SLA_Bid_Timeout (J_{i,j,k}).
32         end
33     end
34     begin
35         Sub-Procedure: SLA_Bid_Timeout(J_{i,j,k})
36         call Post_Resource_Claim (r_{i,j,k}) .
37     end
38 end
```

Figure 7.4: SLA-based GFA application scheduling algorithm.

to the tuple space either periodically or whenever the resource condition changes such as a job completion event occurring.

7.6 Simulation Model

Here, we present a simulation model for evaluating the performance of the proposed coordinated resource provisioning approach. The simulation model is similar to the one considered in Chapter 6. However, the only difference is that in Chapter 6 the Core service layer only included one sub-layer called Index Service, while in this chapter we extend the Core services layer to include the Coordination services too. The message queuing model remains exactly the same in which a Grid peer node (through its Chord routing service) is connected to an outgoing message queue and an incoming link from the Internet (described Chapter 6). The network messages delivered through the incoming link are processed as soon as they arrive. Further, the Chord routing service receives messages from the local publish/subscribe Index service. Similarly, these messages are processed as soon as they arrive at the Chord routing service.

After processing, the Chord routing service queues the message in the local outgoing queue. Basically, this queue models the network latencies that a message encounters as it is transferred from one Chord routing service to another on the overlay. Once a message leaves an outgoing queue, it is directly delivered to a Chord routing service through the incoming link. The distributions for the delays (including queueing and processing) encountered in an outgoing queue are given by the M/M/1/K queue steady state probabilities. The Coordination service directly connects to the Index service. Effectively, there is negligible delay in message transfer between Coordination and Index service.

7.7 Performance Evaluation

In this section, we validate the proposed P2P tuple space-based coordinated resource provisioning model through trace-based simulations. The simulated environment models the Grid-Federation resource sharing environment presented in Chapters 4 and 5 as a case study.

```
1  PROCEDURE: Resource_Provision
2  begin
3      list ← φ
4      begin
5          Sub-Procedure: Event_Resource_Claim_Submit (Claim r_{i,j,k})
6          list ← list ∪ r_{i,j,k}.
7      end
8      begin
9          Sub-Procedure: Match (Ticket U_i)
10         list_m ← φ
11         set index = 0
12         while ( list[index] ≠ null ) do
13             if ( Overlap (list[index], U_i) ) then
14                 list_m ← list_m ∪ list[index]
15             end
16             else
17                 continue
18             end
19             reset index = index + 1
20         end
21         return list_m .
22     end
23     begin
24         Sub-Procedure: Overlap (Claim r_{i,j,k}, Ticket U_i)
25         if (r_{i,j,k} ∩ U_i ≠ null ) then
26             return true.
27         end
28         else
29             return false.
30         end
31     end
32     begin
33         Sub-Procedure: Event_Resource_Ticket_Submit (U_i)
34         call Load_Dist(U_i, Match(U_i)).
35     end
36     begin
37         Sub-Procedure: Load_Dist (U_i, list_m)
38         set index = 0
39         while (R_i is not over-provisioned) do
40             send notification match event to resource claimer: list_m [index]
41             remove(list_m [index])
42             reset index = index + 1.
43         end
44     end
45 end
```

Figure 7.5: Resource provisioning algorithm for coordination service.

7.7.1 Experimental Setup

We start by describing the test environment setup.

Broker Network Simulation:

In a similar fashion to Chapter 6, our simulation infrastructure includes two discrete event simulators; namely *GridSim* [32], and *PlanetSim* [82]. We model the resource brokering service (i.e. a GFA) inside GridSim, that injects resource claims and resource tickets on behalf of both, the users and the resource providers respectively. Every GFA connects to the Core services layer which also has implementations for Coordination service and publish/subscribe Index service as sub-layers. At the Connectivity layer we utilised the Chord implementation provided with PlanetSim.

Experiment configuration:

- Network configuration: The experiments ran a Chord overlay with 32 bit configuration (i.e. number of bits utilised to generate node and key ids). The network size n was fixed at 100 GFA/broker nodes. The network queue message processing rate, μ, at a Grid peer was fixed at 500 messages per second. The message queue size, K, was fixed at 10^4.

- Resource claim and resource ticket injection rate: The GFAs inject resource claim and resource ticket objects based on an exponential inter-arrival time distribution. The value for resource claim inter-arrival delay ($\frac{1}{\lambda_l^{in}}$) is distributed over the interval $[5, 60]$ in step of 5 secs. While the inter-arrival delay ($\frac{1}{\lambda_u^{in}}$) of resource claim object was fixed to 30 secs. The inter-arrival delay in claim/ticket injection is considered same for all GFAs/brokers in the system. The spatial extent of both resource claims and resource ticket objects lies in a 5-dimensional attribute space. The attribute dimension includes the number of processors, p_i, resource access cost, c_i, processor speed, m_i, processor architecture, x_i, and operating system type, ϕ_i. The distributions for these resource dimensions have been obtained from the Top 500

supercomputer list[1].

Note that, in our simulation we did not utilize resource utilization, ρ_i , as the GFA's load indicator. Instead, GFAs encode the metric *"number of available processors"* at time t with the resource ticket object U_i. Specifically, the information on the number of available processor is updated inside the $gindex_i$ object and sent to the coordination service along with ticket object U_i. The coordination service utilizes this metric as the indicator for the current load on the resource R_i. In other words, the coordinator service would stop sending the notifications as the number of processors available with a ticket issuer reaches *zero*.

- Publish/subscribe index configuration: The minimum division, f_{min}, of logical d-dimensional publish/subscribe index was set to 3, while the maximum height of the index tree, f_{max}, was also limited to 3. This means we do not allow the partitioning of the P2P tuples space beyond f_{min} level. In this case, a cell at a minimum division level does not undergo any further division. Hence, no resource claim or resource ticket object is stored beyond the f_{min} level. The index space resembles a Grid-like structure where each index cell is randomly hashed to a Grid peer based on its control point value. The publish/subscribe Cartesian space had 6 dimensions including number of processors, p_i, resource access cost, c_i, processor speed, m_i, processor architecture, x_i, and operating system type, ϕ_i. Hence, this configuration resulted into 243 (3^5) Grid index cells at the f_{min} level. On an average, 2 index cells are hashed to a Grid peer in a network comprising of 100 Grid sites.

Indexed data distribution: We generated a resource type distribution using the resource configuration obtained from the Top 500 Supercomputer list. We utilised the resource attributes including processor architecture, its number, its speed, and installed an operating system from the Supercomputer list. The value for c_i was uniformly distributed over the interval $[0, 10]$.

Workload configuration: We generated the workload distributions across GFAs based on the model given in the paper [116]. The workload model generates the

[1]Top 500 Supercomputer List, http://www.top500.org/

job-mixes having the details on their run times, sizes, and inter-arrival times. This model is statistically derived from existing workload traces and incorporates correlations between job run times and job sizes and daytime cycles in job inter-arrival times. The model calculates for each job its arrival time using a 2-gamma distributions, and the number of nodes using a two-stage-uniform distribution, and the run time using the number of nodes and a hyper-gamma distribution.

Mostly we utilised the default parameters already given by the model except for the number of processors/machines. The processor count for a resource was fed to the workload model based on the resource configuration obtained from the Top 500 list. The simulation environment models 25 jobs at each GFA, and since there are 100 GFAs therefore total number of jobs in the system accounts to 2500. Also note that, we simulated the supercomputing resources in space shared processor allocation mode. More details on how the execution time for jobs are computed on space shared resource facilities can be found in Chapter 4.

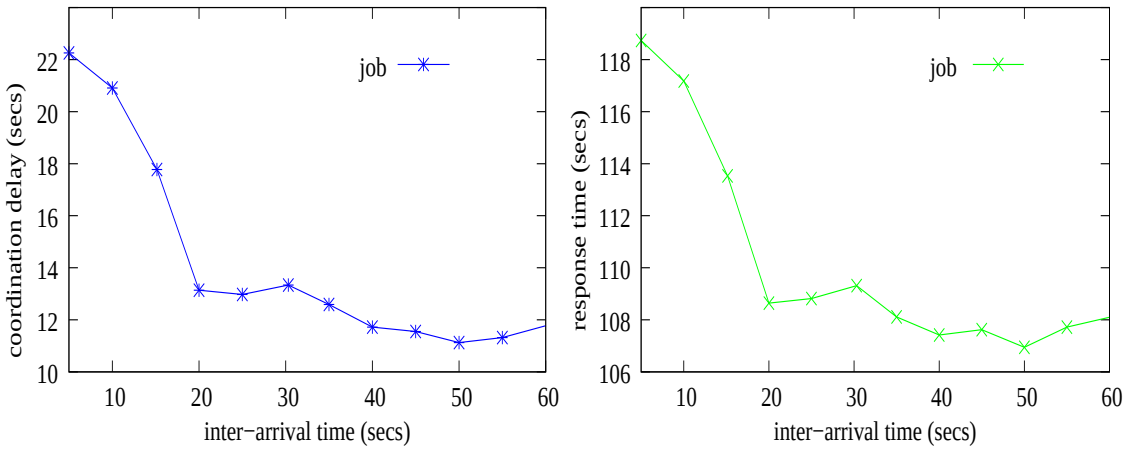
7.7.2 Effect of Job Inter-Arrival Delay: Lightly-Constrained Workloads

The first set of experiments measured the performance of P2P tuple space in coordinating resource provisioning with respect to the following metrics: average coordination delay, average response time and average processing time for jobs. Further, it also quantifies the details about the job migration statistics in the system; the number of jobs executed locally and number jobs executed remotely. In this experiment, the resource claim injection rate is varied from 12 to 1 per minute while the resource ticket injection rate is fixed to 2 per minute. This experiment simulates a lightly-constrained workload or job characteristic. In other words, on an average the simulated jobs did not require large number of processors for execution. For this experiment, the job characteristics were generated by configuring the minimum and maximum processor per job as 2 and 2^6 respectively in the workload model.

Fig. 7.6 and Fig. 7.7 show the measurement for parameters coordination delay, re-

sponse time, processing time and job migration. The metric coordination delay sums up the latencies for: (i) a resource claim to reach the index cell; (ii) the waiting time till a resource ticket matches with the claim; and (iii) the notification delay from coordination service to the relevant GFA. The processing time for a job is defined as the time the job takes to actually execute on a processor or set of processors. The average response time for a job is the delay between the submission and arrival of execution output. Effectively, the response time includes the latencies for coordination and processing delays. Note that these measurements were collected by averaging the values obtained for each job in the system.

Fig. 7.6(a) depicts results for the average coordination delay in seconds with increasing job inter-arrival delay. With increase in average job inter-arrival delay, we observed a decrease in the average coordination delay. The results show that at higher inter-arrival delays, resource claim objects experience less network traffic and competing requests. Thus, this leads to an overall decrease in the coordination delay across the system. The effect of this can also be seen in the response time metric for the jobs (refer to Fig. 7.6(b)), which is also seen to improve with an increase in inter-arrival delays.



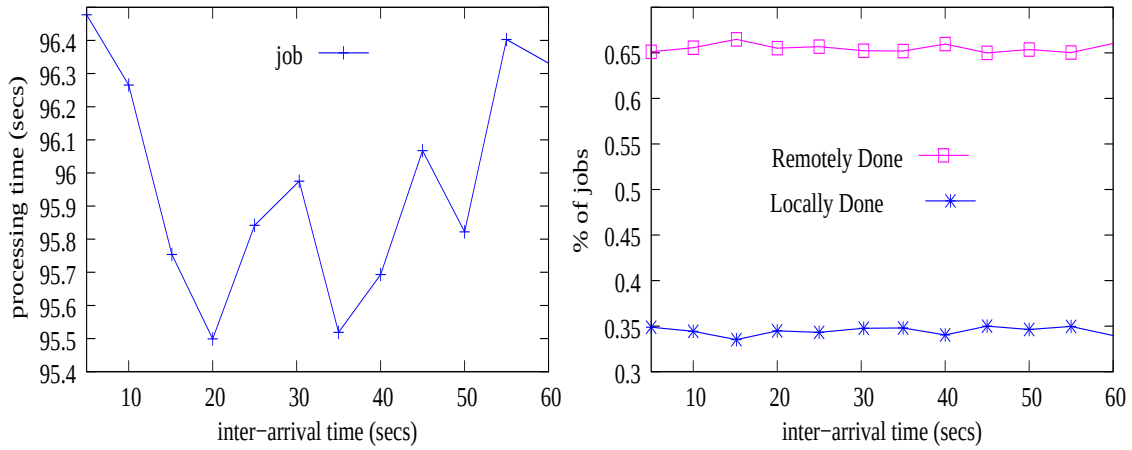
(a) average job inter-arrival delay (secs) vs average coordination delay (secs).

(b) average job inter-arrival delay (secs) vs average response time (secs).

Figure 7.6: Simulation: Effect of job inter-arrival delay: lightly-constrained.

Fig. 7.7(a) depicts results for the average job processing delay in seconds with increasing job inter-arrival delay. As expected, the processing delays do not change significantly with an increase in the inter-arrival delay. This is due to the availability of resources with

similar or near similar processing capabilities in the Top 500 list. Hence, allocation of jobs to any of the resource does not have significant effect on the overall processing time. Further, the job-migration statistics also showed negligible change with increasing job inter-arrival delays (refer to Fig. 7.7(b)). At every step, approximately 65% of jobs were executed remotely while the remaining jobs executed at the originating site itself.



(a) average job inter-arrival delay (secs) vs processing time (secs).

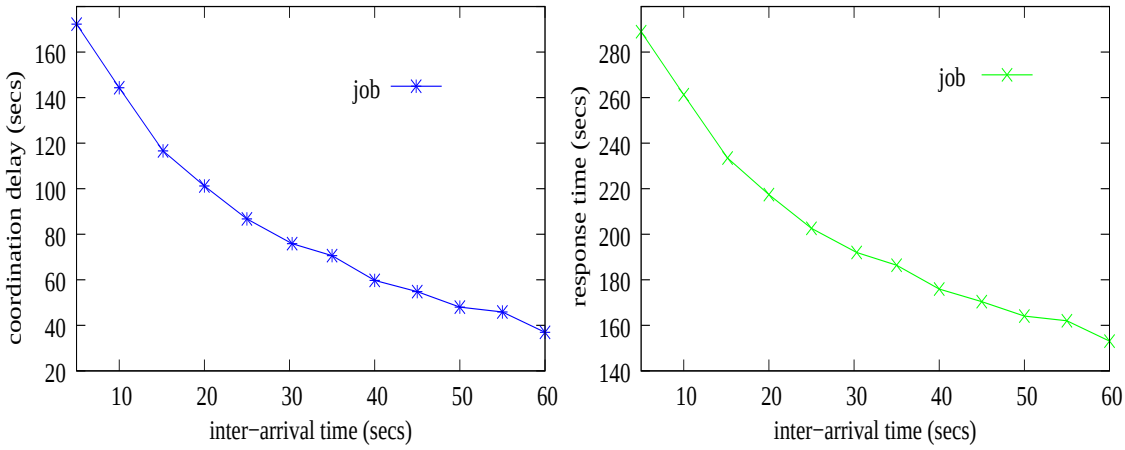
(b) average job inter-arrival delay (secs) vs $\%$ of jobs.

Figure 7.7: Simulation: Effect of job inter-arrival delay: lightly-constrained.

7.7.3 Effect of Job Inter-Arrival Delay: Heavily-Constrained Workloads

This experiment simulates the performance of P2P tuple space in coordinating resource provisioning for highly-constrained workload or job characteristic. The heavily-constrained workloads on an average require relatively larger number of processors on per job-basis as compared to the lightly-constrained ones. For this experiment, the job characteristics were generated by configuring the minimum and maximum processor per job as 2^6 and 2^8 respectively in the workload model. Other simulation configurations stay the same as described for the previous experiment.

Fig. 7.8(a) depicts results for the average coordination delay in secs with increasing job inter-arrival delay. With increase in average job inter-arrival delay, we observed noticeable decrease in the average coordination delay. At an inter-arrival delay of 5 secs, on



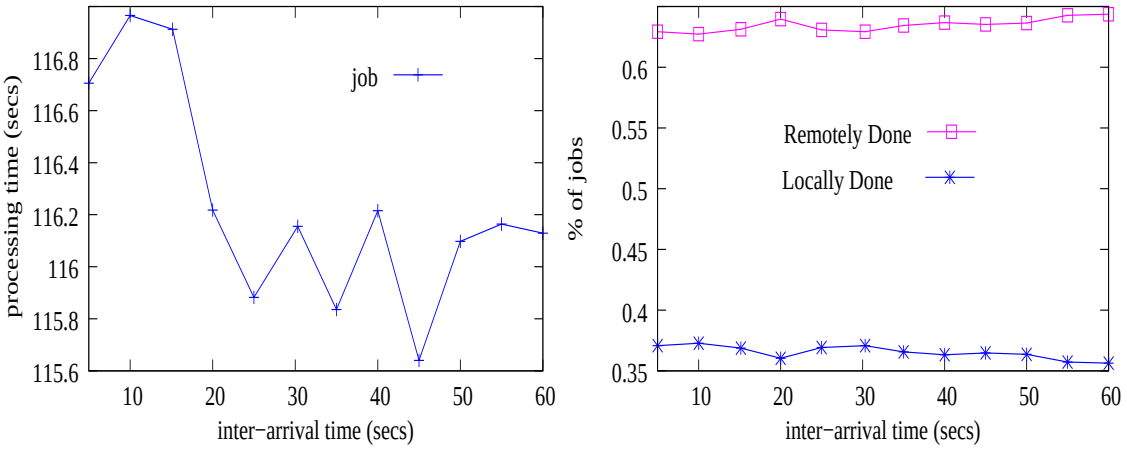
(a) average job inter-arrival delay (secs) vs average coordination delay (secs).

(b) average job inter-arrival delay (secs) vs average response time (secs).

Figure 7.8: Simulation: Effect of job inter-arrival delay: heavily-constrained.

the average job experienced a coordination delay of about 172 secs (refer to Fig. 7.8(a)). At an inter-arrival delay of 50 secs, the coordination delay decreased to 45 secs. The results show that at higher inter-arrival delays, resource claim objects experience less network traffic and competing requests. However, we saw the same trend in the case of lightly-constrained jobs as well, where the decrease in case of heavily-constrained jobs is more significant (about 73%). The chief reason behind this being that there is a higher degree of competition between resource claim requests, as on average they required larger number of processors for execution. The effect of diminishing coordination delay can be seen in the response time metric for the jobs as well (refer to Fig. 7.8(b)), which is also seen to improve with increase in inter-arrival delays.

Similar to the lightly-constrained case, we observed that the processing delays (see Fig. 7.9(a)) does not change significantly with increases in inter-arrival delay. Further, the job-migration statistics also showed negligible or very little change with increasing job inter-arrival delays (refer to Fig. 7.9(b)). At every step, approximately 62% of jobs were executed remotely while remaining executed at the originating site itself.



(a) average job inter-arrival delay (secs) vs processing time (secs).

(b) average job inter-arrival delay (secs) vs % of jobs.

Figure 7.9: Simulation: Effect of job inter-arrival delay: heavily-constrained.

7.8 Conclusion

In this chapter, we described a P2P tuple space framework for efficiently coordinating resource provisioning in a federated Grid system such as the Grid-Federation. The proposed coordination space built upon the resource discovery system presented in Chapter 6. The simulation based study shows that heavily- constrained workloads can experience significant coordination delays due to the competing requests in the system. However, the same is not true when the workloads are lightly-constrained i.e. the resource claim requests for lesser number of processors.

One limitation with our approach is that the current publish/subscribe index can map a resource claim object to at most 2 index cells. In some cases this can lead to generation of unwanted notification messages in the system and may be to an extent sub-optimal load-balancing as well. In our future work, we plan to address this issue by constraining the mapping of a resource claim object to an index cell. Another way to tackle this problem is to make the peers currently managing the same resource claim object communicate with each other before sending the notifications.

Chapter 8

Coordinated Federation of Alchemi Desktop Grids

This chapter presents the design and implementation of the Alchemi-Federation software system. The software serves as a proof of concept for our main results in thesis. In this chapter we start with a brief description of background information on the Alchemi desktop Grid computing system in Section 8.2. Section 8.3 presents the overall software architecture of the Alchemi-Federation system; including details on the individual components. Section 8.4 discusses the implementation of a P2P publish/subscribe based resource discovery service and software interfaces. In Section 8.5, we present details on service deployment and bootstrap. Section 8.6 includes the discussion on the performance evaluation. Finally, chapter ends with a discussion on conclusions and future work.

8.1 Introduction

The Alchemi-Federation system logically connects topologically and administratively separated Alchemi grids as part of a generalised resource sharing system. Fig. 8.3 depicts the proposed decentralised Alchemi-Federation based on a publish/subscribe resource indexing service. Each Alchemi grid site is managed by a software service called Grid-Federation Agent (GFA). A GFA exports an Alchemi grid resource to a wide-area resource sharing environment. A GFA has the following basic software modules: Local Resource

Management System (LRMS), Grid Resource Manager (GRM), and Distributed Information Manager (DIM). We discussed the functionalities of these modules in Chapter 4. In this chapter we elaborate on their software architecture, design and implementation.

The Alchemi-Federation framework is developed with the aim of making distributed Grid resource integration and application programming efficient, flexible, and scalable. The conceived software service can act as a base platform for hosting a variety of distributed applications and programming models. Some of the important application domains include Grid workflow composition, distributed auctions, distributed storage management with trading framework and wide-area parallel programming environment. The unique features of Alchemi-Federation are:

- Internet-based federation of distributed Alchemi grids;
- implementation of a P2P publish/subscribe based resource indexing service that makes the system highly scalable;
- implementation of a P2P tuple space-based distributed load-balancing algorithm.

8.2 Alchemi: A Brief Introduction

Alchemi [117] is a .Net based enterprise Grid computing and runtime machinery for creating a high-throughput resource sharing environment. An Alchemi Manager logically couples the Windows Desktop machines running the instance of Alchemi Executor service. An Executor service can be configured to receive and execute jobs both in voluntary and non-voluntary modes. Alchemi exposes run-time machinery and a programming environment (API) required for constructing Desktop Grid applications. The core Alchemi middleware relies on the master-worker model - a manager is responsible for coordinating the execution of tasks sent to its executors (desktop machines). The layered architecture of the Alchemi system is shown in Fig. 8.1.

8.2.1 Programming and Application Model

Alchemi has supporting APIs for the following job execution models: Thread Model and Job model. The Thread Model is used for applications developed natively using the

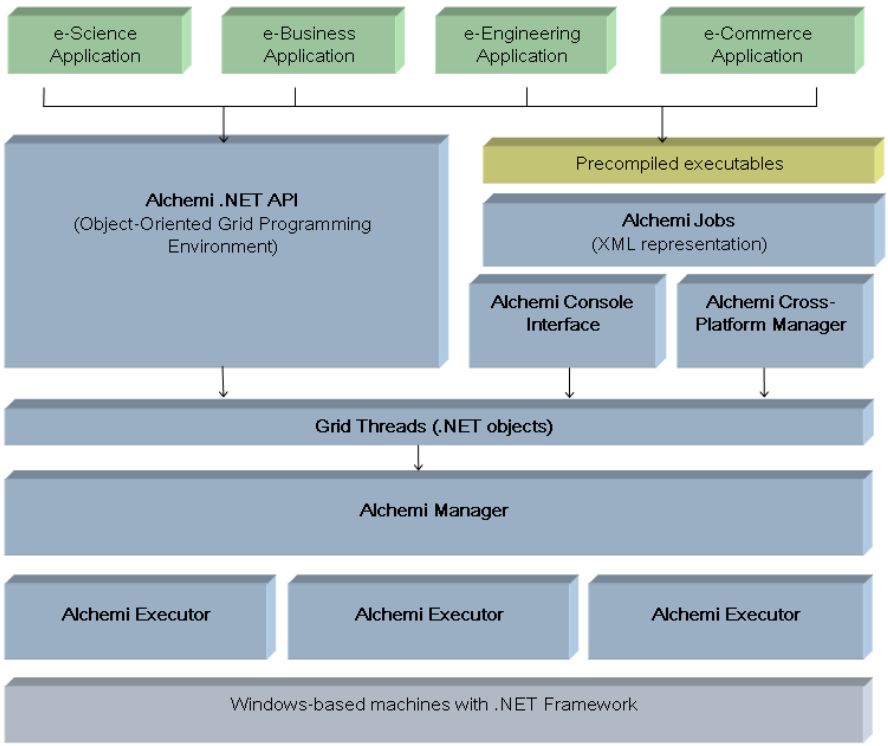


Figure 8.1: Alchemi architecture.

.Net Alchemi application programming framework. This model defines two main classes including GThread and GApplication. A GThread is the simplest unit of task that can be submitted for execution. One or more GThreads can be combined together to form a GApplication such as executing parallel threads over Alchemi to do distributed image rendering. The Job Model has been designed to support legacy tasks developed using different programming platforms (such as C, C++ and Java). These legacy tasks can be submitted to the Alchemi through the Cross Platform Manager. ASP.Net Web service interface host the Cross Platform Manager service which can be invoked by generalised Grid schedulers such as GridBus broker.

Fig. 8.2 illustrates the job submission and execution process involving Alchemi Manager, Executor and users. Application users submit their jobs directly to the local Alchemi Manager. This submission can be done either through Alchemi's API if invoked from .Net platform or Cross Platform Manager's Web service interface. Once the job is submitted,

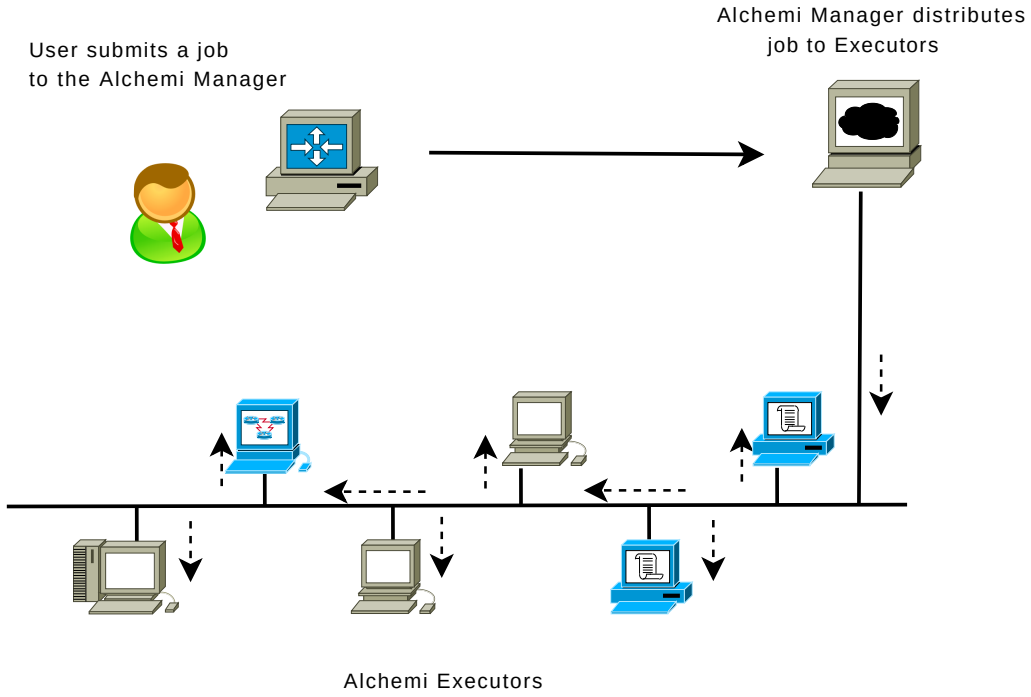


Figure 8.2: Job submission and execution on Alchemi.

the Manager queues it for future consideration by the scheduler. The Alchemi scheduler queries the status of each executor and finally dispatches the job to the available executor. After processing, executors send back back the job output to the owner via the central Manager.

8.3 System Design

This section presents comprehensive details about the software services that govern the overall Alchemi-Federation system. Fig. 8.4 shows the layered architecture of the proposed software system. We start by describing the Grid-Federation Agent service.

8.3.1 Grid-Federation Agent Service

As described in Chapter 4, the GFA service is composed of three software entities including Grid Resource Manager (GRM), Local Resource Management System (LRMS) and Distributed Information Manager (DIM).

Grid Resource Manager (GRM)

The GRM component of a GFA exports the local Alchemi site to the federation and is responsible for coordinating the federation wide application scheduling and resource allocation. We have already discussed in detail the job submission, job queuing and migration in Chapters 2 and 3. This software module is implemented in C-sharp. As shown in Fig. 8.4, GRM interacts with other software modules including LRMS and Grid peer. Both LRMS and Grid peer software modules are implemented in C-sharp so they have no inter-operational issues.

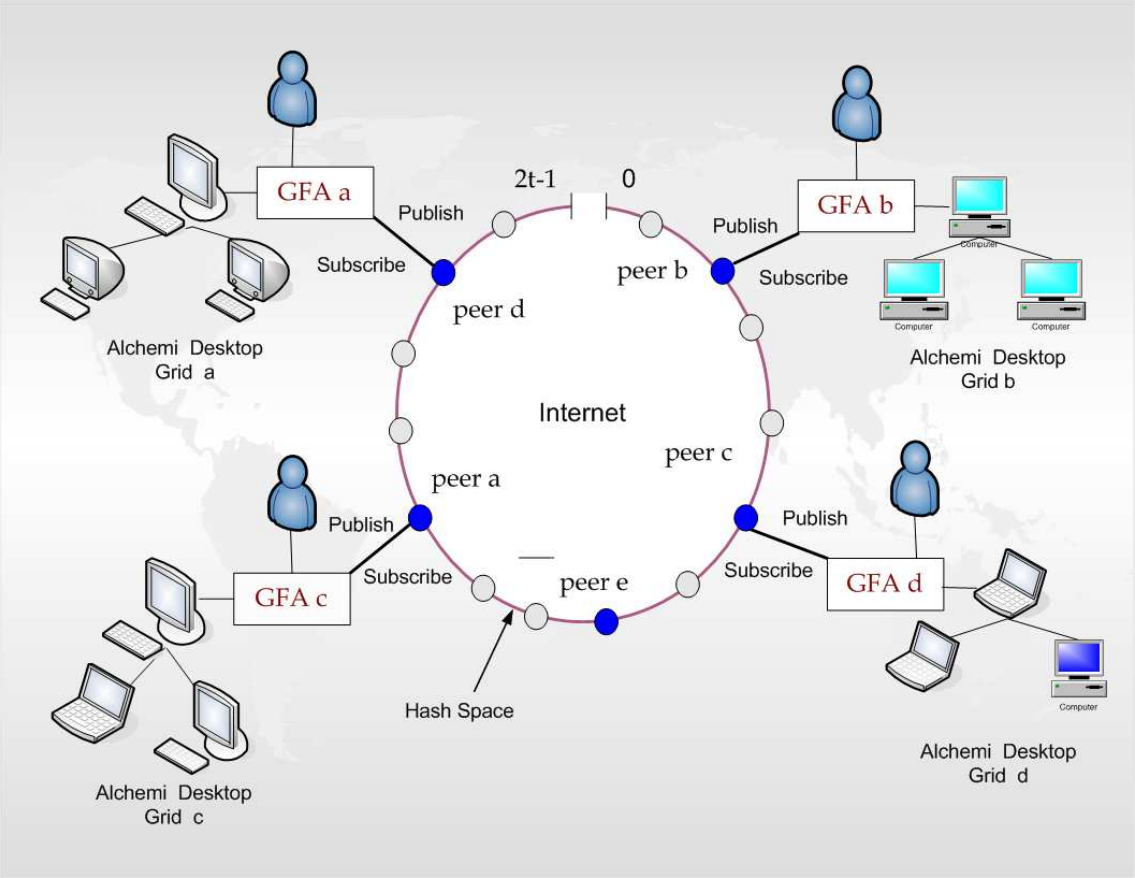


Figure 8.3: Alchemi GFA and Alchemi sites with their Grid peer service and some of the hashings to the Chord ring.

Local Resource Management System (LRMS)

The LRMS software module extends the basic Alchemi Manager module through object oriented software inheritance. Additionally, we implemented the following methods for

facilitating the federation job submission and the migration process: answering the GRM queries related to job queue length, expected response time and current resource utilization status. LRMS inherits the capability to submit applications to Alchemi executors from the basic Alchemi Manager module. The Alchemi executors register themselves with the Manager. This in turn keeps track of their status and availability. In the Alchemi system, a job is abstracted as a Thread object that requires a sequential computation for a fixed duration of time. The executors return the completed threads directly to the LRMS module which in turn sends it to the GRM module. Finally, the GRM module directly returns the thread output to the responsible remote GRM in the federation. In case the job thread is submitted by a user local to a Alchemi grid, then the LRMS directly returns the output without GRM intervention.

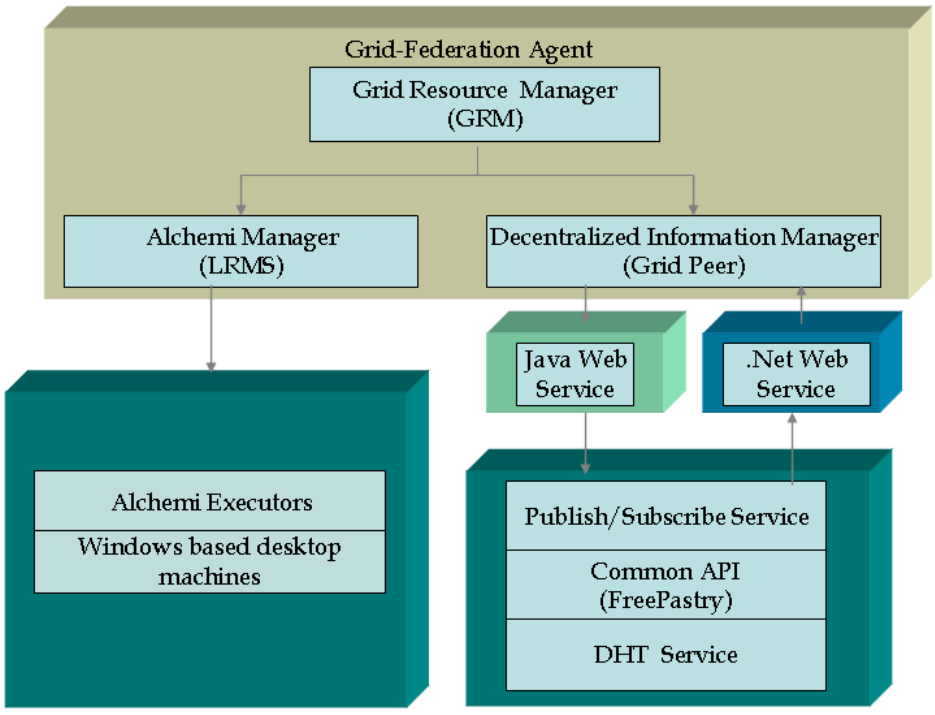


Figure 8.4: Alchemi GFA software interfaces and their interactions.

Grid Peer

The Grid peer module in conjunction with publish/subscribe indexing service performs tasks related to decentralised resource lookups and updates. The details on how the GRM component encapsulates the Resource Lookup Queries (RLQs) and the Resource Update Queries (RUQs) can be found in Chapter 6. Here we discuss the details on interaction protocols between the Grid peer and the publish/subscribe service. The Grid peer module is implemented in C-sharp while the publish/subscribe service is implemented using the Java platform. To resolve the inter-operational issues between these two services we implemented web service interfaces for both modules. The publish/subscribe index service exposes the method for invoking RLQ and RUQ processes through a web service interface (refer to Fig. 8.5).

Apache Tomcat container hosts the publish/subscribe application service. Apache Tomcat is the servlet container that implements the Java Servlet and JavaServer Pages technologies. The specifications for Java Servlet and JavaServer Pages are developed by Sun under the Java Community Process. We utilised the Apache Axis 1.4 SOAP (Simple Object Access Protocol) engine for parsing the XML messages. SOAP is a communication protocol put forward by W3C for exchanging structured information among software entities running in different hosting environment. It is an XML based protocol that is based on three specifications: an envelope that defines a framework for describing what is in a message and how it should be processed, a set of encoding rules for expressing instances of application-defined data types and methods, and a convention for representing Remote Procedure Calls (RPCs) and responses. The Grid peer module implements a .Net web service for receiving the query responses from the publish/subscribe index service. This web service is implemented using ASP.Net and is hosted by the Microsoft Internet Information Service 6.0 (IIS).

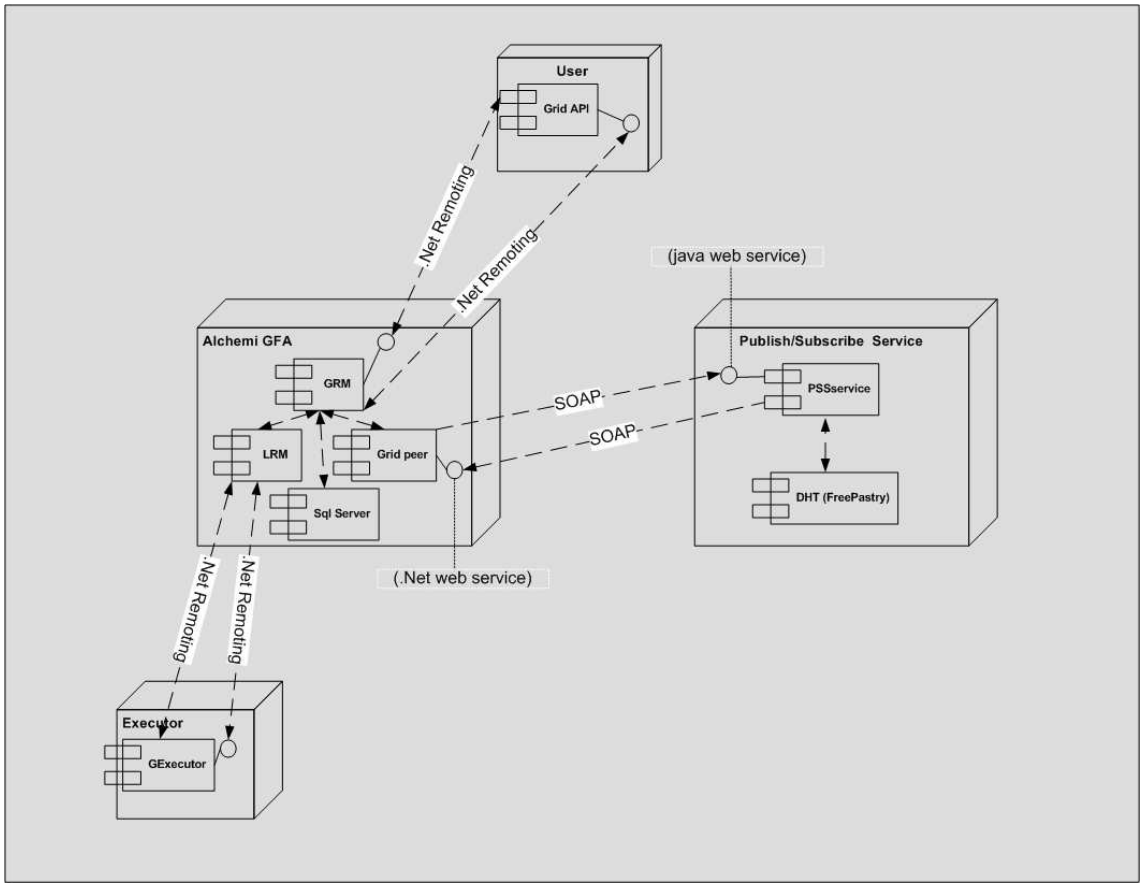


Figure 8.5: Object oriented view of the Alchemi-Federation architecture and the interaction between its components.

8.4 Spatial Index based Peer-to-Peer Publish/Subscribe Resource Discovery Service

The resource discovery service organises data by maintaining a logical d-dimensional publish/subscribe index over a network of distributed Alchemi GFAs. Specifically, GFAs create a Pastry overlay, which collectively maintains the logical publish/subscribe index to facilitate a decentralised resource discovery process. We have presented finer details about the resource discovery service and the spatial index in Chapters 5 and 6. Here, we focus only on implementation details such as design methodology, programming tools, and libraries. The resource discovery service was developed using the core Java programming libraries and FreePastry P2P framework. We utilised the Eclipse Integrated Development Environment (IDE) for system implementation and testing.

Our resource discovery system implementation followed a layered approach known as OPeN architecture. The OPeN architecture consists of three layers: the Application layer, Core Services layer and Connectivity layer. The Application layer implements all the logic that encapsulates the query requirements of the underlying Alchemi Federation environment. The Core Services layer undertakes the consistency and management of virtual d-dimensional indices. The Connectivity layer provides services related to Key-based routing, overlay management and replica placement. The Application service, in conjunction with the Core Services, undertakes the resource discovery tasks including distributed information updates, lookups and virtual index consistency management. While the maintenance of Connectivity layer is left to the basic DHT implementations such as FreePastry, the modules for Application and Core services layer is developed using the standard Java libraries. For Connectivity layer services we utilised the FreePastry framework.

8.4.1 FreePastry

FreePastry is an open source implementation of the well-known Pastry routing substrate. The Pastry protocol was proposed by Microsoft's Systems Research Group Cambridge, United Kingdom and Rice University's Distributed System Group. Pastry offers a generic, scalable and efficient routing substrate for development of P2P applications. It exposes a Key-Based Routing (KBR) API; given the Key K, the Pastry routing algorithm can find the peer responsible for this key in $\log_b n$ messages, where b is the base and n is the number of peers in the network. Nodes in a Pastry overlay form a decentralised, self-organising and fault-tolerant circular network within the Internet. Both data and peers in a Pastry overlay are assigned Ids from a 128-bit unique identifier space. These identifiers are generated by hashing the object's names, a peer's IP address or public key using a cryptographic hash functions such as SHA-1/2. FreePastry is currently available under BSD-like license. FreePastry framework supports the P2P Common API specification proposed in the paper [57].

The Common API (ref to Fig. 8.6) abstracts the design of P2P applications into three layers: tier 0, tier 1 and tier 2. Key-based routing at tier 0 represents the basic capabilities that are common to all structured overlays. The Common API specification hides the

complexity of the low level P2P protocol implementation by defining a common set of interfaces to be invoked by higher level application services. These application services can invoke standard KBR procedures independent of the actual implementation. In other words, a KBR implemented using the Chord, Pastry or CAN will not make any difference to the operation of the higher level application service. Tier 1 abstracts more higher level services built upon the basic KBR or structured overlays. Examples include DHTs, Decentralised Object Location and Routing (DOLR), and group anycast/multicast (CAST). Application services at tier 3 such as CFS, PAST, Scribe can utilise one or more of the abstractions provided by tier 2.

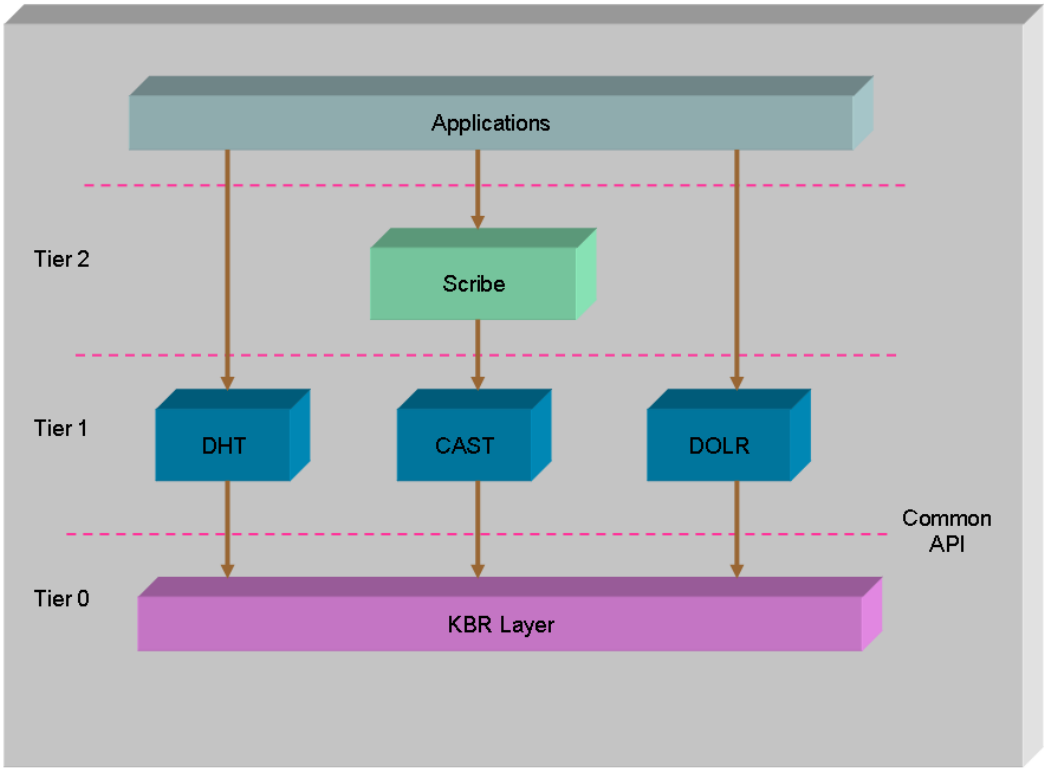


Figure 8.6: Structured P2P systems' CommonAPI architecture.

8.5 Deployment and Bootstrap

8.5.1 Manager Container

The ManagerContainer Class loader is responsible for instantiating the classes that implement the GFA functionality (such as the GRM, LRMS, Grid Peer, and Alchemi Executors) in the Alchemi-Federation system. Additionally, Manager Container additionally initialises the Publish/Subscribe Index web service. The Index service initialisation process includes: (i) booting the node hosting the index service into the existing Pastry overlay if one exists, otherwise start a new overlay; (ii) if this is the first node in the overlay then also compute the division of the logical index space at the fmin level else send a node.join(keys) message to the overlay to undertake the ownership of Index keys. Note that FreePastry takes care of the tasks related to routing table, leaf set and neighbour set maintenance. Our Application service is only concerned with coordinating proper distribution and migration of logical Index keys.

8.5.2 Tomcat Container

Tomcat servlet container hosts the Publish/Subscribe Index service. It exposes an API called TriggerService (int PortName, String BootStrapServerName, int BootStrapPort) to the ManagerContainer service for invoking the Index service. The values for API call parameters PortName, BootStrapServerName and BootSTrapPort are maintained in a configuration file accessible only to the ManagerContainer. Other APIs that Tomcat container exposes include SubmitRLQ(String Object) for submitting RUQs, SubmitRUQ(String Object) for submitting RUQs and SubmitURLQ(String Object) for unsubscribing from the Index service once an application has been successfully scheduled. These methods are invoked by the Grid peer service whenever an application is submitted to the GRM for scheduling consideration.

8.6 Performance Evaluation

In this section, we evaluate the performance of the software system in a resource sharing network that consisted of federation of 5 Alchemi desktop grids as shown in the Fig. 8.7. These desktop grids were created in three different laboratories (labs) including Microsoft .Net Lab, Masters student Lab 1 and 2 within the Computer Science and Software Engineering Department at the University of Melbourne. The machines in these Labs are connected through a Local Area Network (LAN). The LAN environment has a data transfer capability of 100 MB/sec (megabits per second). Ethernet switches of these Labs inter-connect through the firewall router. Various system parameters were configured as follows:

- Pastry network configuration: Both Grid peer nodeIds and publish/subscribe object IDs were randomly assigned and uniformly distributed in the 160-bit Pastry identifier space. Other network parameters were configured to the default values as given in the file *freepastry.params*. This file is provided with the FreePastry distribution.

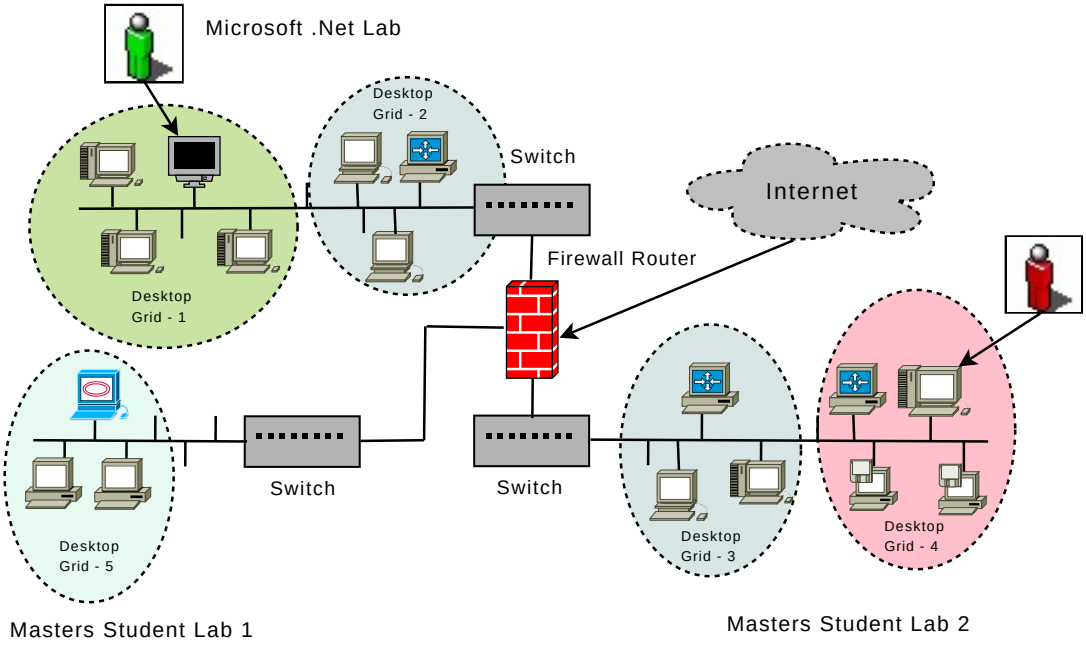
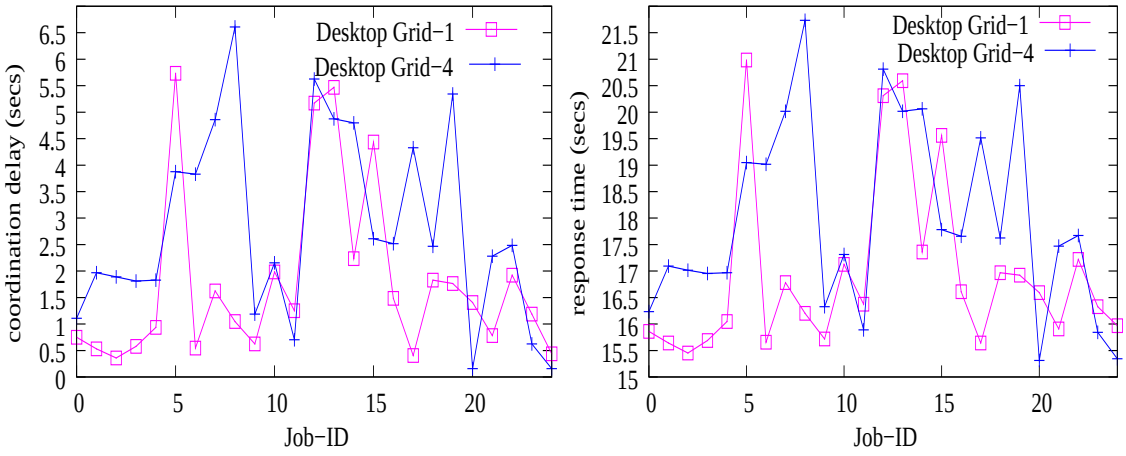


Figure 8.7: Alchemi-Federation testbed setup.

- Resource Configuration: Every GFA/Cluster was configured to connect to different numbers of executors (refer to Fig. 8.7). The Alchemi manager periodically reports

the resource status/configuration to the GFA as given by the resource ticket publish interval. The clusters running the GFA component had Windows XP as the operating system running on Intel chips. The processors were allocated to the jobs in the space-shared mode.

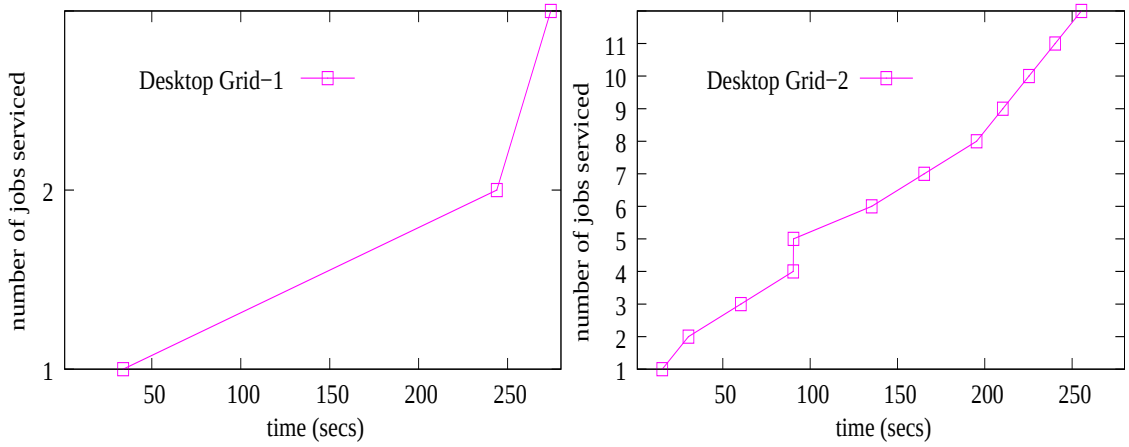
- Publish/Subscribe index space configuration: The minimum division f_{min} of logical d-dimensional publish/subscribe index was set to 2, while the maximum height of the index tree, f_{max} was constrained to 5. The index space had provision for publishing resource information in 4-dimensions including number of processors, p_i their speed, μ_i, operating system type, ϕ_i, and processor architecture, x_i. This index configuration resulted into 16 (2^4) Grid index cells at f_{min} level. On an average, 3 index cells are hashed to a Grid peer in a network of 5 Alchemi sites.



(a) Job-ID vs. average coordination delay (secs) (b) Job-ID vs. average response time (secs)

Figure 8.8: Job perspective.

- Workload configuration: The test application was a Windows executable (source code written using C-sharp) that computed whether a given number is prime or not. In order to introduce processing delays, the process was made to sleep for 10 seconds before it could proceed to check the prime condition. A simple brute force algorithm was implemented to check the prime condition for a number. The brute force algorithm consists of dividing the number by every possible divisor, up to the number. If exactly 2 factors are found, the number is prime. However, if more than 2 factors are found, then the number is not prime (it is composite).



(a) Time (secs) vs. no. of jobs serviced

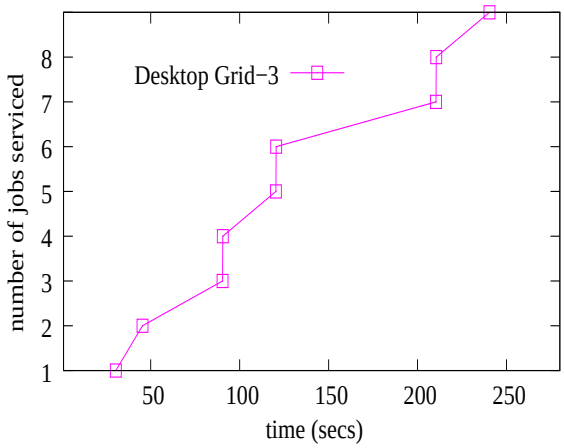
(b) Job-ID vs. no. of jobs serviced

Figure 8.9: Resource perspective.

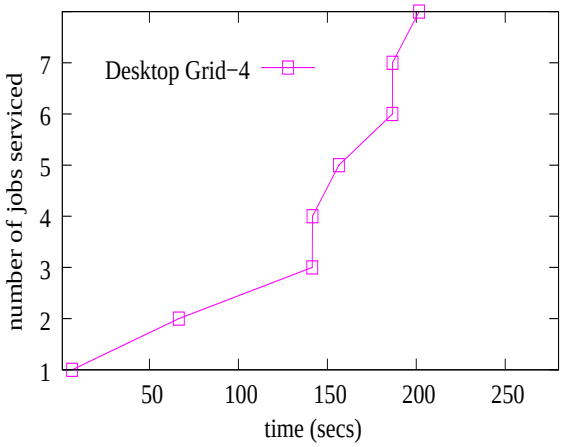
- Resource claim and resource ticket injection rate: The GFAs inject resource claim and resource ticket objects based on the exponential inter-arrival time distribution (recall from the last chapter). The value for resource claim inter-arrival delay ($\frac{1}{\lambda_l^{in}}$) was fixed to 10 secs. While the inter-arrival delay ($\frac{1}{\lambda_u^{in}}$) of resource claim object was fixed to 15 secs. The inter-arrival delay in ticket injection is considered the same for all the GFAs/Grids in the system. We configured 2 GFAs/Grids (Desktop Grid-1 and Desktop Grid-4) to insert resource claim objects into system with the delays as described. The users in Desktop Grids 1 and 2 submit 25 resource claim objects over the experiment run at an exponential inter-arrival delay. While the injection of resource ticket object is done by all the GFAs/Grids in the Alchemi-Federation system.

8.6.1 Discussion

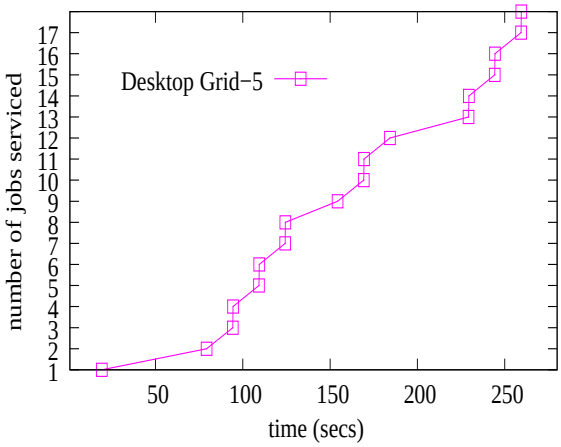
This experiment measures the performance of the software system with respect to the following metrics: average coordination delay and average response time. Recall from the last chapter, the performance metric coordination delay sums up the latencies for: (i) resource claim to reach the index cell; (ii) waiting time untill a resource ticket matches the claim; and (iii) notification delay from coordination service to the relevant GFA. While the average response time for a job is the delay between the submission time and arrival



(a) Time (secs) vs. no. of jobs serviced

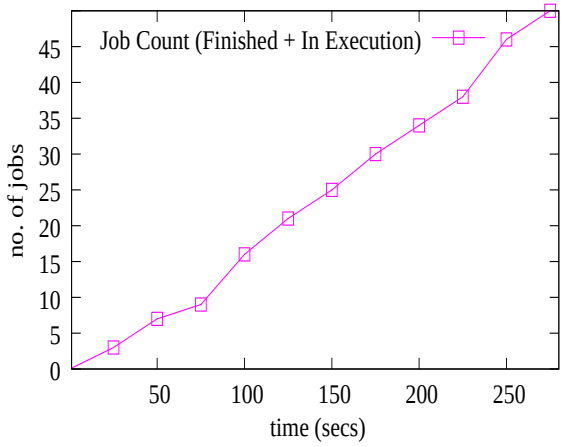


(b) Job-ID vs. no. of jobs serviced



(c) Job-ID vs. no. of jobs serviced

Figure 8.10: Resource perspective.



(a) No. of jobs vs. time (secs)

Figure 8.11: Resource perspective.

of the execution output.

Fig. 8.8(a) depicts the results for the average coordination delay in seconds for each job submitted during the experiment period. We observed that jobs across the Desktop Grids 1 and 2 experienced varying degree of coordination delay. As described earlier, the coordination delay directly affects the overall response time for jobs which is evident from the Fig. 8.8(b).

Fig. 8.9 and 8.10 show how the job load was distributed over the Alchemi grids. We observed that Desktop Grid 1 executed the least number of jobs (refer to Fig. 8.9(a)) i.e. 3 jobs, while Desktop Grid 5, located in Lab 3, executed the highest number of jobs (refer to Fig. 8.10(c)) i.e. 18 jobs over the experiment run. Overall, the resources performed reasonably well as it is seen in Fig. 8.9 and Fig. 8.10. In Fig. 8.11, we show the details on number of jobs finished and under execution across the Alchemi-Federation over the experiment run time.

8.7 Conclusion

In this chapter, we have described the Alchemi-Federation software system, which serves as a proof of the concepts that have been proposed in this thesis. We have strictly followed an Object Oriented Design (OOD) methodology in architecting and implementing the Alchemi-Federation system. Our existing Alchemi-Federation testbed consisted of Alchemi Grids distributed over three different laboratories of the department. These laboratories are protected from malicious users by a firewall router that inhibits any connection from or to the machines that do not belong to the domain. In future work, we intend to overcome this limitation of the Alchemi GFA service by implementing a cross-firewall communication capability. Such extension to the Alchemi GFA will support the creation of Internet-based federation of Alchemi Grids that belong to different firewall controlled domains.

Our software platform can be utilised to develop other distributed applications such as P2P auctions and distributed storage frameworks. Currently, our platform provides services for aggregating distributed computational resources. We also intend to study the query load-imbalance issues at the peers in a Windows-based Grid computing environ-

ment where the resource attribute distribution tends to be skewed. In future work, we intend to incorporate decentralised reputation management frameworks such as PeerReview and JXTA Poblano in the Alchemi-Federation system. These reputation management systems will aid in facilitating a secure and trustworthy system for the participants to interact. Further, we are also considering integrating the PeerMint credit management system. PeerMint is a decentralised credit management application that has been developed using the FreePastry platform.

Chapter 9

Conclusion and Future Directions

9.1 Summary

Grids have evolved as the next generation computing platform for hosting distributed services for computing, content management (including replication and distribution) and data sharing. Grid environment includes the resources that have varying types and capabilities, are topologically and geographically isolated and are under control of separate administrative policies. Federated grids (also known as decentralised or hierarchical grids) constitute a novel and emerging research area. Federated grids aim toward coupling of distributed resources as part of single resource sharing environment. In this thesis, I proposed a new federated Grid system, called Grid-Federation. The Grid-Federation resource sharing model aims toward decentralised and coordinated coupling of distributed Grid resources as a part of single cooperative system.

The P2P network model forms the basis for the design of decentralised protocols for scheduling and resource discovery in the Grid-Federation. The decentralised organisation of the system gives the provider more autonomy and additionally makes the system highly scalable. Two levels of decentralised coordination is presented in this thesis: (i) an SLA based broker-to-broker coordination protocol that inhibits the brokers from over-provisioning the resources and also gives every site the admission control capability; and (ii) a P2P tuple space based coordination protocol that coordinates the scheduling process among the distributed resource brokers. I proposed a novel model for designing decentralised, coordinated and scalable Grid resource management system. I have shown

through extensive simulation based studies that the proposed techniques are favourable for building next generation Grid systems.

9.2 Conclusion

To support the thesis that Grid-Federation model along with its decentralised protocols for scheduling and coordination is better than existing techniques for implementing new generation Grid resource sharing system I have:

1. Outlined Key Taxonomies Related to Designing a Decentralised Grid Resource Sharing System

   Comprehensive taxonomy related to decentralised scheduling, objective function, coordination and security are presented and are later utilised for classifying the current state-of-the-art. This study contributes by providing better understanding of existing Grid scheduling systems with respect to the degree of decentralisation and coordination that they can support. Further, I have also briefly looked at the current security solutions available for building such decentralised Grid systems.

   Further, I also presented a comprehensive study on the current state of the art in P2P-based Grid resource discovery. The main contribution of this study is a survey and classification of P2P-based resource discovery mechanism in a Grid computing system. Existing approaches to tag the DHTs with Grid resource information was discussed and classified based on the presented taxonomy. This study also provides a qualitative comparison of the existing DHT based d-dimensional indices with respect to scalability and load-balancing. The presented comparison can be utilised by the Grid system developers with respect to the kind of indexing system they should follow.

2. Proposed, Modeled and Evaluated a Decentralised Resource Sharing System called Grid-Federation

Grid-Federation aims toward policy based cooperative and coordinated coupling of distributed cluster resources. Computational economy metaphor was utilised for driving the application scheduling and resource allocation within the Grid-Federation. The cost-time scheduling algorithm was applied to simulate the scheduling of jobs using iterative resource queries to the decentralised federation directory. The results showed that:

- While the users from popular resources (fast/cheap) have increased competition and therefore a harder time to satisfy their QoS demands, in general the system provides an increased ability to satisfy QoS demands of all the users in the federation,
- the resource supply and demand patterns affect resource provider's overall incentive in a computational economy based resource sharing system,
- if all users seek either time/cost optimisation (non-uniform demand) then the slowest/most expensive resource owners will not benefit as much. However, if there is uniform distribution of users some seeking time and some seeking cost optimisation then all resource providers gain some benefit from the federation,
- the cost-time scheduling heuristic does not lead to excessive scheduling messages, i.e., to excessive directory accesses and we expect the system to be scalable.

3. Proposed, Modeled and Evaluated an SLA-based GFA-to-GFA Service Contract Negotiation Protocol

Following this, a novel SLA-based GFA-to-GFA SLA contract negotiation protocol was proposed. The well-known agent coordination protocol, called contract-net formed the basis for distributed SLA-based negotiations. The proposed approach modeled a set of resource providers as a contract net while job superschedulers/brokers as the managers, responsible for negotiating SLA contracts and job superscheduling in the net. Superschedulers bid for SLA contracts on the net with a focus on completing the job within the user specified deadline. We analyzed how the

varying degree of SLA bidding time (i.e., admission control decision making time for LRMSes) affects the resource providers' payoff function. The results showed that:

- The proposed approach gives resource owners finer control over resource allocation decisions. However, it also indicated that the proposed approach has a degrading effect on the user's QoS satisfaction mainly due to the time overhead incurred as a result of dynamic bidding,
- SLA contract negotiation protocol based on dynamic bidding in the net is scalable with respect to total scheduling message generated in the system. In general, the proposed approach does not incur excessive messages on a per job basis as compared to the traditional FCFS case.

4. Proposed, Modeled and Evaluated Decentralised Resource Discovery in the Grid-Federation

   I presented a decentralised solution for scalable and robust resource discovery in the Grid-Federation. The resource discovery service utilised a P2P spatial publish/subscribe index for organising the d-dimensional Grid resource data. I analysed through trace driven simulation study how the query arrival rate and Grid system size affects the system performance. The experiment result showed that:

   - The resource query rate i.e. RLQ and RUQ rate directly affects the performance of the decentralised resource discovery system. At higher rates, Grid resource queries can experience considerable latencies,
   - contrary to what one may expect, the Grid system size does not have a significant impact on the performance of the resource discovery system, in particular the query latency and the number of message routing hops.

5. Proposed, Modeled and Evaluated a DHT-based Tuple Space for Decentralised Coordination

Further, the resource discovery system was extended to provide the abstraction/facility of a P2P tuple space for realising a decentralised coordination network. The P2P tuple space can transparently support a decentralised coordination network for distributed application services such as brokers, agents, auctioneers etc. The P2P tuple space provides a global virtual shared space that can be concurrently and associatively accessed by all participants in the system and the access is independent of the actual physical or topological proximity of the tuples or hosts. The effectiveness of the P2P tuple space in coordinating resource provisioning was analysed using extensive simulation study. The results showed that:

- The job inter-arrival delay has significant impact on the coordination overhead for the jobs in the decentralised network. At lower inter-arrival delays, competing requests have to wait for longer time before the notification arrives from the coordination network,
- the performance of coordination network worsens further for highly-constrained workloads as compared to lightly-constrained workloads. However, if there is abundant supply of resources in the system then the system behaviour is independent of workload type.

6. Designed and Implemented the Alchemi-Federation Software System

Finally, the Grid-Federation model is realised using the Alchemi desktop Grid computing system, which I refer to as Alchemi-Federation. The software system serves as a proof of the concepts/models/protocols that have been proposed in this thesis. I have strictly followed an OOD methodology in architecting and implementing the Alchemi-Federation system. Our software platform can be utilised to develop other distributed application systems such as P2P cooperative auction environment and distributed storage framework. Currently, our platform provides services for aggregating the distributed computational resources.

This thesis makes significant contribution towards providing a new model for decentralised Grid resource management, in particular the decentralised protocols for

application scheduling, resource discovery and coordination. The proposed techniques are clearly favourable as compared to the existing literature available for the Grid resource management system design. I have also incorporated the proposed ideas in the software system, Alchemi-Federation. The thesis clearly makes advancement over the current state of the art by proposing novel decentralised protocols and models.

9.3 Open Issues

The current models of distributed systems, including Grid computing and P2P computing, suffer from a knowledge and resource fragmentation problem. By knowledge fragmentation, it means that various research groups in both academia and industry work in an independent manner. They define standards without any proper coordination. They give very little attention to the inter-operational ability between the related systems. Such disparity can be seen in the operation of various Grid systems including Condor-G, Nimrod-G, OurGrid, Grid-Federation, Tycoon and Bellagio. These systems define independent interfaces, different job description languages, communication protocols, superscheduling and resource allocation methodologies. In this case, users have an access only to those resources that can understand the underlying Grid system protocol. Hence, this leads to the distributed resource fragmentation problem.

A possible solution to this can be federating these Grid systems based on universally agreed standards (similar to the TCP/IP model that governs the current Internet). The core to the operation and inter-operational ability of Internet component is the common resource indexing system, i.e., DNS. Both the Grid and P2P communities clearly lack any such global or widely accepted service. These systems do not expose any API or interfaces that can help them to inter-operate. In recent times, we have seen some efforts towards developing a generic Grid service-oriented architecture, more commonly referred to as the Open Grid Service Architecture (OGSA). Core Grid developers also define common standards through the GGF. Web Service Resource Framework (WSRF) defines a new set of specifications for realising the OGSA vision of grid and web services. WSRF can overcome the cross-platform inter-operational ability issues in Grid comput-

ing. However, still it cannot glue the gaps between various Grid systems because of the basic differences in interfaces, communication protocols, superscheduling and resource allocation methodologies.

Possible solutions to overcome knowledge and resource fragmentation problem include: (i) availability of a robust, distributed, scalable resource indexing/organisation system; (ii) evolution of common standards for resource allocation and application superscheduling; (iii) agreement on using common middleware for managing Grid resources such as clusters, SMPs etc; and (iv) defining common interfaces and APIs that can help different related system to inter-operate and coordinate activities.

9.4 Future Directions

This thesis introduces novel protocols to build scalable, robust and decentralised Grid resource management system. It demonstrated the benefits of proposed decentralised protocols in terms of user's QoS satisfaction, provider's profit function, scalability, robustness and coordination. Overall, the proposed models and protocols lay the foundation for architecting next generation Grid and P2P systems.

9.4.1 Coordinated Co-allocation Framework for Synchronous Parallel Applications

Synchronous parallel applications (such as MPI-CH) refer to the class of applications that have run time dependencies and need to do frequent message passing. Traditionally, these kind of applications have been designed for tightly coupled environments such as shared memory processors and computational clusters (that have high bandwidth interconnection network). Given the complexity of Grid environment, including its dynamism, scale, heterogeneity in resource type, storage capability, management policies and communication bandwidth, the efficient execution and adaptation of synchronous parallel applications [112] present significant challenges with respect to scheduling and resource management.

Efficient execution of such class of applications in a Grid environment requires mech-

anisms for coordinated co-allocation across resources having sufficient computational and network bandwidth capabilities. In general, co-allocation across Grid resources belonging to multiple control domain is a complex problem in itself. In this case, the decentralised coordination mechanism proposed in this thesis can be utilised to transparently and dynamically formulate efficient co-allocated schedules. Further, the decentralised co-allocation algorithm can also benefit from the SLA-based protocols proposed in this thesis with respect to guaranteed service delivery and prioritised execution.

9.4.2 Decentralised Storage Management and Replication Framework

With evolutionary growth in the Internet users and commodity computers (such as desktop machines), it is feasible to maintain a pervasive and on-demand content management and replication network. The search engine Google, performs centralized content replication at its data centers to increase data availability, fault-tolerance and performance. In contrast, the proposed decentralised Grid-Federation model can be extended to support an Internet-scale content management and replication network that leverages the storage capability of commodity machines. Coupled with the recent advancement in P2P reputation and trust management schemes such as PeerReview and Poblano, the proposed system can deliver the guaranteed storage services. Next, the Alchemi-Federation software system can be easily extended to support the storage management and replication functionality.

9.4.3 Cooperative Multiple e-Science Workflow Scheduling Framework

In e-Science Grid computing environments, workflow management systems allocate tasks to the resources after negotiating the SLA contracts with every resource provider that executes one or more tasks. Due to this, the service providers need to allocate resources based on negotiated QoS parameters and manage various competing demands from other users. However, the competition among multiple workflow systems may lead to degraded QoS satisfaction for certain users. This limitation can easily be eradicated by utilising the decentralised coordination among e-Science workflow systems. In this case, the conflicting

requests of the workflow systems can be resolved by the decentralised coordinator leading to enhanced QoS satisfaction across the system. To an extent, data-intensive workflow scheduling also needs to consider co-allocation issues similar to synchronous parallel applications.

9.4.4 Cooperative P2P Auction Network

Proposed decentralised protocols for resource discovery and coordination can be extended to support a P2P auction market place [92]. The capability of the resource discovery service to efficiently handle d-dimensional range queries can be utilised to discover available auctions in the system. The auctioneer can advertise their items, auction types, and pricing information through the RUQ, while the interested buyers can subscribe for the auctioned items through the RLQ. In case the match occurs, the interested buyers can directly bid at the auctioneer. P2P reputation management systems such as PeerReview and Poblano can be utilised to establish the credibility of the participants in the system.

9.4.5 P2P Relational Database Management System (RDBMS)

Realising an efficient, scalable and robust P2P RDBMS is an interesting future research problem. Fundamental to P2P RDBMS is the development of distributed algorithms for: (i) query processing; (ii) data consistency, and integrity; and (iii) transaction atomicity, durability, and isolation. First step in designing a P2P RDBMS is to partition the relational tuple space across a set of distributed nodes in the system. The data partition strategy should be such that the query workload is uniformly distributed while efficiently utilising the node's computational and network bandwidth capability. The discovery and partitioning of tuple space across the nodes in the system for hosting the tuple space can be facilitated through the decentralised protocols proposed in this thesis.

REFERENCES

[1] Abawajy, J. H. and Dandamudi, S. P. (2000). Distributed hierarchical workstation cluster co-ordination scheme. In *PARELEC '00: Proceedings of the International Conference on Parallel Computing in Electrical Engineering*, page 111. IEEE Computer Society, Los Alamitos, CA, USA.

[2] Abawajy, J. H. and Dandamudi, S. P. (2003). Parallel job scheduling on multicluster computing systems. In *Cluster'03: Proceedings of the 5th IEEE International Conference on Cluster Computing, Hong Kong*, pages 11–18. IEEE Computer Society, Los Alamitos, CA, USA.

[3] Abramson, D., Buyya, R., and Giddy, J. (2002). A computational economy for grid computing and its implementation in the Nimrod-G resource broker. *Future Generation Computer Systems (FGCS) Journal, Volume 18, Issue 8, pages 1061-1074, Elsevier Science, The Netherlands, October*.

[4] Akram, A., Allan, R., and Rana, O. (2005). Virtual communities and community coordinator. In *2005 International Conference on Semantics, Knowledge and Grid (SKG 2005), 27-29 November 2005, Beijing, China*, page 110. IEEE Computer Society, Los Alamitos, CA, USA.

[5] Al-Ali, R., Hafid, A., Rana, O., and Walker, D. (2004). An approach for quality of service adaptation in service-oriented grids: Research articles. *Concurrency and Computation : Practice and Experience, Volume 16, Issue 5, pages 401-412, John Wiley and Sons Ltd., Chichester, UK*.

[6] Alexander, B. and Buyya, R. (2003). Gridbank: A grid accounting services architecture for distributed systems sharing and integration. In *Workshop on Internet Computing and E-Commerce, Proceedings of the 17th Annual International Parallel and Distributed Processing Symposium (IPDPS 2003), April 22-26*. IEEE Computer Society, Los Alamitos, CA, USA.

[7] Allen, A. O. (1978). *Probability, Statistics and Queuing Theory with computer science applications*. Academic Press, INC.

[8] Andrade, N., Brasileiro, F., Cirne, W., and Mowbray, M. (2004). Discouraging free riding in a peer-to-peer cpu-sharing grid. In *HPDC'13: Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing*. IEEE Computer Society, Los Alamitos, CA, USA.

[9] Andrade, N., Cirne, W., Brasileiro, F., and Roisenberg, P. (2003). OurGrid: An approach to easily assemble grids with equitable resource sharing. In *JSSPP'03: Proceedings of the 9th Workshop on Job Scheduling Strategies for Parallel Processing*. LNCS, Springer, Berlin/Heidelberg, Germany.

[10] Andrzejak, A. and Xu, Z. (2002). Scalable, efficient range queries for grid information services. In *P2P'02: Second IEEE International Conference on Peer-to-Peer Computing*. IEEE Computer Society, Los Alamitos, CA, USA.

[11] Asano, T., Ranjan, D., Roos, T., Welzl, E., and Widmayer, P. (1997). Space-filling curves and their use in the design of geometric data structures. *Theoretical Computer Science, Volume 181, Issue 1, pages 3–15, Elsevier Science Publishers Ltd., Essex, UK*.

[12] Aspnes, J. and Shah, G. (2003). Skip Graphs. In *SODA '03: Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms, Baltimore, Maryland*, pages 384–393. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.

[13] Auyoung, A., Chun, B., Snoeren, A., and Vahdat, A. (2004). Resource allocation in federated distributed computing infrastructures. In *OASIS '04: 1st Workshop on Operating System and Architectural Support for the On-demand IT Infrastructure, Boston, MA, October*.

[14] Balakrishnan, H., Kaashoek, M. F., Karger, D., Morris, R., and Stoica, I. (2003). Looking up data in p2p systems. *Communications of the ACM, Volume 46, Issue 2, pages 43–48, ACM Press, New York, NY, USA*.

[15] Balazinska, M., Balakrishnan, H., and Karger, D. (2002). INS/Twine: A scalable peer-to-peer architecture for intentional resource discovery. In *Pervasive '02: Proceedings of the First International Conference on Pervasive Computing*, pages 195–210. Springer-Verlag, Germany.

[16] Barak, A., Shiloh, A., and Amar, L. (2005). An organizational grid of federated mosix clusters. *Proceedings of the 5th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'05)*.

[17] Barbara, D. (1999). Mobile computing and databases-a survey. *IEEE Transactions on Knowledge and Data Engineering, Volume 11, Issue 1, pages 108–117, IEEE Educational Activities Department, Piscataway, NJ, USA*.

[18] Beckmann, N., Kriegel, H., Schneider, R., and Seeger, B. (1990). The r*-tree: an efficient and robust access method for points and rectangles. In *SIGMOD '90: Proceedings of the 1990 ACM SIGMOD international conference on Management of data, Atlantic City, New Jersey, United States*, pages 322–331. ACM Press, New York, NY, USA.

[19] Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. *Communications of the ACM, Volume 18, Issue 9, pages 509–517, ACM Press, New York, NY, USA*.

[20] Berchtold, S., Bohm, C., and Kriegal, H.-P. (1998). The pyramid-technique: towards breaking the curse of dimensionality. In *SIGMOD '98: Proceedings of the 1998 ACM SIGMOD international conference on Management of data, Seattle, Washington, United States*, pages 142–153. ACM Press, New York, NY, USA.

[21] Berger, M. J. and Bokhari, S. H. (1987). A partitioning strategy for non-uniform problems on multiprocessors.

[22] Berman, F. and Wolski, R. (1997). The apples project: A status report. *Proceedings of the 8th NEC Research Symposium, Berlin, Germany*.

[23] Berman, K. A. and Paul, J. L. (1997). *Fundamentals of Sequential and Parallel Algorithms*. PWS Publishing Company.

[24] Bharambe, A. R., Agrawal, M., and Seshan, S. (2004). Mercury: supporting scalable multi-attribute range queries. In *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications, Portland, Oregon, USA*, pages 353–366. ACM Press, New York, NY, USA.

[25] Bienkowski, M., Korzeniowski, M., and auf der Heide, F. M. (2005). Dynamic load balancing in distributed hash tables. In *4th International Workshop on Peer-to-Peer Systems (IPTPS '05)*, pages 217–225.

[26] Bode, B., Halstead, D., Kendall, R., and Jackson, D. (2000). PBS: The portable batch scheduler and the maui scheduler on linux clusters. *Proceedings of the 4th Linux Showcase and Conference, Atlanta, GA, USENIX Press, Berkley, CA, October*.

[27] Bredin, J., Maheswaran, R. T., c. Imer, Basar, T., Kotz, D., and Rus, D. (2000). A game-theoretic formulation of multi-agent resource allocation. In *AGENTS '00: Proceedings of the fourth international conference on Autonomous agents*, pages 349–356. ACM Press, New York, NY, USA.

[28] Busi, N., Manfredini, C., Montresor, A., and Zavattaro, G. (2003). Peerspaces: data-driven coordination in peer-to-peer networks. In *SAC '03: Proceedings of the 2003 ACM symposium on Applied computing, Melbourne, Florida*, pages 380–386. ACM Press, New York, NY, USA.

[29] Butt, A. R., Zhang, R., and Hu, Y. C. (2003). A self-organizng flock of condors. In *SC '03: Proceedings of the 2003 ACM/IEEE conference on Supercomputing*. IEEE Computer Society, Los Alamitos, CA, USA.

[30] Buyya, R., Abramson, D., and Giddy, J. (2000). Nimrod/g: An architecture for a resource management and scheduling system in a global computational grid. In *The Fourth International Conference on High-Performance Computing in the Asia-Pacific Region*, page 283. IEEE Computer Society, Los Alamitos, CA, USA.

[31] Buyya, R., Abramson, D., Giddy, J., and Stockinger, H. (2002). Economic models for resource management and scheduling in grid computing. *Special Issue on Grid computing Environment, The Journal of concurrency and Computation:Practice and Experience (CCPE), Volume 14, Issue 13-15, Wiley Press*.

[32] Buyya, R. and Murshed, M. (2002). Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *Concurrency and Computation: Practice and Experience (CCPE), Volume 14, Issue 13-15, pages 1175-1220, Wiley Press*.

[33] Cai, M., Frank, M., Chen, J., and Szekely, P. (2003). Maan: A Multi-attribute addressable network for grid information services. *Proceedings of the Fourth IEEE/ACM International workshop on Grid Computing*, pages 184 – 191.

[34] Calvert, K., Doar, M., and Zegura, E. W. (June 1997). Modeling internet topology. In *IEEE Communications Magazine*. IEEE Computer Society, Los Alamitos, CA, USA.

[35] Casanova, H. and Dongara, J. (1997). Netsolve: A network server solving computational science problem. *International Journal of Supercomputing Applications and High Performance Computing, Volume 11, Issue 3, pages 212-223*.

[36] Castro, M., Costa, M., and Rowstron, A. (2004). Should we build gnutella on a structured overlay? *SIGCOMM Computing Communication Reviews, Volume 34, Issue 1, pages 131–136, ACM Press, New York, NY, USA*.

[37] Castro, M., Druschel, P., Ganesh, A., Rowstron, A., and Wallach, D. S. (2002a). Secure routing for structured peer-to-peer overlay networks. *SIGOPS Operating System Review, ACM Press, New York, NY, USA*, 36(SI):299–314.

[38] Castro, M., Druschel, P., Kermarrec, A., and Rowstron, A. (2002b). Scribe: a large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications, Volume 20, Issue 8, pages 1489–1499, IEEE Computer Society, Los Alamitos, CA, USA*.

[39] Chapin, S., Karpovich, J., and Grimshaw, A. (1999). The legion resource management system. *Proceedings of the 5th Workshop on Job Scheduling Strategies for Parallel Processing, San Juan, Puerto Rico, 16 April, Springer:Berlin*.

[40] Chase, J., Grit, L., Irwin, D., Moore, J., and Sprenkle, S. (2003). Dynamic virtual clusters in a grid site manager. *In the Twelfth International Symposium on High Performance Distributed Computing (HPDC-12), June*.

[41] Chawathe, Y., Ratnasamy, S., Breslau, L., Lanham, N., and Shenker, S. (2003). Making gnutella-like p2p systems scalable. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 407–418. ACM Press, New York, NY, USA.

[42] Cheema, A. S., Muhammad, M., and Gupta, I. (2005). Peer-to-peer discovery of computational resources for grid applications. In *Grid'05: 6th IEEE/ACM International Workshop on Grid Computing*. IEEE Computer Society, Los Alamitos, CA, USA.

[43] Cheng, J. Q. and Wellman, M. P. (Aug 1998). The WALRAS Algorithm: A convergent distributed implementation of general equilibrium outcomes. In *Computational Economics, Volume 12, Issue 1*, pages 1 – 24.

[44] Chou, W. (2002). Inside ssl: Accelerating secure transactions. *IT Professional, IEEE Educational Activities Department, Piscataway, NJ, USA*, 4(5):37–41.

[45] Chown, P. (2002). Advanced encryption standard (aes) ciphersuites for transport layer security (tls).

[46] Chun, B. and Culler, D. (2000). A decentralized, secure remote execution environment for clusters. *Proceedings of the 4th Workshop on Communication, Architecture and Applications for Network-based Parallel Computing, Toulouse, France*.

[47] Chun, B., Culler, D., Roscoe, T., Bavier, A., Peterson, L., Wawrzoniak, M., and Bowman, M. (2003). Planetlab: an overlay testbed for broad-coverage services. *SIGCOMM Computer Communication Reviews, Volume 33, Issue 3, pages 3–12, ACM Press, New York, NY, USA*.

[48] Chun, B., Ng, C., Albrecht, J., Parkes, D., and Vahdat, A. (2004). SHARE: Computational resource exchanges for distributed resource allocation.

[49] Clarke, I., Sandberg, O., Wiley, B., and Hong, T. W. (2001). Freenet: a distributed anonymous information storage and retrieval system. In *International workshop on Designing privacy enhancing technologies*, pages 46–66. Springer-Verlag, Germany.

[50] Cormen, T., Leiserson, C., Rivest, R., and Stein, C. (2001). *Introduction to Algorithms, Second Edition*. The MIT Press.

[51] Courcoubetis, C. and Siris, V. (1999). Managing and pricing service level agreements for differentiated services. In *Proc. of 6th IEEE/IFIP International Conference of Quality of Service (IWQoS'99), London, UK, May-June*.

[52] Crainiceanu, A., Linga, P., Gehrke, J., and Shanmugasundaram, J. (2004). Querying peer-to-peer networks using p-trees. In *WebDB '04: Proceedings of the 7th International Workshop on the Web and Databases*, pages 25–30. ACM Press, New York, NY, USA.

[53] Cugola, G. and Picco, G. (2001). PeerWare: Core middleware support for peer-to-peer and mobile systems. *Technical Report, Politecnico Di Milano*.

[54] Czajkowski, K., Fitzgerald, S., Foster, I., and Kesselman, C. (2001). Grid information services for distributed resource sharing. In *HPDC '01: Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10)*, page 181. IEEE Computer Society, Los Alamitos, CA, USA.

[55] Czajkowski, K., Foster, I., and Kesselman, C. (2005). Agreement-based resource management. In *Proceedings of the IEEE, Volume 93, Issue 3*. IEEE Computer Society, Los Alamitos, CA, USA.

[56] Dabek, F., Kaashoek, M. F., Karger, D., Morris, R., and Stoica, I. (2001). Wide-area cooperative storage with cfs. In *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles, Banff, Alberta, Canada*, pages 202–215. ACM Press, New York, NY, USA.

[57] Dabek, F., Zhao, B., Druschel, P., Kubiatowicz, J., and Stoica, I. (2003). Towards a common api for structured peer-to-peer overlays. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS03)*, Berkeley, CA.

[58] Dash, R. K., Jennings, N. R., and Parkes, D. C. (2003). Computational-mechanism design: A call to arms. *IEEE Intelligent Systems*, pages 40–47. Special Issue on Agents and Markets.

[59] Druschel, P. and Rowstron, A. (2001). Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility. In *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles, Banff, Alberta, Canada*, pages 188–201. ACM Press, New York, NY, USA.

[60] Durschel, P. (2006). The Renaissance of Decentralized Systems, Keynote talk at the 15th IEEE International Symposium on High Performance Distributed Computing, Paris, France.

[61] Epema, D. H. J., Livny, M., van Dantzig, R., Evers, X., and Pruyne, J. (1996). A worldwide flock of condors: load sharing among workstation clusters. *Future Generation Computer Systems, Volume 12, Issue 1, pages 53-65, Elsevier Science Publishers B. V., Amsterdam, The Netherlands*.

[62] Eugster, P. T., Felber, P. A., Guerraoui, R., and Kermarrec, A.-M. (2003). The many faces of publish/subscribe. *ACM Computing Surveys, Volume 35, Issue 2, pages 114–131, ACM Press, New York, NY, USA*.

[63] Eymann, T., Reinicke, M., Ardaiz, O., Artigas, P., de Cerio, L. D., Freitag, F., Messeguer, R., Navarro, L., Royo, D., and Sanjeevani, K. (2004). Decentralized vs. centralized economic coordination of resource allocation in grids. In *Grid Computing*, volume 2970/2004, pages 9–16. LNCS, Springer, Berlin/Heidelberg, Germany.

[64] Feigenbaum, J. and Shenker, S. (2002). Distributed algorithmic mechanism design: recent results and future directions. In *DIALM '02: Proceedings of the 6th international workshop on Discrete algorithms and methods for mobile computing and communications, Atlanta, Georgia, USA*, pages 1–13. ACM Press, New York, NY, USA.

[65] Feitelson, D. G., Rudolph, L., Schwiegelshohn, U., Sevcik, K. C., and Wong, P. (1997). Theory and practice in parallel job scheduling. In *IPPS '97: Proceedings of the Job Scheduling Strategies for Parallel Processing*, pages 1–34. Springer-Verlag, London, UK.

[66] Feitelson, D. G. and Tsafrir, D. (2006). Workload sanitation for performance evaluation. In *IEEE Intl. Symposium Performance Analysis of Systems and Software*, pages 221–230. Springer-Verlag, London, UK.

[67] Fitzgerald, S., Foster, I., Kesselman, C., von Laszewski, G., Smith, W., and Tuecke, S. (1997). A directory service for configuring high-performance distributed computations. In *Proc. 6th IEEE Symposium on High Performance Distributed Computing*, pages 365–375. IEEE Computer Society Press.

[68] Forman, G. H. and Zahorjan, J. (1994). The challenges of mobile computing. *IEEE Computer, Volume 27, Issue 4, pages 38–47, IEEE Computer Society Press, Los Alamitos, CA, USA*.

[69] Foster, I. and Iamnitchi, A. (2003). On death, taxes, and the convergence of peer-to-peer and grid computing. In *Peer-to-Peer Systems II, Second International Workshop, IPTPS 2003, Berkeley, CA, USA, February 21-22,2003, Revised Papers*, pages 118–128. Lecture Notes in Computer Science.

[70] Foster, I. and Kesselman, C. (1997). Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications, Volume 11, Issue 2, pages115-128*.

[71] Foster, I. and Kesselman, C. (1998). The grid: Blueprint for a new computing infrastructure. *Morgan Kaufmann Publishers, USA*.

[72] Foster, I., Kesselman, C., and Tuecke, S. (2001). The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of Supercomputer Applications, Volume 15, Issue 3*.

[73] Frerot, C. D., Lacroix, M., and Guyennet, H. (2000a). Federation of resource traders in objects-oriented distributed systems. *(PARELEC'00) August 27 - 30, Quebec, Canada*.

[74] Frerot, C. D., Lacroix, M., and Guyennet, H. (2000b). Resource balancing using trader federation. *(ISSC'00) july 09 - 06, Antibes, France*.

[75] Frey, J., Tannenbaum, T., Livny, M., Foster, I., and Tuecke, S. (2001). Condor-G: A computation management agent for multi-institutional grids. In *10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10 '01), 2001*, pages 237 – 246. IEEE Computer Society, Los Alamitos, CA, USA.

[76] Fu, Y., Chase, J., Chun, B., Schwab, S., and Vahdat, A. (2003). SHARP: an architecture for secure resource peering. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles, Bolton Landing, NY, USA*, pages 133–148. ACM Press, New York, NY, USA.

[77] Gaede, V. and Gunther, O. (1998). Multidimensional access methods. *ACM Computing Surveys, Volume 30, Issue 2, pages 170–231, ACM Press, New York, NY, USA*.

[78] Gambosi, G., Postiglione, A., and Talamo, M. (2001). Algorithms for the relaxed online bin-packing model. *SIAM Journal of Computing, Volume 30, Issue 5, pages 1532–1551*.

[79] Ganesan, P., Bawa, M., and Garcia-Molina, H. (2004a). Online balancing of range-partitioned data with applications to peer-to-peer systems. Technical report, Stanford U.

[80] Ganesan, P., Yang, B., and Garcia-Molina, H. (2004b). One torus to rule them all: multi-dimensional queries in p2p systems. In *WebDB '04: Proceedings of the 7th International Workshop on the Web and Databases, Paris, France*, pages 19–24. ACM Press, New York, NY, USA.

[81] Gao, J. and Steenkiste, P. (2004). An adaptive protocol for efficient support of range queries in dht-based systems. In *ICNP '04: Proceedings of the Network Protocols, 12th IEEE International Conference on (ICNP'04)*, pages 239–250. IEEE Computer Society, Los Alamitos, CA, USA.

[82] Garca, P., Pairot, C., Mondjar, R., Pujol, J., Tejedor, H., and Rallo, R. (2005). Planetsim: A new overlay network simulation framework. In *Software Engineering and Middleware, SEM 2004, Linz, Austria*, pages 123–137. Lecture Notes in Computer Science (LNCS), Springer, Germany.

[83] Gelernter, D. (1985). Generative communication in linda. *ACM Transactions on Programming Languages and Systems, Volume 7, Issue 1, pages 80–112, ACM Press, New York, NY, USA*.

[84] Gentzsch, W. (2001). Sun grid engine: Towards creating a compute power grid. In *CCGRID'01: 1st IEEE International Symposium on Cluster Computing and the Grid, Brisbane, Australia*, page 35. IEEE Computer Society, Los Alamitos, CA, USA.

[85] Gong, L. (2001). JXTA: a network programming environment. *IEEE Internet Computing, Volume 05, Issue 3, pages 88–95, IEEE Computer Society, Los Alamitos, CA, USA*.

[86] Grosu, D. and Chronopoulos, T. (2004). Algorithmic mechanism design for load balancing in distributed systems. In *IEEE Transactions on Systems Man and Cybernetics Part B*, pages 77–84. IEEE Computer Society, Los Alamitos, CA, USA.

[87] Gupta, A., Liskov, B., and Rodrigues, R. (2003a). One hop lookups for peer-to-peer overlays. In *Ninth Workshop on Hot Topics in Operating Systems (HotOS-IX), Lihue, Hawaii*, pages 7–12.

[88] Gupta, A., Sahin, O. D., Agrawal, D., and Abbadi, A. E. (2004). Meghdoot: content-based publish/subscribe over p2p networks. In *Middleware '04: Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware*, pages 254–273. Springer-Verlag New York, Inc.

[89] Gupta, I., Birman, K., Linga, P., Demers, A., and van, R. (2003b). Kelips: Building an efficient and stable p2p dht through increased memory and background overhead. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS03)*.

[90] Hand, S., Kotsovinos, T. H. E., and Pratt, I. (2003). Controlling the xenoserver open platform. In *2003 IEEE Conference on Open Architectures and Network Programming*, pages 3–11. IEEE Computer Society, Los Alamitos, CA, USA.

[91] Harrison, A. and Taylor, I. (2006). The web services resource framework in a peer-to-peer context. *Journal of Grid Computing*, 4(4):425–445.

[92] Hausheer, D. and Stiller, B. (2005). PeerMart: the technology for a distributed auction-based market for peer-to-peer services. In *ICC 2005: 2005 IEEE International Conference on Communications*, pages 1583–1587, Washington, DC, USA. IEEE Computer Society.

[93] Housley, R., Polk, W., Ford, W., and Solo, D. (2002). Internet X.509 public key infrastructure certificate and certificate revocation list (crl) profile.

[94] Howell, F. and McNab, R. (1998). Simjava: A discrete event simulation package for java applications in computer systems modeling. *First International Conference on Web-based modeling and Simulation, San Diego, CA, 1998, SCS Press: San Diego, CA*.

[95] Huebsch, R., Hellerstein, J. M., Boon, N. L., L., T., Shenker, S., and Stoica, I. (2003). Querying the internet with pier. In *Proceedings of 19th International Conference on Very Large Databases (VLDB'03), Berlin, Germany*. Morgan Kaufmann Publishers.

[96] Iamnitchi, A. and Foster, I. (2001). On fully decentralized resource discovery in grid environments. In *Grid'01:International Workshop on Grid Computing, Denver, CO*. IEEE Computer Society, Los Alamitos, CA, USA.

[97] Iamnitchi, A. and Foster, I. (2004). *A peer-to-peer approach to resource location in Grid environments*. Kluwer Academic Publishers, Norwell, MA, USA.

[98] In, J., Avery, P., Cavanaugh, R., and Ranka, S. (2004). Policy based scheduling for simple quality of service in grid computing. In *IPDPS'04: Proceedings of the 18th International Parallel and Distributed Processing Symposium*. IEEE Computer Society, Los Alamitos, CA, USA.

[99] Jagadish, H. V. (1990). Linear clustering of objects with multiple attributes. In *SIGMOD '90: Proceedings of the 1990 ACM SIGMOD international conference on Management of data, Atlantic City, New Jersey, USA*, pages 332–342. ACM Press, New York, NY, USA.

[100] Jagadish, H. V. (1997). Analysis of the hilbert curve for representing two-dimensional space. *Information Processing Letters, Volume 62, Issue 1, pages 17–22, Elsevier North-Holland Inc., Amsterdam, The Netherlands*.

[101] Jovanovic, M., Annexstein, F., and Berman, K. (2001). Scalability issues in large peer-to-peer networks-a case study of gnutella. Technical report, University of Cincinnati.

[102] Juhasz, Z. and Paul, P. (2002). Scalability analysis of the contract net protocol. In *CCGRID '02: Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*. IEEE Computer Society, Los Alamitos, CA, USA.

[103] Karger, D., Lehman, E., Leighton, T., Panigrahy, R., Levine, M., and Lewin, D. (1997). Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web. In *STOC '97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing, El Paso, Texas, United States*, pages 654–663. ACM Press, New York, NY, USA.

[104] Korn, F., Pagel, B., and Faloutsos, C. (2001). On the 'dimensionality curse' and the 'self-similarity blessing'. *IEEE Transactions on Knowledge and Data Engineering, Volume 13, Issue 1, pages 96–111, IEEE Educational Activities Department, Piscataway, NJ, USA*.

[105] Krauter, K., Buyya, R., and Maheswaran, M. (2002). A taxonomy and survey of grid resource management systems for distributed computing. *Software Practice and Experience (SPE), Volume 32, Issue 2, pages 135–164, John Wiley and Sons, Inc., New York, NY, USA*.

[106] Krych, D. (2003). Calculation and analysis of nash equilibria of vickrey-based payment rules for combinatorial exchanges. Undergraduate thesis, Harvard University.

[107] Kubiatowicz, J., Bindel, D., Chen, Y., Czerwinski, S., Eaton, P., Geels, D., Gummadi, R., Rhea, S., Weatherspoon, H., Weimer, W., Wells, C., and Zhao, B. (2000). Oceanstore: an architecture for global-scale persistent storage. *SIGPLAN Notes, Volume 35, Issue 11, pages 190–211, ACM Press, New York, NY, USA*.

[108] L. Chan and S. Karunasekera (2007). Designing Configurable Publish-Subscribe Scheme for Decentralised Overlay Networks. In *AINA'07: Proceedings of the IEEE 21st International Conference on Advanced Information Networking and Applications*. IEEE Computer Society, Los Alamitos, CA, USA.

[109] Lai, K., Huberman, B. A., and Fine, L. (2004). Tycoon: A distributed market-based resource allocation system. *Technical Report, HP Labs*.

[110] Li, J., Stribling, J., Gil, T. M., Morris, R., and Kaashoek, M. F. (2004). Comparing the performance of distributed hash tables under churn. In *Proceedings of the 3rd International Workshop on Peer-to-Peer Systems (IPTPS04)*.

[111] Li, Z. and Parashar, M. (2005). Comet: A scalable coordination space for decentralized distributed environments. In *HOT-P2P '05: Proceedings of the Second International Workshop on Hot Topics in Peer-to-Peer Systems*, pages 104–112. IEEE Computer Society, Los Alamitos, CA, USA.

[112] Li, Z. and Parashar, M. (2007). A computational infrastructure for grid-based asynchronous parallel applications. In *HPDC '07: Poster Proceedings of the 16th IEEE International Symposium on High Performance Distributed Computing (HPDC'07)*, Washington, DC, USA. IEEE Computer Society.

[113] Litzkow, J., Livny, M., and Mukta, M. W. (1988). Condor- a hunter of idle workstations. In *ICDCS'88: Proceedings of the 8th IEEE International Conference on Distributed Computing Systems*. IEEE Computer Society, Los Alamitos, CA, USA.

[114] Liu, B., Lee, W., and Lee, D. L. (2005). Supporting complex multi-dimensional queries in p2p systems. In *ICDCS'05: Proceedings of the 25th IEEE International Conference on Distributed Computing Systems, Columbus, OH, USA*, pages 155– 164. IEEE Computer Society, Los Alamitos, CA, USA.

[115] Lua, K., Crowcroft, J., Pias, M., Sharma, R., and Lim, S. (2005). A survey and comparison of peer-to-peer overlay network schemes. *IEEE Communications Surveys and Tutorials, Volume 7, Issue 2, Second Quarter, pages 72-93, IEEE Communications Society Press*.

[116] Lublin, U. and Feitelson, D. G. (2003). The workload on parallel supercomputers: modeling the characteristics of rigid jobs. *Journal of Parallel and Distributed Computing, Volume 63, Issue 11, pages 1105–1122, Academic Press, Inc., Orlando, FL, USA.*

[117] Luther, A., Buyya, R., Ranjan, R., and Venugopal, S. (2004). *Peer-to-Peer Grid Computing and a .NET-based Alchemi Framework, High Performance Computing: Paradigm and Infrastructure*. Laurence Yang and Minyi Guo (editors), Wiley Press, New Jersey, USA. Fall 2004.

[118] Lv, Q., Cao, P., Cohen, E., Li, K., and Shenker, S. (2002). Search and replication in unstructured peer-to-peer networks. In *ICS '02: Proceedings of the 16th international conference on Supercomputing*, pages 84–95. ACM Press, New York, NY, USA.

[119] Mastroianni, C., Talia, D., and Verta, O. (2005). A super-peer model for resource discovery services in large-scale grids. *Future Generation Computer Systems*, 21(8):1235–1248.

[120] McLaughry, S. W. and Wycko, P. (1999). T spaces: The next wave. In *HICSS '99: Proceedings of the Thirty-second Annual Hawaii International Conference on System Sciences-Volume 8*, page 8037. IEEE Computer Society, Los Alamitos, CA, USA.

[121] MicroSystems, S. (2003). Javaspace specification 2.0, http://www.sun.com/software/jini/specs/js2_0.pdf. *Technical Report*.

[122] Milojicic, D., Kalogeraki, V., Lukose, R., and Nagarajan, K. (2002). Peer-to-peer computing. Technical Report HPL-2002-57, HP Labs.

[123] Mislove, A., Post, A., Haeberlen, A., and Druschel, P. (2006). Experiences in building and operating epost, a reliable peer-to-peer application. In *EuroSys '06: Proceedings of the 2006 EuroSys conference, Leuven, Belgium*, pages 147–159. ACM Press, New York, NY, USA.

[124] Mondal, A., Lifu, Y., and Kitsuregawa, M. (2004). P2PR-Tree: An r-tree-based spatial index for peer-to-peer environments. In *Proceedings of the International Workshop on Peer-to-Peer Computing and Databases (held in conjunction with EDBT)*, pages 516 – 525. Springer-Verlag, Germany.

[125] Moore, D. and Hebeler, J. (2001). *Peer-to-Peer:Building Secure, Scalable, and Manageable Networks*. McGraw-Hill Osborne.

[126] Murphy, A. L., Picco, G. P., and Roman, G. (2006). LIME: A coordination model and middleware supporting mobility of hosts and agents. *ACM Transactions on Software Engineering Methodology, Volume 15, Issue 3, pages 279–328, ACM Press, New York, NY, USA.*

[127] Noy, N. F. (2004). Semantic integration: a survey of ontology-based approaches. *SIGMOD Records, Volume 33, Issue 4, pages 65–70, ACM Press, New York, NY, USA.*

[128] Oppenheimer, D., Albrecht, J., Vahdat, A., and Patterson, D. (July 2005). Design and implementation trade-offs for wide-area resource discovery. In *Proceedings of 14th IEEE Symposium on High Performance, Research Triangle Park, NC*. IEEE Computer Society, Los Alamitos, CA, USA.

[129] Orenstein, J. (1990). A comparison of spatial query processing techniques for native and parameter spaces. In *SIGMOD '90: Proceedings of the 1990 ACM SIGMOD international conference on Management of data, Atlantic City, New Jersey, United States*, pages 343–352. ACM Press, New York, NY, USA.

[130] Ouelhadj, D., Garibaldi, J., MacLaren, J., Sakellariou, R., and Krishnakumar, K. (2005). A multi-agent infrastructure and a service level agreement negotiation protocol for robust scheduling in grid computing. In *Proceedings of the European Grid Conference*. Lecture Notes in Computer Science, Springer-Verlag, Germany.

[131] Pallis, G. and Vakali, A. (2006). Insight and perspectives for content delivery networks. *Communications of the ACM, Volume 49, Issue 1, pages 101–106, ACM Press, New York, NY, USA*.

[132] Pearlman, L., Welch, V., Foster, I., Kesselman, C., and Tuecke, S. (2002). A community authorization service for group collaboration. In *POLICY '02: Proceedings of the 3rd International Workshop on Policies for Distributed Systems and Networks (POLICY'02)*, page 50. IEEE Computer Society, Los Alamitos, CA, USA.

[133] Preneel, B. (1999). The state of cryptographic hash functions. In *Lectures on Data Security, Modern Cryptology in Theory and Practice, Summer School, Aarhus, Denmark, July 1998*, pages 158–182. Springer-Verlag, UK.

[134] Ramabhadran, S., Ratnasamy, S., Hellerstein, J. M., and Shenker, S. (2004). Brief announcement: Prefix Hash Tree. In *PODC'04: Proceedings of ACM PODC, St. Johns, Canada*. ACM Press, New York, NY, USA.

[135] Raman, R., Livny, M., and Solomon, M. (1998). Matchmaking: Distributed resource management for high throughput computing. In *HPDC '98: Proceedings of the The Seventh IEEE International Symposium on High Performance Distributed Computing*, page 140. IEEE Computer Society, Los Alamitos, CA, USA.

[136] Ranjan, R., Buyya, R., and Harwood, A. (2005a). A case for cooperative and incentive based coupling of distributed clusters. In *Cluster'05: Proceedings of the 7th IEEE International Conference on Cluster Computing, Boston, MA, USA*. IEEE Computer Society, Los Alamitos, CA, USA.

[137] Ranjan, R., Buyya, R., and Harwood, A. (2005b). A model for cooperative federation of distributed clusters. In *HPDC'14: Proceedings of the 14th IEEE International Conference on High Performance Distributed Computing, Research Triangle Park, North Carolina*. IEEE Computer Society, Los Alamitos, CA, USA.

[138] Ranjan, R., Harwood, A., and Buyya, R. (2006). SLA-based coordinated super-scheduling scheme for computational grids. In *Proceedings of the 8th IEEE International Conference on Cluster Computing (CLUSTER'06), Barcelona, Spain*. IEEE Computer Society, Los Alamitos, CA, USA.

[139] Rao, A., Lakshminarayanan, K., Surana, S., Karp, R., and Stoica, I. (2003). Load balancing in structured p2p systems. In *2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*.

[140] Ratnasamy, S., Francis, P., Handley, M., Karp, R., and Schenker, S. (2001). A scalable content-addressable network. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, San Diego, California, United States*, pages 161–172. ACM Press, New York, NY, USA.

[141] Ratnasamy, S., Stoica, I., and Shenker, S. (2002). Routing algorithms for dhts: Some open questions. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 45–52. Springer-Verlag, London, UK.

[142] Rivest, R. (1976). Partial match retrieval algorithms. *SIAM Journal of Computing, Volume 5, Issue 1, pages 19–50*.

[143] Rowstron, A. and Druschel, P. (2001). Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware'01: Proceedings of IFIP/ACM International Conference on Distributed Systems Platforms*, pages 329–359. SpringerLink, Heidelberg, Germany.

[144] Sabin, G., Kettimuthu, R., Rajan, A., and Sadayappan, P. (2003). Scheduling of Parallel Jobs in a Heterogeneous Multi-site Environment. In *Job Scheduling Strategies for Parallel Processing, volume 2862*, pages 87–104, Seattle, WA, USA.

[145] Sacerdoti, F., Katz, M., Massie, M., and Culler, D. (2003). Wide area cluster monitoring with ganglia. In *Cluster'03: Proceedings of the 5th IEEE International Conference on Cluster Computing, Tsim Sha Tsui, Kowloon, Hong Kong*, pages 289–298. IEEE Computer Society,Los Alamitos, CA, USA.

[146] Sahin, O. D., Gupta, A., Agrawal, D., and Abbadi, A. E. (2004). A peer-to-peer framework for caching range queries. In *ICDE '04: Proceedings of the 20th International Conference on Data Engineering*, page 165. IEEE Computer Society, Los Alamitos, CA, USA.

[147] Samet, H. (1989). *The Design and Analysis of Spatial Data Structures*. Addison-Wesley Publishing Company.

[148] Saroiu, S., Gummadi, P., and Gribble, S. (2003). A measurement study of peer-to-peer file sharing systems. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*. ACM Press, New York, NY, USA.

[149] Satyanarayanan, M. (2001). Pervasive computing: vision and challenges. *IEEE Personal Communications, Volume 8, Issue 4, pages 10–17, IEEE Computer Society, Los Alamitos, CA, USA*.

[150] Schmidt, C. and Parashar, M. (2003). Flexible information discovery in decentralized distributed systems. In *HPDC'12: In the Twelfth International Symposium on High Performance Distributed Computing, June*. IEEE Computer Society, Los Alamitos, CA, USA.

[151] Schopf, J. (2001). Ten actions when superscheduling. In *Global Grid Forum*.

[152] Shan, H., Oliker, L., and Biswas, R. (2003). Job superscheduler architecture and performance in computational grid environments. In *SC '03: Proceedings of the 2003 ACM/IEEE conference on Supercomputing*, pages 44–51. IEEE Computer Society, Los Alamitos, CA, USA.

[153] Sherwani, J., ALi, N., Lotia, N., Hayat, Z., and Buyya, R. (2002). Libra: An economy driven job scheduling system for clusters. *Proceedings of 6th International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia'02)*.

[154] Sit, E. and Morris, R. (2002). Security considerations for peer-to-peer distributed hash tables. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 261–269. Springer-Verlag, Germany.

[155] Smale, S. (May 1976). Convergent process of price adjustment and global newton methods. In *American Economic Review, 66(2)*, pages 284–294.

[156] Smith, R. (1980). The contract net protocol: high level communication and control in distributed problem solver. In *IEEE Transactions on Computers*, pages 1104–1113. IEEE Computer Society, Los Alamitos, CA, USA.

[157] Smith, R. G. (1988). The contract net protocol: high-level communication and control in a distributed problem solver. *Distributed Artificial Intelligence, Morgan Kaufman Publishers Inc., San Francisco, CA, USA*, pages 357–366.

[158] Spence, D., Crowcroft, J., Hand, S., and Harris, T. (2005). Location based placement of whole distributed systems. In *CoNEXT'05: Proceedings of the 2005 ACM conference on Emerging network experiment and technology, Toulouse, France*, pages 124–134. ACM Press, New York, NY, USA.

[159] Spence, D. and Harris, T. (2003). Xenosearch: Distributed resource discovery in the xenoserver open platform. In *HPDC '03: Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing (HPDC'03)*, page 216. IEEE Computer Society, Los Alamitos, CA, USA.

[160] Srinivasan, S., Kettimuthu, R., Subramani, V., and Sadayappan, P. (2002). Selective reservation strategies for backfill job scheduling. In *JSSPP '02: Revised Papers from the 8th International Workshop on Job Scheduling Strategies for Parallel Processing*, pages 55–71. Springer-Verlag, London, UK.

[161] Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., and Balakrishnan, H. (2001). Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, San Diego, California, USA*, pages 149–160. ACM Press, New York, NY, USA.

[162] Stonebraker, M., Devine, R., Kornacker, M., Litwin, W., Pfeffer, A., Sah, A., and Staelin, C. (1994). An economic paradigm for query processing and data migration in maiposa. *Proceedings of 3rd International Conference on Parallel and Distributed Information Systems, Austin, TX, USA, September 28-30, IEEE CS Press*.

[163] Subramani, V., Kettimuthu, R., Srinivasan, S., and Sadayappan, P. (2002). Distributed job scheduling on computational grids using multiple simultaneous requests. In *11th IEEE International Symposium on High Performance Distributed Computing (HPDC-11), 23-26 July*. IEEE Computer Society, Los Alamitos, CA, USA.

[164] Talia, D. and Trunfio, P. (2003). Toward a synergy between p2p and grids. *IEEE Internet Computing*, 7(4):96–95.

[165] Tam, D., Azimi, R., and Jacobsen, H. (2003). Building content-based publish/subscribe systems with distributed hash tables. In *International Workshop on Databases, Information Systems and Peer-to-Peer Computing*. Springer-Verlag, Germany.

[166] Tanin, E., Harwood, A., and Samet, H. (2005). A distributed quad-tree index for peer-to-peer settings,. In *ICDE'05: Proceedings of the International Conference on Data Engineering*, pages 254–255. IEEE Computer Society, Los Alamitos, CA, USA.

[167] Tanin, E., Harwood, A., and Samet, H. (2007). Using a distributed quadtree index in peer-to-peer networks. *VLDB Journal, Volume 16, Issue 2, pages 165–178*.

[168] Teo, Y., March, V., and Wang, X. (2005). A DHT-based grid resource indexing and discovery scheme. In *Singapore-MIT Alliance Annual Symposium*.

[169] Tolksdorf, R. and Glaubitz, D. (2001). Coordinating web-based systems with documents in xmlspaces. In *CooplS '01: Proceedings of the 9th International Conference on Cooperative Information Systems*, pages 356–370. Springer-Verlag, London, UK.

[170] Triantafillou, P. and Aekaterinidis, I. (2004). Content-based publish/-subscribe over structured p2p networks. *1st International Workshop on Discrete Event-Based Systems*.

[171] Venugopal, S., Buyya, R., and Winton, L. (2006). A Grid Service Broker for Scheduling distributed e-Science Applications on Global Data Grids. *Concurrency and Computation: Practice and Experience, Volume 18, Issue 6, pages 685–699, Wiley Press, New York, NY, USA*.

[172] Waldspurger, C., Hogg, T., Huberman, B., Kephart, J., and Stornetta, W. (1992). Spawn: A distributed computational economy. *IEEE Transactions on Software Engineering , Volume 18, Issue 2, IEEE CS Press, USA, February*.

[173] Waterhouse, S., Doolin, D. M., K., G., and Faybishenko, Y. (2002). Distributed search in p2p networks. *IEEE Internet Computing, Volume 06, Issue 1, pages 68–72, IEEE Computer Society, Los Alamitos, CA, USA*.

[174] Weissman, J. B. and Grimshaw, A. (1996). Federated model for scheduling in wide-area systems. *HPDC'96: Proceedings of the Fifth IEEE International Symposium on High Performance Distributed Computing, pages542-550, August*.

[175] Welch, V., Siebenlist, F., Foster, I., Bresnahan, J., Czajkowski, K., Gawor, J., Kesselman, C., Meder, S., Pearlman, L., and Tuecke, S. (2003). Security for grid services. In *HPDC '03: Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing*, pages 48– 57. IEEE Computer Society, Los Alamitos, CA, USA.

[176] Wolski, R., Plank, J. S., Brevik, J., and Bryan, T. (2001). G-commerce: Market formulations controlling resource allocation on the computational grid. In *IPDPS '01: Proceedings of the 15th International Parallel & Distributed Processing Symposium*, page 46. IEEE Computer Society, Los Alamitos, CA, USA.

[177] Yang, B. and Garcia-Molina, H. (2003). Designing a super-peer network. In *ICDE'03: Proceedings of the 19th IEEE International Conference on Data Engineering*, volume 00, page 49. IEEE Computer Society, Los Alamitos, CA, USA.

[178] Yeager, W. and Williams, J. (2002). Secure peer-to-peer networking: The JXTA example. *IT Professional, IEEE Educational Activities Department, Piscataway, NJ, USA*, 4(2):53–57.

[179] Yeo, C. and Buyya, R. (2005). Service level agreement based allocation of cluster resources: Handling penalty to enhance utility. In *Cluster'05: Proceedings of the 7th IEEE International Conference on Cluster Computing, Boston, MA, USA*. IEEE Computer Society, Los Alamitos, CA, USA.

[180] Yeo, C. S., Buyya, R., Assuncao, M. D., Yu, J., Sulistio, A., Venugopal, S., and Placek, M. (2006). *Utility Computing on Global Grids, Handbook of Computer Networks*. Hossein Bidgoli (editor), John Wiley and Sons, New York, USA.

[181] Yu, J., Venugopal, S., and Buyya, R. (2006). Grid market directory: A web and web services based grid service publication directory. *The Journal of Supercomputing, Volume 36, Issue 1, pages 17–31, Springer Science and Business Media, Berlin, Germany*.

[182] Zanikolas, S. and Sakellariou, R. (2005). A taxonomy of grid monitoring systems. *Future Generation Computer Systems (FGCS) Journal, Volume 21, Issue 1, pages 163-188, Elsevier Science, The Netherlands, January*.

[183] Zhang, X., Freschl, J. L., and Schopf, J. M. (2003). A performance study of monitoring and information services for distributed systems. In *The Twelfth International Symposium on High Performance Distributed Computing (HPDC-12), June*. IEEE Computer Society, Los Alamitos, CA, USA.

[184] Zhao, B. Y., Kubiatowicz, J. D., and Joseph, A. D. (2001). Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley.

[185] Zhou, S. (1992). LSF: Load sharing in large-scale heterogeneous distributed systems. *In Proceedings of the Workshop on Cluster Computing, Tallahassee, FL,*.