Scheduling Distributed Data-Intensive Applications on Global Grids

by

Srikumar Venugopal

Submitted in total fulfilment of
the requirements for the degree of

Doctor of Philosophy

THE UNIVERSITY OF
MELBOURNE

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

July 2006

Scheduling Distributed Data-Intensive Applications on Global Grids

Srikumar Venugopal

Supervisors: Dr. Rajkumar Buyya, Prof. Rao Kotagiri

Abstract

The next generation of scientific experiments and studies are being carried out by large collaborations of researchers distributed around the world engaged in analysis of huge collections of data generated by scientific instruments. Grid computing has emerged as an enabler for such collaborations as it aids communities in sharing resources to achieve common objectives. Data Grids provide services for accessing, replicating and managing data collections in these collaborations. Applications used in such Grids are distributed data-intensive, that is, they access and process distributed datasets to generate results. These applications need to transparently and efficiently access distributed data and computational resources. This thesis investigates properties of data-intensive computing environments and presents a software framework and algorithms for mapping distributed data-oriented applications to Grid resources.

The thesis discusses the key concepts behind Data Grids and compares them with other data sharing and distribution mechanisms such as content delivery networks, peer-to-peer networks and distributed databases. This thesis provides comprehensive taxonomies that cover various aspects of Data Grid architecture, data transportation, data replication and resource allocation and scheduling. The taxonomies are mapped to various Data Grid systems not only to validate the taxonomy but also to better understand their goals and methodology.

The thesis concentrates on one of the areas delineated in the taxonomy – scheduling distributed data-intensive applications on Grid resources. To this end, it presents the design and implementation of a Grid resource broker that mediates access to distributed computational and data resources running diverse middleware. The broker is able to discover remote data repositories, interface with various middleware services and select suitable resources in order to meet the application requirements. The use of the broker is illustrated by a case study of scheduling a data-intensive high energy physics analysis application on an Australia-wide Grid.

The broker provides the framework to realise scheduling strategies with differing objectives. One of the key aspects of any scheduling strategy is the mapping of jobs to the appropriate resources to meet the objectives. This thesis presents heuristics for mapping jobs with data dependencies in an environment with heterogeneous Grid resources and multiple data replicas. These heuristics are then compared with performance evaluation metrics obtained through extensive simulations.

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

This thesis is the culmination of a long journey throughout which I have received support from many people whom I wish to acknowledge here. First and foremost, I am deeply indebted to my principal supervisor, Rajkumar Buyya, for his advice and for being a ceaseless motivator during the last four years. He has always encouraged me to keep an open mind and to push the envelope at all times. I owe the progress made in my research career to his tutelage, and to the peer interactions enabled with his support.

I would like to express my gratitude towards Rao Kotagiri, my co-supervisor, for the encouragement, support and advice that I have received from him during my candidature. His questions and insights have often snapped me from acute tunnel vision and have led to explorations into other areas of distributed systems. These have revealed interesting ideas for the work in this thesis and for the future.

I would like to thank Lyle Winton (School of Physics, University of Melbourne) for being a great source of ideas during the early days of my candidature. The principles behind the Gridbus broker were shaped during discussions with him and his experience with the Globus Toolkit proved invaluable during its development. Krishna Nadiminti and Hussein Gibbins (GRIDS Laboratory, University of Melbourne) enthusiastically took up the development of the broker and greatly improved its design and features. Their irreverent attitude has made the GRIDS Lab a fun place to be in. I would also like to express my gratitude to Choon Hoong Ding and Tianchi Ma (GRIDS Lab), Brett Beeson, Glenn Moloney, and Martin Sevior (School of Physics), Benjamin Khoo (IBM, Singapore), and students of the Cluster and Grid Computing subject with whom I have worked and exchanged ideas over the years.

I would like to thank Reagan Moore (San Diego Supercomputing Center), Heinz Stockinger (University of Vienna), Chris Mattman (JPL, NASA) and William Allcock (Argonne National Lab) for their extensive comments on the taxonomy that has been incorporated into this thesis. I would also like to thank Sushil Prasad (Georgia State University) for his instructive comments on the scheduling heuristics in this thesis. I also would like to express my gratitude towards Marcos Assuncao, Kyong-Hoon Kim and Marco Netto (GRIDS Lab, University of Melbourne) for proof-reading this thesis and for their extensive comments.

Chee Shin, Jia and Anthony have been my office-mates for most of my candidature. I have received ready answers from them whether it be questions about Java, GridSim or Chinese food and have had fun during long discussions about nothing in particular. I would like to thank them for simply being good friends and Chee Shin, in particular, for being a good listener. Elan and Rajiv also have been good mates and the long trips that we have all taken together have been the highest points of my Australian experience so far. Shoaib deserves to be thanked for helping me adapt to Melbourne in the initial days after my arrival. I would also like to thank other friends in the GRIDS Lab and the Department for their support during my candidature.

I would like to thank the University of Melbourne and the Australian Government for providing scholarships to pursue doctoral studies. Also, I would like to acknowledge the support received from the ARC (Australian Research Council) Discovery Project and StorageTek Fellowship grants at various times during my candidature. I would like to express my gratitude to the Department for the infrastructural support and for the travel scholarships that have helped me attend international conferences. In particular, I would like to thank the administrative staff for being very helpful at all times.

I would like to thank Sreeraj, my (ex) housemate of nearly four years, for his friendship, and for patiently enduring my tyranny in the kitchen. We started our PhD candidatures together and have shared many experiences that will be cherished memories for many years. I would like to thank John Kuruvilla and his family, Biju and Julie George, and Hari and Indu for their support, and for the frequent meals that I have had at their homes.

Last but never the least, I would like to thank my family for their love and support at all times. My brother's wry humor and wisecracks have never failed to uplift my mood. My mother and father have been always optimistic about everything and their dogged persistence has been a constant source of inspiration during my candidature. I am what I am only due to their efforts.

Srikumar Venugopal
Melbourne, Australia
July 2006.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

This chapter introduces the context of the research to be presented in this thesis. It starts off with an introduction to the general area of Grid computing and Data Grids, and discusses the motivation and challenges for scheduling distributed data-intensive applications in such environments. Then, it presents a short overview of resource brokers and scheduling, and presents the primary contributions of this research. The chapter ends with a discussion on the organisation of the rest of this thesis.

1.1 Grid Computing

The next generation of scientific applications in domains as diverse as high energy physics, molecular modelling, and earth sciences involve the production of large datasets from simulations or large-scale experiments. Analysis of these datasets and their dissemination among researchers located over a wide geographic area requires high capacity resources such as supercomputers, high bandwidth networks, and mass storage systems. Collectively, these large scale applications are now part of e-Science [101], a discipline that envisages using high-end computing, storage, networking and Web technologies together to facilitate collaborative and data-intensive scientific research. e-Science requires new paradigms in Internet computing that address issues such as multi-domain data sharing applications, co-operation and co-ordination of resources and operations across system boundaries.

Grid computing [84] paradigm unites geographically-distributed and heterogeneous

computing, storage, and network resources and provide unified, secure, and pervasive access to their combined capabilities. Therefore, Grid platforms enable sharing, exchange, discovery, selection, and aggregation of distributed heterogeneous resources such as computers, databases, visualisation devices, and scientific instruments. Grid computing, therefore, leads to the creation of virtual organisations [88] by allowing geographically-distributed communities to pool resources in order to achieve common objectives.

Grid computing has the potential to support different kinds of applications. They include compute-intensive applications, data-intensive applications and applications requiring distributed services. Various types of Grids have been developed to support these applications and are categorized as Computational Grids, Data Grids and Service Grids [123]. A large number of e-Science applications require capabilities supported by Data Grids. Realizing such Grids requires challenges to be overcome in security, user management, resource management, resource discovery, application scheduling, high-speed network protocols, and data management. However, from the user's perspective, two important barriers that need to be overcome are the complexity of developing Grid applications and their scheduling on distributed resources. This thesis presents a software framework for creating and composing distributed data-intensive applications, and scheduling algorithms for effectively deploying them on global Grids.

1.2 Data Grids and Application Scheduling

Data Grids [56, 106] primarily deal with providing services and infrastructure for distributed data-intensive applications that need to access, transfer and modify massive datasets stored in distributed storage resources. A Data Grid aims to present the following capabilities to its users: (a) ability to search through numerous available datasets for the required dataset and to discover suitable data resources for accessing the data, (b) ability to transfer large-sized datasets between resources in a minimal time, (c) ability for users to manage multiple copies of their data, (d) ability to select suitable computational resources and process data on them, and (e) ability to manage access permissions for the data. Therefore, Data Grids aim to combine high-end computing technologies with high-performance networking and wide-area storage management techniques.

To realise these abilities, a Data Grid needs to provide tools, services and APIs (Application Programming Interfaces) for orchestrating collaborative access to data and computational resources. These include administration tools to make it less cumbersome to manage authenticating and authorising widely dispersed members for accessing disparate resources and data collections; data search tools to allow users to discover datasets of interest out of the hundreds and thousands that may be available within a collaboration; intelligent data replication and caching services to ensure that the users can access the required datasets in the fastest and/or cheapest manner; data management tools and services to allow users to upload data back into the collaboration, provide useful descriptions for other researchers and if required, enforce access controls; and resource management services and APIs to allow applications and users to utilise the infrastructure effectively by processing the data at idle resources that offer better turnaround times and reduced costs. This thesis, however, concentrates on the challenges of application deployment on Data Grids.

Scheduling and deployment of Grid applications is performed by *resource brokers* that hide the complexity of the underlying infrastructure by transforming users' requirements into Grid operations, that are then carried out without their intervention. Users describe requirements such as the type of analysis, required executables, data dependencies, deadline for the execution and the maximum available budget through simple interfaces. The resource broker creates jobs corresponding to the analysis requirements and discovers suitable computational resources to execute the jobs and appropriate data repositories for accessing the data required for the jobs. It then deploys the jobs on selected Grid resources, monitors their execution, and upon their completion, collates and presents the results of the analysis to the users. By abstracting the low-level details of dealing with Grid resources, a resource broker helps its users focus on designing scenarios and experiments that utilise the infrastructure thereby allowing them to realise maximum utility from their collaboration.

Figure 1.1 shows such a scenario that involves an application with distributed data requirements. The data is generated by an instrument such as a particle accelerator or a telescope and is replicated at distributed locations. The broker discovers the data replicas by querying a directory such as Replica Location Services (RLS) [55] and available

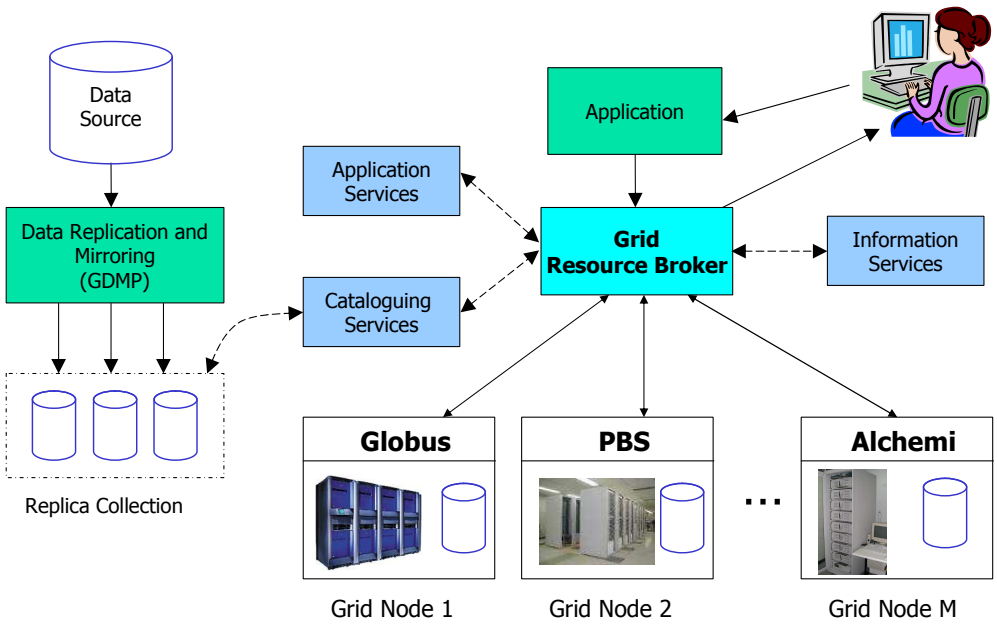


Figure 1.1: A Grid resource broker for Data Grids.

computational resources by querying information services such as Grid Index Information Service (GIIS) [65]. Additionally, it may consult other information services such as Grid Market Directory [221] for resource prices, resource monitoring services such as Ganglia [178] for performance data, and application catalogues for information about locations of applications. It then devises a schedule for executing the application taking into consideration the computational performance, data transfer requirements and costs associated with resource usage.

In recent years, many resource brokers have been developed for different applications and to achieve different objectives [1, 12, 52, 74, 91, 183]. However, the needs of distributed data-intensive applications have not been taken into account by these in either the process of resource discovery or job scheduling. This thesis presents the architecture and design of a Grid resource broker that discovers suitable data sources and computational resources for a given distributed data-intensive application scenario; maps such jobs to resources in order to achieve user-specified Quality of Service (QoS) metrics; deploys and monitors job execution on selected resources; accesses data from local or remote data sources during job execution; and collates and presents results of the execution to the user.

The execution of distributed data-intensive applications involves requirements for discovering, processing, storing and managing large distributed datasets and is guided by

factors such as cost and speed of accessing, transferring and processing data. There may be multiple datasets involved in a computation, each replicated at multiple locations that are connected to one another and to the compute resources by networks with varying costs and capabilities. Consequently, this explosion of choices makes it difficult to identify appropriate resources for retrieving and performing the required computation on the selected datasets. This thesis, therefore, develops and presents scheduling algorithms for applications that require accessing massive datasets replicated on multiple Grid resources.

1.3 Contributions

This thesis makes several contributions towards improving the understanding of data-intensive computing environments and towards advancing the area of scheduling distributed data-intensive applications on Grid resources. These are as follows:

1. This thesis discusses the key concepts behind Data Grids and compares them with content delivery networks, peer-to-peer networks and distributed databases. It provides a systematic characterisation of Data Grids and a thorough examination of their differences with these distributed data-intensive mechanisms. The objective of this exercise is to delineate the uniqueness of Data Grids and to identify technologies and algorithms developed in related areas that can be applied to the target research area.

2. This thesis provides comprehensive taxonomies that cover various aspects of architecture, data transportation, data replication and resource allocation and scheduling. The proposed taxonomy is mapped to various Data Grid systems not only to validate the taxonomy but also to better understand their goals and their methodology. This also helps evaluate their applicability to similar problems.

3. This thesis presents the design and development of a Grid resource broker for executing distributed data-oriented applications on a Grid. The broker discovers computational and data resources, schedules jobs based on users' requirements and returns results back to the user. The broker follows a simple yet extensible object-oriented model that is based on the strict separation of logic and data.

4. This thesis presents a comprehensive resource and application model for the problem of scheduling distributed data intensive Bag of Task applications on Data Grids. The application can be split up or "decomposed" to obtain a collection of independent jobs that each require multiple datasets that are each replicated on multiple data repositories. The model takes into account the economic costs of processing a job along with the execution time and the times for transferring the required datasets from different data hosts to the compute resource on which the job will be executed.

5. This thesis presents heuristics for mapping and scheduling distributed data-intensive jobs on Data Grid resources. It introduces a greedy heuristic that aims to minimise either the total execution cost or time depending on the user's preference, subject to the user's deadline and budget constraints. It introduces another heuristic that is based on a solution to the well-known Set Covering Problem. These are evaluated both on real Grid testbeds and via extensive simulations.

1.4 Thesis Organisation

The rest of the thesis is organised as follows: Chapter 2 presents an overview of Data Grids and the comparison with other data distribution and processing technologies. This is followed by Chapter 3 which proposes a taxonomy of Data Grid research and classifies some of the publications within this field accordingly. The thesis then concentrates on one of the areas delineated in the taxonomy - that of resource allocation and scheduling - and introduces the design and architecture of the Gridbus broker in Chapter 4. Chapter 5 discusses the scheduling problem and also introduces a greedy heuristic for deadline and budget constrained cost and time minimisation scheduling of data-intensive applications. Chapter 6 then discusses a graph-based approach towards the scheduling problem and presents a heuristic and its evaluation via simulation. Finally, the thesis concludes and presents ideas for future work in Chapter 7.

The core chapters are derived from various articles published during the course of the Ph.D. candidature as detailed below:

Chapter 2 and *Chapter 3* are derived from:

- **Srikumar Venugopal**, Rajkumar Buyya, and Kotagiri Ramamohanarao, "A Taxonomy of Data Grids for Distributed Data Sharing, Management and Processing", *ACM Computing Surveys*, Vol. 38, No. 1, ACM Press, New York, USA, March 2006.

Chapter 4 is partially derived from:

- **Srikumar Venugopal**, Rajkumar Buyya, and Lyle Winton, "A Grid Service Broker for Scheduling Distributed Data-Oriented Applications on Global Grids", *Proceedings of the 2nd International Workshop on Middleware for Grid Computing (MGC 04)*, Oct. 2004, Toronto, Canada, ACM Press, USA.

- **Srikumar Venugopal**, Rajkumar Buyya, and Lyle Winton, "A Grid Service Broker for Scheduling e-Science Applications on Global Data Grids", *Concurrency and Computation: Practice and Experience*, Vol. 18, No. 6, pp 685-699, Wiley Press, New York, USA, May 2006.

- Krishna Nadiminti, **Srikumar Venugopal**, Hussein Gibbins, and Rajkumar Buyya, *The Gridbus Grid Service Broker and Scheduler (2.0) User Guide*, Technical Report, GRIDS-TR-2005-4, Grid Computing and Distributed Systems Laboratory, University of Melbourne, Australia, April 22, 2005.

  Comments: Krishna Nadiminti and Hussein Gibbins as members of the Gridbus project have extended the Gridbus broker to operate with recent developments in low-level middleware and added various features required for production Grid usage. Prior to their involvement, I was the primary developer of the broker and applied the same to many application studies including the Belle High Energy Physics application study reported in this thesis.

Chapter 5 and *Chapter 6* are partially derived from:

- **Srikumar Venugopal** and Rajkumar Buyya, "A Deadline and Budget Constrained Scheduling Algorithm for eScience Applications on Data Grids", *Proceedings of the 6th International Conference on Algorithms and Architectures for Parallel Processing*, Oct. 2005, Melbourne, Australia, Springer-Verlag, Berlin, Germany.

- **Srikumar Venugopal** and Rajkumar Buyya, "A Set Coverage-based Mapping Heuristic for Scheduling Distributed Data-Intensive Applications on Global Grids", *Proceedings of the 7th IEEE/ACM International Conference on Grid Computing (Grid 2006)*, Sept. 2006, Barcelona, Spain, IEEE Computer Society Press, Los Alamitos, CA (accepted and in print).

Chapter 2

Data Grids: An Overview and Comparison

This chapter provides a general overview of Data Grids that covers topics such as key concepts, characteristics and a layered architecture. This chapter presents an analysis of the differences between Data Grids and other distributed data-intensive paradigms such as content delivery networks, peer-to-peer file-sharing networks and distributed databases. It ends with a discussion on the convergence between the former and the latter and how techniques in other data-intensive networks are finding application in Data Grids.

2.1 Terms and Definitions

A data intensive computing environment consists of applications that produce, manipulate or analyse data in the range of hundreds of MegaBytes (MB) to PetaBytes (PB) and beyond [149]. The data is organised as collections or *datasets* and are typically stored on mass storage systems (also called *repositories*) such as tape libraries or disk arrays. The datasets are accessed by users in different locations who may create local copies or *replicas* of the datasets to reduce latencies involved in wide-area data transfers and therefore, improve application performance. A replica may be a complete or a partial copy of the original dataset. A *replica management system* or *data replication mechanism* allows users to create, register and manage replicas and may also update the replicas if the

original datasets are modified. The system may also create replicas on its own guided by *replication strategies* that take into account current and future demand for the datasets, locality of requests and storage capacity of the repositories. *Metadata*, or "data about data", is information that describes the datasets and could consist of attributes such as name, time of creation, size on disk and time of last modification. Metadata may also contain specific information such as details of the process that produced the data. A *replica catalog* contains information about locations of datasets and associated replicas and the metadata associated with these datasets. Users query the catalog using metadata attributes to conduct operations such as locating the nearest replica of a particular dataset.

In the context of Grid computing, any hardware or software entity such as supercomputers, storage systems or applications that are shared between users of a Grid is called a *resource*. However, for the rest of this thesis and unless otherwise stated, the term resource means hardware such as computers or storage systems. Resources are also *nodes* in the network and hence, these terms are used interchangeably. The network-enabled capabilities of the resources that can be invoked by users, applications or other resources are called *services*.

2.2 Data Grids

A Data Grid provides services that help users discover, transfer and manipulate large datasets stored in distributed repositories and also, create and manage copies of these datasets. At the minimum, a Data Grid provides two basic functionalities: a high performance and reliable data transfer mechanism, and a scalable replica discovery and management mechanism [56]. Depending on application requirements, various other services need to be provided. Examples of such services include consistency management for replicas, metadata management and data filtering and reduction mechanism. All operations in a Data Grid are mediated by a security layer that handles authentication of entities and ensures conduct of only authorized operations.

Another aspect of a Data Grid is to maintain shared collections of data distributed across administrative domains. These collections are maintained independent of the underlying storage systems and are able to include new sites without major effort. More

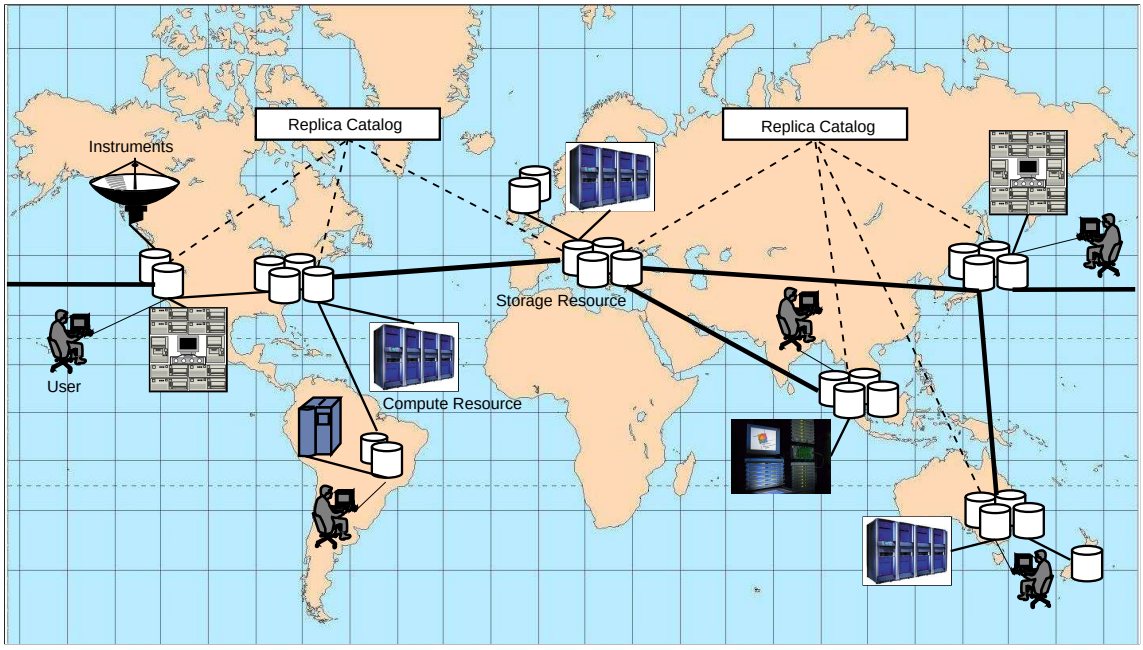


Figure 2.1: A High-Level view of a Data Grid.

importantly, it is required that the data and information associated with data such as metadata, access controls and version changes be preserved even in the face of platform changes. These requirements lead to the establishment of persistent archival storage [150].

Figure 2.1 shows a high-level view of a worldwide Data Grid consisting of computational and storage resources in different countries that are connected by high speed networks. The thick lines show high bandwidth networks linking the major centres and the thinner lines are lower capacity networks that connect the latter to their subsidiary centres. The data generated from an instrument, experiment or a network of sensors is stored in its principal storage site and is transferred to the other storage sites around the world on request through the data replication mechanism. Users query their local replica catalog to locate datasets that they require. If they have been granted the requisite rights and permissions, the data is fetched from the repository local to their area, if it is present there; otherwise it is fetched from a remote repository. The data may be transmitted to a computational site such as a cluster or a supercomputer facility for processing. After processing, the results may be sent to a visualisation facility, a shared repository or to the desktops of the individual users.

A Data Grid, therefore, provides a platform through which users can access aggregated computational, storage and networking resources to execute their data-intensive applica-

tions on remote data. It promotes a rich environment for users to analyse data, share the results with their collaborators and maintain state information about the data seamlessly across institutional and geographical boundaries. Often cited examples for Data Grids are the ones being set up for analysing the huge amounts of data that will be generated by the CMS (Compact Muon Solenoid), ATLAS (A Toroidal LHC AppratuS), ALICE (A Large Ion Collider Experiment) and LHCb (LHC beauty) experiments at the Large Hadron Collider (LHC) [131] at CERN when they will begin production in 2007. These Data Grids will involve thousands of physicists spread over hundreds of institutions worldwide and will be replicating and analysing terabytes of data daily.

Resources in a Grid are heterogeneous in terms of operating environments, capability and availability and are under the control of their own local administrative domains. These domains are autonomous and retain the rights to grant users access to the resources under their control. Therefore, Grids are concerned with issues such as: sharing of resources, authentication and authorization of entities, and resource management and scheduling for efficient and effective use of available resources. Naturally, Data Grids share these general concerns, but have their own unique set of characteristics and challenges listed below:

- *Massive Datasets:* Data-intensive applications are characterised by the presence of large datasets of the size of Gigabytes (GB) and beyond. For example, the CMS experiment at the LHC is expected to produce 1 PB (10^{15} bytes) of RAW data and 2 PB of Event Summary Data (ESD) annually when it begins production [104]. Resource management within Data Grids therefore extends to minimizing latencies of bulk data transfers, creating replicas through appropriate replication strategies and managing storage resources.

- *Shared Data Collections:* Resource sharing within Data Grids also includes, among others, sharing distributed data collections. For example, participants within a scientific collaboration would want to use the same repositories as sources for data and for storing the outputs of their analyses.

- *Unified Namespace:* The data in a Data Grid share the same logical namespace in which every data element has a unique logical filename. The logical filename is mapped to one or more physical filenames on various storage resources across a

Data Grid.

- *Access Restrictions:* Users might wish to ensure confidentiality of their data or restrict distribution to close collaborators. Authentication and authorization in Data Grids involves coarse to fine-grained access controls over shared data collections.

However, certain characteristics of Data Grids are specific to the applications for which they are created. For example, for astrophysics or high energy physics experiments, the principal instrument such as a telescope or a particle accelerator is the single site of data generation. This means that all data is written at a single site, and then replicated to other sites for read access. Updates to the source are propagated to the replicas either by the replication mechanism or by a separate consistency management service.

A lot of challenges in Grid computing revolve around providing access to different types of resources. Foster, Kesselman and Tuecke [88] have proposed a Grid architecture for resource sharing among different entities based around the concept of *Virtual Organizations (VOs)*. A VO is formed when different organisations pool resources and collaborate in order to achieve a common goal. A VO defines the resources available for the participants and the rules for accessing and using the resources and the conditions under which the resources may be used. Resources here include not just compute, storage or network resources, they may also be software, scientific instruments or business data. A VO also provides protocols and mechanisms for applications to determine the suitability and accessibility of available resources. In practical terms, a VO may be created using mechanisms such as Certificate Authorities (CAs) and trust chains for security, replica management systems for data organisation and retrieval and centralised scheduling mechanisms for resource management.

The existence of VOs impacts the design of Data Grid architectures in many ways. For example, a VO may be stand alone or may be composed of a hierarchy of regional, national and international VOs. In the latter case, the underlying Data Grid may have a corresponding hierarchy of repositories and the replica discovery and management systems will be structured accordingly. More importantly, sharing of data collections is guided by the relationships that exist between the VOs that own each of the collections. Subsequent sections will discuss how Data Grids are differentiated by such design choices and how

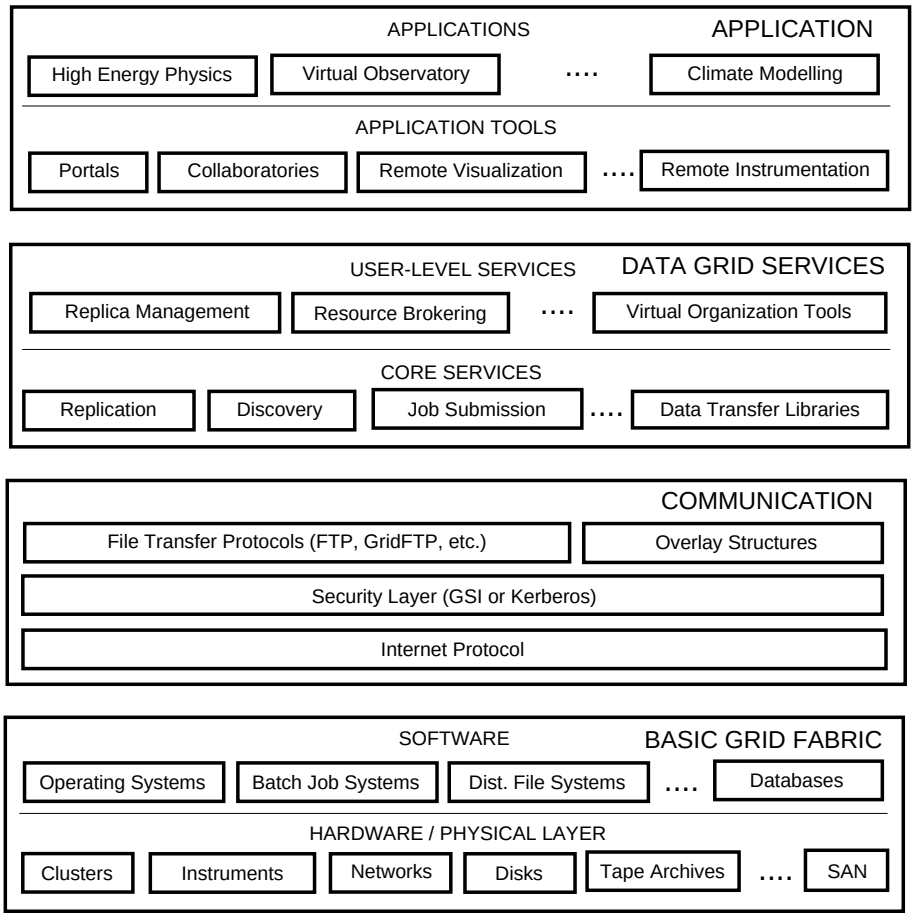


Figure 2.2: A Layered Architecture.

these affect underlying technologies.

2.2.1 Layered Architecture

The components of a Data Grid can be organised in a layered architecture as shown in Figure 2.2. This architecture follows from similar definitions given by Foster et al. [88] and Baker et al. [23]. Each layer builds on the services offered by the lower layer in addition to interacting and co-operating with components and the same level (eg. Resource broker invoking VO tools). These layers can be described from bottom to top as below:

1. *Grid Fabric*: Consists of the distributed computational resources (clusters, supercomputers), storage resources (RAID arrays, tape archives) and instruments (telescope, accelerators) connected by high-bandwidth networks. Each of the resources runs system software such as operating systems, job submission and management

systems and relational database management systems (RDBMS).

2. *Communication*: Consists of protocols used to query resources in the Grid Fabric layer and to conduct data transfers between them. These protocols are built on core communication protocols such as TCP/IP and authentication protocols such as PKI (Public Key Infrastructure), passwords or SSL (Secure Sockets Layer). The cryptographic protocols allow verification of users' identities and ensure security and integrity of transferred data. These security mechanisms form part of the Grid Security Infrastructure (GSI) [87]. File transfer protocols such as GridFTP (Grid File Transfer Protocol), among others, provide services for efficient transfer of data between two resources on the Data Grid. Application-specific overlay structures provide efficient search and retrieval capabilities for distributed data by maintaining distributed indexes.

3. *Data Grid Services*: Provides services for managing and processing data in a Data Grid. The core level services such as replication, data discovery and job submission provide transparent access to distributed data and computation. User-level services such as resource brokering and replica management provide mechanisms that allow for efficient resource management hidden behind inituitive commands and APIs (Application Programming Interfaces). VO tools provide easy way to perform functions such as adding new resources to a VO, querying the existing resources and managing users' access rights.

4. *Applications*: Specific services cater to users by invoking services provided by the layers below and customising them to suit the target domains such as high energy physics, biology and climate modelling. Each domain provides a familiar interface and access to services such as visualisation. Portals are web interfaces that provide single-point access to available VO services and domain-specific applications and tools. Collaboratories [121] have similar intent and also provide applications that allow users to conduct joint operations with their colleagues.

The security layer and Data Grid services provide applications uniform access to resources in the Fabric layer while abstracting out much of the inherent complexity and heterogeneity. Formation of VOs requires interoperability between the resources and components

that are provided by different participants. This motivates the use of standard protocols and service interfaces for information exchange among VO entities. Service interfaces themselves have to be separated from implementation details and have to be described in language- and platform-independent format. Realization of these requirements have led the Grid computing research community, through forums such as Global Grid Forum (GGF), to adopt a new Open Grid Services Architecture (OGSA) [86] that is based on the *Web services* paradigm. Web services are self-contained, stateless components that use standard mechanisms for representation and exchange of data. OGSA builds on Web service properties such as vendor and platform neutral service definition using XML (eXtensible Markup Language) [39] and standard communication protocols such as SOAP (Simple Object Access Protocol) to create *Grid services*. Grid services are standardized Web service interfaces that provide Grid capabilities in a secure, reliable and stateful manner. Grid services may also be potentially transient and service instances support service lifetime management and state notification. OGSA utilizes standard Web service mechanisms for discovering and invoking Grid services.

The OGSA Data Services [89] deal with accessing and managing data resources in a Grid environment. A *data service* implements one or more of a set of basic interfaces that describe the data and provide operations to manipulate it. The same data can be represented in many ways by different data services that implement different set of operations and data attributes. This abstract view of data created by a data service is termed *data virtualisation*. Subsequent efforts through the Data Access and Integration Services Working Group (DAIS-WG) at GGF have produced a set of more concrete standards [18] for representing data through services. These standards provide the consumers of these services the advantage of being isolated from the inner workings of Data Grids and therefore, be able to develop complex applications that consume data in different ways.

2.2.2 Related Data-Intensive Research

Three related distributed data-intensive research areas that share similar requirements, functions and characteristics are described below. These have been chosen because of the similar properties and requirements that they share with Data Grids.

Content Delivery Network

A Content Delivery Network (CDN) [68, 71] consists of a "collection of (non-origin) servers that attempt to offload work from origin servers by delivering content on their behalf" [124]. That is, within a CDN, client requests are satisfied from other servers distributed around the Internet (also called edge servers) that cache the content originally stored at the source (origin) server. A client request is rerouted from the main server to an available server closest to the client likely to host the content required [71]. This is done by providing a DNS (Domain Name System) server that resolves the client DNS request to the appropriate edge server. If the latter does not have the requested object then it retrieves the data from the origin server or another edge server. The primary aims of a CDN are, therefore, load balancing to reduce effects of sudden surges in requests, bandwidth conservation for objects such as media clips and reducing the round-trip time to serve the content to the client. CDNs are generally employed by Web content providers and commercial providers such as Akamai Inc., Speedera Inc. and IntelliDNS Inc. have built dedicated infrastructure to serve multiple clients. However, CDNs haven't gained wide acceptance for data distribution because, currently CDN infrastructures are proprietary in nature and owned completely by the providers.

Peer-to-Peer Network

Peer-to-peer (P2P) networks [156] are formed by ad hoc aggregation of resources to form a decentralised system within which each peer is autonomous and depends on other peers for resources, information and forwarding requests. The primary aims of a P2P network are: to ensure scalability and reliability by removing the centralised authority, to ensure redundancy, to share resources and to ensure anonymity. An entity in a P2P network can join or leave anytime and therefore, algorithms and strategies have to be designed keeping in mind the volatility and requirements for scalability and reliability. P2P networks have been designed and implemented for many target areas such as compute resource sharing (e.g. SETI@Home [15], Compute Power Market [47]), content and file sharing (Napster, Gnutella, Kazaa [57]) and collaborative applications such as instant messengers (Jabber [112]). Milojicic et al. [145] present a detailed taxonomy and survey of peer-to-peer

systems. The discussion here focuses mostly on content and file-sharing P2P networks as these involve data distribution. Such networks have mainly focused on creating efficient strategies to locate particular files within a group of peers, to provide reliable transfers of such files in the face of high volatility and to manage high load caused due to demand for highly popular files. Currently, major P2P content sharing networks do not provide an integrated computation and data distribution environment.

Distributed Databases

A distributed database (DDB) [53, 157] is a logically organised collection of data stored at different sites of a computer network. Each site has a degree of autonomy, is capable of executing a local application, and also participates in the execution of a global application. A distributed database can be formed either by taking an existing single site database and splitting it over different sites (top-down approach) or by federating existing database management systems so that they can be accessed through a uniform interface (bottom-up approach) [185]. The latter are also called multidatabase systems. Varying degrees of autonomy are possible within DDBs ranging from tightly-coupled sites to complete site independence. Distributed databases have evolved to serve the needs of large organisations which need to remove the need for a centralised computer centre, to interconnect existing databases, to replicate databases to increase reliability, and to add new databases as new organisational units are added. This technology is very robust and provides distributed transaction processing, distributed query optimisation and efficient management of resources. However, these systems cannot be employed in their current form at the scale of Data Grids envisioned as they have strong requirements for ACID (Atomicity, Consistency, Isolation and Durability) properties [99] to ensure that the state of the database remains consistent and deterministic.

2.2.3 Analysis of Data-Intensive Networks

This section compares the data-intensive paradigms described in the previous sections with Data Grids in order to bring out the uniqueness of the latter by highlighting their respective similarities and differences. Also, each of these areas have their own mature

solutions which may be applicable to the same problems in Data Grids either wholly or with some modification. These properties are summarised in Table 2.1 and are explained below:

Purpose - Considering the purpose of the network, it is generally seen that P2P content sharing networks are vertically integrated solutions for a single goal (for example, file-sharing). CDNs are dedicated to caching web content so that clients are able to access it faster. DDBs are used for integrating existing diverse databases to provide a uniform, consistent interface for querying and/or replicating existing databases for increasing reliability or throughput. In contrast to these single purpose networks, Data Grids are primarily created for enabling collaboration through sharing of distributed resources including data collections and support various activities including data transfer and computation over the same infrastructure. The overall goal is to bring together existing disparate resources in order to obtain benefits of aggregation.

Aggregation - All the networks are formed by aggregating individual nodes to form a distributed system. The aggregation can be created through an *ad hoc* process wherein nodes subscribe to the network without prior arrangements or a *specific* process where they are brought together for a particular purpose. The aggregation can be *stable* or *dynamic*. P2P networks, by definition, are ad hoc in nature with nodes entering and leaving at will. A CDN provider creates the infrastructure by setting up dedicated servers for caching content. DDBs are created by either federating existing databases or by establishing a tightly-coupled network of databases by a single organisation. In the case of a CDN or a DDB system, the entire network is managed by a single entity that has the authority to add or remove nodes and therefore, these have stable configurations. Data Grids are created by institutions forming VOs by pooling their resources for achieving a common goal. However, within a Data Grid, dynamic configurations are possible due to introduction or removal of resources and services.

Organisation - The organisation of a CDN is hierarchical with the data flowing from the origin to the edges. Data is cached at the various edge servers to exploit locality of data requests. There are many models for organisation of P2P content sharing network

Table 2.1: Comparison between various data distribution networks.

Property	**P2P (Content sharing)**	**CDN**	**DDB**	**Data Grids**
Purpose	File sharing	Reducing web latency	Integrating existing databases, Replicating database for reliability & throughput	Analysis, collaboration
Aggregation	Ad hoc, Dynamic	Specific, Stable	Specific, Stable	Specific, Dynamic
Organisation	Centralised, two-level hierarchy, flat	Hierarchical	Centralised, federation	Hierarchy, federation, monadic, hybrid
Data Access Type	Mostly read with frequent writes	Read only	Equally read and write	Mostly read with rare writes
Data Discovery	Central directory, Flooded requests or document routing	HTTP Request	Relational Schemas	Catalogues
Latency Management & Performance	Replication, Caching, Streaming	Caching, Streaming	Replication, Caching	Replication, Caching, Streaming, Pre-staging, Network tuning
Consistency Requirements	Weak	Strong (read only)	Strong	Weak
Transaction Support	None	None currently	Yes	None currently
Computational Requirements	None currently	None (Client-side)	Transaction Processing	Data Production and Analysis
Autonomy	Operational, Participation	None (Dedicated)	Operational (federated)	Access, Operational, Participation
Heterogeneity	System, Structural	System	System	System, - Syntactic, Structural, Semantic
Management Entity	Individual	Single Organisation	Single Organisation	VO
Security Requirements	Anonymity	Data Integrity	Authentication, Authorisation, Data Integrity	Authentication, Authorisation, Data Integrity

and these are linked to the searching methods for files within the network. Within Napster, a peer has to connect to a centralised server and search for an available peer that has the required file. The two peers then directly communicate with each other. Gnutella avoids the centralised directory by having a peer broadcast its request to its neighbours and so on until the peer with the required file is obtained. Kazaa and FastTrack limit the fan-out in Gnutella by restricting broadcasts to SuperPeers who index a group of peers. Freenet [59] uses content-based hashing, in which a file is assigned a hash based on its contents and nearest neighbour search is used to identify the required document. Thus, three different models of organisation, viz. centralised, two-level hierarchy and flat (structured and unstructured) can be seen in the examples presented above. Distributed databases provide a relational database management interface and are therefore organised accordingly. Global relations are split into fragments that are allocated to either one or many physical sites. In the latter case, replication of fragments is carried out to ensure reliability of the database. While distribution transparency may be achieved within top-down databases, it may not be the case with federated databases that have varying degrees of heterogeneity and autonomy. As will be shown in the taxonomy section, there are 4 different kinds of organisation present in a Data Grid: monadic, hierarchical, federated, and hybrid combinations of these.

Data Access Type - Access type distinguishes the type of data access operations conducted within the network. P2P content sharing networks are mostly read-only environments and write operations occur when an entity introduces new data into the network or creates copies of existing data. CDNs are almost exclusively read-only environments for end-users and updating of data happens at the origin servers only. In DDBs, data is both read and written frequently. Data Grids are similar to P2P networks as they are mostly read-only environments into which either data is introduced or existing data is replicated. However, a key difference is that depending on application requirements, Data Grids may also support updating of data replicas if the source is modified.

Data Discovery - Another distinguishing property is how the data is discovered within the network. The three approaches for searching within P2P networks have been mentioned previously. Current research focuses on the document routing model and the four algorithms proposed for this model: Chord [190], CAN [176], Pastry [177] and

Tapestry [222]. CDNs fetch data which has been requested by a browser through HTTP (Hyper Text Transfer Protocol). DDBs are organised using the same relational schema paradigm as single-site databases and thus, data can be searched for and retrieved using SQL (Structured Query Language). Data in Data Grids are organised into catalogues which map the logical description of data to the actual physical representation. One form of these catalogues is the replica catalogue which contains a (possibly) one-to-many mapping from the logical (or device-independent) filename to the actual physical filenames of the datasets. Data can be located by querying these catalogues and resolving the physical locations of the logical datasets.

In addition to these mechanisms, the use of metadata for searching data is supported by certain individual products in each of the four data-intensive networks. Data can be queried for based on attributes such as description or content type. In Data Grids, metadata catalogues offer another means for querying for data. In such cases, metadata has to be curated properly as otherwise it would affect the efficiency and accuracy of data discovery. The role of metadata and catalogues will be looked at in detail in the next chapter.

Latency Management & Performance - A key element of performance in distributed data-intensive networks is the manner in which they reduce the latency of data transfers. Some of the techniques commonly used in this regard are replicating data close to the point of consumption, caching of data, streaming data and pre-staging the data before the application starts executing. Replication is different from caching as the former involves creation and maintenance of copies of data at different places in the network depending on access rates or other criteria while the latter involves creating just one copy of the data close to the point of consumption. Replication is, therefore, done mostly from the source of the data (provider side) and caching is done at the data consumer side. While both replication and caching seek to increase performance by reducing latency, the former also aims to increase reliability by creating multiple backup copies of data.

CDNs employ caching and streaming to enhance performance especially for delivering media content [182]. While several replication strategies have been suggested for a CDN, Karlsson and Mahalingam [114] experimentally show that caching provides equivalent or even better performance than replication. In the absence of requirements for consistency or availability guarantees in CDNs, computationally expensive replication

strategies do not offer much improvement over simple caching methods. P2P networks also employ replication, caching and streaming of data in various degrees. Replication and caching are used in distributed database systems for optimizing distributed query processing [119].

In Data Grids, all of the techniques mentioned are implemented in one form or another. However, additionally, Data Grids are differentiated by the requirement for transfer of massive datasets. This is either absent in the other data-intensive networks or is not considered while designing these networks. This motivates use of high-speed data transfer mechanisms that have separation of data communication - that is, sending of control messages happens separately from the actual data transfer. In addition, features such as parallel and striped data transfers among others, are required to further reduce time of data movement. Optimization methods to reduce the amount of data transfers, such as accessing data close to the point of its consumption, are also employed within Data Grids.

Consistency - Consistency is an important property which determines how "fresh" the data is. Grids and P2P networks generally do not provide strong consistency guarantees because of the overhead of maintaining locks on huge volumes of data and the ad hoc nature of the network respectively. Among the exceptions for Data Grids is the work of Dullmann et al. [72] which discusses a consistency service for replication in Data Grids. In P2P networks, Oceanstore [125] is a distributed file system that provides strong consistency guarantees through expensive locking protocols. In CDNs, while the data in a cache may go stale, the system always presents the latest version of the data when the user requests it. Therefore, the consistency provided by a CDN is strong.

Distributed databases, as mentioned before, have strong requirements for satisfying ACID properties. While these requirements can be relaxed in the case of unstable conditions such as those found in mobile networks [163], even then the semantics for updating are much stricter within distributed databases than in other distribution networks. Also, updates are more frequent and can happen from within any site in the network. These updates have to be migrated to other sites in the network so that all the copies of the data are synchronised. There are two methods for updating that are followed [98]: *lazy*, in which the updates are asynchronously propagated and *eager*, in which the copies are synchronously updated.

Transaction Support - A transaction is a set of operations (actions) such that all of them succeed or none of them succeed. Transaction support implies the existence of check-pointing and rollback mechanisms so that a database or data repository can be returned to its previous consistent state in case of failure. It follows from the discussion of the previous property that transaction support is essential for distributed databases. CDNs have no requirements for transaction support as they only support read only access to data to the end users. P2P Networks and Data Grids currently do not have support for recovery and rollback. However, efforts are on to provide transaction support within Data Grids to provide fault tolerance for distributed transactions [207].

Computational Requirements - Computational requirements in data intensive environments originate from operations such as query processing, applying transformations to data and processing data for analysis. CDNs are exclusively data-oriented environments with a client accessing data from remote nodes and processing it at its own site. While current P2P content sharing networks have no processing of the data, it is possible to integrate such requirements in the future. Computation within DDBs involves transaction processing which can be conducted in two ways: the requested data is transmitted to the originating site of the transaction and the transaction is processed at that site, or the transaction is distributed among the different nodes which have the data. High volumes of transactions can cause heavy computational load within DDBs and there are a variety of optimisation techniques to deal with load balancing in parallel and distributed databases.

Data Grids have heavy computational requirements that are caused by workloads involving analysis of datasets. Many operations in Data Grids, especially those involving analysis, can take long intervals of time (measured in hours or even days). This is in contrast to the situation within DDBs where the turnaround time of requests is short and for applications such as OLTP (On Line Transaction Processing), measured in milliseconds. High performance computing sites, that generally constitute existing Data Grids, are shared facilities and are oversubscribed most of the time. Therefore, application execution within Data Grids has to take into account the time to be spent in queues at these sites as well.

Autonomy - Autonomy deals with the degree of independence allowed to different nodes within a network. However, there could be different types and different levels of

autonomy provided [13, 185]. *Access autonomy* allows a site or a node to decide whether to grant access to a user or another node within the network. *Operational autonomy* refers to the ability of a node to conduct its own operations without being overridden by external operations of the network. *Participation autonomy* implies that a node has the ability to decide the proportion of resources it donates to the network and the time it wants to associate or disassociate from the network. Data Grid nodes have all the three kinds of autonomy to the fullest extent. While nodes in a P2P network do not have fine-grained access controls against users, they have maximum independence in deciding how much share will they contribute to the network. CDNs are dedicated networks and so, individual nodes have no autonomy at all. Tightly coupled databases retain all control over the individual sites whereas multidatabase systems retain control over local operations.

Heterogeneity - Network environments encompass heterogeneous hardware and software configurations that potentially use different protocols. This impacts applications which have to be engineered to work across multiple interfaces, multiple data formats and multiple protocols wherever applicable. Interoperability of the system therefore, refers to the degree of transparency a system provides for a user to access this information while being unaware of the underlying complexity.

Heterogeneity can also be split into many types depending on the differences at various levels of the network stack. Koutrika [120] has identified four different types of heterogeneity in the case of data sources within digital libraries.

1. *System heterogeneity* - arises from different hardware platforms and operating systems.
2. *Syntactic heterogeneity* - arises from the presence of different protocols and encodings used with the system.
3. *Structural heterogeneity* - originates from the data organised according to different models and schemas.
4. *Semantic heterogeneity* - originates from different meanings given to the same data, especially because of the use of different metadata schemas for categorising the data.

It can be seen from the definitions of the data-intensive networks that the same classifi-

cation is applicable in the current context. System heterogeneity is a feature of all the data-intensive networks discussed here. Though P2P networks, CDNs and DDBs can simultaneously store data in different formats, they require the establishment of common protocols within individual networks. CDNs and DDBs are also homogeneous when it comes to structure of data as they enforce common schema (Web content schema for CDNs and relational schema for DDBs). P2P networks offer structural and semantic heterogeneity as they unify data from various sources and allow the user to query across all of the available data.

The existence of different components including legacy and otherwise, that speak a variety of protocols and store data in their own (sometimes proprietary) formats with little common structure or consistent metadata information means that Data Grids contain data that is syntactically, structurally and semantically heterogeneous. However, where Data Grids truly differ from other data intensive networks in this regard is the level of interoperability required. Users within a Data Grid expect to have an integrated view of data which abstracts out the underlying complexity behind a simple interface. Through this interface, they would require manipulating the data by applying transformations or by conducting analysis. The results of the analysis or transformation need to be viewed and may provide feedback to conduct further operations. This means that not only should a Data Grid provide interoperability between different protocols and systems, it should also be able to extract meaningful information from the data according to users' requirements. This is different to P2P content sharing networks where the user only queries for datasets matching a particular criterion and downloads them.

Management Entity - The management entity administers the tasks for maintaining the aggregation. Generally, this entity is a collection of the stakeholders within the distribution network. While this body usually does not have control over individual nodes, nevertheless, it provides services such as a common data directory for locating content and an authentication service for the users of the network. For the Data Grid, the concept of VOs has already been discussed in the previous section. Though entities in a P2P network are independent, a central entity may provide directory service as in the case of Napster. CDNs are owned and maintained by a corporation or a single organisation. Likewise, DDBs are also maintained by single organisations even though the constituent

databases may be independent.

Security Requirements - Security requirements differ depending on perspective. In a data distribution network, security may have to be ensured against corruption of content (data integrity), for safeguarding users' privacy (anonymity), and for resources to verify users' identities (authentication). P2P Networks such as Freenet are more concerned with preserving anonymity of the users as they may be breaking local censorship laws. A CDN primarily has to verify data integrity as access for manipulating data is granted only to the content provider. Users have to authenticate against a DDB for carrying out queries and transactions and data integrity has to be maintained for deterministic operation.

Since Data Grids are multi-user environments with shared resources, the main security concerns are authentication of both users and resources, and granting of permissions for specific types of services to a user (authorisation). Data Grids resources are also spread among various administrative entities and therefore, accepting security credentials of a user also involves trusting the authority that issued the credentials in the first place. Many VOs have adopted community-based authorization [6] where the VO itself provides the credentials or certifies certain authorities as trusted and sets the access rights for the user. While these are issues within Grids in general, Data Grids also need verification while accessing data and need to guard against malicious operations on data while in transit. Also, more elaborate access controls than those currently deployed in general Grids are needed for safeguarding confidential data in Data Grids.

2.3 Discussion and Summary

Thus, it can be seen that though Data Grids share many characteristics with other types of data intensive network computing technologies, they are differentiated by heavy computational requirements, wider heterogeneity, higher autonomy of individual entities and the presence of VOs. Most of the current Data Grid implementations focus on scientific applications. Recent approaches have, however, explored the integration of the above-mentioned technologies within Data Grids to take advantage of the strengths that they offer in areas such as data discovery, storage management and data replication. This is possible as Data Grids already encompass and build on diverse technologies. Foster and

Iamnitchi [81] discuss the convergence of P2P and Grid computing and contend that the latter will be able to take advantage of the failure resistance and scalability offered by the former which gains from the experience in managing diverse and powerful resources, complex applications and the multitude of users with different requirements. Ledlie et al. [132] present a similar view and discuss the areas of aggregation, algorithms and maintenance where P2P research can be beneficial to Grids. Practical Grid technologies such as Narada Brokering [90] have used P2P methods for delivering a scalable event-service.

Based on the detailed investigation conducted on the architecture of Data Grids, a taxonomy has been developed and is discussed in the next chapter.

Chapter 3

A Taxonomy of Data Grids

The rapid emergence of Data Grids in scientific and commercial settings has led to a variety of systems offering solutions for dealing with distributed data-intensive applications. Unfortunately, this has also led to difficulty in evaluating these solutions because of the confusion in pinpointing their exact target areas. The taxonomy provided in Section 3.1 breaks down the overall research in Data Grids into specialised areas and categorizes each of them in turn. The following section, Section 3.2 then surveys some representative projects and publications and classifies them according to the taxonomy.

3.1 Taxonomy

The properties of a Data Grid are determined by its underlying organization. The organizational attributes not only determine the placement, replication, and propagation of data throughout a Data Grid but also the interaction of the users with the infrastructure. The actual work of transferring, processing and managing data is done by the core mechanisms such as data transport, data replication and resource management. These core mechanisms, therefore, define the capabilities of a Data Grid. Accordingly, this taxonomy is split into four sub-taxonomies as shown in Figure 3.1. The first sub-taxonomy is from the point of view of Data Grid organization. This classifies ongoing scientific Data Grid efforts worldwide. The next sub-taxonomy deals with the transport technologies used within Data Grids. This not only covers well-known file transfer protocols but

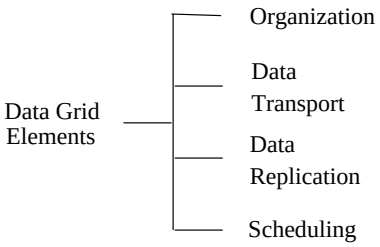


Figure 3.1: Data Grid Elements.

also includes other means of managing data transportation. A scalable, robust and intelligent replication mechanism is crucial to the smooth operation of a Data Grid and the sub-taxonomy presented next takes into account concerns of Grid environments such as metadata and the nature of data transfer mechanisms used. The last sub-taxonomy categorizes resource allocation and scheduling research and looks into issues such as locality of data.

While each of the areas of data transport, replica management and resource management are independent fields of research and merit detailed investigations on their own, in this chapter, these are studied from the point of view of the specific requirements of Data Grid environments that have been provided in the previous chapter.

3.1.1 Data Grid Organization

Figure 3.2 shows a taxonomy based on the various organizational characteristics of Data Grid projects. These characteristics are central to any Data Grid and manifest in different ways in different systems.

Model - The model is the manner in which data sources are organised in a system. A variety of models are in place for the operation of a Data Grid. These are dependent on: the source of data, whether single or distributed, the size of data and the mode of sharing. Four of the common models found in Data Grids are shown in Figure 3.3 and are discussed as follows:

1. *Monadic:* This is the general form of a Data Grid in which all the data is gathered at a central repository that then answers user queries and provides the data. The data can be from many sources such as distributed instruments and sensor networks and is made available through a centralised interface such as a web portal which

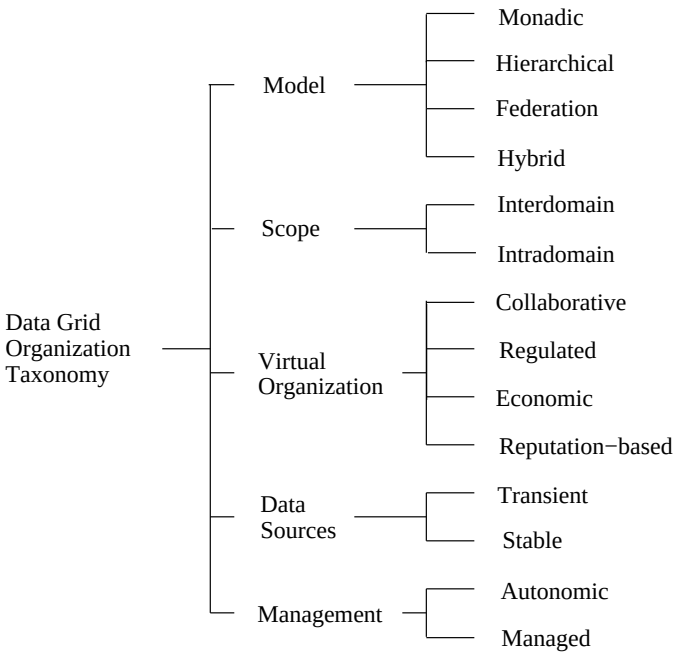
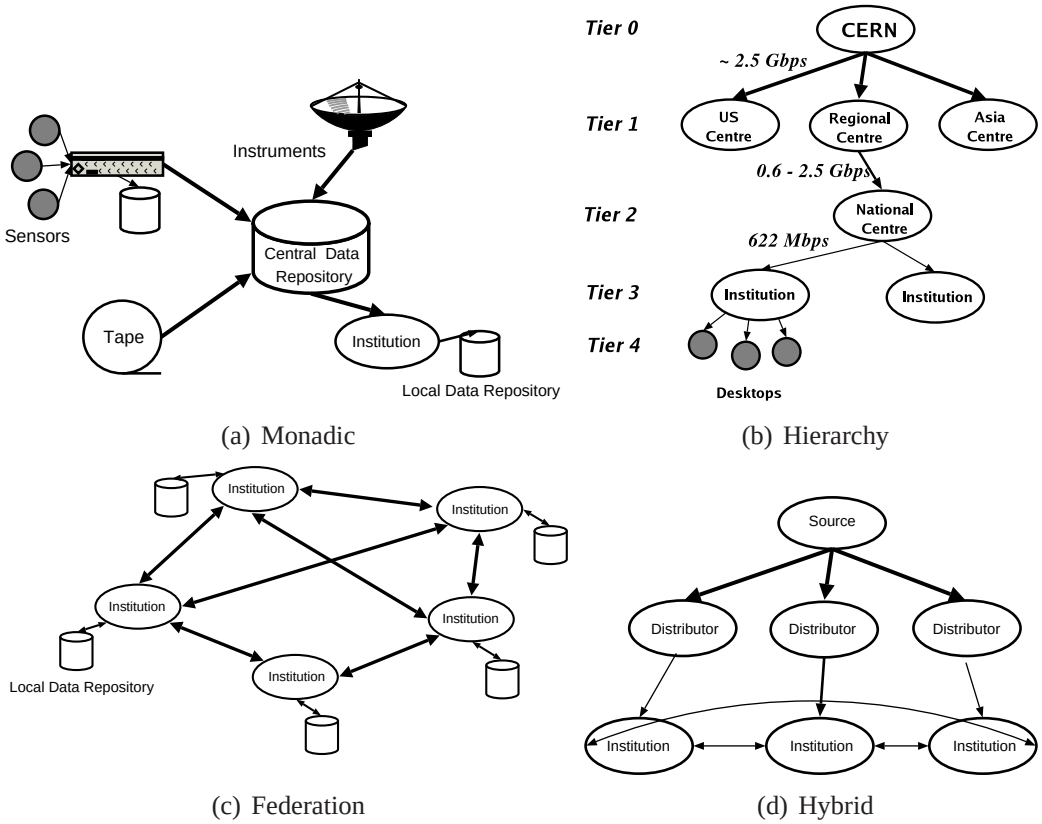


Figure 3.2: Data Grid Organization Taxonomy.



(a) Monadic (b) Hierarchy

(c) Federation (d) Hybrid

Figure 3.3: Possible models for organization of Data Grids.

also verifies users and checks for authorization. This model is shown in Figure 3.3(a) and has been applied in the NEESgrid (Network for Earthquake Engineering Simulation) project [161] in the United States.

The difference between this and other models of Data Grid organisation is that there is only a single point for accessing the data. In contrast, within other models, the data can be wholly or partially accessed at different points where it is made available through replication. The central repository may be replicated in this case for fault tolerance but not for improving locality of data. Thus, this model serves better in scenarios where the overhead of replication is not compensated by an increase in efficiency of data access such as the case wherein all accesses are local to a particular region.

2. *Hierarchical:* This model is used in Data Grids where there is a single source for data and the data has to be distributed across collaborations worldwide. For example, the MONARC (Models of Networked Analysis at Regional Centres) group within CERN has proposed a tiered infrastructure model for distribution of CMS data [5]. This model is presented in Figure 3.3(b) and specifies requirements for transfer of data from CERN to various groups of physicists around the world. The first level is the compute and storage farm at CERN which stores the data generated from the detector. This data is then distributed to sites, called Regional Centres (RCs), located around the world. From the RCs, the data is then passed downstream to the national and institutional centres and finally onto the physicists. A Tier 1 (RC) or a Tier 2 (national) centre has to satisfy certain bandwidth, storage and computational requirements as shown in the figure.

   The massive amounts of data generated in these experiments motivate the need for a robust data distribution mechanism. Also, researchers at participating institutions may be interested only in subsets of the entire dataset that may be identified by querying using metadata. One advantage of this model is that maintaining consistency is much simpler as there is only one source for the data.

3. *Federation:* The federation model [171] is presented in Figure 3.3(c) and is prevalent in Data Grids created by institutions who wish to share data in already existing

databases. One example of a federated Data Grid is the BioInformatics Research Network (BIRN) [35] in the United States. Researchers at a participating institution can request data from any one of the databases within the federation as long as they have the proper authentication. Each institution retains control over its local database. Varying degrees of integration can be present within a federated Data Grid. For example, Moore et al. [148] discuss about 10 different types of federations that are possible using the Storage Resource Broker (SRB) [26] in various configurations. The differences are based on the degree of autonomy of each site, constraints on cross-registration of users, degree of replication of data and degree of synchronization.

4. *Hybrid:* Hybrid models that combine the above models are beginning to emerge as Data Grids mature and enter into production usage. These come out of the need for researchers to collaborate and share products of their analysis. A hybrid model of a hierarchical Data Grid with peer linkages at the edges is shown in Figure 3.3(d).

Scope - The scope of a Data Grid can vary depending on whether it is restricted to a single domain (*intradomain*) or if it is a common infrastructure for various scientific areas (*interdomain*). In the former case, the infrastructure is adapted to the particular needs of that domain. For example, special analysis software may be made available to the participants of a domain-specific Data Grid. In the latter case, the infrastructure provided will be generic.

Virtual Organizations - Data Grids are formed by VOs and therefore, the design of VOs reflects on the social organization of the Data Grid. A VO is *collaborative* if it is created by entities who have come together to share resources and collaborate on a single goal. Here, there is an implicit agreement between the participants on the usage of resources. A *regulated* VO may be controlled by a single organization which lays down rules for accessing and sharing resources. In an *economy-based* VO, resource providers enter into collaborations with consumers due to profit motive and the latter select providers based on their advertised level of service and cost. In such cases, service-level agreements dictate the rights of each of the participants. A *reputation-based* VO may be created by inviting entities to join a collaboration based on the level of services

that they are known to provide.

Data Sources - Data sources in a Data Grid may be *transient* or *stable*. A scenario for a transient data source is a satellite which broadcasts data only at certain times of the day. In such cases, applications need to be aware of the short life of the data stream. As will be revealed later, most of the current Data Grid implementations have always-on data sources such as mass storage systems or production databases. In future, with diversification, Data Grids are also expected to handle transient data sources.

Management - The management of a Data Grid can be *autonomic* or *managed*. Present day Data Grids require plenty of human intervention for tasks such as resource monitoring, user authorization and data replication. However, research is leading to autonomic [20, 158] or self-organizing, self-governing systems whose techniques may find applications in future Data Grids.

3.1.2 Data Transport

The data transport mechanism is one of the fundamental technologies underlying a Data Grid. Data transport involves not just movement of bits across resources but also other aspects of data access such as security, access controls and management of data transfers. A taxonomy for data transport mechanisms within Data Grids is shown in Figure 3.4.

Functions - Data transport in Grids can be modelled as a three-tier structure that is similar to the networking stacks such as the Open System Interconnection (OSI) reference model. At the bottom is the *Transfer Protocol* that specifies a common language for two nodes in a network to initiate and control data transfers. This tier takes care of simple bit movement between two hosts on a network. The most widely-used transport protocols in Data Grids are FTP (File Transfer Protocol) [167] and GridFTP [10]. The second tier is an optional *Overlay Network* that takes care of routing the data. An overlay network provides its own semantics over the Internet protocol to satisfy a particular purpose. In P2P networks, overlays based on distributed hash tables provide a more efficient way of locating and transferring files [14]. Overlay networks in Data Grids provide services such as storage in the network, caching of data transfers for better reliability and the ability for applications to manage transfer of large datasets. The topmost tier provides application-

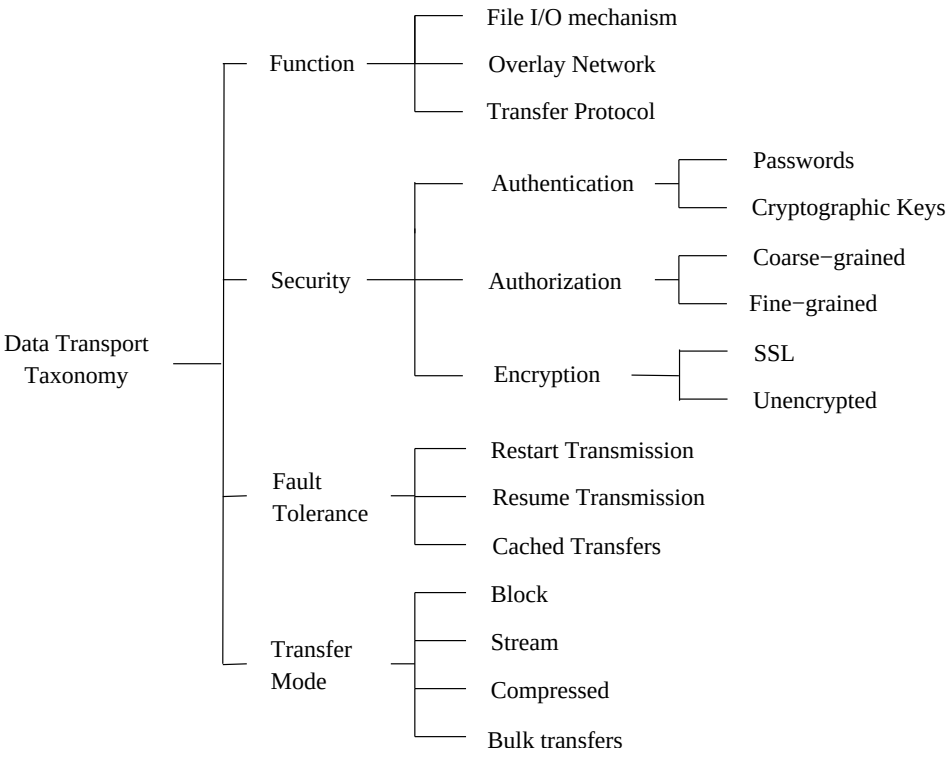


Figure 3.4: Data Transport Taxonomy.

specific functions such as *File I/O*. A file I/O mechanism allows an application to access remote files as if they are locally available. This mechanism presents to the application a transparent interface through APIs that hide the complexity and the unreliability of the networks. A data transport mechanism can therefore perform one of these functions.

Security - Security is an important requirement while accessing or transferring files to ensure proper authentication of users, file integrity and confidentiality. Transport security can be divided into three main categories: *authentication* and *authorization* of users and *encryption* of data transfer. Authentication can be based on either *passwords* or symmetric or asymmetric *public key* cryptographic protocols such as Kerberos [153] or X.509 [107] mechanisms respectively. In the context of data movement, authorization of users is enforced by mechanisms such as access controls on the data that is to be transferred. *Coarse-grained* authorization methods use traditional methods such as UNIX file permissions to restrict the number of files or collections that are accessible to the user. However, expansion of Data Grids to fields such as medical research that have strict controls on the distribution of data have led to requirements for *fine-grained* authorization. Such requirements include restricting the number of accesses even for authorised users, delegating

read and write access rights to particular files or collections and flexible ownership of data [148]. Fine-grained access control methods that may be employed to achieve these requirements include time- and usage-limited tickets, Access Control Lists (ACLs), Role Based Access Control (RBAC) methods [181] and Task-Based Authorization Controls (TBAC) [206]. Data encryption may be present or absent within a transfer mechanism. The most prevalent form of data encryption is through SSL (Secure Sockets Layer) [212].

Fault Tolerance - Fault tolerance is also an important feature that is required in a Data Grid environment especially when transfers of large data files occur. Fault tolerance can be subdivided into restarting over, resuming from interruption and providing caching. *Restarting* the transfer all over again means that the data transport mechanism does not provide any failure tolerance. However, all data in transit would be lost and there is a slight overhead for setting up the connection again. Protocols such as GridFTP allow for *resuming* transfers from the last byte acknowledged. Overlay networks provide *caching* of transfers via store-and-forward protocols. In this case, the receiver does not have to wait until the connections are restored. However, caching reduces performance of the overall data transfer and the amount of data that can be cached is dependent on the storage policies at the intermediate network points.

Transfer Mode - The last category is the transfer modes supported by the mechanism. *Block*, *stream* and *compressed* modes of data transfer have been available in traditional data transmission protocols such as FTP. However, it has been argued that transfers of large datasets such as those that are anticipated within Data Grids are restricted by vanilla FTP and underlying Internet protocols such as Transmission Control Protocol (TCP) which were initially designed for low bandwidth, high latency networks. As such, these are unable to take advantage of the capabilities of high bandwidth, optical fibre networks that are available for Data Grid environments [134]. Therefore, several optimisations have been suggested for improving the performance of data transfers in Grid environments by reducing latency and increasing transfer speed. Some of them are listed below:

- *Parallel data transfer* - is the ability to use multiple data streams over the same channel to transfer a file. This also saturates available bandwidth in a channel while completing transfer.

- *Striped data transfer* - is the ability to use multiple data streams to simultaneously access different blocks of a file that is partitioned among multiple storage nodes (also called *striping*). This distributes the access load among the nodes and also improves bandwidth utilisation.
- *Auto-resizing of buffers* - is the ability to automatically resize sender and receiver TCP window and buffer sizes so that the available bandwidth can be more effectively utilised.
- *Container operations* - is the ability to aggregate multiple files into one large dataset that can be transferred or stored more efficiently. The efficiency gains come from reducing the number of connections required to transfer the data and also, by reducing the initial latency.

The first three are protocol-specific optimisations while the last one is applied to the transfer mechanism. These enhancements are grouped under the *bulk transfer* mode. A mechanism may support more than one mode and its suitability for an application can be gauged by the features it provides within each of the transfer modes.

3.1.3 Data Replication and Storage

A Data Grid is a geographically-distributed collaboration in which all members require access to the datasets produced within the collaboration. Replication of the datasets is therefore a key requirement to ensure scalability of the collaboration, reliability of data access and to preserve bandwidth. Replication is bounded by the size of storage available at different sites within the Data Grid and the bandwidth between these sites. A replica management system therefore ensures access to the required data while managing the underlying storage.

A replica management system, shown in Figure 3.5, consists of storage nodes which are linked to each other via high-performance data transport protocols. The replica manager directs the creation and management of replicas according to the demands of the users and the availability of storage, and a catalog or a directory keeps track of the replicas and their locations. The catalog can be queried by applications to discover the number and the locations of available replicas of a particular dataset. In some systems, the man-

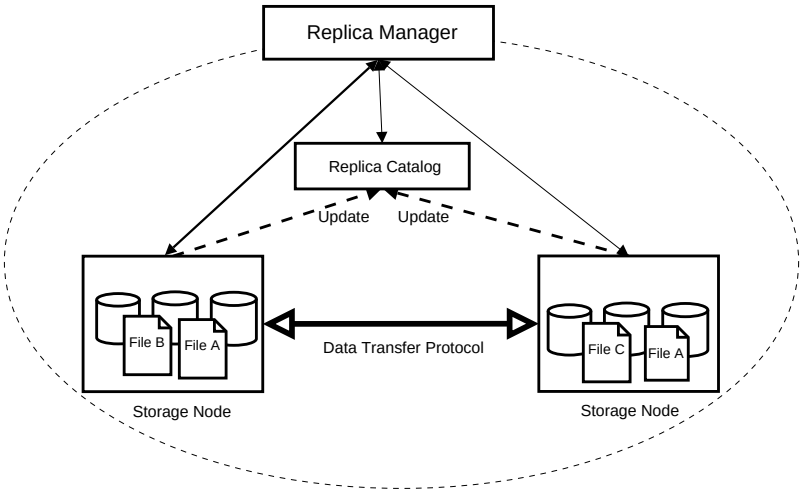


Figure 3.5: A Replica Management Architecture.

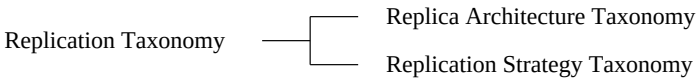


Figure 3.6: Replication Taxonomy.

ager and the catalog are merged into one entity. Client-side software generally consists of a library that can be integrated into applications and a set of commands or GUI utilities that are built on top of the libraries. The client libraries allow querying of the catalog to discover datasets and to request replication of a particular dataset.

The important elements of a replication mechanism are therefore the architecture of the system and the strategy followed for replication. The first categorization of Data Grid replication is therefore, based on these properties as is shown in Figure 3.6. The architecture of a replication mechanism can be further subdivided into the categories shown in Figure 3.7.

Model & Topology - The model followed by the system largely determines the way in which the nodes are organized and the method of replication. A *centralized* system would have one master replica which is updated and the updates are propagated to the other nodes. A *decentralized* or peer-to-peer mechanism would have many copies, all of which need to be synchronized with each other. Nodes under a replica management system can be organised in a variety of topologies which can be grouped chiefly into three: *Hierarchy*, *Flat* and *Hybrid*. Hierarchical topologies have tree-like structure in which updates propagate through definite paths. Flat topologies are found within P2P

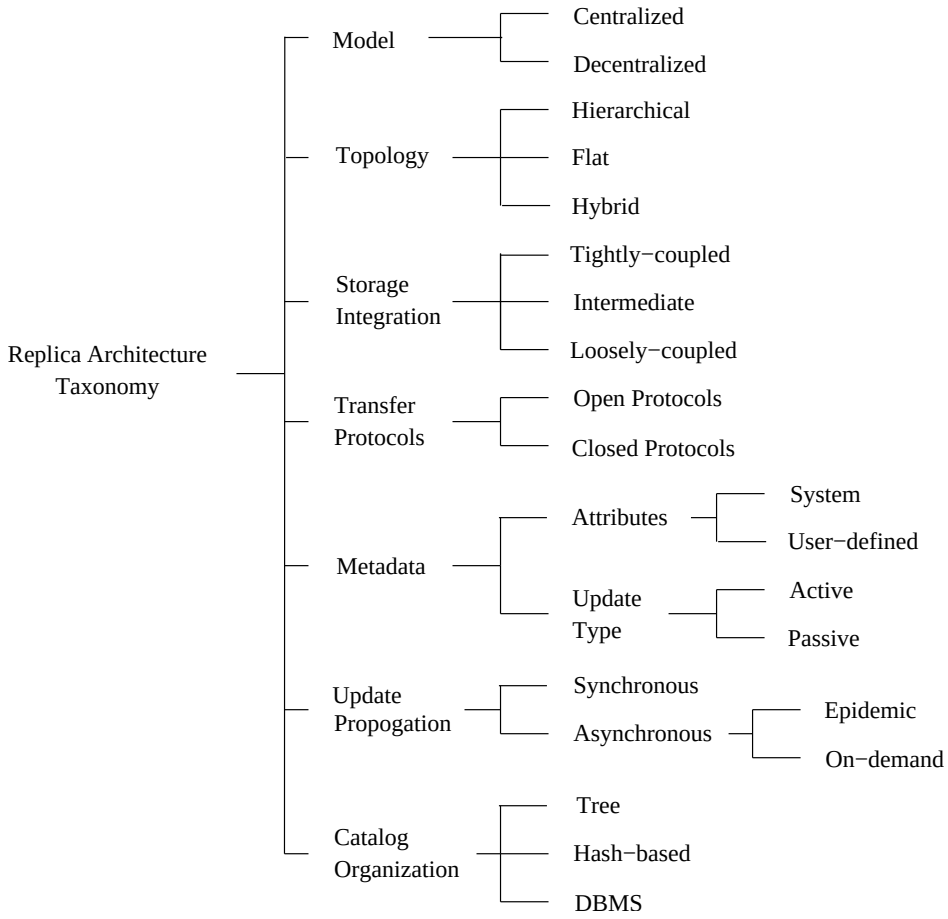


Figure 3.7: Replica Architecture Taxonomy.

systems and progression of updates is entirely dependent on the arrangements between the peers. These can be both structured and unstructured. Hybrid topologies can be achieved in situations such as a hierarchy with peer connections at different levels as has been discussed by Lamehamedi et al. [129].

Storage Integration - The relation of replication to storage is very important and determines the scalability, robustness, adaptability and applicability of the replication mechanism. *Tightly-coupled* replication mechanisms that exert fine-grained control over the replication process are tied to the storage architecture on which they are implemented. The replication system controls the filesystem and I/O mechanism of the local disk. The replication is conducted at the level of processes and is often triggered by a read or write request to a file at a remote location by a program. Such systems more or less try to behave as a distributed file system such as NFS (Network File System) as they aim to provide transparent access to remote files to applications. An example of such a mech-

anism is Gfarm [199]. *Intermediately-coupled* replication systems exert control over the replication mechanism but not over the storage resources. The filesystems are hosted on diverse storage architectures and are controlled by their respective systems. However, the replication is still initiated and managed by the mechanism, and therefore it interacts with the storage system at a very low-level. Such mechanisms work at the level of individual applications and data transfer is handled by the system. While replication can be conducted transparent to users and applications, it is also possible for the latter to direct the mechanism, and thereby, control the replication process. Example of such a system is the SRB. *Loosely-coupled* replication mechanisms are superimposed over the existing filesystems and storage systems. The mechanism exerts no control over the filesystem. Replication is initiated and managed by applications and users. Such mechanisms interact with the storage systems through standard file transfer protocols and at a high level. The architecture is capable of complete heterogeneity.

Transfer Protocols - The data transport protocols used within replica management systems is also a differentiating characteristic. *Open protocols* for data movement such as GridFTP allow clients to transfer data independent of the replica management system. The replicated data is accessible outside of the replica management system. Systems that follow *closed* or unpublished protocols restrict access to the replicas to their client libraries. Tightly-coupled replication systems are mostly closed in terms of data transfer. RLS (Replica Location Service) [55] and GDMP (Grid Data Mirroring Pilot) [180] use GridFTP as their primary transport mechanism. But the flip-side to having open protocols is that the user or the application must take care of updating the replica locations in the catalog if they transfer data without involving the replication management system.

Metadata - It is difficult, if not impossible, for users to identify particular datasets out of hundreds and thousands that may be present in a large, distributed, collection. From this perspective, having proper metadata about the replicated data aids users in querying for datasets based on attributes that are more familiar to them. Metadata can have two types of attributes: one is *system-dependent* metadata, which consists of file attributes such as creation date, size on disk, physical location(s) and file checksum and the other is *user-defined* attributes which consist of properties that depend on the experiment or VO that the user is associated with. For example in a High-Energy Physics experiment, the

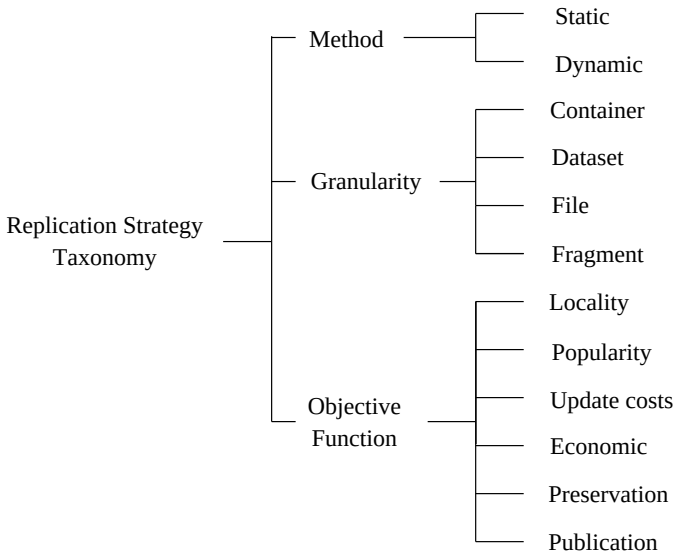


Figure 3.8: Replication Strategy Taxonomy.

metadata could describe attributes such as experiment date, mode of production (simulation or experimental) and event type. The metadata can be *actively* updated by the replica management system or else updated *passively* by the users when they create new replicas, modify existing ones or add a new file to the catalog.

Replica Update Propagation - Within a Data Grid, data is generally updated at one site and the updates are then propagated to the rest of its replicas. This can be in *synchronous* or in *asynchronous* modes. While synchronous updating is followed in databases, it is not practiced in Data Grids because of the expensive wide-area locking protocols and the frequent movement of massive data required. Asynchronous updating can be epidemic [103], that is, the primary copy is changed and the updates are propagated to all the other replicas or it can be on-demand as in Grid Data Mirroring Pilot (GDMP) [189] wherein replica sites subscribe to update notifications at the primary site and decide themselves when to update their copies.

Catalog Organization - A replica catalog can be distinguished on the basis of its organization. The catalog can be organized as a *tree* as in the case of LDAP (Lightweight Directory Access Protocol) based catalogs such as the Globus Replica Catalog [7]. The data can be catalogued on the basis of *document hashes* as has been seen in P2P networks. However, SRB and others follow the approach of storing the catalog within a *database*.

Replication strategies determine when and where to create a replica of the data. These

strategies are guided by factors such as demand for data, network conditions and cost of transfer. The replication strategies can be categorized as shown in Figure 3.8.

Method - The first classification is based on whether the strategies are *static* or *dynamic*. Dynamic strategies adapt to changes in demand and bandwidth and storage availability but induce overhead due to larger number of operations that they undertake as these are run at regular intervals or in response to events (for example, increase in demand for a particular file). Dynamic strategies are able to recover from failures such as network partitioning. However, frequent transfers of massive datasets that result due to such strategies can lead to strain on the network resources. There may be little gain from using dynamic strategies if the resource conditions are fairly stable in a Data Grid over a long time. Therefore, in such cases, static strategies are applied for replication.

Granularity - The second classification relates to the level of subdivision of data that the strategy works with. Replication strategies that deal with multiple files at the same time work at the granularity of *datasets*. The next level of granularity is individual *files* while there are some strategies that deal with smaller subdivisions of files such as objects or *fragments*.

Objective Function - The third classification deals with the objective function of the replication strategy. Possible objectives of a replication strategy are to maximise the *locality* or move data to the point of computation, to exploit *popularity* by replicating the most requested datasets, to minimize the *update costs* or to maximize some *economic* objective such as profits gained by a particular site for hosting a particular dataset versus the expense of leasing the dataset from some other site. *Preservation* driven strategies provide protection of data even in the case of failures such as corruption or obsolescence of underlying storage media or software errors. Another possible objective function for a replication strategy is to ensure effective *publication* by propagating new files to interested clients.

3.1.4 Resource Allocation and Scheduling

The requirements for large datasets and the presence of multiple replicas of these datasets scattered at geographically-distributed locations makes scheduling of data-intensive jobs

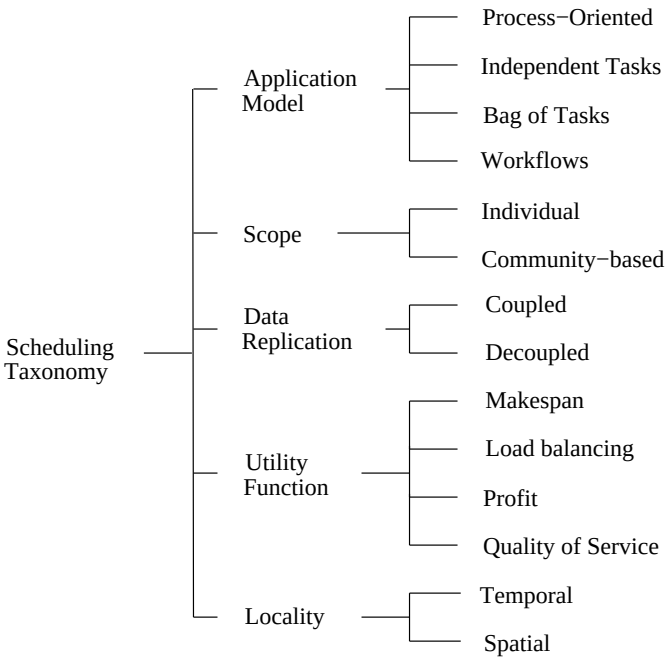


Figure 3.9: Data Grid Scheduling Taxonomy.

different from that of computational jobs. Schedulers have to take into account the bandwidth availability and the latency of transfer between a computational node to which a job is going to be submitted and the storage resource(s) from which the data required is to be retrieved. Therefore, the scheduler needs to be aware of any replicas close to the point of computation and if the replication is coupled to the scheduling, then create a new copy of the data. A taxonomy for scheduling of data-intensive applications is shown in Figure 3.9. The categories are explained as follows:

Application Model - Scheduling strategies can be classified by the application model that they are targeted towards. Application models are defined in the manner in which the application is composed or distributed for scheduling over Grid resources. These can range from fine-grained levels such as processes to coarser levels such as individual tasks to sets of tasks such as workflows. Here, a task is considered as the smallest independent unit of computation. Each level has its own scheduling requirements. *Process-oriented* applications are those in which the data is manipulated at the process level. Examples of such applications are MPI (Message Passing Interface) programs that execute over global Grids [82]. *Independent tasks* having different objectives are scheduled individually and it is ensured that each of them get their required share of resources. A *Bag-of-Tasks*

(BoT) application consists of a set of independent tasks all of which must be executed successfully subject to certain common constraints such as a deadline for the entire application. Such applications arise in parameter studies [1] wherein a set of tasks is created by running the same program on different inputs. In contrast, a *workflow* is a sequence of tasks in which each task is dependent on the results of its predecessor(s). The products of the preceding tasks may be large datasets themselves (for example, a simple two-step workflow could be a data-intensive simulation task and the task for analysis of the results of simulation). Therefore, scheduling of individual tasks in a workflow requires careful analysis of the dependencies and the results to reduce the amount of data transfer.

Scope - Scope relates to the extent of application of the scheduling strategy within a Data Grid. If the scope is *individual*, then the scheduling strategy is concerned only with meeting the objectives from a user's perspective. In a multi-user environment therefore, each scheduler would have its own independent view of the resources that it wants to utilise. A scheduler is aware of fluctuations in resource availability caused by other schedulers submitting their jobs to common resources and it strives to schedule jobs on the least-loaded resources that can meet its objectives. With the advent of VOs, efforts have moved towards *community-based* scheduling in which schedulers follow policies that are set at the VO level and enforced at the resource level through service level agreements and allocation quotas [73, 214].

Data Replication - The next classification relates to whether job scheduling is *coupled* to data replication or not. Assume a job is scheduled to be executed at a particular compute node. When job scheduling is coupled to replication and the data has to be fetched from remote storage, the scheduler creates a copy of the data at the point of computation so that future requests for the same file that come from the neighbourhood of the compute node can be satisfied more quickly. Not only that, in the future, any job dealing with that particular data will be scheduled at that compute node if available. However, one requirement for a compute node is to have enough storage to store all the copies of data. While storage management schemes such as LRU (Least Recently Used) and FIFO (First In First Out) can be used to manage the copies, the selection of compute nodes is prejudiced by this requirement. There is a possibility that promising computational resources may be disregarded due to lack of storage space. Also, the process of creation

of the replica and registering it into a catalog adds further overheads to job execution. In a decoupled scheduler, the job is scheduled to a suitable computational resource and a suitable replica location is identified to request the data required. The storage requirement is transient, that is, disk space is required only for the duration of execution. A comparison of decoupled against coupled strategies by Ranganathan and Foster [173] has shown that decoupled strategies promise increased performance and reduce the complexity of designing algorithms for Data Grid environments.

Utility function - A job scheduling algorithm tries to minimize or maximize some form of a utility function. The utility function can vary depending on the requirements of the users and architecture of the distributed system that the algorithm is targeted at. Traditionally, scheduling algorithms have aimed at reducing at the total time required for computing all the jobs in a set, also called its *makespan*. *Load balancing* algorithms try to distribute load among the machines so that no machine is either idle or overburdened. Scheduling algorithms with economic objectives try to maximize the users' economic utility usually expressed as some *profit* function that takes into account economic costs of executing the jobs on the Data Grid. Another possible objective is to meet the *Quality-of-Service (QoS)* requirements specified by the user. QoS requirements that can be specified include minimising the cost of computation, meeting a deadline, meeting strict security requirements and/or meeting specific resource requirements [42].

Locality - Exploiting the locality of data has been a tried and tested technique for scheduling and load-balancing in parallel programs [102, 144, 166] and in query processing in databases [184, 191]. Similarly, data grid scheduling algorithms can be categorized as whether they exploit the *spatial* or *temporal* locality of the data requests. Spatial locality is locating a job in such a way that all the data required for the job is available on data hosts that are located close to the point of computation. Temporal locality exploits the fact that if data required for a job is close to a compute node, subsequent jobs which require the same data are scheduled to the same node. Spatial locality can also be termed as "moving computation to data" and temporal locality can be called as "moving data to computation". It can be easily seen that schedulers which couple data replication to job scheduling exploit the temporal locality of data requests.

3.2 Mapping of Taxonomy to Various Data Grid Systems

This section classifies various Data Grid research projects according to the taxonomies developed in Section 3.1. While the list of example systems is not exhaustive, it is representative of the classes that have been discussed. The projects in each category have been chosen based on several factors such as broad coverage of application areas, project support for one or more applications, scope and visibility, large-scale problem focus and ready availability of documents from project web pages and other sources.

3.2.1 Data Grid Projects

This space studies and analyses the various Data Grid projects that have been developed for various application domains around the world. While many of these projects cover aspects of Data Grid research such as middleware development, advanced networking and storage management, however, here the focus is only on those projects which are involved in setting up infrastructure. A list of these projects and a brief summary about each of them is provided in Table 3.1. These are also classified according to the taxonomy provided in Figure 3.2.

Some of the scientific domains that are making use of Data Grids are as follows:

High Energy Physics (HEP) The computational and storage requirements for HEP experiments have already been covered in previous literature [41]. Other than the four experiments at the LHC already mentioned, the Belle experiment at KEK, Japan, the BaBar experiment at the Stanford Linear Accelerator Center (SLAC) and the CDF and D0 experiments at Fermi National Laboratory, US are also adopting Data Grid technologies for their computing infrastructure. There have been numerous Grid projects around the world that are setting up the infrastructure for physicists to process data from HEP experiments. Some of these are the LHC Computing Grid (LCG) led by CERN, the Particle Physics Data Grid (PPDG) and Grid Physics Network (GriPhyN) in the United States, GridPP in the UK and Belle Analysis Data Grid (BADG) in Australia. These projects have common features such as a tiered model for distributing the data, shared facilities for computing and storage and personnel dedicated towards managing the infrastructure. Some of them are entering

Table 3.1: Data Grid Projects around the world.

Name	Domain	Grid Type	Remarks	Country / Region
LCG [136]	High Energy Physics	Hierarchical model, Intradomain, Collaborative VO, Stable Sources, Managed	To create and maintain a data movement and analysis infrastructure for LHC users.	Global
EGEE [76]	High Energy Physics, Biomedical Sciences	Hierarchical model, Interdomain, Collaborative VO, Stable Sources, Managed	To create a seamless common Grid infrastructure to support scientific research.	Global
BIRN [35]	Bio-Informatics	Federated model, Intradomain, Collaborative VO, Stable Sources, Managed	To foster collaboration in biomedical science through sharing of data.	United States
NEESgrid [161]	Earthquake Engineering	Monadic model, Intradomain, Collaborative VO, Transient Sources, Managed	To enable scientists to carry out experiments in distributed locations and analyse data through a uniform interface.	United States
GriPhyn [22]	High Energy Physics	Hierarchical model, Intradomain, Collaborative VO, Stable Sources, Managed	To create an infrastructure integrating computational and storage facilities for high energy physics experiments.	United States
Grid3 [94]	Physics, Biology	Hierarchical model, Interdomain, Collaborative VO, Stable Sources, Managed	To provide a uniform, scalable and managed grid infrastructure for science applications	United States
BioGrid, Japan [34]	Protein Simulation, Brain Activity Analysis	Federated model, Intradomain, Collaborative VO, Stable Sources, Managed	Grid infrastructure for medical and biological research.	Japan
Virtual Observatories [196]	Astronomy	Federated model, Intradomain, Collaborative VO, Stable Sources, Managed	Infrastructure for accessing diverse astronomy observation and simulation archives through integrated mechanisms.	Global
Earth System Grid [9]	Climate Modelling	Federated model, Intradomain, Collaborative VO, Stable Sources, Managed	Integrating computational and analysis resources for next generation climate research.	United States
GridPP [108]	High Energy Physics	Hierarchical model, Intradomain, Collaborative VO, Stable Sources, Managed	Grid infrastructure for Particle Physics in the UK.	United Kingdom
eDiaMoND [37]	Breast Cancer Treatment	Federated model, Intradomain, Collaborative VO, Stable Sources, Managed	To provide medical professionals and researchers access to distributed databases of mammogram images.	United Kingdom
Belle Analysis Data Grid [216]	High Energy Physics	Hierarchical model, Intradomain, Collaborative VO, Stable Sources, Managed	Grid infrastructure for Australian physicists involved in the Belle and ATLAS experiments.	Australia

or are being tested for production usage.

Astronomy The community of astrophysicists around the globe are setting up Virtual Observatories for accessing the data archives that has gathered by telescopes and instruments around the world. These include the National Virtual Observatory (NVO) in the US, Australian Virtual Observatory, Astrophysical Virtual Observatory in Europe and AstroGrid in the UK [197]. The International Virtual Observatory Alliance (IVOA) is coordinating these efforts around the world for ensuring interoperability. Commonly, these projects provide uniform access to data repositories along with access to software libraries and tools that may be required to analyse the data. Other services that are provided include access to high-performance computing facilities and visualization through desktop tools such as web browsers. Other astronomy grid projects include those being constructed for the LIGO (Laser Interferometer Gravitational-wave Observatory) [130] and SDSS (Sloan Digital Sky Survey) [187] projects.

BioInformatics The increasing importance of realistic modeling and simulation of biological processes coupled with the need for accessing existing databases has led to Data Grid solutions being adopted by bioinformatics researchers worldwide. These projects involve federating existing databases and providing common data formats for the information exchange. Examples of these projects are BioGrid project in Japan for online brain activity analysis and protein folding simulation, the eDiaMoND project in the UK for breast cancer treatment and the BioInformatics Research Network (BIRN) for imaging of neurological disorders using data from federated databases.

Earth Sciences Researchers in disciplines such as earthquake engineering and climate modeling and simulation are adopting Grids to solve their computational and data requirements. NEESgrid is a project to link earthquake researchers with high performance computing and sensor equipment so that they can collaborate on designing and performing experiments. Earth Systems Grid aims to integrate high-performance computational and data resources to study the petabytes of data resulting from climate modelling and simulation.

Table 3.2: Comparison between various data transport technologies.

Project	**Function**	**Security**	**Fault Tolerance**	**Transfer Mode**
GASS	File I/O	PKI, Unencrypted, Coarse-grained	Caching	Block, Stream append
IBP	Overlay Mechanism	Password, Unencrypted, Coarse-grained	Caching	Block
FTP	Transfer Protocol	Password, Unencrypted, Coarse-grained	Restart	All
SFTP	Transfer Protocol	PKI, SSL, Coarse-grained	Restart	All
GridFTP	Transfer Protocol	PKI, SSL, Coarse-grained	Resume	All
Kangaroo	Overlay Mechanism	PKI, Unencrypted, Coarse-grained	Caching	Block
Legion	File I/O	PKI, Unencrypted, Coarse-grained	Caching	Block
SRB	File I/O	PKI, SSL, Fine-grained	Restart	Block, Stream, Bulk transfer

3.2.2 Data Transport Technologies

Within this subsection, various projects involved in data transport over Grids are discussed and classified according to the taxonomy provided in Section 3.1.2. The data transport technologies studied here range from protocols such as FTP to overlay methods such as Internet Backplane Protocol to file I/O mechanisms. Each technology has unique properties and is representative of the categories in which it is placed. A summary of these technologies and their categorization is provided in Table 3.2.

GASS

Global Access to Secondary Storage (GASS) [33] is a data access mechanism provided within the Globus toolkit for reading local data at remote machines and for writing data to remote storage and moving it to a local disk. The goal of GASS is to provide a uni-

form remote I/O interface to applications running at remote resources while keeping the functionality demands on both the resources and the applications limited.

GASS conducts its operations via a file cache which is an area on the secondary storage where the remote files are stored. When a remote file is requested by an application for reading, GASS by default fetches the entire file into the cache from where it is opened for reading as in a conventional file access. It is retained in the cache as long as applications are accessing it. While writing to a remote file, the file is created or opened within the cache where GASS keeps track of all the applications writing to it via reference count. When the reference count is zero, the file is transferred to the remote machine. Therefore, all operations on the remote file are conducted locally in the cache, which reduces demand on bandwidth. A large file can be *prestaged* into the cache, that is, fetched before an application requests it for reading. Similarly, a file can be transferred out via *poststaging*. GASS operations also allow access to permitted disk areas other than the file cache and are available through an API and also through Globus commands. GASS is integrated with the Globus Resource Access and Monitoring (GRAM) service [64] and is used for staging executables, staging in files and retrieving the standard output and error streams of the jobs.

GASS provides a limited ability for data transfer between remote nodes. As it prefetches the entire file into the cache, it is not suitable as a transfer mechanism for large data files (of GigaByte upwards) as the required cache capacity might not be available. Also, it does not provide features such as file striping, third-party transfer, TCP tuning, etc. provided by protocols such as GridFTP. However, because of its lightweight functionality, it is suitable for applications where the overhead of setting up a GridFTP connection dominates.

IBP

Internet Backplane Protocol (IBP) [27, 164] allows applications to optimize data transfer and storage operations by controlling data transfer explicitly by storing the data at intermediate locations. IBP uses a “store-and-forward” protocol to move data around the network. Each of the IBP nodes has a temporary buffer into which data can be stored for a fixed amount of time. Applications can manipulate these buffers so that data is moved to locations close to where it is required.

IBP is modelled after the Internet Protocol. The data is handled in units of fixed-size byte arrays which are analogous to IP datagrams or network packets. Just as IP datagrams are independent of the data link layer, byte arrays are independent of the underlying storage nodes. This means that applications can move data around without worrying about managing storage on the individual nodes. IBP also provides a global addressing space that is based on global IP addressing. Thus, any client within an IBP network can make use of any IBP node.

IBP can also be thought of as a virtualisation layer or as an access layer built on top of storage resources. IBP provides access to heterogeneous storage resources through a global addressing space in terms of fixed block sizes thus making access to data independent of the storage method and media. The storage buffers can grow to any size, and thus the byte arrays can also be thought of as files which live on the network.

IBP also provides a client API and libraries that provide semantics similar to UNIX system calls. A client connects to an IBP "depot", or a server, and requests storage allocation. In return, the server provides it three *capabilities*: for reading from, writing to and managing the allocation. Capabilities are cryptographically secure byte strings which are generated by the server. Subsequent calls from the client must make use of the same capabilities to perform the operations. Thus, capabilities provide a notion of security as a client can only manipulate its own data. Capabilities can be exchanged between clients as they are text. Higher-order aggregation of byte arrays is possible through exNodes which are similar to UNIX inodes. exNodes allow uploading, replicating and managing of files on a network with an IBP layer above the networking layer [165].

Beyond the use of capabilities, IBP does not have an address mechanism that keeps track of every replica generated. There is no directory service that keeps track of every replica and no information service that can return the IBP address of a replica once queried. Though exNodes store metadata, IBP itself does not provide a metadata searching service. Therefore, IBP is a low-level storage solution that functions just above the networking layer.

FTP

FTP (File Transfer Protocol) [167] is one of the fundamental protocols for data movement in the Internet. FTP is therefore ubiquitous and every operating system ships with an FTP client.

FTP separates the process of data transfer into two channels, the control channel used for sending commands and replies between a client and a server and the data channel through which the actual transfer takes place. The FTP commands set up the data connection by specifying the parameters such as data port, mode of transfer, data representation and structure. Once the connection is set up the server then initiates the data transfer between itself and the client. The separation of control and data channels also allows third-party transfers to take place. A client can open two control channels to two servers and direct them to start a data transfer between themselves bypassing the client. Data can be transferred in three modes: stream, block and compressed. In the stream mode, data is transmitted as is and it is the responsibility of the sending host to notify the end of stream. In the block mode, data is transferred as a series of blocks preceded by header bytes. In the compressed mode, a preceding byte denotes the number of replications of the following byte and filler bytes are represented by a single byte.

Error recovery and restart within FTP does not cover corrupted data but takes care of data lost due to loss of network or a host or of the FTP process itself. This requires the sending host to insert markers at regular intervals within the data stream. A transmission is restarted from the last marker sent by the sender before the previous transfer crashed. However, restart is not available within the stream transfer mode. Security within FTP is very minimal and limited to the control channel. The username and password are transmitted as clear text and there is no facility for encrypting data while in transit within the protocol. This limits the use of FTP for confidential transfers.

Numerous extensions to FTP have been proposed to offset its limitations. RFCs 2228 [105] and 2389 [100] propose security and features extensions to FTP respectively. However, these are not implemented by popular FTP servers such as wu-ftpd. SSH File Transfer Protocol (SFTP) [93] is a secure file transfer protocol that uses the Secure Shell (SSH) Protocol for both authentication and data channel encryption. SFTP is designed to

be both a transfer protocol and a remote file system access protocol. However, it does not support features required for high-performance data transfer such as parallel and striped data transfer, resuming interrupted transmissions or tuning of TCP parameters.

GridFTP

GridFTP [8, 10] extends the default FTP protocol by providing features that are required in a Data Grid environment. The aim of GridFTP is to provide secure, efficient, and reliable data transfer in Grid environments.

GridFTP extends the FTP protocol by allowing GSI and Kerberos based authentication. GridFTP provides mechanisms for parallel and striped data transfers and supports partial file transfer that is, the ability to access only part of a file. It allows changing the sizes of the TCP buffers and congestion windows to improve transfer performance. Transfer of massive data-sets is prone to failures as the network may exhibit transient behaviour over long periods of time. GridFTP sends restart markers indicating a byte range that has been successfully written by the receiver every 5 seconds over the control channel. In case of a failure, transmission is resumed from the point indicated by the last restart marker received by the sender.

GridFTP provides these features by extending the basic FTP protocol through new commands, features and a new transfer mode. The Striped Passive(SPAS) command is an extension to the FTP PASV command wherein the server presents a list of ports to connect to rather than just a single port. This allows for multiple connections to download the same file or for receiving multiple files in parallel. The Extended Retrieve (ERET) command supports partial file transfer among other things. The Set Buffer (SBUF) and AutoNegotiate Buffer (ABUF) extensions allow the resizing of TCP buffers on both client and server sides. The Data Channel Authentication (DCAU) extension provides for encrypting of data channels for confidential file transfer. DCAU is used only when the control channel is authenticated through RFC 2228 [105] mechanisms. Parallel and striped data transfers are realised through a new transfer mode called the extended block mode (mode E). The sender notifies the receiver of the number of data streams by using the End of Data (EOD) and End of Data Count (EODC) codes. The EODC code signifies how many EOD codes should be received to consider a transfer closed. An additional protocol is therefore re-

quired from the sender side to ensure that the receiver obtains the data correctly. GridFTP implements RFC 2389 [100] for negotiation of feature sets between the client and the server. Therefore, the sender first requests the features supported by the receiver and then sets connection parameters accordingly. GridFTP also supports restart for stream mode transfers which is not provided in the vanilla FTP protocol.

The only public implementation for the GridFTP server-side protocols is provided in the Globus Toolkit [83]. The Globus GridFTP server is a modified wu-ftpd server that supports most of GridFTP's features except for striped data transfer and automatic TCP buffer size negotiation. The Globus Toolkit provides libraries and APIs for clients to connect to GridFTP servers. A command-line tool, *globus-url-copy*, developed using these libraries, functions as a GridFTP client. Another examples of a GridFTP clients is the UberFTP [152] client from NCSA.

Evaluation of GridFTP protocols alongside FTP has shown that using the additional features of GridFTP increases performance of data transfer [75]. Particularly, the usage of parallel threads dramatically improves the transfer speed over both loaded and unloaded networks. Also, parallel transfers saturate the bandwidth thus improving the link utilisation.

Kangaroo

Kangaroo [202] is an end-to-end data movement protocol that aims to improve the responsiveness and reliability of large data transfers within the Grid. The main idea in Kangaroo is to conduct the data transfer as a background process so that failures due to server crashes and network partitions are handled transparently by the process instead of the application having to deal with them.

Kangaroo uses memory and disk storage as buffers to which data is written to by the application and moved out by a background process. The transfer of data is performed concurrently with CPU bursts thereby improving utilization. The transfer is conducted through *hops*, or stages where an intermediate server is introduced between the client and the remote storage from which the data is to be read or written. Data received by the intermediate stage is spooled into the disk from where it is copied to the next stage by a background process called the *mover*. This means that a client application writing data to

a remote storage is isolated from the effects of a network crash or slow-down as long as it can keep writing to the disk spool. However, it is also possible for a client to write data to the destination server directly over a TCP connection using the Kangaroo primitives.

Kangaroo services are provided through an interface which implements four simple file semantics: `get` (non-blocking read), `put`(non-blocking write), `commit` (block until writes have been delivered to the next stage) and `push` (block until all writes are delivered to the final destination). However, this only provides weak consistency since it is envisioned for grid applications in which data flow is primarily in one direction. As can be seen, Kangaroo is an output-oriented protocol which primarily deals with reliability of data transfer between a client and a server.

The design of Kangaroo is similar to that of IBP even though their aims are different. Both of them use store-and-forward method as a means of transporting data. However, while IBP allows applications to explicitly control data movement through a network, Kangaroo aims to keep the data transfer hidden through the usage of background processes. Also, IBP uses byte arrays whereas Kangaroo uses the default TCP/IP datagrams for data transmission.

Legion I/O model

Legion [54] is a object-oriented grid middleware for providing a single system image across a collection of distributed resources. The I/O mechanism within Legion [215] aims to provide transparent access to files stored on distributed resources through APIs and daemons that can be used by native and legacy applications alike.

Resources within the Legion system are represented by objects. BasicFileObjects correspond to files in a conventional file system while ContextObjects correspond to directories. However, these are separated from the actual file system. A datafile is copied to a BasicFileObject to be registered within the context space of Legion. The context space provides location-independent identifiers which are bound to human-readable context names. This presents a single address space and hierarchy from which users can request files without worrying about their location. Also, the representation of BasicFileObject is system-independent, and therefore provides interoperability between heterogeneous systems.

Access to a Legion file object is provided through various means. Command-line utilities provide a familiar interface to the Legion context space. Application developers can use APIs which closely mimic C and C++ file primitives and Unix system calls. For legacy codes, a buffering interface is provided through which applications can operate on local files copied from the Legion objects and the changes are copied back. Another method is to use a modified NFS daemon that translates client request to appropriate Legion invocations.

Security for file transfer is provided through means of X.509 proxies which are delegated to the file access mechanisms [79]. Data itself is not encrypted while in transit. Caching and prefetching is implemented for increasing performance and to ensure reliability.

SRB I/O

The Storage Resource Broker (SRB) [26] developed at the San Diego Supercomputing Centre (SDSC) focuses on providing a uniform and transparent interface to heterogenous storage systems that include disks, tape archives and databases. A study of SRB as a replication mechanism is provided in the following section, however, this description focuses on the data transport mechanism within SRB.

Data transport within SRB provides features such as parallel data transfers for performing bulk data transfer operations across geographically distributed sites. If parallel transfer is requested by a client, the SRB server creates a number of parallel streams depending on bandwidth availability and speed of the storage medium. SRB also allows streaming data transfer and supports bulk ingest operations in which multiple files are sent using multiple streams to a storage resource. SRB I/O can transfer multiple files as containers and can stage files from tape or archival storage to disk storage for faster access.

SRB provides for strong security mechanisms supported by fine-grained access controls on data. Access security is provided through credentials such as passwords or public key and private key pair which can be stored within MCAT itself. Controlled authorization for read access is provided through tickets issued by users who have control privileges on data. Tickets are time-limited or use-limited. Users can also control access privileges

Table 3.3: Comparison between various data replication mechanisms.

Project	**Model**	**Topology**	**Storage Integration**	**Data Transport**	**Metadata**	**Update**	**Catalog**
Gfarm	Centralised	Hierarchy	Tightly-coupled	Closed	System, Active	Async., epidemic	DBMS
RLS	Centralised	Hierarchy	Loosely-coupled	Open	User-defined, Passive	Async., on-demand	DBMS
GDMP	Centralised	Hierarchy	Loosely-coupled	Open	User-defined, Passive	Async., on-demand	DBMS
SRB	Decentralised	Flat	Intermediate	Closed	User-defined, Passive	Async., on-demand	DBMS

along a collection hierarchy.

SRB also provides support for remote procedures. These are operations which can be performed on the data within SRB without having to move it. Remote procedures include execution of SQL queries, filtering of data and metadata extraction. This also provides for an additional level of access control as users can specify certain datasets or collections to be accessible only through remote procedures.

3.2.3 Data Replication and Storage

In this subsection, four of the data replication mechanisms used within Data Grids are studied in depth and classified according to the taxonomy given in Section 3.1.3. These were chosen not only because of their wide usage but also because of the wide variations in design and implementation represented by them. A summary is given in Table 3.3. Table 3.4 encapsulates the differences between the various replication mechanisms on the basis of the replication strategies that they follow. Some of the replication strategies have been only simulated and therefore, these are explained in a separate subsection.

Grid DataFarm

Grid Datafarm (Gfarm) [199] is an architecture that couples storage, I/O bandwidth and processing to provide scalable computing to process petabytes (PB) of data. The architecture consists of nodes that have a large disk space (in the order of terabytes (TB)) coupled with computing power. These nodes are connected via a high speed interconnect such as Myrinet or Fast Ethernet. Gfarm consists of the Gfarm filesystem, process scheduler and the parallel I/O APIs.

The Gfarm filesystem is a parallel filesystem that unifies the file addressing space over all the nodes. It provides scalable I/O bandwidth by integrating process scheduling with data distribution. A Gfarm file is a large file that is stored throughout the filesystem on multiple disks as fragments. Each fragment has arbitrary length and can be stored on any node. Individual fragments can be replicated and the replicas are managed through Gfarm metadata. Individual fragments may be replicated and the replicas are managed through the filesystem metadata and replica catalog. Metadata is updated at the end of each operation on a file. A Gfarm file is write-once, that is, if a file is modified and saved, then internally it is versioned and a new file is created.

Gfarm targets data-intensive applications in which the same program is executed over different data files and where the primary task is of reading a large body of data. The data is split up and stored as fragments on the nodes. While executing a program, the process scheduler dispatches it to the node that has the segment of data that the program wants to access. If the nodes that contain the data and its replicas are under heavy CPU load, then the filesystem creates a replica of the requested fragment on another node and assigns the process to it. In this way, I/O bandwidth is gained by exploiting the access locality of data. This process can also be controlled through the Gfarm APIs. It is also possible to access the file using a local buffer cache instead of replication.

On the whole, Gfarm is a system that is tuned for high-speed data access within a tightly-coupled yet large-scale architecture such as clusters consisting of hundreds of nodes. It requires high-speed interconnects between the nodes so that bandwidth-intensive tasks such as replication do not cause performance hits. This is evident through experiments carried out over clusters and wide-area testbeds [200, 218]. The scheduling in

Gfarm is at the process level and applications have to use the API though a system call trapping library is provided for inter-operating with legacy applications. Gfarm targets applications such as High Energy Physics where the data is "write-once read-many". For applications where the data is constantly updated, there could be problems with managing the consistency of the replicas and the metadata though an upcoming version aims to fix them [201].

RLS

Giggle (GIGa-scale Global Location Engine) [55] is an architectural framework for a Replica Location Service (RLS) that maintains information about physical locations of copies of data. The main components of RLS are the Local Replica Catalog (LRC) which maps the logical representation to the physical locations and the Replica Location Index (RLI) which indexes the catalog itself.

The actual data is represented by a *logical file name (LFN)* and contain some information such as the size of the file, its creation date and any other such metadata that might help users to identify the files that they seek. A logical file has a mapping to the actual physical location(s) of the data file and its replicas, if any. The physical location is identified by a unique *physical file name (PFN)* which is a URL (Uniform Resource Locator) to the data file on storage. Therefore, a LRC provides the PFN corresponding to an LFN. The LRC also supports authenticated queries that is, information about the data is not available in the absence of proper credentials.

A data file may be replicated across several geographical and administrative boundaries and information about its replicas may be present in several replica catalogs. An RLI creates an index of replica catalogs as a set of logical file names and a pointer to a replica catalog entries. Therefore, it is possible to define several configurations of replica indexes, for example a hierarchical configuration or a central, single-indexed configuration or a partitioned index configuration. Some of the possible configurations are listed by Chervenak et al. [55]. The information within an RLI is periodically updated using soft-state mechanisms similar to those used in Globus MDS (Monitoring and Discovery System). In fact, the structure of the replica catalog is quite similar to that of MDS [65].

RLS is aimed at replicating data that is "write once read many". Data from scientific

instruments that needs to be distributed around the world is falls into this category. This data is seldom updated and therefore, strict consistency management is not required. Soft-state management is enough for such applications. RLS is also a standalone replication service that is it does not handle file transfer or data replication itself. It provides only an index for the replicated data.

GDMP

GDMP [180, 189] is a replication manager that aims to provide secure and high-speed file transfer services for replicating large data files and object databases. GDMP provides point-to-point replication capabilities by utilizing the capabilities of other Data Grid tools such as replica catalogs and GridFTP.

GDMP is based on the publish-subscribe model, wherein the server publishes the set of new files that are added to the replica catalog and the client can request a copy of these after making a secure connection to the server. GDMP uses GSI as its authentication and authorization infrastructure. Clients first register with the server and receive notifications about new data that are available which are then requested for replication. Failure during replication is assumed to be handled by the client. For example, if the connection fails while replicating a set of files, the client may reconnect with the server and request a re-transfer. The file transfer is conducted through GridFTP.

GDMP deals with object databases created by High Energy Physics experiments. A single file may contain up to a billion (10^9) objects and therefore, it is advantageous for the replication mechanisms to deal with objects rather than files. Objects requested by a site are copied to a new file at the source. This file is then transferred to the recipient and the database at the remote end is updated to include the new objects. The file is then deleted at the origin. In this case, replication is static as changing Grid conditions are not taken into account by the source site. It is left up to the client site to determine the time and the volume of replication.

GDMP was originally conceived for the CMS experiment at the LHC in which the data is generated at one point and has to be replicated globally. Therefore, consistency of replicas is not a big issue as there are no updates and all the notifications are in a single direction. The data for this experiment was in the form of files containing objects

where each object represented a collision. GDMP can interact with the object database to replicate specific groups of objects between sites.

SRB

The purpose of the SRB is to enable the creation of shared collections through management of consistent state information, latency management, load leveling, logical resources usage and multiple access interfaces [26, 170]. SRB also aims to provide a unified view of the data files stored in disparate media and locations by providing the capability to organise them into virtual collections independent of their physical location and organization. It provides a large number of capabilities that are not only applicable to Data Grids but also for collection building, digital libraries and persistent archival applications.

An SRB installation follows a three-tier architecture - the bottom tier is the actual storage resource, the middleware lies in between and at the top is the Application Programming Interface (API) and the Metadata CATalog (MCAT). File systems and databases are managed as *physical storage resources (PSRs)* which are then combined into *logical storage resources (LSRs)*. Data items in SRB are organised within a hierarchy of collections and sub-collections that is analogous to the UNIX filesystem hierarchy. Collections are implemented using LSRs while the data items within a collection can be located on any PSR. Data items within SRB collections are associated with metadata which describe system attributes such as access information and size, and descriptive attributes which record properties deemed important by the users. The metadata is stored within MCAT which also records attributes of the collections and the PSRs. Attribute-based access to the data items is made possible by searching MCAT.

The middleware is made up of the SRB Master daemon and the SRB Agent processes. The clients authenticate to the SRB Master and the latter starts an Agent process that processes the client requests. An SRB agent interfaces with the MCAT and the storage resources to execute a particular request. It is possible to create a federation of SRB servers by interconnecting the masters. In a federation, a server acts as a client to another server. A client request is handed over to the appropriate server depending on the location determined by the MCAT service.

SRB implements transparency for data access and transfer by managing data as col-

Table 3.4: Comparison between replication strategies.

Project	**Method**	**Granularity**	**Objective Function**
Grid Datafarm	Static	File, Fragment	Locality
RLS	Static	Datasets, File	Popularity, Publication
GDMP [189]	Static	Datasets, File, Fragment	Popularity, Publication
SRB	Static	Containers, Datasets, File	Preservation, Publication
Lamehamedi et. al ([129]; [128])	Dynamic	File	Update Costs
Bell et al. [30]	Dynamic	File	Economic
Lee and Weissman [133]	Dynamic	File	Popularity
Ranganathan et al. [175]	Dynamic	File	Popularity

lections which own and manage all of the information required for describing the data independent of the underlying storage system. The collection takes care of updating and managing consistency of the data along with other state information such as timestamps and audit trails. Consistency is managed by providing synchronisation mechanisms that lock stale data against access and propagates updates throughout the environment until global consistency is achieved.

SRB is one of the most widely used Data Grid technologies in various application domains around the world including the UK eScience (eDiaMoND), BaBar, BIRN, IVOA and the California Digital Library [168].

Other Replication Strategies

Lamehamedi, et. al [128, 129] study replication strategies based on the replica sites being arranged in different topologies such as ring, tree or hybrid. Each site or node maintains an index of the replicas it hosts and the other locations of these replicas that it knows. Replication of a dataset is triggered when requests for it at a site exceed some threshold. The replication strategy places a replica at a site that minimises the total access costs including both read and write costs for the datasets. The write cost considers the cost of

updating all the replicas after a write at one of the replicas. They show through simulation that the best results are achieved when the replication process is carried out closest to the users.

Bell et al. [30] present an file replication strategy based on an economic model that optimises the selection of sites for creating replicas. Replication is triggered by the number of requests received for a dataset. Access mediators receive these requests and start auctions to determine the cheapest replicas. A Storage Broker (SB) participates in these auctions by offering a price at which it will sell access to a replica if it is present. If the replica is not present at the local storage element, then the broker starts an auction to replicate the requested file onto its storage if it determines that having the dataset is economically feasible. Other SBs then bid with the lowest price that they can offer for the file. The lowest bidder wins the auction but is paid the amount bid by the second-lowest bidder. This is a Vickrey second price auction [209] with descending bids.

Lee and Weissman [133] present an architecture for dynamic replication within a service Grid. The replicas are created on the basis of each site evaluating whether its performance can be improved by requesting one more replica. The most popular services are, therefore, most replicated as this will entail a performance boost by lessening the load requirements on a particular replica.

Ranganathan et al. [175] present a dynamic replication strategy that creates copies based on trade-offs between the cost and the future benefits of creating a replica. The strategy is designed for peer-peer environments where there is a high-degree of unreliability and hence, considers a minimum number of replicas that might be required given the probability of a node being up and the accuracy of information possessed by a site in a peer-peer network.

3.2.4 Resource Allocation and Scheduling

This subsection deals with the study of resource allocation and scheduling strategies within Data Grids. While Grid scheduling has been a well-researched topic, this study is limited to only those strategies that explicitly deal with transfer of data during processing. Therefore, the focus here is on features such as adapting to environments with varied

data sources and scheduling jobs in order to minimise the movement of data. Table 3.5 summarises the scheduling strategies surveyed in this section and their classification.

Table 3.5: Comparison between scheduling strategies.

Work/Project	**Application Model**	**Scope**	**Data Replication**	**Utility Function**	**Locality**
Casanova, et al. [51]	Bag-of-Tasks	Individual	Coupled	Makespan	Temporal
GrADS [67]	Process-level	Individual	Decoupled	Makespan	Spatial
Ranganathan & Foster [173]	Independent Tasks	Individual	Decoupled	Makespan	Spatial
Kim and Weissman [116]	Independent Tasks	Individual	Decoupled	Makespan	Spatial
Takefusa, et. al [198]	Process-level	Individual	Coupled	Makespan	Temporal
Pegasus [69]	Workflows	Individual	Decoupled	Makespan	Temporal
Thain et al. [203]	Independent Tasks	Community	Coupled	Makespan	Both
Chameleon [160]	Independent Tasks	Individual	Decoupled	Makespan	Spatial
SPHINX [110, 111]	Workflows	Community	Decoupled	QoS	Spatial

Scheduling strategies for data-intensive applications can be distinguished on the basis of whether they couple data movement to job submission or not. As mentioned earlier in Section 3.1.4, in the former case, the temporal locality of data requests is exploited. Initial work focused on reuse of cached data. An example of this direction is the work by Casanova et al. [51] who introduce heuristics for scheduling independent tasks sharing common files, on a Grid composed of interconnected clusters. Here, the strategy is to prefer nodes within clusters to which the data has already been transferred rather than those clusters where the data is not present. The source of the data is considered to be the client node, i.e., the machine which submits the jobs to the Grid. Later efforts looked at extending this to data replication where copies of the data are maintained over a longer term to benefit requests coming from future job submissions. Takefusa et al. [198] have simulated job scheduling and data replication policies for central and tier model organization of Data Grids based on the Grid Datafarm [199] architecture. Out of the several

policies simulated, the authors establish that the combination of *OwnerComputes* strategy (job is executed on the resource that contains the data) for job scheduling along with background replication policies based on number of accesses (*LoadBound-Replicate*) or on the node with the maximum estimated performance (*Aggressive-Replication*) provides the minimum execution time for a job.

Similar in intent, Thain et al. [203] describe a means of creating I/O communities which are groups of CPU resources such as Condor pools clustered around a storage resource. The storage appliance satisfies the data requirements for jobs that are executed on both the processes within and outside the community. The scheduling strategy in this work allows for both the data to be staged to a community where the job is executed and the job to migrate to a community where the data required is already staged. The decision is made by the user after comparing the overheads of either staging the application or replicating the data. This is different to the policies previously mentioned wherein the replication process is based on heuristics and requires no user intervention. Again, improving temporal locality of data by replicating it within a community improves the performance. Later, this section looks at another coupled strategy proposed by Phan et al. [162] that uses Genetic Algorithms as a scheduling heuristic.

Strategies that decouple job submission from data movement attempt to reduce the data transfer time either by scheduling the job close to or at the source of the data, or by accessing the data from a replica site which is closest to the site of computation. Here, the term "close" refers to a site with minimum transfer time. Ranganathan and Foster [173] propose a decoupled scheduling architecture for data intensive applications which consists of 3 components: the External Scheduler (ES) that decides to which node the jobs must be submitted, the Local Scheduler (LS) on each node that decides the priority of the jobs arriving at that node and the Dataset Scheduler (DS) that tracks the popularity of the datasets and decides which datasets to replicate or delete. Through simulation, they evaluate combinations of 4 job scheduling algorithms for the ES and 3 replication algorithms for the DS. The results show that the worst performance is given by executing a job at the source of data in the absence of replication. This is because a few sites which host the data are overloaded in this case. The best performance is given by same job scheduling strategy but with data replication. A similar strategy is proposed in Chameleon [160]

wherein a site on which the data has already been replicated is preferred for submitting a job over one where the data is not present.

Most of the strategies studied try to reduce the *makespan* or the Minimum Completion Time (MCT) of the task which is defined as the difference between the time when the job was submitted to a computational resource and the time it completed. Makespan also includes the time taken to transfer the data to the point of computation if that is allowed by the scheduling strategy. Takefusa et al. [198] and Grid Application Development Software (GrADS) project [67] are makespan schedulers that operate at the system process level. Scheduling within the latter is carried out in three phases: before the execution, there is an initial matching of an application's requirements to available resources based on its performance model and this is called *launch-time scheduling*; then, the initial schedule is modified during the execution to take into account dynamic changes in the system availability which is called *rescheduling*; finally, the co-ordination of all schedules is done through *meta-scheduling*. Contracts [211] are formed to ensure guaranteed execution performance. The mapping and search procedure presented by Dail et al. [66] forms Candidate Machine Groups (CMG) consisting of available resources which are then pruned to yield one suitable group per application. The mapper then maps the application data to physical location for this group. Therefore, spatial locality is primarily exploited. The scheduler is tightly integrated into the application and works at the process level. However, the algorithms are themselves independent of the application. Recent work however has suggested extending the GrADS scheduling concept to workflow applications [61]. However, the treatment of data still remains the same.

Casanova et al. [51] extend three heuristics for reducing makespan — *Min-Min*, *Max-Min* and *Sufferage* that were introduced by Maheswaran et al. [142] — to consider input and output data transfer times. Min-Min assigns tasks with the least makespan to those nodes which will execute them the fastest whereas Max-Min assigns tasks with maximum makespan to fastest executing nodes. Sufferage assigns tasks on the basis of how much they would "suffer" if they are not assigned to a particular node. This "sufferage" value is computed as the difference between the best MCT for a task on a particular node and the second-best MCT on another node. Tasks with higher sufferage values receive more priority. The authors introduce another heuristic, *XSufferage*, which is an extended version

of Sufferage that takes into account file locality before scheduling jobs by considering MCT on the cluster level. Within XSufferage, a job is scheduled to a cluster if the file required for the job has been previously transferred to any node within the cluster.

Kim and Weissman [116] introduce a Genetic Algorithm (GA) based scheduler for reducing makespan of Data Grid applications decomposable into independent tasks. The scheduler targets an application model wherein a large dataset is split into multiple smaller datasets and these are then processed in parallel on multiple "virtual sites", where a virtual site is considered to be a collection of compute resources and data servers. The solution to the scheduling problem is represented as a chromosome in which each gene represents a task allocated to a site. Each sub-gene is associated with a value that represents the fraction of a dataset assigned to the site and the whole gene is associated with a value denoting capability of the site given the fraction of the datasets assigned, the time taken to transfer these fractions and the execution time. The chromosomes are mutated to form the next generation of chromosomes. At the end of an iteration, the chromosomes are ranked according to an objective function and the iteration stops at a predefined condition. Since the objective of the algorithm is to reduce the completion time, the iterations tend to favour those tasks in which the data is processed close to or at the point of computation thereby exploiting the spatial locality of datasets. Phan et al. [162] apply a similar GA based strategy, but in their case, data movement is coupled to job submission. The chromosome that they adopt represents job ordering, assignments of jobs to compute nodes and the assignment of data to replica locations. At the end of a specified number of iterations (100 in this case), the GA converges to a near-optimal solution that gives a job order queue, job assignments and data assignments that minimize makespan.

While the strategies before have concentrated on independent tasks or BoT model of Grid applications, Pegasus [69] concentrates on reducing makespan for workflow-based applications. The strategy reduces an *abstract workflow* that contains the order of execution of components into a *concrete workflow* where the component is turned into an executable job and the locations of the computational resources and the data are specified. The abstract workflow goes through a process of *reduction* where the components whose outputs have already been generated and entered into a Replica Location Service are removed from the workflow and substituted with the physical location of the products.

The emphasis is therefore on the reuse of already produced data products. The planning process selects a source of data at random, that is, neither the temporal nor the spatial locality is exploited.

Other projects aim to achieve different scheduling objectives such as achieving a specific QoS demanded by the application. SPHINX (Scheduling in Parallel for a Heterogeneous Independent NetworX) [110] is one such middleware project for scheduling data-intensive applications on the Grid. Scheduling within SPHINX is based on a client-server framework in which a scheduling client within a VO submits a meta-job as a Directed Acyclic Graph (DAG) to one of the scheduling servers for the VO along with QoS requirements such as number of CPUs required and deadline of execution. QoS privileges that a user enjoys may vary with the groups he or she belongs to. The server is allocated a portion of the VO resources and in turn, it reserves some of these for the job submitted by the client based on the allocated QoS for the user and sends the client an estimate of the completion time. The server also reduces the DAG by removing tasks whose outputs are already present. If the client accepts the completion time, then the server begins execution of the reduced DAG. The scheduling strategy in SPHINX [111] considers VO policies as a four dimensional space with the resource provider, resource properties, user and time forming each of the dimensions. Policies are expressed in terms of quotas which are tuples formed by values of each dimension.The optimal resource allocation for a user request is provided by a linear programming solution which minimizes the usage of the user quotas on the various resources.

3.3 Discussion and Summary

Figures 3.10 – 3.14 pictorially represent the mapping of the systems that were analysed in Section 3.2 to the taxonomy. Each of the boxes at the “leaves” of the taxonomy “branches” contains those systems that exhibit the property at the leaf. A box containing “(All)” implies that all the systems studied satisfy the property given by the corresponding leaf. From the figures it can be seen that the taxonomy is shown to be complete with respect to the systems studied as each of them can be fully described by the categories within this taxonomy.

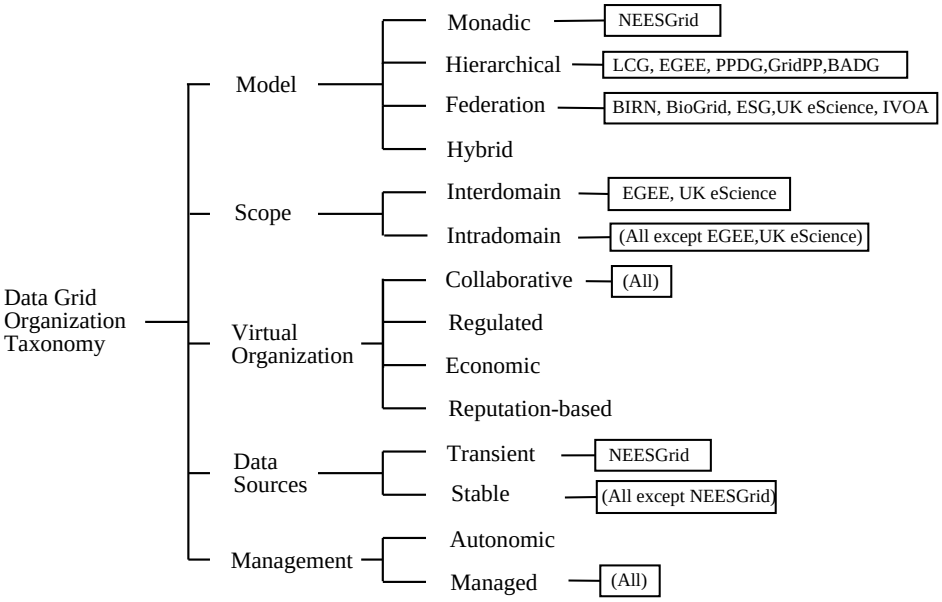


Figure 3.10: Mapping of Data Grid Organization Taxonomy to Data Grid Projects.

Figure 3.10 shows the organizational taxonomy annotated with the Data Grid projects that were studied in Section 3.2.1. As can be seen from the figure, current scientific Data Grids mostly follow the hierarchical or the federated models of organization because the data sources are few and well-established. These data sources are generally mass storage systems from which data is transferred out as files or datasets to other repositories. From a social point of view, such Data Grids are formed by establishing collaborations between researchers from the same domain. In such cases, any new participants willing to join or contribute have to be part of the particular scientific community to be inducted into the collaboration.

The mapping of various Data Grid transport mechanisms studied in Section 3.2.2 to the proposed taxonomy is shown in Figure 3.11. The requirement to transfer large datasets has led to the development of high-speed, low latency transfer protocols such as GridFTP which is rapidly becoming the default transfer protocol for all Data Grid projects. While FTP is also used within certain projects for data with lesser size and security constraints, and SRB I/O is applicable in any SRB installation, IBP and Kangaroo are not deployed in existing Data Grids. This is due to the fact that the latter are research projects rather than products and do not meet all requirements of a Data Grid environment.

Figures 3.12 and 3.13 show mapping of the data replication systems covered in Sec-

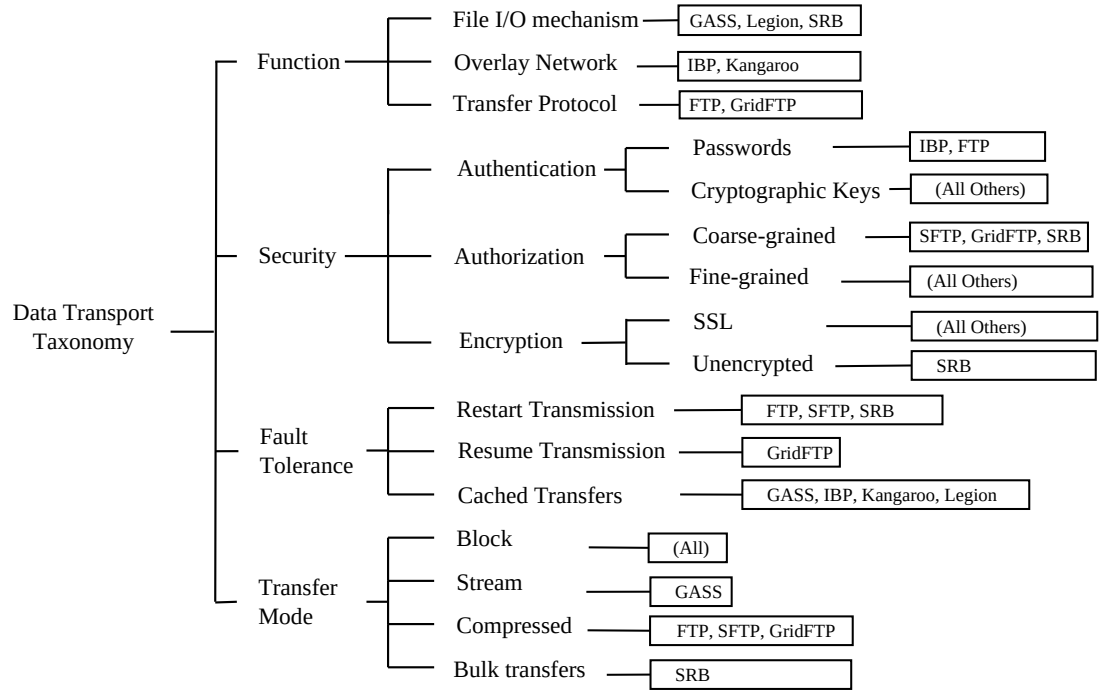


Figure 3.11: Mapping of Data Transport Taxonomy to Various Projects.

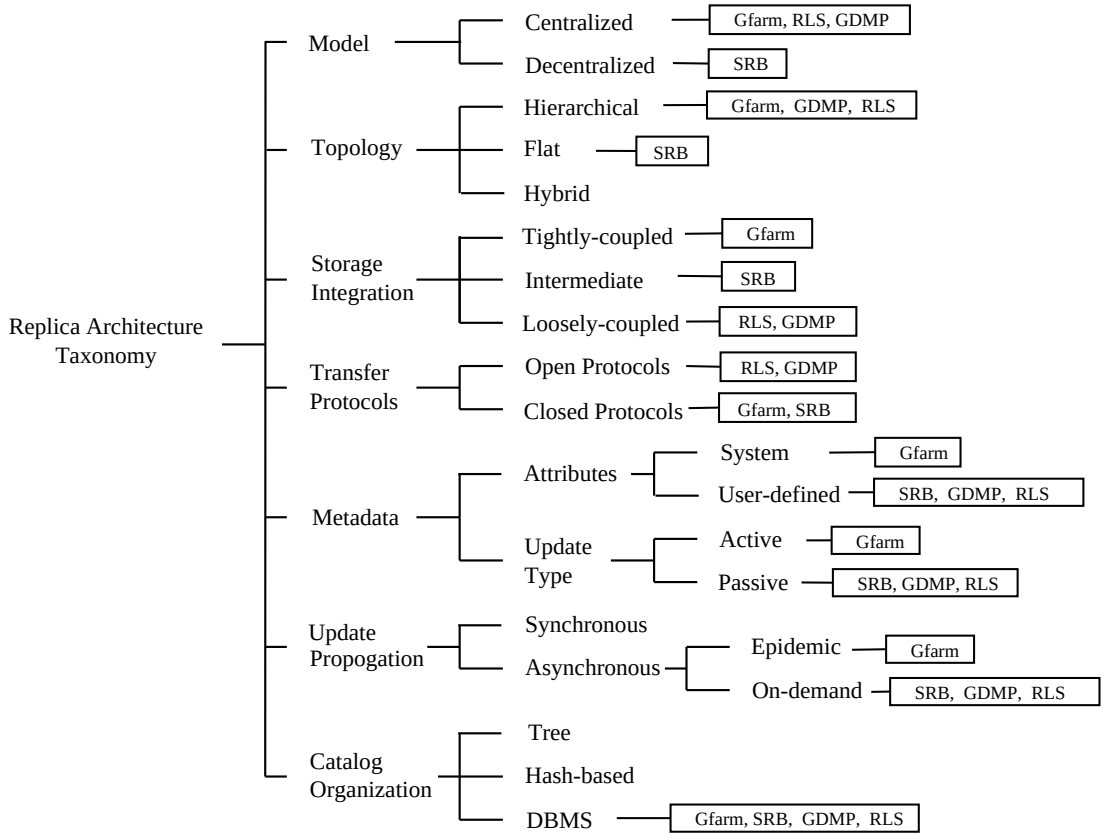


Figure 3.12: Mapping of Data Replication Architecture Taxonomy to Various Systems.

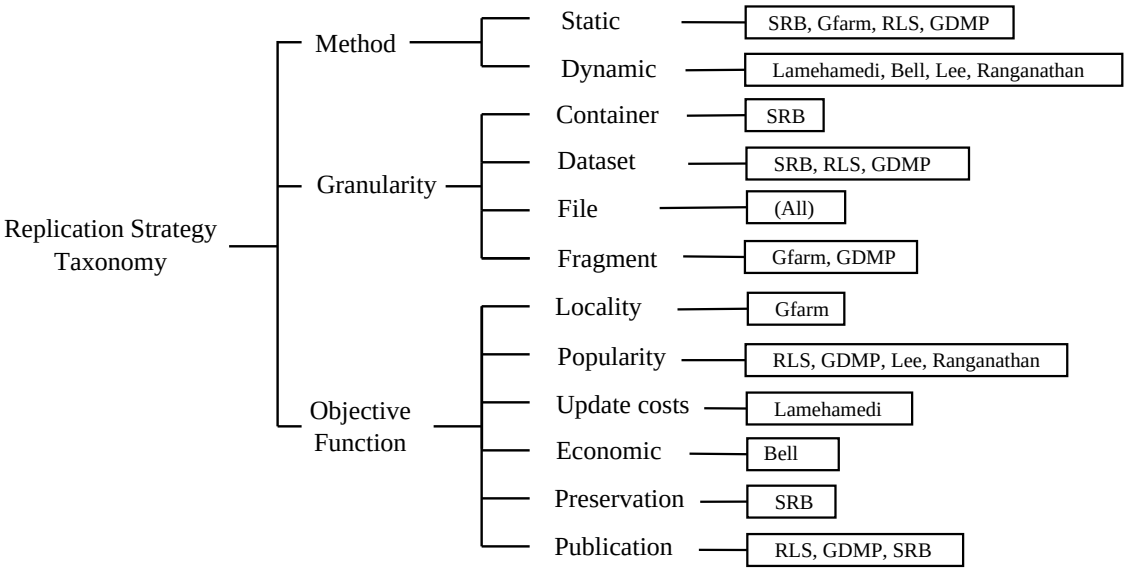


Figure 3.13: Mapping of Data Replication Strategy Taxonomy to Various Systems.

tions 3.2.3 to the replica architecture and strategy taxonomy. The hierarchical model of the HEP experiments in Figure 3.10 has motivated the development of tree-structured replication mechanisms that are designed to be top-down in terms of organization and data propagation. Many of the projects that have followed the federation model have used SRB which offers more flexibility in the organization model of replica sites. SRB is also used by many HEP experiments such as Belle and BaBar but configured as a hierarchy of sites. Currently massive datasets are being replicated statically by project administrators in select locations for all the projects, and intelligent and dynamic replication strategies have not yet found a place in production Data Grids. The static replication strategy is guided by the objective of increasing locality of datasets.

Figure 3.14 maps the Data Grid scheduling efforts discussed in the previous section to the scheduling taxonomy. It can be inferred that almost all of these efforts have concentrated on reducing the makespan of the schedule. Mostly, the algorithms have been developed to take care of individual jobs such as those submitted to job queueing systems. This corresponds with traditional workloads in scientific domains such as High Energy Physics that consist of batch submission of analysis jobs.

However, the maturing of Grid services and Application Programming Interfaces (APIs) is bringing about data-intensive Grid applications that work at the level of aggregated tasks/jobs. These applications also require to satisfy more sophisticated objectives than

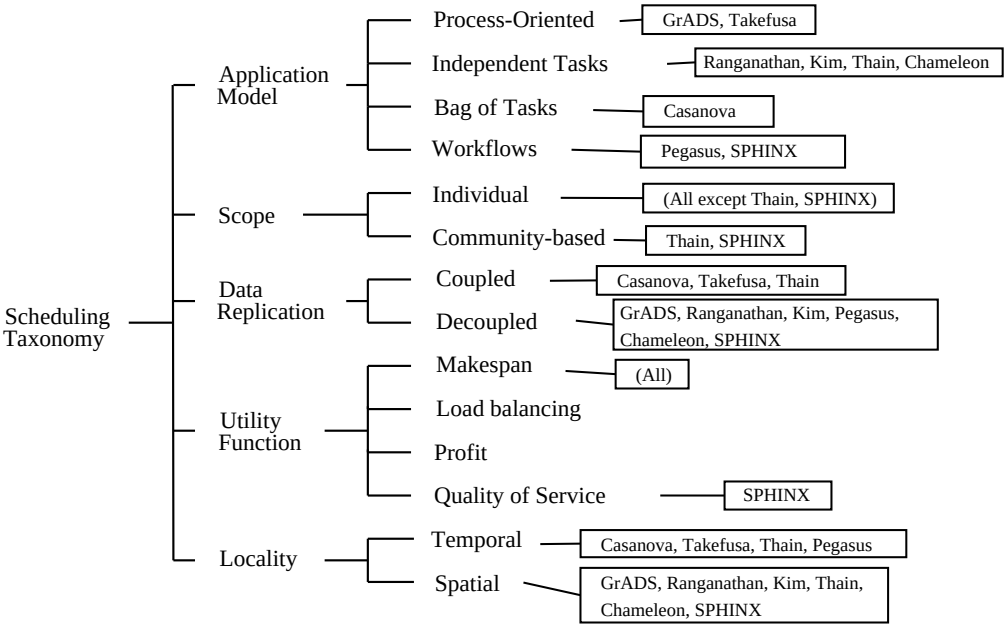


Figure 3.14: Mapping of Resource Allocation and Scheduling Taxonomy to Various Systems.

simple makespan reduction. The next chapter presents a software framework called a resource broker that allows users to create data-intensive Grid applications without knowing the underlying details of Grid services. The resource broker translates application requirements to tasks that are then carried out by invoking the appropriate Grid services. The broker also allows creation of schedulers with varied objectives such as deadline and budget constrained scheduling of applications. A case study involving the scheduling of a data-intensive High Energy Physics applications is also presented to illustrate the utility of the broker.

Chapter 4

A Grid Resource Broker for Data-Intensive Applications

This chapter presents the design and implementation of the Gridbus Grid resource broker for distributed data-intensive applications. It first discusses the motivation for developing a resource broker and the requirements that need to be satisfied by the broker, especially for distributed data-intensive applications. Then, it discusses the architecture and design of the broker in detail. Next, the implementation of the concepts within the design is illustrated with several examples. The Gridbus broker is then compared to other Grid brokers on the basis of several properties. Finally, this chapter presents a case study involving a High Energy Physics (HEP) application called the Belle Analysis Software Framework (BASF) deployed on resources around Australia, to illustrate the usage of the broker for distributed data-intensive scheduling.

4.1 Resource Brokers: Challenges

The capabilities that need to be provided in order to realise a Grid environment include: uniform authentication and authorisation; resource management and job submission; large-scale data management and transfer; and resource allocation and scheduling. Software tools and services that provide these capabilities are collectively called *Grid middleware*, and mediate between users and the underlying Grid fabric consisting of heterogeneous

computing and storage resources connected by networks of varying capabilities. Figure 4.1 shows the evolution of Grid application development using Grid middleware.

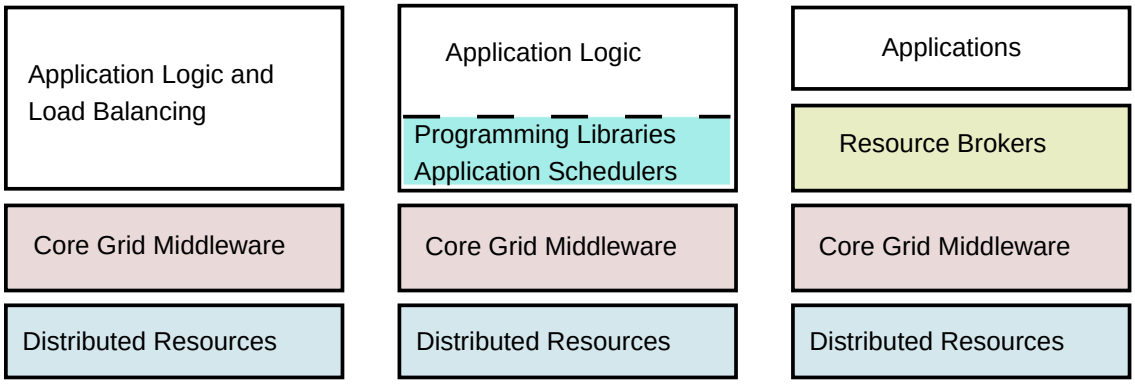


Figure 4.1: Evolution of application development on Grids. (a) Direct to core middleware (1995-). (b) Through programming libraries (1999-). (c) Using user-level middleware (2000-).

The first set of Grid middleware aimed to present a secure and standard method of invocation that abstracted the underlying heterogeneity of distributed resources. These *core grid middleware* such as Globus [83] and Legion [54] (Figure 4.1(a)) provided services for performing low-level Grid functions such as data access, job submission and authorisation. Some of the early Grid applications directly invoked the functionalities presented by these *core middleware* through their APIs. However, they were still too complex and low-level to become popular for general application development. These were followed by programming libraries such as NetSolve [21], Ninf [151], Cactus [11] and GrADS [31] (Figure 4.1(b)) that provided a software environment to create applications accessing distributed services in a transparent fashion. While these hid the problems of having to deal with varying Grid conditions from the user, significant effort was required to develop schedulers and task managers for each application. Projects such as AppLeS (Application Level Schedulers) [32] produced some of the early work in this regard. The next step was to move the scheduling algorithms to a generic framework that also provided capabilities such as resource selection, job monitoring and data access to any application. Such frameworks are called *resource brokers* (Figure 4.1(c)) and some examples of these are Nimrod/G [43], AppLeS Parameter Sweep Template (APST) [52], Condor-G [91] and the Gridbus resource broker which is discussed in this thesis.

The creation of a generic resource broker framework for a data-intensive Grid application runs into several challenges that arise out of the diversity inherent in Grid environments. These challenges are listed as follows:

Service Heterogeneity: With the introduction of the OGSA [88], Grids have progressed from being aggregations of heterogeneous resources to collections of stateful services. These services can be grouped into several categories such as job submission and monitoring, information, data management and application deployment. Standardisation of service interfaces has been a recent development in Grid computing and is still an ongoing process. Also, rapid developments in this field have meant that middleware itself changes frequently, and therefore service interface changes are the norm rather than the exception. Due to this heterogeneity of services, supporting diverse service semantics is still a serious challenge.

Variety of Application Models: As was discussed in Chapter 3, Section 3.1.4, data-intensive Grid applications tend to follow a variety of models such as bag of tasks, workflows and independent jobs. However, these are still required to interact with the same set of service interfaces. Enabling this interaction requires reconciliation between different application directives and constructs. Also, applications may invoke services in a variety of ways. A brokering system must avoid imposing constraints on applications as far as possible so as to not to limit its own applicability.

Multiple User Objectives: Applications and users may wish to satisfy different objectives at the same time. Some possible objectives include receiving results in the minimum possible time or within a set deadline, reducing the amount of data transfer and duplication, or ensuring minimum expense for an execution or minimum usage of allocated quota of resources. Different tasks within an application may be associated with different objectives and different QoS (Quality of Service) requirements. Examples of such requirements have been discussed in Chapter 3, Section 3.1.4. A brokering system must, therefore, ensure that different scheduling strategies meeting different objectives can be employed whenever required.

The discussion in previous chapters showed that one of the characteristics of a Data Grid is the presence of replicated data that is potentially widely dispersed through-

out the network. Resource brokering for distributed data-intensive applications should provide automatic discovery of data sources for a given dataset or a file, scheduling of jobs with respect to location of data and late binding of data locations to jobs so that the data resources can be assigned by taking into consideration the current availability of the underlying network.

Interface Requirements: The interface presented to the user may take several forms. Many scientists are comfortable with traditional command-line tools and require that Grid tools be command line- and script-friendly as well. Web portals that allow users to invoke Grid and application capabilities within one interface, have gained popularity in recent times as they enable portability of working environments. Recent applications are also able to seamlessly access Grid functions whenever required by invoking Grid/Web services or Grid middleware APIs. A resource broker should be able to support as many of these interfaces as possible in order to be useful to the largest community possible.

Infrastructural Concerns: The quintessential properties of Grid environments such as absence of administrative control over resources, dynamic system availability and high probabilities of failure have been described extensively in previous publications [23, 211]. A resource broker has to be able to handle these properties while abstracting them as much as possible from the end-user. This is a significant challenge to developing any Grid middleware.

The following sections present the architecture, design and implementation of a Grid resource broker that takes into account the challenges mentioned before in order to abstract the vagaries of the environment from the end-user.

4.2 Architecture of the Gridbus Broker

The Gridbus broker follows an open and extensible object-oriented architecture designed with the twin objectives of *flexibility* and *dependability* in mind. The architecture of the Gridbus broker and its interaction with external entities is shown in Figure 4.2. The components of the broker are grouped into three layers according to the level of abstraction

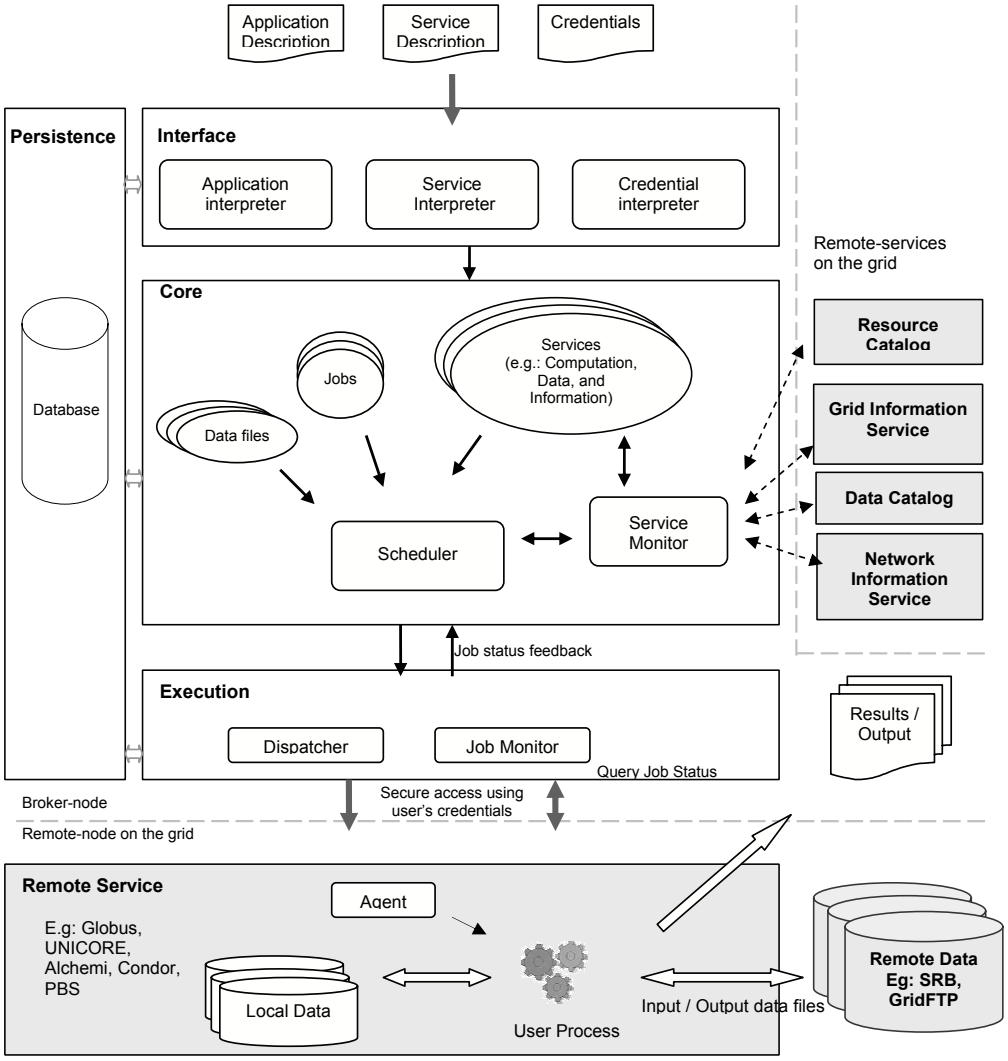


Figure 4.2: Gridbus broker architecture and its interaction with other Grid entities.

they provide from the underlying Grid resources. Each layer is decoupled from its underlying layer and their interaction is conducted through standard interfaces. The overall flow of control is from top to bottom while any events and exceptions that occur during execution being filtered by each layer from the bottom to the top. The layers are described in detail as follows.

4.2.1 Interface Layer

Applications, web portals and other such interfaces external to the broker interact with the components of the Interface layer. The inputs from the external entities are translated by this layer to create the objects in the Core layer. Three kinds of inputs are provided to the

broker: a description of the application requirements, a list of services that can be utilised for executing the application, and the list of credentials for accessing the services.

The application description provides details of the execution such as the location of executables, description of task inputs including required remote data files and information about task outputs. This description can be provided in one of the XML-based languages supported by the broker or be given programmatically through the broker's APIs. Similarly, the set of services required for the user objectives can be provided through the APIs or as an XML-based service description file containing information such as service location, service type and specific details such as remote batch job submission systems for computational services. The services can also be discovered by the broker at runtime from remote information services such as the Grid Market Directory (GMD) [221] or Grid Index Information Service (GIIS) [65] among others. The list of credentials is provided in another file. File-based inputs are handled by the respective interpreters which convert the descriptions to entities within the broker. The Application Interpreter converts the application description file to Task objects while the Service Interpreter converts the service description to Service objects. These objects are described as a part of the Core layer in the following section.

4.2.2 Core Layer

This layer contains entities that represent the properties of the Grid infrastructure independent of the middleware and the functionality of the broker itself. Therefore, it abstracts the details of the actual interaction with the Grid resources performed by the Execution layer. This interaction is driven by the decisions made by the functional components of the broker present in the Core layer.

Tasks represent sets of activities to be carried out in the execution. Examples of such activities include copying a file to the remote node, running the executable and storing the output in a remote repository. Tasks are associated with input parameters and may have requirements that need to be fulfilled before their execution. The task specification provides the template for creating jobs which are the actual units of work sent to the remote Grid resource. That is, a task is an abstraction of the work performed for executing

the application. The conversion of task to jobs is scenario-dependent; for example, a parameter sweep task is converted into a set of jobs, while a single, independent task representing a simple application with a single function is converted into one job.

It is possible to represent different types of services within the broker mirroring the variety of services that are available in Grids. A computational service represents a computational resource with properties such as architecture, operating system and available job submission systems. Data services describe storage repositories and the details of the data files stored within these. These details include attributes such as the path of the files in the repository and the protocol used to access the files. The size and location of input datasets for the application that are replicated on different repositories are also tracked by the broker. Services that provide meta-information such as resource information services, data catalogs and market directories are depicted as information services. Properties such as bandwidth and economic cost of the network paths between the computational and data resources are provided by network information services. Authorised access to all services is mediated by user-supplied Credentials that are associated with one or more services. The Service Monitor keeps track of the state of the services by querying them at regular intervals to determine properties such as availability, current price and performance.

The Scheduler maps jobs to the appropriate services depending on the strategy employed. The Scheduler may also take into account the user's QoS requirements such as deadline and budget. The Scheduler makes use of the information gathered by the service monitor to make its decisions.

4.2.3 Execution Layer

The actual task of dispatching the jobs is taken care of by the Execution layer which provides Dispatchers for various middleware. These dispatchers create middleware-specific Agents from the jobs and are executed on the remote resources. If there are any data files associated with the job, then the agents request them from the data repositories that have been selected to access those files. During execution, the Job Monitor keeps track of the job status - whether a job is queued, executing, has finished successfully or has failed on the remote resource. On completion of job execution, the associated agent returns any

results to the broker and provides debugging information.

4.2.4 Persistence Sub-system

The persistence subsystem extends across the three layers described previously and maintains the state of the various entities within the broker. It is primarily used to interface with the database into which the state is stored at regular intervals. The persistence sub-system satisfies two purposes: it allows for recovery in case of unexpected failure of the broker and is also used as a medium of synchronisation among the components in the broker.

4.3 Design of the Gridbus Broker

The broker design is based on the architecture described above. Objects in the broker can be broadly classified into two categories - *entities* and *workers*. This terminology is derived from the UML (Unified Modelling Language) Process for Business Modeling [77]. Entities exist as information containers representing the properties, functions and instantaneous states of the various architectural elements that are proxies for the actual Grid entities and constructs involved in the execution. Therefore, entities are stored in the persistence database and are updated periodically. Workers represent the functionality of the broker, that is, they implement the actual logic and manipulate the entities in order to achieve the application objectives. Therefore, workers can be considered as active objects and the entities as passive objects. In Figure 4.2, workers are represented by rectangles with rounded corners and entities by ovals. Examples of entities are ApplicationContext, Job, Service (eg: ComputeServer, DataHost, InformationService, ApplicationService) and DataFile. Workers within the broker include FarmingEngine, Scheduler, Dispatcher, JobMonitor and ServiceMonitor. The following subsections take a closer look at each of these objects. These are accompanied by UML 2.0 [36] diagrams that illustrate the relationships between the various objects.

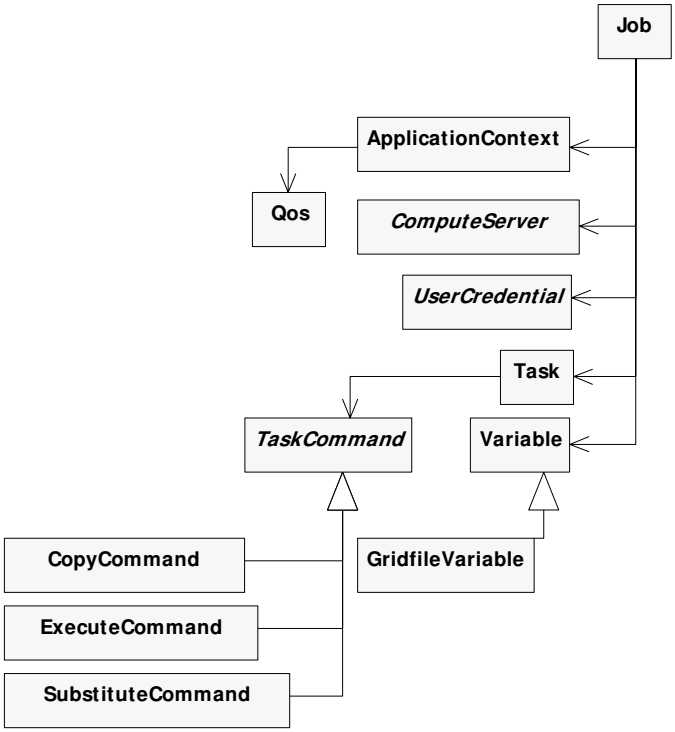


Figure 4.3: ApplicationContext, Tasks and Jobs.

4.3.1 Entities

Application Context and Tasks

Figure 4.3 shows the class diagram of the broker's task structure and its associated entities. An ApplicationContext represents the attributes of a user's application specification such as its task specification, one or more credentials for accessing the services and QoS requirements. The QoS requirements codify user expectations of the execution such as a deadline by which the execution must be completed. Task specifications are represented by Task objects. A Task is structured as a set of Commands which describe the activities to be undertaken. Three types of commands are available at present - CopyCommand, ExecuteCommand, and SubstituteCommand. The CopyCommand instructs the broker to copy a file from any resource in the Grid to any other resource. A number of file transfer protocols are supported including GridFTP, GASS and SRB copy. An ExecuteCommand specifies the application to be executed on the remote node. A SubstituteCommand is used for substituting variables whose values are determined at runtime, in local files on the broker. The task structure within the broker is based on and extended from the task

specification followed by Nimrod [2].

Jobs

A Job represents an instantiation of the task at the remote node and is therefore associated with a single Task object that describes its function. A Job may be associated with one or more Variable objects which describe its input data. A Variable is one of various types including integer, string and float and is associated with a single value derived from its domain. New types of variables can be introduced into the broker by extending the Variable class. For example, a Variable of type "gridfile" describes a file or a dataset that is stored on a repository that is accessible through any of the supported file transfer protocols. A gridfile Variable is associated with a DataFile object representing the remote file in the broker.

A Job is also associated with a JobWrapper that represents the interface for translating the user task specification to create an Agent that can be executed by the middleware running on the designated compute resource. The JobWrapper is, therefore, necessarily middleware-specific. The JobWrapper will be discussed in more detail in relation to the Dispatcher. Other than these, the Job is associated with a set of Services and a UserCredential.

In the course of its lifetime, a Job passes through many states as is outlined in Figure 4.4. A Job is an input to the Scheduler which allocates it to a set of resources based on its requirements. The Job's status is then changed to SCHEDULED. During the STAGE_IN state, input files and executables required for the job are staged to the remote resource. When this process is completed successfully and a handle is obtained, then a job is considered to be SUBMITTED. The Job may be queued while waiting for an available processor and its state changes to PENDING. When the Job starts its execution, it is considered ACTIVE. After the job has finished executing, it enters the STAGE_OUT stage where its output files are transferred back to the broker. If all its outputs are received and are as expected by the task requirements, then the job is considered as "DONE". If one of state transitions fails on the remote side or the job has completed on the remote side but has not produced the expected result files, then it is considered FAILED and is reset and marked for re-scheduling.

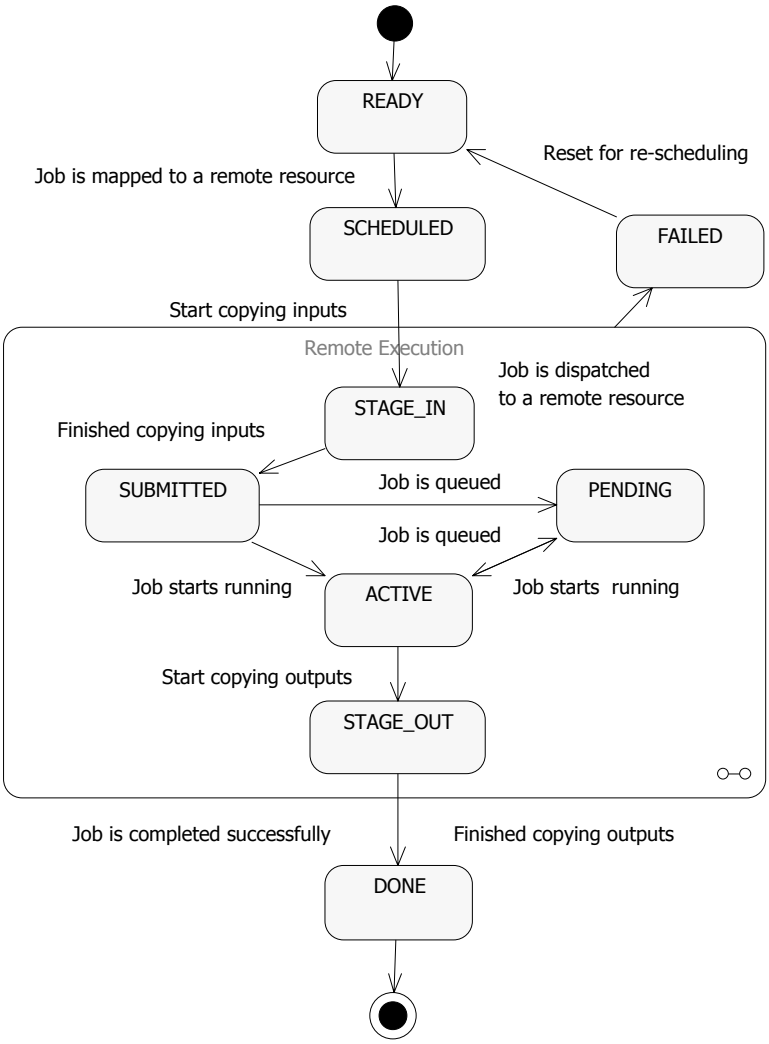


Figure 4.4: State transition diagram for a job.

Services

Services within the broker are represented in a hierarchy, shown in Figure 4.5 in which the first tier is an abstract Service whose attributes are the Service ID and the location of the service (hostname or URI). The next level groups the services into categories depending on the type of the service. Computational resources are represented by the abstract ComputeServer object, data repositories are represented by the abstract DataHost object and information services are represented by abstract InformationService object. The lowest tier then contains specific implementations of these services that provide the functions of associated middleware. For example, interaction with a computational resource running Globus middleware is implemented in the GlobusComputeServer class that extends the

abstract ComputeServer. However, it should be noted that these are not the only services possible. The flexibility of the structure encourages the introduction of newer services as the diversity of services increases. For example, Application Services could be introduced to represent remotely hosted applications that can invoked using through web services.

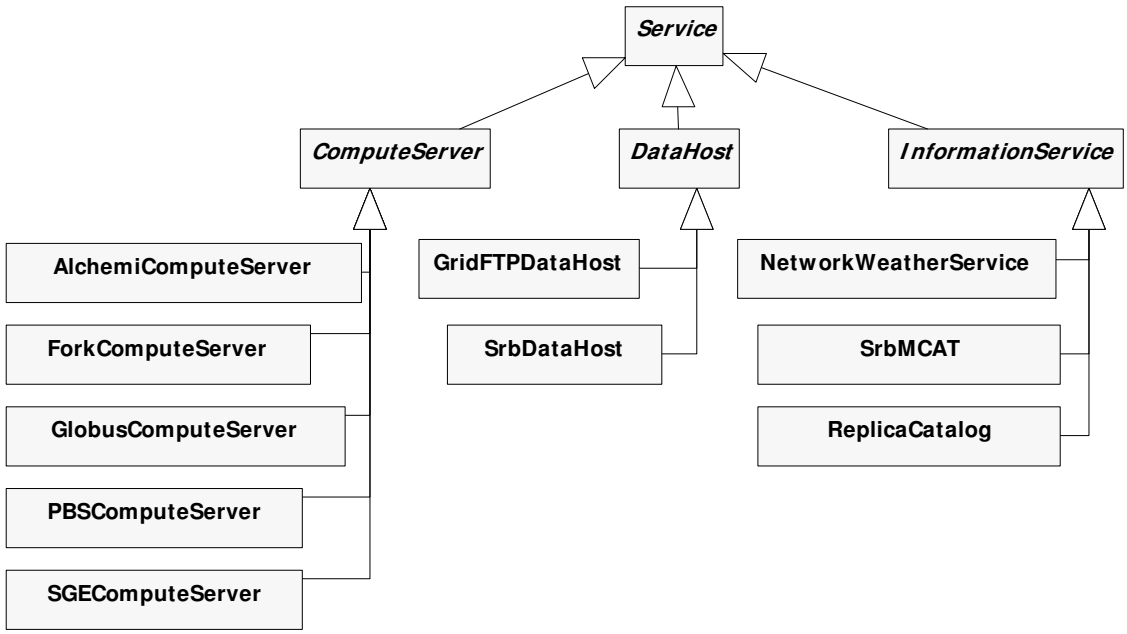


Figure 4.5: Service object hierarchy.

Each ComputeServer is associated with two active entities - a Dispatcher for jobs mapped to that service and a JobMonitor that monitors the jobs so dispatched. The abstract ComputeServer has been implemented for different middleware and job managers such as Globus, Alchemi [138], Unicore [78], PBS (Portable Batch Scheduler) [28], SGE (Sun Grid Engine) [95], Condor [137] and XGrid [122]. The DataHost has been implemented for providing access to data stores running SRB and those enabled by GridFTP.

InformationServices are categorised depending on the type of information they serve. For example, ReplicaCatalog services such as Globus Replica Catalogue and SRB MCAT provide information on different copies of required datasets that are stored on distributed repositories. Information about network properties is gathered from the NetworkInformationService and is stored in data structures called NetworkLinks that keep track of the changing network conditions between various resources.

Credentials

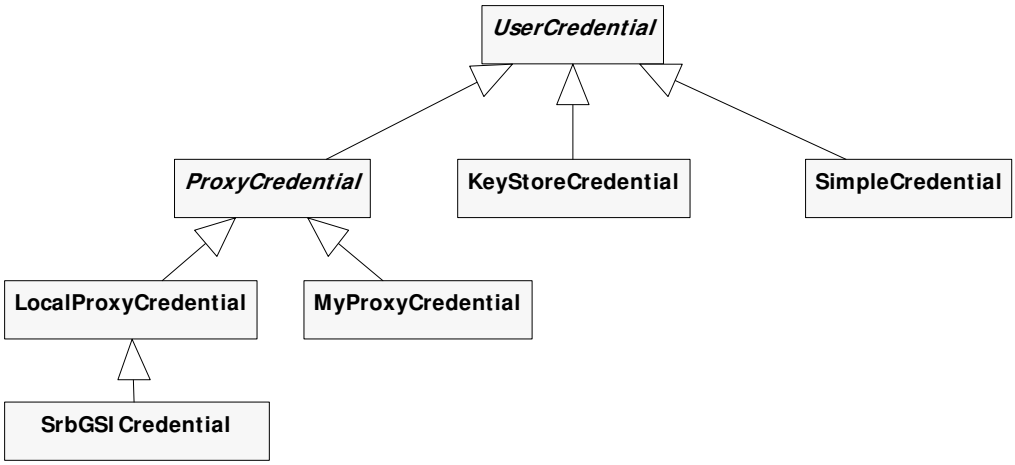


Figure 4.6: Credentials supported by the broker.

The Gridbus broker defines the concept of a UserCredential, representing an authentication token to access remote services. The base UserCredential object has been extended to realise the different types of credentials that are accepted by different middleware. For example, the LocalProxyCredential represents the Globus GSI (Grid Security Infrastructure) X.509 proxy object created on the client side. On the other hand, a SimpleCredential is used for accessing resources that require a username and a password for authentication such as those enabled through SSH (Secure Shell).

The set of credentials available to the user is associated with his/her ApplicationContext. Even though the credentials are passive objects, to ensure the security of users credentials, these are transient and are not saved in the persistence storage. This also means that the user has to provide the broker with a fresh set of credentials when recovering from a previously paused/failed run of a Grid application. Figure 4.6 shows the various types of credentials supported by the broker.

4.3.2 Workers

GridbusFarmingEngine

The GridbusFarmingEngine is the first component to be initialised and is a container for the other objects in the broker. It is responsible for managing the lifecycles of other

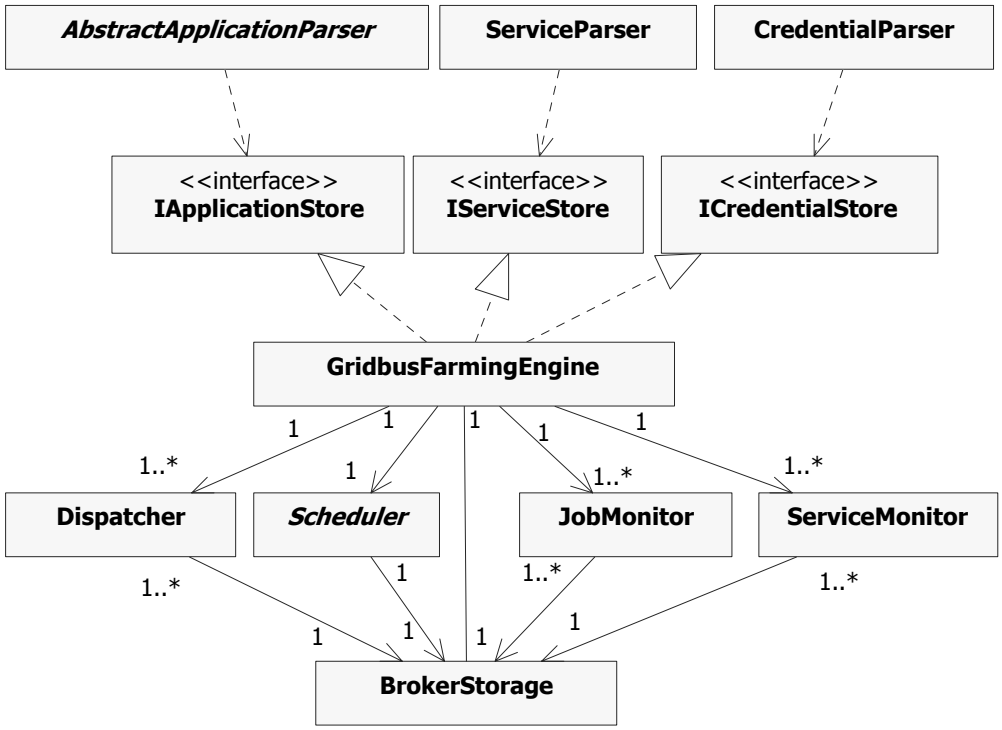


Figure 4.7: GridbusFarmingEngine and its associated entities.

active objects –the Scheduler, the Dispatchers, the JobMonitors and the ServiceMonitors– from start-up to shutdown. It is also a front end to the persistent storage and therefore, maintains the overall state of the broker by saving the state of various passive entities. The FarmingEngine and its associations with other entities within the broker is shown in Figure 4.7. In the figure, BrokerStorage is the frontend to the persistence system and the Store interfaces are used by the various interpreters to interact with the database.

ServiceMonitor

The ServiceMonitor component periodically checks the availability of the specified remote services and discovers new services which may become available. The sequence of operations in the ServiceMonitor is shown in Figure 4.8. Initially, it polls all available services by invoking the `discoverProperties` operation that is provided within each Service entity. This operation does a search for service attributes that are specific to the service and middleware type. If the operation is successful, the values that are so retrieved are set within the Service object and the Monitor is notified that the service is available.

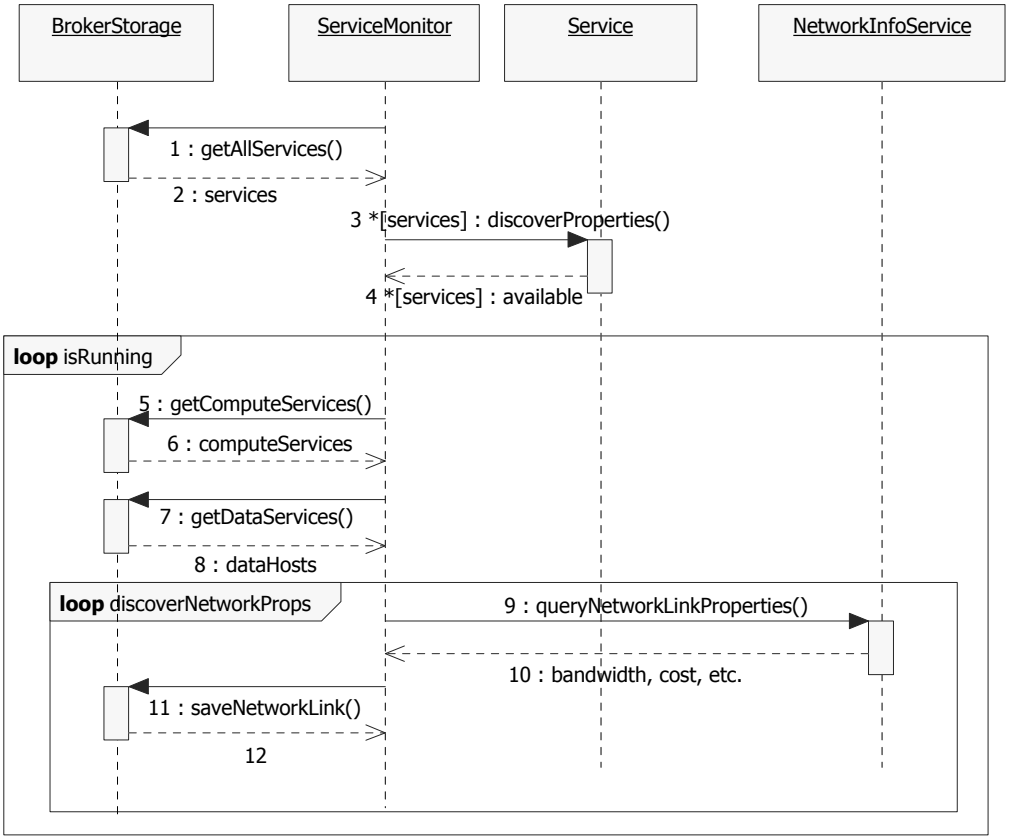


Figure 4.8: Service monitoring sequence diagram.

While running, the ServiceMonitor is also able to retrieve information from various InformationServices about the Grid environment. The example in Figure 4.8 shows how the NetworkInformationService is polled for information about available bandwidth and cost (if applicable) of the network links between various ComputeServers and DataHosts.This information is stored into the database from where it is retrieved by the Scheduler for making decisions.

Scheduler

The scheduling component is designed as two separate components running simultaneously: the Scheduler and the Dispatcher. The Scheduler matches the jobs individually to the services and also, decides the order of execution of the jobs on the resources. Figure 4.9 shows the basic sequence of operations that is performed by the Scheduler. The Scheduler gets the list of ready jobs from the persistent storage and a list of services, depending on the strategy, that it is interested in. The mapping is an assignment of a job to

appropriate computational and data services. At the very least, the job has to be mapped to a ComputeServer where it is to be executed. If the application specifies data files to be processed, then the mapping also includes the assignment of Data Hosts from which each of the files should be accessed. The mapping is saved back to the database as an attribute of the Job. At the same time, the state of the Job is changed from READY to SCHEDULED.

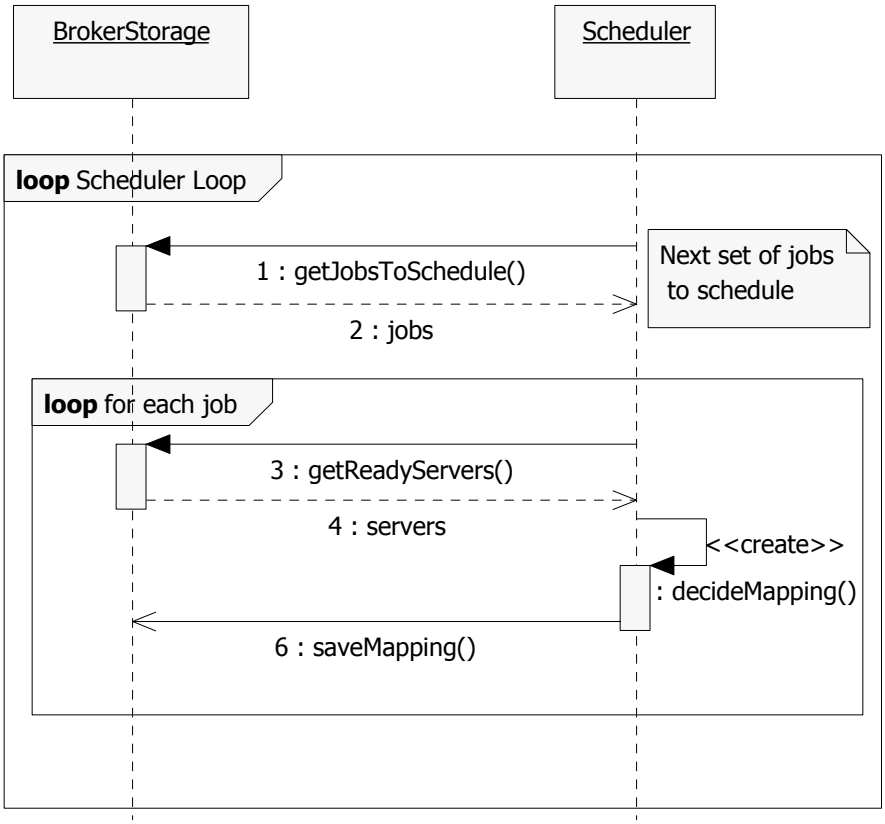


Figure 4.9: Schedule sequence diagram.

The sequence of operations for the Dispatcher is shown in Figure 4.10. The Dispatcher for each ComputeServer retrieves from the persistence database, the list of jobs that are in the SCHEDULED state and have been mapped to that server. It checks the status of the remote resource and available queue slots, if applicable. If the compute resource has an available slot, then the Dispatcher creates a JobWrapper depending on the ComputeServer selected for the job. The JobWrapper creates an Agent for the job that is specific to the remote resource architecture and middleware type by converting the Task Commands to middleware-specific invocations or system calls on the resource. It also performs job

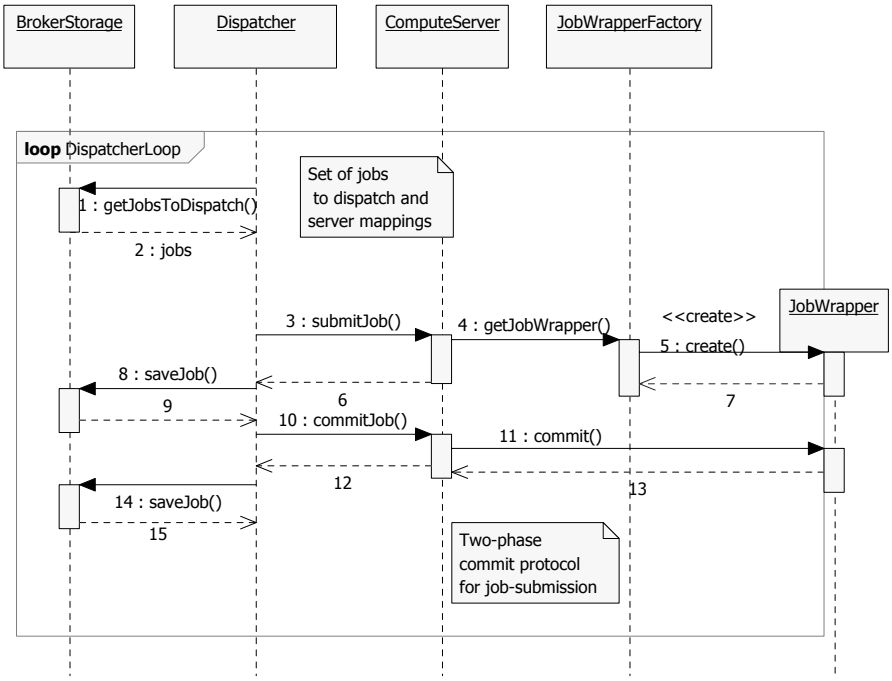


Figure 4.10: Dispatch sequence diagram.

submission in accordance with the protocols followed by the remote middleware.

Prior to the actual submission of the job, the files required for the job such as input files and executables are copied on to the remote resource. The files can be transferred on to the resource by the broker (push model) or they can be requested and copied by the remote resource (pull model). This allows a great deal of flexibility in implementing transfer modes, three of which are illustrated in Figure 4.11. For example, many Grid resources are behind firewalls that prohibit any connections to any port on the resource. In this case, using the pull model has the advantage that a transfer program on the resource can make an outbound connection through the firewall (Figure 4.11(b)). Another advantage of the pull model is that the source of the files can be a file server that is separate from the resource on which the broker is running thereby supporting a scenario in which the broker is behind a firewall as well (Figure 4.11(c)). The resource manager is a middleware component that is able to send and receive messages from the outside world through the firewall. During this process, the Job state is changed to STAGE_IN.

The JobWrapper then submits the Agent to the remote resource manager and waits for confirmation of acceptance. This is obtained as a remote handle to the job that uniquely identifies the job at the resource. The job state is changed to SUBMITTED and the job

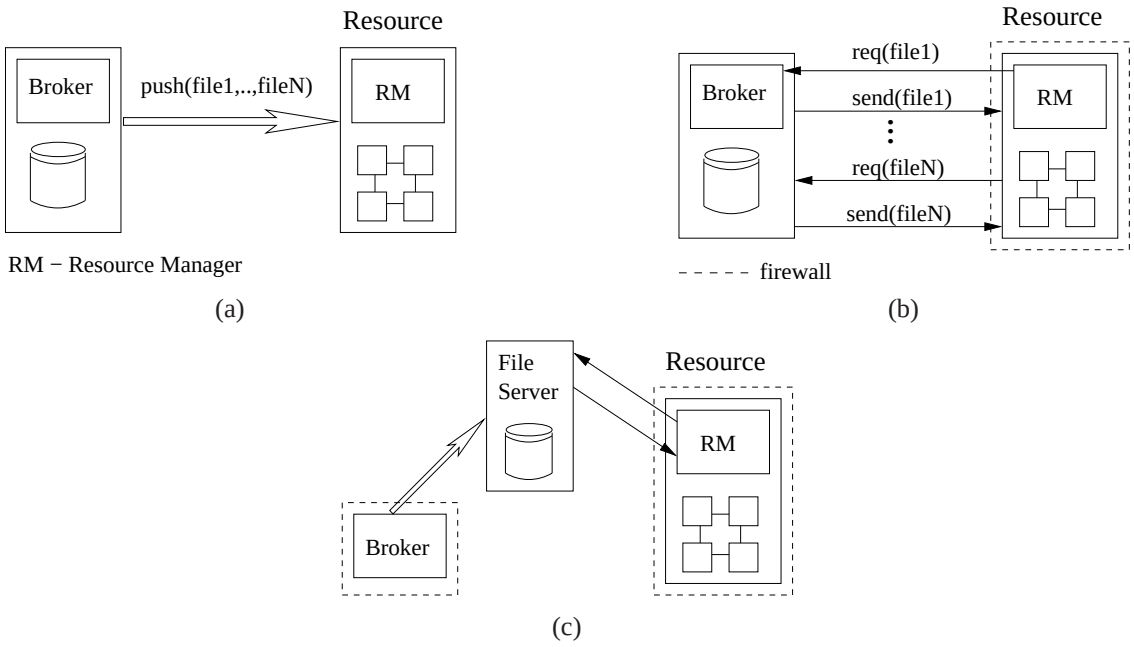


Figure 4.11: File transfer modes supported by the broker. (a) Push model of file transfer. (b) Pull model of file transfer. (c) A file transfer using an intermediate server.

along with the handle is saved back to the persistent storage. The number of available slots within the ComputeServer is decremented to reflect the submission and the server also saved to the persistent storage. If the handle is not received, then the job is considered as FAILED and is marked for re-scheduling.

The Dispatcher is also able to follow two-phase commit protocol for job submission as described by Czajkowski et al. [63]. In the first phase, the stage-in of the files and the job submission are performed and the dispatcher waits for an “agreement” message from the remote resource in the form of the remote handle. After the remote handle is received, the dispatcher sends a “commit” message to the remote resource which proceeds with the job submission and sends an acknowledgement back. The job state is changed to SUBMITTED only after the receipt of the acknowledgement. For resources running middleware that do not support the two-phase protocol, the receipt of remote handle is considered as acknowledgement of submission. In this case, if there is a network failure before the handle is received at the broker, the job will be marked as failed though it may have started execution at the remote node. In two phase protocol, the job is not processed by the Grid resource until the broker sends a commit message. Thus, two-phase commits ensure that job submission is carried out only once even in the face of network problems.

Prior to passing control to the JobWrapper, the Dispatcher selects a credential from the set of UserCredentials and binds it to the job just prior to dispatching it to the remote resource. This decision is based on the type of the middleware and the type of credential. Alternatively, a user may specify mapping of credentials to resources. This feature aids a user to seamlessly run jobs on different types of middleware, or even to use different credentials for the same type of middleware, at the same time.

JobMonitor

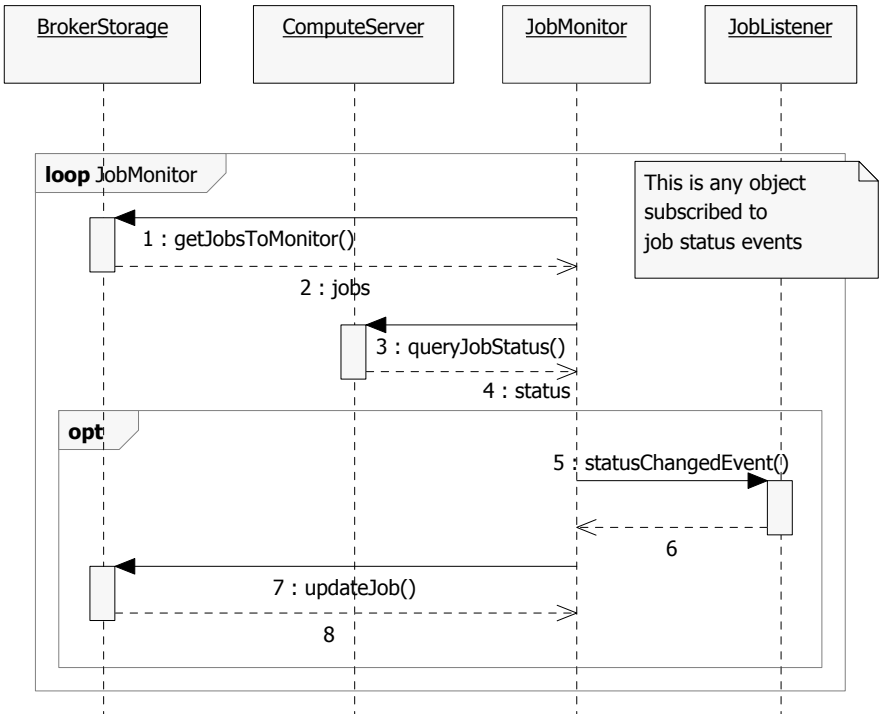


Figure 4.12: Job monitoring sequence diagram.

The JobMonitor for each ComputeServer keeps track of a job's progress after submission to that remote resource. As shown in Figure 4.12, the JobMonitor periodically requests the list of jobs that are in the SUBMITTED state on a particular resource, from the persistent storage. It uses the remote handle to query the status of the job using middleware-specific functionality. The query is a blocking call, and is therefore provided with a timeout period after which the JobMonitor cancels the query to proceed to the next job. Failure to contact the job on the remote resource is not considered as a failure of the job immediately. The JobMonitor tries to contact the job again for a set number of times

before giving up and marking the job as failed.

It is possible to implement JobListeners that will receive events from the JobMonitor when a job status is changed. The listeners can be entities inside the broker such as an event-driven scheduler, or outside the broker such as an applet within an application Web portal using the Gridbus broker [96].

4.3.3 Design Considerations and Solutions

The primary aim of the Gridbus broker is to provide a generic resource discovery and scheduling framework that abstracts the heterogeneous and dynamic nature of the Grid infrastructure and allows users to achieve different objectives. In Section 4.1, some of the challenges in achieving this aim are outlined. These challenges provide the requirements against which the broker is designed. The broker meets these requirements in the manner outlined as follows.

Service Heterogeneity

The Gridbus broker tackles the problem of heterogeneous Grid resources and service interfaces by adopting the principle of *minimal assumptions*. This means that throughout the broker, there are as few assumptions as possible about the nature of the environment in which it operates. The relationship between the objects in the broker is generic and independent of interaction models followed by any middleware. The broker, therefore, does not impose a particular configuration requirement on Grid resources and is thus able to use as many resources as possible. For example, a resource running any Unix-based operating system and with one of the supported middleware operational would be immediately useable by the broker as the latter requires only a POSIX-compliant shell environment that is standard on such machines.

The three-layer architecture of the broker also helps in maintaining this independence. For example, the assignment of jobs to resources is performed by the Scheduler which, as a component of the Core layer, has a middleware-independent view of the resources, while the Dispatcher dispatches the jobs to the resources. However, the actual interface with the remote resource happens through the middleware-specific JobWrapper which is

a part of the Execution layer.

The broker can also be made to interface to any new service or middleware by extending the appropriate classes. For example, support for a new middleware can be added by extending the abstract ComputeServer and JobWrapper classes. The Scheduler will then be able to immediately utilise any resource running that middleware. The interaction with the remote middleware is also independent of the rest of broker. Using this method, the broker has been extended to support a wide variety of both computational and data Grid middleware such as Globus, PBS and SRB among others [70, 117, 138]. With the support for Alchemi [138], which is a .NET-based desktop Grid computing framework for Windows platforms, and XGrid, a similar framework for Mac OS X, the Gridbus broker can already schedule jobs across almost all of the platforms in use today. Similarly, new information sources and data repositories can be supported by extending the InformationService and the DataHost classes respectively.

Support for Different Application Models and User Interfaces

The components within the broker are designed to be modular and there is a clean separation on the basis of functionality. Particularly, the division between workers and entities clearly delineates responsibilities among the components of the broker. Since the passive entities are just holders of information, the logic within the workers can be changed without affecting the former. It is also possible to introduce new workers that either use or extend the existing entities in different ways or introduce new entities of their own *without structural or design changes to the rest of the broker*. This loose coupling between components provides a lot of flexibility and enables the realisation of different application and system models [188].

The components within the Interface Layer convert the application, service and credential descriptions to broker entities and store them into the persistence database through the interfaces provided in the FarmingEngine. Thus, it is possible to support any form of description by mapping it to the entities within the broker. At present, the broker supports XML-based description of parameter sweep and bag-of-task applications and has a set of XML-based files for describing services and credentials. These inputs can also be provided directly to the broker through its APIs. The APIs allow direct manipulation of the

entities in the broker and thus, can be used to realise different application models.

The same mechanisms that allow the creation of application models also enable the creation of different user interfaces for the broker. The broker can be interfaced through command line, desktop clients or web portals. It is also possible to talk to the broker's persistence database directly through its APIs. This enables any component to retrieve information about the broker entities through normal SQL (Structured Query Language) queries. This ability is useful in scenarios such as an application Web portal requiring information about an ongoing execution through a broker installed on a machine different from the portal server.

Realisation of Different User Objectives

The broker allows for schedulers to be plugged-in, and hence is able to support new scheduling algorithms and policies. This enables the broker to adapt to new user requirements and objectives.The separation of the dispatch component from the scheduling provides a lot of flexibility for implementing the scheduling logic. These two components are also designed to be independent of the resource and middleware details. However, a developer can still choose to create schedulers that may require certain middleware-dependent services such as resource information services.

One of the main requirements of the design was the ability to execute generic data-oriented applications that may require access to one or more distributed datasets at the same time. Data services, represented by DataHosts, have the same level of importance as ComputeServers. Location of available replicas of a dataset can be gathered by querying the appropriate replica catalogs that are represented as InformationServices. Information about current network conditions such as available bandwidth, are important when large datasets are to be transferred. This is available through the NetworkLinks data structure as are properties such as pricing, classes of service and availability. CopyCommands in the Task allow the application to perform third-party (not involving the broker machine) point-to-point data transfers on the Grid. The application interpreter allows users to specify datasets as input parameters to their applications. In this case, the Interpreter will discover all the data repositories that host these datasets and the Scheduler will select one of the repositories for accessing the datasets. The data repositories are bound to the respective

jobs just prior to their dispatch and therefore, these can be changed at any time during scheduling. More importantly, the combination of all these features enables the broker to satisfy different requirements such as selecting data repositories for accessing datasets on the basis of price and/or performance and selecting network links offering a particular class of service. Examples of scheduling algorithms implemented in the broker that satisfy such requirements will be discussed later in the case study and in the next chapter.

The Gridbus broker has been designed from the ground up to support the computational economy paradigm [44], and assigns costs to various services including computation, data storage, and information services. The default scheduling policy uses these parameters to decide on an appropriate mapping strategy to schedule jobs on resources. The design also allows the ability to plug-in a market directory [221] which offers information about various priced services, and a Grid bank [25] that manages users credit in a Grid market. The ServiceMonitor is able to periodically refresh pricing information from market information services and therefore, provides the ability to make decisions in a highly-volatile Grid economy. The Gridbus broker also extends the notion of computational economy to data-intensive applications and has the ability to keep up with dynamic pricing and resource conditions.

The Gridbus broker is also able to support users who would like to use resources spread across multiple VOs by natively managing multiple credentials for multiple resources. This ability also allows users to access legacy data repositories that are not "Grid-enabled" and follow their own authentication mechanism, rather than Grid proxies.

Infrastructure

Grid environments are dynamic in nature. As a result, transient behaviour is not only a feature of the resources but also of the middleware itself. The broker design considers various types of failures that may occur either on a remote resource to which a job is submitted or on the broker machine itself during various stages of its operation. Remote failures include failure during job submission, execution and monitoring, or retrieving the outputs. These could be due to various reasons, such as incorrect configuration of the remote resource, incorrect job descriptions, network problems, unavailable data files, or usage policy restrictions on the resource. Local failures include unexpected system

crashes which lead to abrupt termination of the broker, unavailability of local input files and invalid input parameters.

The broker applies different fault recovery methods in different cases of remote failure. A job is not considered completed until it is determined that each of its constituent task activities have successfully exited. Thus, even if the middleware on the remote resource has signalled a successful completion, the JobMonitor checks to see if the job has generated required results and only then it is deemed successful. The JobMonitor may not be able to contact an executing job in case of transient network conditions. In such cases, the JobMonitor polls the job a set number of times, and if it is able to re-establish contact, then check the job's current status. In all other cases of job failure, the job is rescheduled on another available resource. The current implementation of the broker focuses on applications that consist of independent tasks, and therefore it is assumed that the failure of a job has no cascadng effect on the rest of the jobs that consitute the application.

The persistence system provides insurance against failure of the broker itself. At any point in the execution, the state of the broker is completely described by the contents of the entities and therefore, only these need to be stored within the persistence database. Any change in the status of a Job or a Service, as discovered by the respective Monitors, is immediately written back to the database so that the latter reflects the true state of the broker. Workers have no state of their own and hence, can be resumed from the point at which they failed by reading the entities from the persistent storage.

The broker tracks variations in resource availability by monitoring resource performance locally. Nothing is required to be installed at the remote resources and dependencies on metrics provided by the middleware are avoided by default. Thus, the broker is able to compare resources in a heterogeneous environment based on metrics that are independent of middleware and relative to the requirements of the current execution. However, it is to be noted there is no mechanism in the broker to inhibit usage of external performance monitors and other services if required.

4.4 Implementation

The Gridbus broker has been implemented in Java so that it can be deployed in Web-enabled environments such as Tomcat [19]-driven portals and also be used from the command line. A typical operation flow of the broker is as follows: The user describes the application in one of the supported input formats which is used to generate the Jobs. Jobs can be generated in different ways depending on the application model. The user's resource description generates a set of services, including ComputeServers, DataHosts, and InformationServices. The FarmingEngine starts the Scheduler, the ScheduleDispatchers, the JobMonitors and the ServiceMonitors. The Scheduler decides on a mapping of a Job to the appropriate Services. The ScheduleDispatcher then submits the mapped job to the chosen computational resource. Once the job is submitted to the mapped ComputeServer, its remote handle is provided to the JobMonitor for that service which uses it to query the job's execution status. When the job is done/failed, the results are collected and job cleanup is initiated. This cycle continues till all the jobs are scheduled and are done or failed.

This section illustrates the interaction of the broker with external entities using code samples. External input to the broker is explained using an example of composing a parameter sweep application. Interaction with remote middleware is explained using the example of the broker's operation with resources Grid-enabled using Globus Toolkit 2.4.3. The aim of this discussion is to present some of the challenges that were encountered during implementation and how they were handled.

4.4.1 Providing Input

Many scientific studies consist of repeating the same set of tasks for different scenarios. In computational terms, this means executing the same set of applications for different sets of data that are generated by different parameter values. This model is called the "parameter sweep model" of computation and offers a simple, yet powerful abstraction for creating distributed applications. A parameter sweep application can be easily converted into a set of independent jobs that are suited for deploying in Grid computing environments where challenges such as load volatility and long response times of individual nodes make it

difficult to adopt an application model that favours tightly-coupled components. Consequently, the parameter sweep model has proved to be very popular for Grid execution, and is therefore considered as one of the “killer applications” for Grids [1].

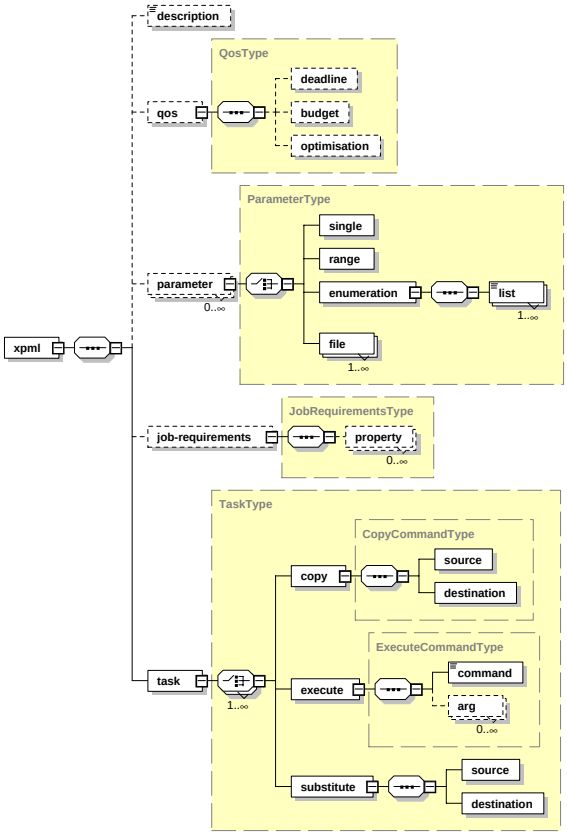


Figure 4.13: XPML Schema representation.

Parameter sweep applications are described in the broker using an XML-based declarative format called the eXtended Parametric Modelling Language (XPML). The XPML structure is shown in Figure 4.13 and consists of parameters, tasks and job requirements. An XPML parameter represents a set of values and has a name, a type and a domain from which the values are derived. An XPML task is a codification of the task specification as has been described previously in Section 4.3. XPML job requirements are conditions that need to be satisfied before a job is executed on a resource such as a specification of a par-

ticular machine architecture or minimum memory, bandwidth or disk space requirements. An example of an XPML file is shown in Figure 4.14. This input file describes a set of remote files, described through the **infile** parameter, provided as an input to the "wc" (word count) command on Unix systems. The output of the "wc" command is available through a file called "output.$jobname" where "$jobname" is a variable that has the value of the current job-id and is substituted at runtime.

```
<?xml version="1.0" encoding="UTF-8"?>
<xpml xmlns="http://schemas.gridbus.org/xpml/2006/01/xpml"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://schemas.gridbus.org/xpml/2006/01/xpml
     xml/schemas/XPMLSchema.xsd">
     <qos>
        <deadline value=""/>
        <budget value="789.0" />
        <optimisation value="TIME_DATA" />
     </qos>
      <parameter name="infile" type="gridfile" domain="file">
         <file protocol="srb" url="srb://db*"/>
      </parameter>
     <task>
        <copy>
           <source location="remote" file="$infile"/>
           <destination location="node" file="someFileName"/>
        </copy>
        <execute>
           <command value="wc"/>
           <arg value="someFileName"/>
           <arg value=" > "/>
           <arg value="output"/>
        </execute>
        <copy>
          <source location="node" file="output"/>
          <destination location="local" file="output.$jobname"/>
        </copy>
     </task>
</xpml>
```

Figure 4.14: XPML Example.

While XPML is based on the Nimrod "plan" file format [2], it introduces several new extensions on its own. These include: 1) Dynamic parameters, 2) a new "file" parameter that aids the parametrisation of data-intensive applications, 3) XPML job-requirements which help in narrowing down the list of resources that are useful for the application, and 4) the integration of QoS parameters within the application specification itself.

A *dynamic* parameter type has either an undefined or an unbounded domain whose definition or boundary conditions respectively, have to be established at runtime. In contrast, a *static* parameter is a variable whose domain is well-defined either as a range of values, as a single static value or as one among a set of values. One such dynamic parameter type is "gridfile" that describes a set of files over which the application has to be executed. This set can be described as a wild card search within a physical or a logical directory to be resolved at runtime. In the example XPML file shown in Figure 4.14, the section highlighted within the inner box shows the declaration of a parameter variable of type "gridfile" named **infile** which is described as a set of files with names starting with "db" that are stored in an SRB repository. The SRB repository/repositories from which information such as the individual filenames and locations of the files are available, have to be provided within the service declaration. The XPMLParser module in the broker queries the SRB MCAT (Metadata CATalog) service to resolve the parameter values. The parameter is later used in the "copy" section as a file to be copied to the remote node.

```
<resource type="compute" id="r2">
   <compute domain="remote">
       <remote middleware="globus">
             <globus hostname="belle.cs.mu.oz.au"
                     jobmanager="jobmanager-fork"
                     version="2.4">
             </globus>
        </remote>
   </compute>
</resource>
```

Figure 4.15: Example of a compute service description.

Other than XPML, the broker also supports a subset of the standard GGF JSDL (Job Submission Description Language) v.1.0 [17] to describe a single independent job. Multiple JSDLs can be combined to create Bag-of-Task applications. Details of available services such as location, cost and middleware type can be provided to the broker via another XML-based file. Figure 4.15 shows a description of a compute resource that is Grid-enabled through the Globus Toolkit version 2.4. Credentials are described in a separate file and are bound to the services based on the middleware type.

The XPMLParser converts the application description into a set of jobs that is then

stored into the persistence database. Similarly, the service descriptions are converted to service objects such as ComputeServers, DataHosts and InformationServices that are stored in the database as well. The user credentials however are not stored in the database to protect against exposure to inadvertent or deliberate access by other programs.

4.4.2 Middleware Interface

As mentioned previously in Section 4.3, the ComputeServers are subclassed depending on the type of middleware to be supported. Each of the subclasses implements middleware-specific functions for discovering properties, submitting jobs and monitoring them. The broker's interaction with resources running Globus Toolkit (GT) version 2.4 is presented as an example here. The Gridbus broker interfaces to Globus 2.4 resources through the Java Commodity Grids (CoG) Kit [210] that provides APIs for accessing Globus services through the Java framework.

The paragraphs that follow present and discuss code fragments relating to the implementation of three functions – resource discovery, job submission and job monitoring – for Globus Toolkit 2.4. The code fragments only highlight those features that illustrate the implementation of design concepts presented in the previous section. The two classes featured here are the GlobusComputeServer and the GlobusJobWrapper. The GlobusComputeServer extends the abstract ComputeServer and implements the functionality for managing a Globus resource including discovering properties and monitoring jobs. The GlobusJobWrapper provides the functions to create a Globus-specific job and submit it to the remote resource.

Figure 4.16 shows a fragment of `Service.discoverProperties()` as implemented in the GlobusComputeServer class. The ServiceMonitor (described in Section 4.3) invokes this function on all the Service objects. In the case of Globus, the first verification is whether the compute resource is alive and if so, is the provided UserCredential valid for accessing it. Failure of this verification means the resource is unusable as either the resource is not reachable or the user does not have the proper credentials for accessing it. Next, the Grid Resource Information Service (GRIS) on the remote resource is queried to obtain resource attributes which are later set within the ComputeServer object.

```
/**
 * Checks if the compute server is up, and sets all its attributes
 * @return true if the properties have been discovered
 */
protected boolean discoverProperties(UserCredential uc) {
  try{
     ProxyCredential pc = (ProxyCredential)uc;
     // Check if server is alive and the user can access it
     if (!checkPing(pc.getProxy())) {
          logger.info("Could not ping " + this.getHostname());
          return false;
     }
     // Building the query string
     String filter = "(&(objectclass=MdsHost)(Mds-Host-hn="+this.getHostname()+"))";
     // Querying the remote Globus Resource Information service
     NamingEnumeration results=MDSUtil.search("ldap://"+this.getHostname()+":2135",
                                filter, HOST_ATTRIBUTES);
     if(results==null) {
          logger.error("setValues() - Error in accessing MDS!!" ,null);
     } else {
          while(results.hasMore()) {
               // Set the attributes  in GlobusComputeServer
       ......
       ......
       }
     }
     return true;
}
```

Figure 4.16: The implementation of `Service.discoverProperties()` in GlobusComputeServer.

In the case of misconfigured GRIS services, it is possible that this query may block until its timeout expires or may fail altogether. However, the resource itself is not considered failed by the broker. The attributes are set to default values in the exception handler (not shown in Figure 4.16) and the resource is still considered alive.

Figure 4.17 shows a fragment of the implementation of the JobWrapper interface for Globus resources. The fragment shown in Figure 4.17 shows the workflow for a Globus job submission and also, the implementation of the two phase commit protocol for Globus. The first phase of job submission is represented by `execute()` function and the second phase by the `commit()` function. On the invocation of the `execute()` function, the JobWrapper creates the executable Agent for remote submission. Since Globus 2.4 is used to interface with resources running Unix and Unix-like (e.g. Solaris, HP-UX, AIX, Linux, etc.) operating systems, the GlobusJobWrapper creates a generic POSIX standards-compliant shell script that is supported by all Unix systems. The task commands are translated into Unix system commands within the script. For example, the

```
public class GlobusJobWrapper extends JobWrapper {
     ..............
     /** The 2-phase commit for GT2 has the following steps:
    * phase 1: Entry point: execute()*/
     protected void execute(Job job) throws Exception {

          /* 1. Create RSL and job shell script.*/
          GramAttributes rsl = new GramAttributes();
          ..............
          /*a) Translate Task Commands to Job Script commands
       *b) Set attributes within the Job RSL */
          ..............
          rsl.setExecutable(shellFileOnStagingServer);
          ..............
          rsl.set("twoPhase","yes");

          /* 2. stageIn input files to an intermediate server*/
          stageIn(job);

          /* 3. Request Gram Job using RSL, get the job handle and set it in the job */
          GramJob gramJob = new GramJob(rsl);
          gramJob.setCredentials(proxy);
          try {
               gramJob.request(contactString, true);
          } catch (WaitingForCommitException e) {
          /* 4. Return: remote handle for job*/
               job.setHandle(gramJob.getIDAsString());
          ..............
     }

     /** phase 2: Entry point: commit()*/
     protected void commit(Job job) throws JobCommitException {
          try {
               /* Send a commit-request signal to GRAM, to start actual execution */
               ProxyCredential pc = (ProxyCredential) job.getUserCredential();
               GSSCredential proxy = pc.getProxy();
               GramJob gramJob = new GramJob(proxy, "");
               gramJob.setID(job.getHandle());
               gramJob.signal(GramJob.SIGNAL_COMMIT_REQUEST);
               /* Throw exception in case of failure*/
          ..............
}
```

Figure 4.17: The implementation of `GlobusJobWrapper`.

Execute Command translates to an invocation of an executable already provided by the remote system or by the user. In the first case, the absolute path of the executable is either discovered from the remote system path or provided by the user in the application description or obtained from a remote application information service such as Grid Market Directory. In the second case, the executable itself is provided by the user for which the broker sets the relative path in the shell script. Dependencies on resource configurations with particular system library and shell interpreter versions is avoided by using only standard Unix system and shell commands.

After the creation of the script, the job itself is encoded as a Globus RSL (Resource Specification Language) [83] specification. The Globus RSL allows different job at-

tributes to be provided including the location of the executable, minimum resource requirements such as memory, and number of CPUs, and the preferred queue to which the job may be submitted. The Globus RSL also supports the pull model of file transfer. Therefore, it is possible to encode the location of the input files to be staged, in the RSL which will be parsed by the Globus Gatekeeper running on the remote node to stage-in the required files onto the resource. The location is encoded as a GASS (Globus Access to Secondary Storage) URL and hence, this also requires that there be a GASS server running on the machine where the input files are stored. Newer version of the RSL supplied with Globus version 4.0 supports file transfers from GridFTP servers which are better suited for transferring large-sized datasets. In a similar fashion, RSL also offers the ability to specify files that need to be transferred out of the resource on completion of the execution. After the required RSL attributes are set (through the Java CoG helper class), a `GramJob` is created which is then submitted in batch mode to the remote resource. If the request is successful, a string uniquely identifying the job on the remote resource is obtained. This is set as the handle of the job.

As an aside, it should be mentioned that Globus allows both batch and interactive job submission. In the latter mode, a *callback handler* is returned to the client. This can be used to keep track of the remote job. However, a temporary network failure between the client and the resource would not only mean the loss of the callback handler but also along with it, loss of all contact with the job. Therefore, the interactive method was considered not robust enough to be used in the broker.

The receipt of the job handle ends the first phase of submission. But, at the remote end, the job has not been processed yet. The ScheduleDispatcher then invokes the `commit()` function in the JobWrapper to signal the remote Gatekeeper to go ahead with the job. Acknowledgement of this request from the Gatekeeper means that the job submission is completed. The lifecycle of the JobWrapper comes to an end and control over the job now passes onto the JobMonitor.

The JobMonitor invokes the `queryJobStatus()` function in the middleware-specific implementation of the ComputeServer abstract class. Figure 4.18 shows the implementation of `queryJobStatus()` for Globus resources. For querying, a temporary Gramjob is created with the same handle as that was received after submitting the job. Globus it-

```
public int queryJobStatus(Job job) {
     int counter=3;
     // Create a short-lived Globus Gram Job
     // with the same handle as the job to query
     GSSCredential proxy =
         ((ProxyCredential)job.getUserCredential()).getProxy();
     GramJob gramjob = new GramJob(proxy,"");
     gramjob.setID(job.getHandle());

     while (counter-- > 0){
      try{
          // Query remote job
          Gram.jobStatus(gramjob);
          switch (gramjob.getStatus()){
          // Handle various Globus statuses and map to broker's Job status
          ......................
          ......................
          case GramJob.STATUS_FAILED:
               status=JobStatus.FAILED;
          break;
          case GramJob.STATUS_DONE:
               status=JobStatus.STAGE_OUT;
          break;
         }
      }catch (GramException ge){
           if (ge.getErrorCode()==GramException.ERROR_CONTACTING_JOB_MANAGER){
             if(counter!=0){
                 // If first attempt at contacting job is unsuccessful
                 //  try 3 more times.
                 continue;
             }else{
                 //Assume the job is completed initially
            //Check the error output later.
                 status = JobStatus.STAGE_OUT;
             }
     ......................
     ......................
   }
        return status;
}
```

Figure 4.18: The implementation of `ComputeServer.queryJobStatus()` in GlobusComputeServer.

self has a set of job states that are mapped to potentially different job states within the broker because of the different state transitions within each of the systems. For example, when the status of a Globus job (or GramJob) is "DONE", it merely means that the execution finished with a zero exit status on the remote resource. However, this does not mean that the job has successfully completed. Success is determined by the broker on the basis of whether the job has completed all of its constituent task activities. Therefore, on GramJob.STATUS_DONE, the status of the job within the broker is set to STAGE_OUT which signals the JobMonitor to retrieve the standard output (`stdout`) and error (`stderr`) files from the remote resource through a separate process, which are then examined to determined the actual job completion status.

While the job is polled at regular intervals, it may complete on the remote resource or fail in between job queries. The remote Globus Gatekeeper also closes down the JobManager responsible for the job to conserve memory and CPU usage. This means that the next query from the JobMonitor will throw an exception as it tries to contact the closed JobManager. In this case, the JobMonitor persists with two more attempts to rule out possibilities such as a temporary network failure. After this, it assumes that the job has had a success or a failure at the remote node and proceeds to the stage out process to examine the outputs. A failure to obtain the output files is considered as a failure of the job itself.

4.5 Related Work

The challenges presented in Section 4.1 have motivated the development of a large number of Grid resource brokering and application deployment systems. Examples of such systems are Nimrod/G [43], Condor-G [91], APST [50, 52] and EU-DataGrid Broker [16] (later succeeded by the gLite). These are chosen for detailed comparison against the Gridbus broker as their objectives and approaches are similar to that of the broker. These are compared to the broker against the manner in which they handle the challenges outlined in Section 4.1.

4.5.1 Condor-G

Condor-G is a computational management system that allows users to manage multi-domain, heterogeneous resources running Globus [85] and Condor middleware, as if they belong to a single domain. It combines the harnessing of resources in a single administrative domain provided by Condor with the resource discovery, resource access and security protocols provided by the Globus Toolkit. At the user side, Condor-G provides API and command line tools to submit jobs, cancel them, query their status, and to access log files. A new Grid Manager daemon is created for each job request which then submits the job to the remote Globus gatekeeper that starts a new JobManager process. Condor-G provides "execute once" semantics by using a two phase commit protocol for job submission and completion. Fault tolerance is provided on the submission side by a persistent job queue and on the remote side by keeping persistent state of the active job within the JobManager.

Jobs are executed on the remote resource within a *mobile sandbox* that traps system calls issued by the task back to the originating system and are checkpointed periodically using Condor mechanisms. This technology called Condor GlideIn effectively treats a collection of Grid resources as a Condor pool. Resource brokering is provided by matching user requirements with information available from services such as GRIS and GIIS through the ClassAds [172] mechanism. Condor-G is a part of many projects such as EGEE, VDT, UK e-Science and Grid2003 among others, and is used by workflow management systems such as Pegasus [69].

Condor-G operates in a Globus and Condor-only environment and installs a virtualization layer at each node at runtime that traps system calls and provides checkpointing facilities. Condor can utilise batch queueing systems such as LSF, PBS and NQE but only through Globus GRAM protocols. Condor-G provides strong fault tolerance mechanisms as a result of its close integration with the low-level Grid middleware. It implements the two-phase commit protocol for Globus job submission for ensuring that the job is executed only once. Through the GlideIn mechanism, it is able to provide libraries that perform checkpointing and job migration and maintains a persistent queue to guard against local failures.

Though Condor-G by itself does not provide any data access functions, it can interface to services such as Kangaroo [202] and Stork [118] that enable it to mediate access to remote files and manage data transfers. Condor-G allows for creation of applications belonging to different models such as workflows and supports different scheduling strategies. However, it does not natively support resource costs and has no functions for optimisations based on pricing.

4.5.2 AppLeS Parameter Sweep Template (APST)

APST is an environment for scheduling and deploying large-scale parameter sweep applications (PSAs) on Grid platforms. APST provides mechanisms for deploying applications on different Grid middleware and schedulers that take into account PSAs with data requirements. APST consists of two processes: the daemon, which deploys and manages applications and the client, which is a console for the users to enter their input. The input

is XML-based and no modification of the application is required for it to be deployed on Grid resources. The APST Scheduler allocates resources based on several parameters including predictions of resource performance, expected network bandwidths and historical data. Examples of scheduling strategies include algorithms that take into account PSAs with shared input files [51] and Divisible Load Scheduling-based algorithms [219]. The scheduler uses a Data Manager and a Compute Manager to deploy and monitor data transfers and computations respectively. These in turn use Actuators to talk to the various Grid middleware. A Metadata Manager talks to different information sources such as Network Weather Service (NWS) [217] and the Globus Monitoring and Discovery Service (MDS) [65] and supplies the gathered data to the scheduler.

APST supports different low-level Grid middleware through the use of Actuators and also allows for different scheduling algorithms to be implemented. However, it is focused towards parameter sweep applications. APST provides the ability to specify data repositories of different types in the input file and has a separate data manager to manage data transfers. However, it does not seem to consider the possibility of multiple sources for any datafile other than those created by replication of the data files during the execution of an application.

4.5.3 Nimrod/G

Nimrod/G [1, 43] is a tool for automated scheduling and execution of parameter sweep applications on Grids. It provides a declarative parametric modelling language through which the task specifications can be provided for an “experiment” or execution of an application. Scheduling within Nimrod/G follows an economic model in which the resources have costs associated with them and the users have to expend their budgets in order to execute their jobs on the resources [45]. The user can also specify Quality of Service (QoS) requirements such as a deadline for finishing the experiment and an option for choosing between a faster yet more expensive execution vis-a-vis a slower but cheaper process. Architecture-wise, Nimrod/G consists of a Task Farming Engine (TFE) for managing an execution, a Scheduler that talks to various information services and decides on resource allocations, and a Dispatcher that creates Agents and sends them to remote nodes

for execution. An Agent can manage more than one job at a remote site.

Nimrod/G works with UNIX-based resources enabled through Globus middleware only. At the time of writing, Nimrod/G does not take into account location of data during scheduling and does not have parametric representation for an application's data requirements. It does, however, have the ability to specify data transfers from the client node to the remote resource and back. Nimrod/G follows the computational economy paradigm and provides four algorithms [45] - *time optimisation*, *cost optimisation*, *cost-time optimisation* and *conservative time optimisation* - for scheduling parameter sweep computationally-intensive applications.

4.5.4 gLite

gLite is an integrated middleware package for the EGEE project that consists of modules for security, information and management, data and job management services. Here the focus is on the gLite's WMS (Workload Management System) package that provides access to resources running various middleware such as Globus, Condor and Storage Resource Manager (SRM) [186]. gLite treats resources as Compute Elements (CE) or Storage Elements (SEs) depending on whether they are computational or data resources respectively. Jobs are generally non-interactive and batch oriented. The gLite Workload Management System (WMS) handles job scheduling and resource allocation and uses Condor-G for job dispatch and management. The WMS accepts job requests and stores them in its Task Queue. A Matchmaker sub-component matches job requests against resource information stored in an Information Super Market (ISM) sub-component, using the Condor ClassAds mechanism. The WMS uses both eager scheduling (jobs are 'pushed' to the resource) and lazy scheduling (resource 'pulls' or requests for jobs). Data required by a job scheduled at a CE is replicated to the nearest SE.

gLite works within a standardised Grid environment running EGEE middleware and has a standardised client configuration that requires external services such as R-GMA (Relational Grid Monitoring Architecture) Information System [60]. gLite is installed on a dedicated machine and accepts job requests from local and remote clients. Thus, it is a centralised resource brokering system and therefore, differs considerably from the

other resource brokers which are primarily user-directed, client-focused resource brokering mechanisms.

gLite automatically schedules replication of the required data for a job to the closest Storage Element to the Compute Element where the job has been scheduled. But, the locations of the data are not taken into account during the selection of Compute Element itself. That is, gLite does not perform any optimisation for reducing the amount of data to be transferred for an execution. gLite interfaces with an Accounting module that enables it to keep track of usage and charge users. However, it does not provide any economy-based scheduling of Grid applications.

4.5.5 Comparison

Table 4.1 compares the Gridbus broker and the related work discussed previously against characteristics derived from the challenges listed at the beginning of this chapter. While it may seem unfair to compare the other brokers against requirements that they were not designed for, this comparison is only a discussion of how the design of the Gridbus broker is different and not a measure of the applicability of the brokers to any situation.

From the table, it can be seen that the design of the Gridbus broker was motivated by different considerations than that of the other brokers. The focus of the Gridbus broker has been on scheduling and executing distributed data-intensive applications on potentially heterogenous Grid resources. This is in contrast to Condor-G and gLite, that are primarily job management systems, or Nimrod/G, that focuses on computationally intensive parameter sweep applications. APST schedules jobs so as to reuse data that has already been transferred but the initial location of the data is the client machine or the machine on which the broker is executing. Also, the Gridbus broker has been designed to enable economy-based strategies for Grid scheduling, going beyond the resource pricing provided by Nimrod-G, by supporting services such as market directories and resource accounting. The Gridbus broker is a single user system, each application execution requires a different instantiation of the broker. This is different to systems such as gLite which is a centralised resource broker that handles multiple users.

One of the design principles that differentiate the Gridbus broker from the other re-

Table 4.1: Comparison of different resource brokers.

Characteristics	Condor-G	APST	Nimrod/G	gLite	Gridbus
I Service Heterogeneity					
a. Support for different low-level computational middleware	Through Globus and Condor	Yes	Through Globus	Only EGEE	Yes
b. Support for different Data Grid middleware	Through Stork	Yes	No	Only EGEE	Yes
c. Equality of different service types	No	No	No	No	Yes
II Support for Application Models					
a. Basic application model	Single Job	Parameter Sweep	Parameter Sweep	Single Job	Independent Tasks
b. Internal support for workflows	None	None	None	None	Available
c. Allow internal entities to be accessed through APIs	No	Unknown	Yes	No	Yes
III Realisation of Different User Objectives					
a. Job scheduling based on location of data	No	Yes	No	No	Yes
b. Third-party data transfers	Through Stork	No	No	Data Replication	Yes
c. Late binding of data locations to jobs	Through Stork and Kangaroo	No	No	No	Yes
d. Access to dynamic network information	No	Yes	No	Yes	Yes
e. Resource pricing and cost-based scheduling	No	No	Yes	No	Yes
f. Managing multiple credentials across VOs	No	No	No	No	Yes
IV Fault tolerance					
a. Checkpointing of jobs	Yes	No	No	Through Condor-G	No
b. Execute-once semantics	Yes	No	No	Yes	Yes
c. Local persistent store	Yes	Yes	Yes	Yes	Yes
d. Dependencies on remote information services	No	No	Yes	Yes	No

source brokers is the support for different Grid middleware. Except APST, the others work with only Globus services, or in the case of gLite, with resources running only EGEE middleware. This decoupling in the Gridbus broker has been achieved by limiting all middleware dependencies to the Execution layer and by not assuming the presence of specific services or libraries on the remote resources. The benefit of this loose coupling of the broker to low-level Grid middleware is that it can utilize a greater number and range of resources. The object-oriented nature of the broker also makes it easy to support any low-level Grid middleware, if required. For example, for compute Grid middleware, all that is required is to extend the ComputeServer and the JobWrapper classes. However, this approach has its disadvantages as well. As mentioned before, Condor-G is able to provide stronger fault tolerance semantics due to its close integration with Globus and because it is able to install a virtualisation layer that checkpoints jobs on the resources. Such a feature would require the broker to assume the availability of certain libraries on the resources.

Another distinctive design feature of the Gridbus broker is the equality of all types of services, whether they are compute, data or information services. That is, all the Grid services are treated as first-class citizens. This enables the broker to achieve different kinds of strategies such as those which give more prominence to data rather than computational requirements. The other brokers, with the exception of APST, focus on the computational aspect of the jobs. While these handle data in different ways - for example, Condor-G presents the data requirements of an application to Stork to handle while gLite simply replicates it on demand - they do not generally have strategies to choose a specific data repository at runtime based on current network conditions. The Gridbus broker has been designed to provision for such requirements.

The next section describes a case study on using the Gridbus broker to create and deploy a distributed data-intensive High Energy Physics application on Grid resources where many of the above-mentioned features were utilised. It also illustrates how jobs were scheduled with respect to the location of required data and how this approach led to better turnaround times than strategies that do not take data location into account.

4.6 A Case Study in High Energy Physics

High Energy Physics (HEP) is a fundamental science studying matter at the very smallest scales. Experiments in HEP involve studying collisions between fundamental particles and are conducted at particle accelerators that are built and manned by large collaborations involving thousands of physicists from institutes around the world. Accelerators record millions of collisions (also called events) per second that are then filtered on-site to limit data output to "interesting" events. The filtered data is then distributed to the members of collaborations for analysis.

Computing resource requirements for HEP are increasing exponentially because of advancements in the efficiencies of particle accelerators and the increasing size of collaborations. The CERN Large Hadron Collider will generate events at the rate of PetaBytes per sec (PB/sec) which will be filtered to create a data stream of 100 MB/sec. The CMS and ATLAS experiments have the largest collaborations among the experiments at LHC, each consisting of 2000 members from 150 institutions from 30 countries worldwide who have to be provided access to the data that is generated at the LHC. The CERN LHC particle accelerator is therefore, frequently cited as a justification for the need for Data Grids in experimental high energy physics [41, 106].

4.6.1 The Belle Project

Charge-Parity (CP) violation was first observed in 1964, by studying the decays of K-mesons. Briefly C is the symmetry operation of particle - antiparticle inversion, and P that of space inversion. The issue today is whether the Standard Model (SM) of Physics offers a complete description of CP violation, or, more importantly, whether new physics is needed to explain it. Answering this question requires very detailed study of this subtle effect.

The Belle experiment, built and operated by a collaboration of 400 researchers across 50 institutes from 10 countries, is probing CP-violation by studying the decay of the B-mesons produced in the KEKB accelerator at the Japanese High Energy Accelerator Research Organization (KEK) in Tsukuba. The increasing efficiencies of the KEKB accelerator have led to an increase in the rate of data production from the Belle experiment.

The current experiment and simulation data set is tens of terabytes in size. While this increase is extremely desirable for the study of B-meson decays, it begins to pose problems for the processing and access of data at geographically remote institutions, such as those within Australia. Hence, it is important for Data Grid techniques to be applied in this experiment [216].

4.6.2 The Application Model

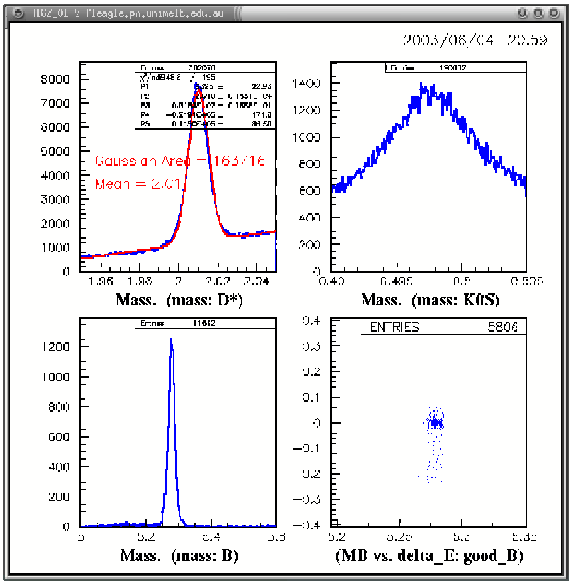


Figure 4.19: A histogram produced from Belle data analysis.

A typical analysis workload in the Belle experiment is split into two streams: data and simulation. Raw data is recorded from various sensors within a detector and stored as separate measurements or “events”. Simulated or Monte-Carlo data involves the generation of events and then detailed detector simulation. From this point on, the analysis streams are very similar. The data is reconstructed, which involves the correlation of sensor information. Data summaries are generated for ease of analysis. Adachi, et. al [4] report that the size of raw data for each event is 35 kB which increases to 60 kB after reconstruction. For user analyses, this is reduced to 12 kB. However, since millions of events are recorded, the aggregate size of the data is still quite large. As an example, within the Belle experiment, 10 TB of data summary information exists at present. These are “skimmed” to produce subsets of the data of most interest to each physicist’s analysis.

These are around 100 GB in size for Belle users. These are then analysed to generate plots and histograms and can then be used for statistical analysis by applying further cuts. A histogram that is produced from data analysis is shown in Figure 4.19. For simulated data, this process is repeated until the analysis is perfected. The simulated data can then be used for systematic error analysis. The same analysis process is performed on data to obtain a result, provided there are no large differences between data and simulation.

The Belle computing effort within Australia is spearheaded by groups at the University of Melbourne and University of Sydney. This effort involves both generation of data from simulation and analysis of simulated and actual event data. The former is very CPU-intensive while the latter is both CPU and I/O-intensive. The event data is obtained from the Storage Facility at KEK while the simulated data and the results of the analysis should be made available to entire collaboration. This results in heavy network requirements for the collaboration both into and out of Australia.

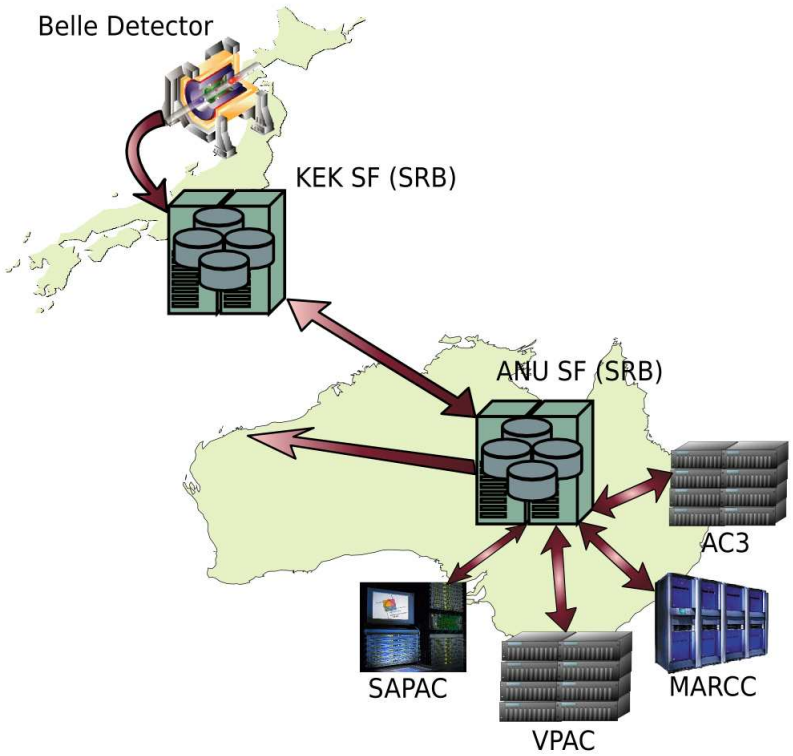


Figure 4.20: The infrastructure for the Belle Data Grid.

The computing, storage and networking requirements for the Australian side of the Belle collaboration have resulted in the Data Grid infrastructure shown in Figure 4.20 [127].

The storage middleware used throughout the Belle collaboration is the Storage Resource Broker (SRB). The main SRB repository in Australia is at the Australian National University Storage Facility (ANUSF) which is federated with the SRB repository at KEK. The required experimental data is downloaded from KEK to ANUSF via SRB protocols. This data is accessed by analysis jobs which are scheduled onto the High Performance Computing (HPC) resources around Australia. The results of the analysis are stored back into ANUSF by the jobs. Similarly, data generated by simulation jobs is also stored back to ANUSF. This data is then retrieved by members of the Belle collaboration at other sites around the world. However, the export and to a certain extent, the import of large data is limited by the expense involved in international transfers from and to Australia. Data transfers in terabytes are still conducted through airmail [147].

The Australian HPC resources are shared between users from all educational institutions in the country belonging to several application domains. Therefore, scheduling of Belle jobs must take into account variations in resource availability and job queueing time due to the varying load on these resources. Data transfer requirements must also be taken into account for analysis jobs. Specifically, data should be accessed from the storage repository nearest to the point of computation to reduce data transfer time and network usage. Transferred data must also be reused, if possible, by successive jobs. These requirements for selecting computational and data resources motivate the use of a resource broker for scheduling analysis jobs on the Grid resources. The following sections discuss in detail the use of the Gridbus broker for scheduling Belle analysis jobs on an Australian Grid testbed.

4.6.3 Experimental Setup

The experiment was conducted using the Belle Analysis Data Grid (BADG) [205] testbed that was set up in Australia in collaboration with IBM. The resources in the testbed and their configurations (circa early 2004) is shown in Figure 4.21. The testbed resources are located in Sydney (Dept. of Physics, University of Sydney), Canberra (Australian National University), Melbourne (School of Physics and the Dept. of Computer Science, University of Melbourne) and Adelaide (Dept. of Computer Science, University of Ade-

laide). At the time of the experiment, all the nodes in the testbed, except for the one in Adelaide, were connected via GrangeNet (Grid And Next Generation Network) [155]. GrangeNet is a three year program to install, develop and operate a multi-gigabit network supporting Grid and advanced communications services across Australia. Hence, there was a higher bandwidth between the Melbourne, Canberra and Sydney resources. Two of these resources (Adelaide and Sydney) were effectively functioning as single processor machines as the Symmetric Multi-Processing (SMP) Linux kernel was not running on them.

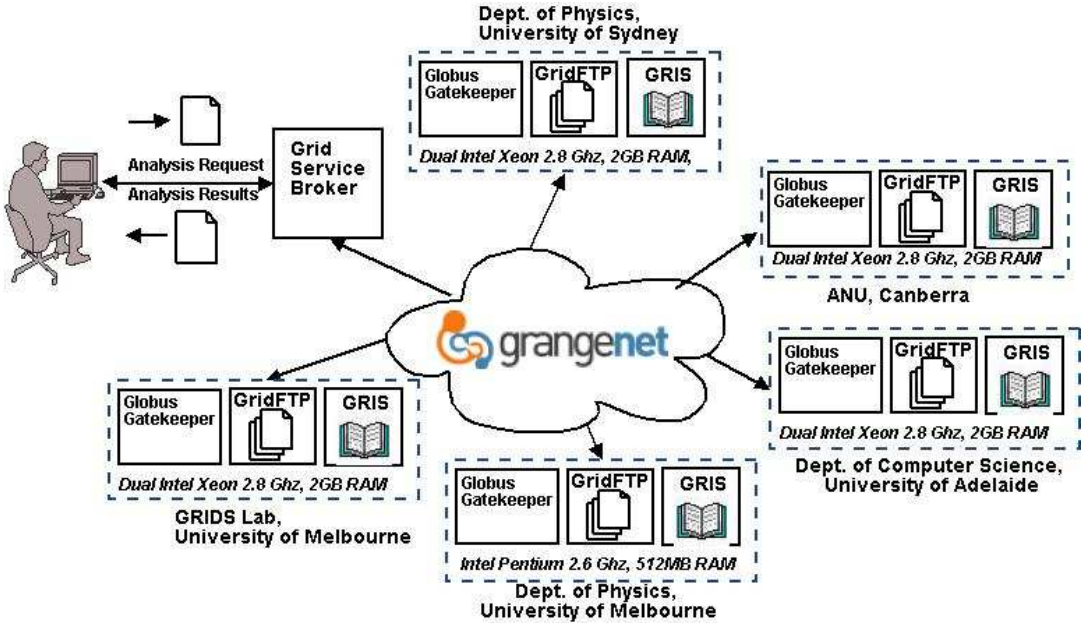


Figure 4.21: Australian Belle Analysis Data Grid testbed.

All the resources in this testbed were Grid-enabled through Globus Toolkit v. 2.4.3. Network conditions between the resources were monitored through the Network Weather Service (NWS) [217]. An NWS sensor was started on each of the resources which reported to the NWS name server located in Melbourne. An NWS activity for monitoring bandwidth was defined at the name server within which a clique containing all the resources on the testbed was created. Members of the clique conduct pairwise experiments one at a time to determine network conditions between them. Querying the name server at any point provides the bandwidth and latency between any 2 members of the clique. Data that was produced on one site in BADG had to be shared with the other sites. For this purpose, a Data Catalog was set up for the Belle Data Grid by the School of Physics using

the Globus Replica Catalog (RC) mechanism [216]. The Globus RC is described in Chapter 3, Section 3.2.3. The Gridbus broker itself was deployed on the Melbourne Computer Science machine and broker agents were dispatched at runtime to the other resources for executing jobs and initiating data transfers.

The Belle experiment uses a software framework, called the Belle AnalySis Framework (BASF) [4] and written in C++ and FORTRAN, for the entire event processing workflow from simulation and event filtering to user analysis. Programs can be written to provide specific functionality and can be defined in scripts as modules to be loaded dynamically at runtime. BASF has been extended to access data from SRB and GridFTP-enabled data repositories through one such module, developed at the School of Physics [216]. This extension also enables it to access streaming data thus reducing considerably the delay that is incurred before the data is completely available on the executing node. This application was installed prior to the execution on all the nodes of the testbed.

4.6.4 Scheduling Belle Jobs on BADG

Figure 4.22 lists the algorithm developed for scheduling Belle analysis jobs on the BADG testbed. The "network proximity" of a compute resource to a data host is a measure of the available bandwidth between the resources. Some of the data resources also have computation facilities, in which case the data transfer time is assumed to be zero as the data host and the compute resource reside at the same site. The scheduler minimises the amount of data transfer involved for executing a job by dispatching jobs to compute servers which are close to the source of data. A naïve way of achieving this is to run the jobs only on those machines that contain their data. But, the data hosts may not have the best computational resources. Therefore, this algorithm considers both the computation time for the job and the data transfer time.

The most important measure in this evaluation was the completion time of the jobs. From multiple runs, it was determined that the Belle analysis jobs had similar computation times even with different datasets. Therefore, the scheduler used the simple measure of the job completion ratio - the ratio of the number of jobs completed to the number of jobs allocated - to evaluate the performance of the computational resources. At every regular

INITIALISATION

1. Identify characteristics, configuration, capability, and suitability of compute resources using the Grid information services (GIS). From the task definition, obtain the data query parameters.
2. Resolve the data query parameter to obtain the list of Logical Data Files (LDFs) from the Data Catalog
3. For each LDF, get the data sources or Data Hosts that store the LDF by querying the Data Catalog.

SCHEDULING

Repeat while there exist unprocessed jobs *(This step is triggered for each scheduling event. The event period is a function of job processing time, rescheduling overhead, resource share variation, etc.)*

1. For each **compute resource**
   (a) Predict and establish the job consumption rate or the available resource share through the measure and extrapolation strategy taking into account the time taken to process previous jobs.
   (b) Use this estimate along with its current commitment to determine expected job completion time.
2. If any of the compute resource has jobs that are yet to be dispatched for execution and there is variation in resource availability in the Grid, then move such jobs to the `Unassigned-Jobs-List`.
3. Repeat until *all unassigned jobs are scheduled or all compute resources have reached their maximum job limit*
   (a) Select the next job from the Unassigned-Jobs-List.
   (b) Identify all Data Hosts that contain the LDF associated with the job.
   (c) Create a Data-ComputeResource-List for the selected job: ” For each data host, identify a compute resource that can complete the job earliest given its current commitment, job completion rate, and data transfer time using current available bandwidth estimates.
   (d) Select a Data Host and compute resource pair with the earliest job completion time from the Data- ComputeResource-List.
   (e) If there exists such a resource pair, then assign the job to the compute resource and remove it from the Unassigned-Jobs-List.
4. End of Scheduling Loop.

Figure 4.22: A Scheduling Algorithm for Belle Analysis jobs.

polling interval, the scheduler evaluated the progress of job completion for each compute resource in the following manner:

$$r_S = \frac{J_C}{J_Q}$$

where r_S was the job completion ratio for a particular resource, J_C is the number of jobs that were completed on that particular resource in the previous polling interval and J_Q is the number of jobs that were queued on that resource in the previous allocation.

The scheduler then calculated the average job completion ratio, R_S at the N^{th} polling

interval as:

$$R_S = R'_S * (1 - \frac{1}{N}) + \frac{r_S}{N}$$

where R'_S was the average job completion ratio for the $(N-1)^{th}$ polling interval. The averaging of the ratio provides a measure of the resource performance from the beginning of the scheduling process and can be considered as an approximate indicator of the future performance of that resource.

Each resource was assigned a *job limit*, the maximum number of jobs that can be allocated out of current list of jobs waiting for execution, proportional to its average job completion ratio. The scheduler then iterates through the list of unassigned jobs one at a time. For each job, it first selects the data host that contains the file required for the job and then, selects a compute resource that has the highest available bandwidth to that data host. If this allocation plus previously allocated jobs and current running jobs on the resource exceeds the job limit for that resource, then the scheduler looks for the next available nearest compute resource.

4.6.5 Evaluation

Trial Dataset

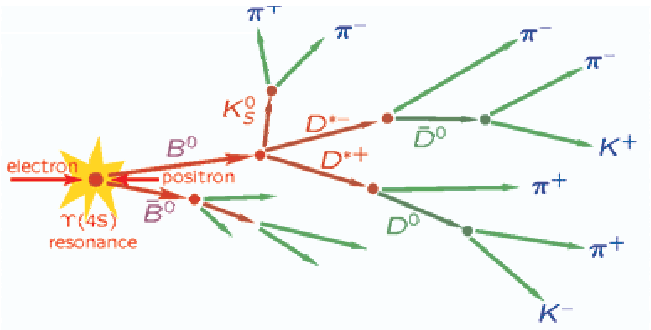


Figure 4.23: The $B^0 \rightarrow D^{*+}D^{*-}K_s$ decay chain.

For validating the broker, a simulation of a "decay chain" of particles has been used. A decay chain occurs when an unstable particle decays into another and so on until a stable particle state is reached. This is typical of the events in a particle accelerator. The experiment consists of 2 parts, both of which involve execution over the Grid. In the

first part, 100,000 events of the decay chain $B^0 \rightarrow D^{*+}D^{*-}K_s$ shown in Figure 4.23 are simulated via distributed generation and this data is entered into the replica catalog. In the analysis part, the replica catalog is queried for the generated data and this is analysed over the Belle Data Grid. The histograms resulting from this analysis are then returned as output. Here only the results of the analysis are discussed as it involved accessing remote data.

```
<parameter name="INFILE" type="gridfile" domain="file" >
    <file protocol="lfn"
          url="lfn:/users/winton/fsimddks/fsimdata*.mdst" />
</parameter>
<job-requirements>
    <property name="minmemory" value="500"/>
</job-requirements>
<task>
    <copy>
        <source location="local" file="ddks_ana.so" />
        <destination location="node" file="ddks_ana.so" />
    </copy>
    <copy>
        <source location="local" file="libanalyser.so" />
        <destination location="node" file="libanalyser.so" />
    </copy>
    <copy>
        <source location="local" file="libbase_analyser.so" />
        <destination location="node" file="libbase_analyser.so" />
    </copy>
    ..........................
    <execute>
            <command value="./runme.ddksana" />
            <arg value="$INFILE" />
            <arg value="$jobname" />
    </execute>
    <copy>
       <source location="node" file="ddks-$jobname.hbook" />
       <destination location="local" file="ddks-$jobname.hbook" />
    </copy>
</task>
```

Figure 4.24: A XPML file for HEP analysis.

An XPML file describing requirements for the analysis is shown in Figure 4.24. The parameter **$INFILE** describes a logical file location, listed in Globus Replica Catalog, that can be either a directory or a collection of files. The broker resolves the logical file location to the actual filenames and their physical locations. The XPML file also instructs

copying of user defined analysis modules and configuration files to the remote sites before any execution is started. The main task involves executing a user-defined shell script (`runme.ddksana`) at the remote site which has 2 input parameters: the full network path to the data file and the name of the job itself. The shell script invokes BASF at the remote site to conduct the analysis over the data file and produce histograms (`*.hbook`). The histograms are then copied over to the broker host machine.

The Logical file name in this particular experiment resolved to 100 Monte Carlo simulation data files. Therefore, the experiment set consisted of 100 jobs, each dealing with the analysis of one data file using BASF. Each of these input data files was 30 MB in size. The entire data set was equally distributed among the five data hosts i.e. each of them has 20 data files each. The data was also not replicated between the resources, therefore, the dataset on each resource remained unique to it.

Results of Evaluation

Three scheduling scenarios were evaluated: (1) scheduling with computation limited to only those resources with data, (2) scheduling without considering location of data, and (3) the adaptive scheduling (presented in Figure 4.22) that optimises computation based on the location of data. The experiments were carried out on April 19th, 2004 between 18:00 and 23:00 AEST. At that time, the Globus gatekeeper service on the Adelaide machine was down and so, it could not be used as a computational resource. However, it was possible to obtain data from it through GridFTP. Hence, jobs that depended on data hosted on the Adelaide server were able to be executed on other machines in the second and third strategies. A graph depicting the comparison of the total time taken for each strategy to execute all the jobs is shown in Figure 4.25 and another comparing resource performance for different scheduling strategies is shown in Figure 4.26.

In the first strategy (scheduling limited to resources with the data for the job), jobs were executed only on those resources which hosted the data files related to those jobs. No data transfers were involved in this scenario. As is displayed in the graph in Figure 4.26, all of the resources except the one in Adelaide were able to execute 20 jobs each. The jobs that were scheduled on that resource failed, as its computational service was unavailable. Hence, Figure 4.25 shows the total time taken for only 80 successful jobs out of 100.

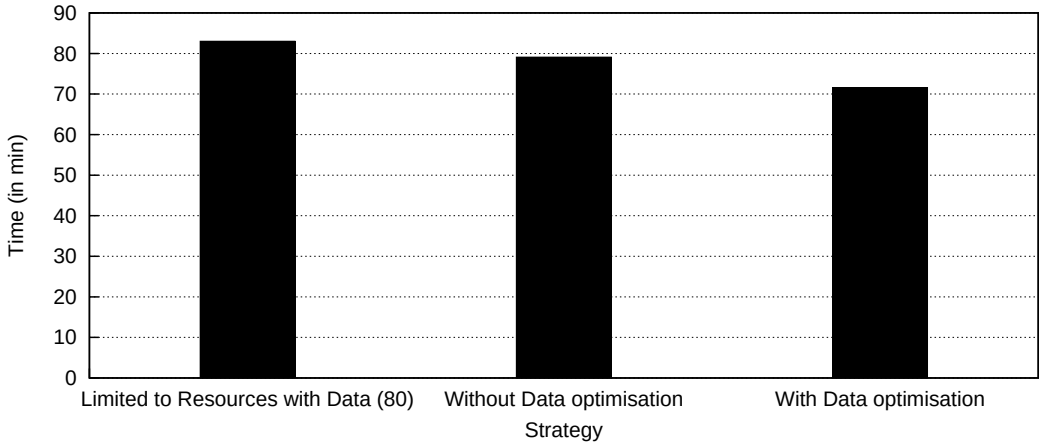


Figure 4.25: Total time taken for each scheduling strategy.

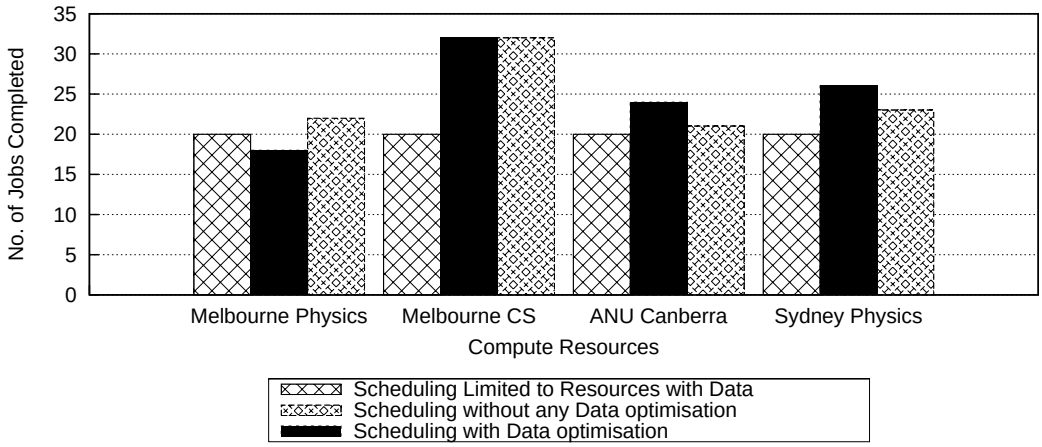


Figure 4.26: Comparison of resource performance under different scheduling strategies.

However, this time also includes the time taken by the scheduler to conclude that the remaining 20 jobs have failed. In this setup, the related data was exclusively located on that resource and hence, these jobs were not reassigned to other compute resources. Thus, a major disadvantage of this scheduling strategy was exposed.

In the second strategy (scheduling without any data optimisation), the jobs were executed on those nodes that have the most available computational resources. That is, there was no optimisation based on location of data within this policy. The Adelaide server was considered a failed resource and was not given any jobs. However, the jobs that utilised data files hosted on this machine were executed on other resources. This strategy involves the maximum amount of data transfer which makes it unsuitable for applications involving large data transfers and utilising resources connected by slow networks.

The last evaluation was carried out using the adaptive scheduling algorithm presented in Figure 4.22. In this case, as there were no multiple data hosts for the same data, the policy was reduced to dispatching jobs to the best available compute resource that had the best available bandwidth to the host for the related data. It can be seen from Figure 4.26 that most of the jobs that accessed data present on the Adelaide resource were scheduled on the Melbourne Physics and CS resources because the latter had consistently higher available bandwidth to the former. This is shown in the plot of the available bandwidth from the University of Adelaide to other resources within the testbed measured during the execution, given in Figure 4.27. The NWS name server was polled every scheduling interval for the bandwidth measurements. As can be seen from Figure 4.25, this strategy took the least time of all three.

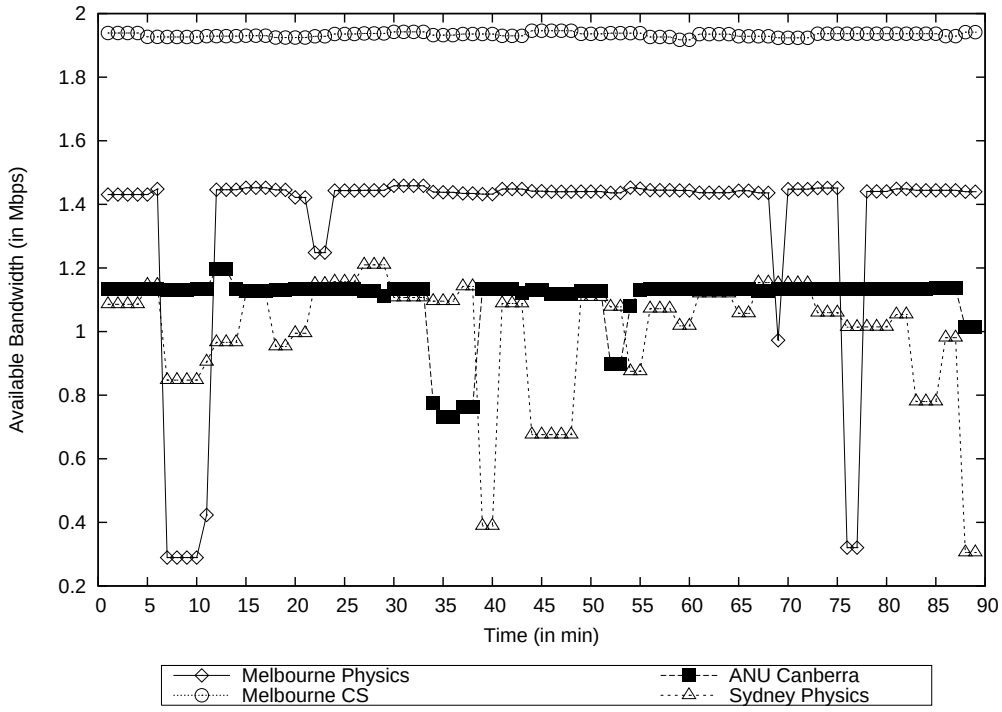


Figure 4.27: Available bandwidth from University of Adelaide to other resources in the testbed.

4.7 Summary

While Grids provide users access to distributed resources in a secure and unified manner, the problems of application composition for Grid environments, selection of appropriate

resources and heterogeneity of infrastructure present significant barriers to widespread usage of Grids for problem solving. Therefore, resource brokers have been developed to bridge the gap between users and Grid computing environments. However, resource brokers developed so far have either not taken into account or given less importance to factors such as location of data, available bandwidth or cost of data transfer that come into picture while scheduling distributed data-intensive applications on Grid resources. Such applications are common in scientific domains where the workload consists of analysing experimental data to produce results.

This chapter presented one such Grid resource broker that was designed with the aim of dealing with the afore mentioned issues as well as allowing users to achieve multiple scheduling objectives. The broker is designed as a collection of loosely-coupled components that are assigned definite roles and interact through standard interfaces. The design enables any of the components to be extended without affecting the rest of the broker. This, therefore, enables creation of different application models and schedulers that are able to achieve different objectives without design changes to the broker and allows extending the broker to support any middleware interfaces required.

The use of the broker to schedule data intensive applications is shown through a case study of Grid-based analysis of Belle experimental data. This case study introduced an adaptive scheduling algorithm that considered the time of data transfer along with the time taken for computation while scheduling a job on to a compute resource. Experimental evaluation showed that this algorithm performed better than scheduling jobs without regard to location of data.

The positive results from this experiment encourage further exploration into scheduling distributed data-intensive applications. The next chapter presents a formal model for this problem that takes into account both time and economic cost (expense) of job execution. This model is then applied to develop deadline and budget constrained cost and time minimization algorithms for the scheduling problem. The broker's ability to support multiple objective functions is then illustrated by its usage in evaluating the performance of these algorithms on a Grid testbed.

Chapter 5

The Scheduling Model and Cost-Aware Algorithms

Distributed data-intensive applications commonly process datasets, which may be each replicated on various storage repositories that are connected to each other and to the computational sites through networks of varying capability. Also, the datasets are generally large enough (of the order of GigaBytes (GB) and higher) that transferring them from storage resources to the eventual point of execution produces a noticeable impact on the execution time of the application. There may be costs involved in the usage of various computational, storage and networking resources for activities such as transferring data and executing jobs. This chapter defines the problem of scheduling distributed data-intensive applications on to Grid resources and presents a formal resource and application model for the problem.

The model is then applied to present an algorithm for scheduling a Bag-of-Tasks (BoT) application on a set of geographically distributed, heterogeneous compute and data resources. Each of the tasks within the application depends on multiple datasets that may be distributed anywhere within the Grid. The algorithm aims to minimise either the overall cost or the time of execution depending on the user's preference subject to two user-defined constraints - the deadline by which the processing must be completed and the overall budget for performing the computation. This algorithm is then evaluated on a real Grid testbed through the Gridbus broker and the results of the experiments are presented.

5.1 The Scheduling Problem

Section 4.6 of Chapter 4 discussed the scheduling of a High Energy Physics (HEP) application on Grid resources involving simultaneous selection of computational and storage resources. The jobs created for this application were data-intensive and processed one dataset that was fetched from a remote repository if required. It was also shown how adaptive scheduling with regard to the location of data is able to offer better performance.

This scenario motivates exploration of scheduling of data-intensive applications on Grid resources. As discussed in Chapter 3, Section 3.2.1, HEP is only one of the scientific domains that is making use of Data Grids. Other areas such as Astronomy, Climate Modeling and BioInformatics have computational and data requirements similar to that of HEP.

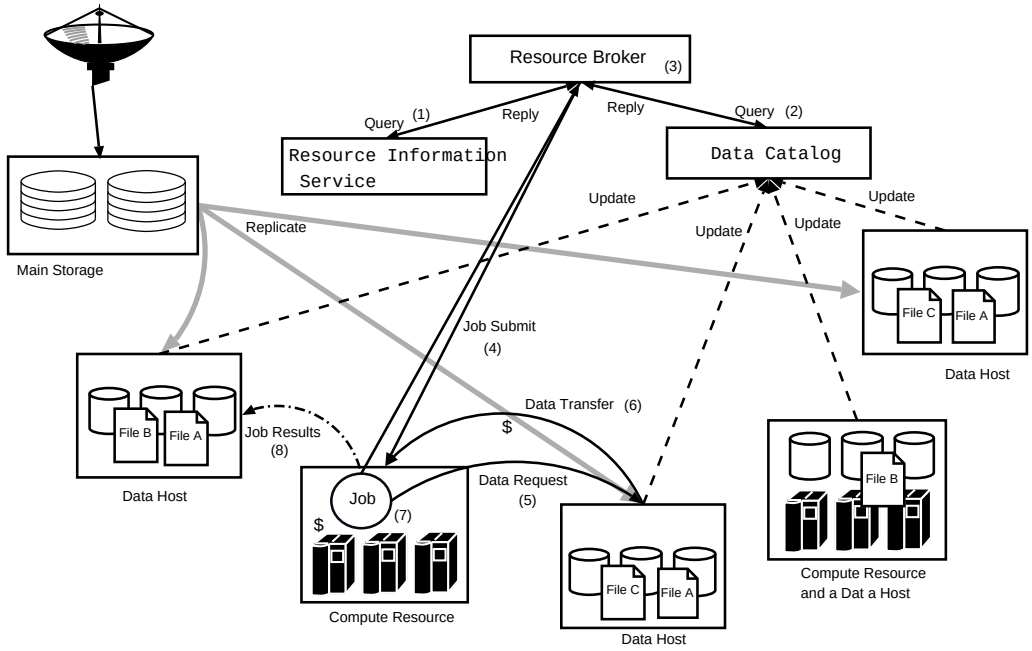


Figure 5.1: The scheduler's perspective of a Data Grid environment.

Based on application deployment experiences and on the scenarios drawn up for users of the production Data Grid projects such as LHC Grid [106], it is possible to arrive at the picture of a typical Data Grid environment as shown in Fig. 5.1. This environment is composed mainly of storage resources, or *data hosts*, which store the data and compute resources which run the jobs that execute upon the data. It is possible that the same resource may contain both storage and computation capabilities. For example, it could be

a supercomputing centre which has a Mass Storage Facility attached to it.

Initially, data generated by an experiment or through simulation may be stored at a few *master* data hosts as *datasets* and made available to the members of the scientific collaboration. Examples of such master repositories are the Tier 0 and Tier 1 centres in the MONARC model for the LHC Grid (see Chapter 3, Section 3.1.1). Over the course of time and after several user requests, the datasets may be replicated on several repositories in the Data Grid. This replication can also occur due to the access and storage policies of the Virtual Organisation (VO) for the collaboration that are guided by various criteria such as minimum bandwidth, storage and computational requirements, data security and access restrictions and data locality issues. Information about the datasets and their replicas are registered into a Data Catalog such as the Globus Replica Catalog [208] or the SRB Metadata Catalog [26] that can be queried by members of the collaboration.

A data-intensive computing environment can also be perceived as a real-world economic system wherein there are producers and consumers of data distributed geographically across multiple organisations. Producers are entities which generate the data and control its distribution via mirroring at various replica locations around the globe. The consumers in this system would be the users or, by proxy, their applications which need to analyse this data to produce meaningful results. The users may want to investigate specific datasets out of a set of hundreds and thousands and may have specific application requirements that need not be fulfilled at every computational site. In such large collaborations, there can be a lot of pressure on the data infrastructure (i.e., network and storage elements). The pressure becomes more acute when a nontrivial percentage of the users are interested in the same datasets simultaneously, thus causing appearance of "network hotspots". Such an effect is commonly observed in the Internet and the World Wide Web [140]. While a robust and adaptive replication mechanism can alleviate some of the above problems, the same problems of data access and transfer costs affect the effectiveness and efficiency of such a mechanism. Pricing resources to reflect supply and demand in order to regulate their usage has been explored in previous publications [42, 154, 191, 213]. Therefore, in such an economy-based system, there are costs or expenses associated with accessing, processing and transferring data.

On the consumer side, a scientist who wants to analyse some of the available datasets

specifies his requirements to the Resource Broker. These requirements may take the form of data specifications such as date of generation, experiment type and data type; application specifications such as a particular version; resource specifications such as architecture, minimum walltime required and queue type; and Quality of Service (QoS) specifications such as the deadline for the analysis job, the budget available and preference, if any, for the cheapest or the fastest processing according to needs and priorities. The step-by-step procedure for executing the analysis is shown in Figure 5.1. The Resource Broker gathers information about the available computational resources through a resource information service (1) and about the datasets through the Data Catalog (2). Here, only resources that meet the specifications and minimum requirements such as minimum free memory and storage threshold are considered as suitable candidates for job execution. It then creates the jobs according to the application description provided by the user. The scheduler within the broker then makes decision on where to submit a job based on the availability and cost of the computational resource, the minimisation preference and the location, access and transfer costs of the data required for the job (3). The job is dispatched to the selected remote computational resource (4) where it requests the data from the replica location selected by the scheduler (5 & 6). After the job has finished processing (7), the results are sent back to the Resource Broker or another storage resource which then updates the data catalog (8). This process is repeated until all the jobs within the set have completed.

Chapter 3, Section 3.1.4 has classified data-intensive Grid applications as belonging to either process-oriented, independent jobs, bag-of-tasks or workflow models. This thesis primarily deals with applications that belong to the bag-of-tasks paradigm wherein each application can be "decomposed" into a set of non-interdependent (or independent) tasks. The tasks are indivisible and therefore, each task is translated into a job that is scheduled onto a computational resource (or a *compute resource*) and requests datasets from the storage resources (or *data hosts*). Each of these datasets may be replicated at several locations that are connected to each other and to the compute resources through networks of varying capability. Therefore, there is an "explosion of choices" for selecting resources to execute a job and to access the datasets it requires. The scheduler within the resource broker has to make decisions at two levels. At the level of an individual job, the scheduling

strategy has to navigate through the multitude of choices to select a compute resource for executing the job and a subset of data hosts such that each dataset required for the job can be obtained from one of the data hosts in the set. This scenario is illustrated in Figure 5.2. This is termed as *matching* or allocation of resources to jobs. The entire set of jobs must be scheduled in such a manner that the user objectives, that is common to the entire set, must be met. This problem therefore becomes one of *ordering* or assigning the set of jobs that have already been matched to the resources previously.

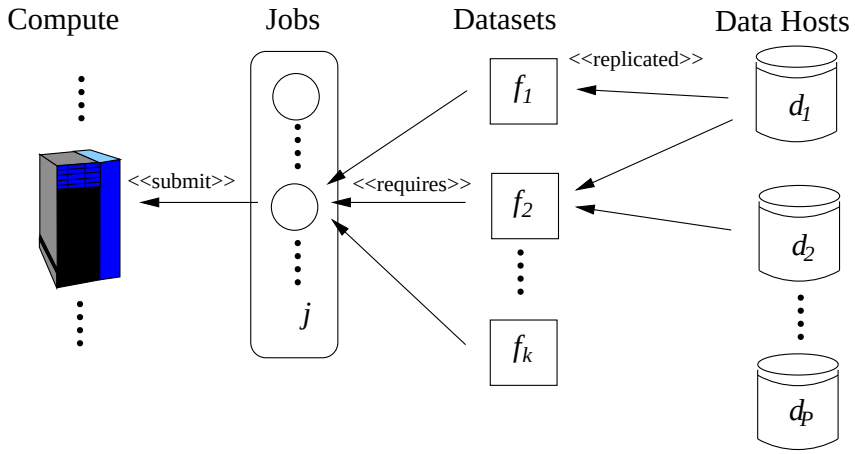


Figure 5.2: Mapping Problem.

5.2 Model

5.2.1 Resource Model

A data-intensive computing environment, as described previously, can be considered to consist of a set of M *compute resources*, $R = \{r_1, r_2, \ldots, r_M\}$ and a set of P *data hosts*, $D = \{d_1, d_2, \ldots, d_P\}$. Within production Grids, a compute resource is commonly a high performance computing platform such as a cluster consisting of processing nodes that are connected in a private local area network and are managed by a batch job submission system hosted at the "head" or "front-end" node connected to the public Internet. However, it is possible to have other types of compute resources as well as symmetric multi-processing systems such as the testbed resources used in the HEP case study in Chapter 4.

A *data host* can be a dedicated storage resource such as a Mass Storage Facility con-

nected to the Internet. At the very least, it may be a storage device attached to a compute resource in which case it inherits the network properties of the latter. It is important to note that even in the second case, the data host is considered as a separate entity from the compute resource. Figure 5.3 shows a simplified data-intensive computing environment consisting of four compute resources and an equal number of data hosts connected by links of different bandwidths.

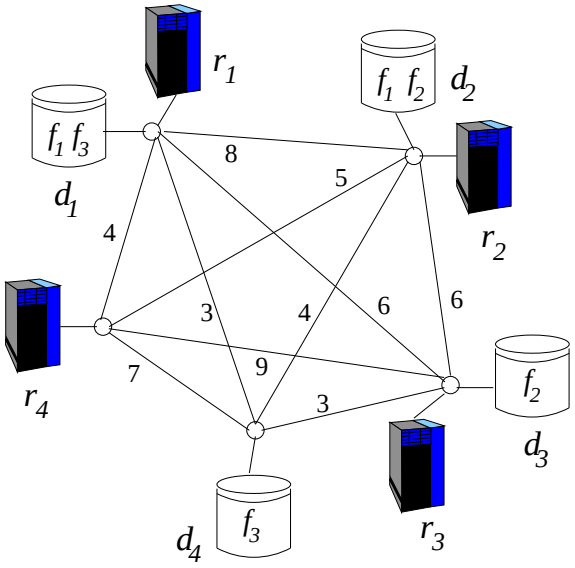


Figure 5.3: A data-intensive environment.

The physical network between the resources consists of entities such as routers, switches, links and hubs. However, the model in this thesis abstracts the physical network to consider the logical network topology wherein each compute resource is connected to every other data host by a distinct network link as shown in Figure 5.3. This logical link is denoted by $Link(r_m, d_p), r_m \in R, d_p \in D$. The bandwidth of the logical link between two resources is the bottleneck bandwidth of the actual physical network between the resources and is given by $BW(Link(r_m, d_p))$. This information is available from various information sources such as the Network Weather Service [217]. The numbers alongside the links in Figure 5.3 depict the bandwidths of the various logical links in the network.

The time taken by a compute resource to access a dataset located on the storage resource at the same site is limited only by the intra-site bandwidth if the storage is a separate physical machine or by the bandwidth between the hard disk and other peripherals if the storage is on the compute machine itself. In both cases, it is considered to be an

order of magnitude lower than the time taken to access a dataset through the Internet from other sites as there is contention for bandwidth among the various sites. Therefore, for the purpose of this study, only the bandwidth between different physical sites is taken into account.

Data is organised in the form of datasets. A dataset can be an aggregated set of files, a set of records or even a part of a large file. Datasets are replicated on the data hosts by a separate replication process that follows a strategy such as one of those described in Chapter 3, Section 3.1.3 which takes into consideration various factors such as locality of access, load on the data host and available storage space. Information about the datasets and their location is available through a catalog such as the Storage Resource Broker Metadata Catalog [169].

5.2.2 Application Model

The application is composed of a set of N jobs, $J = \{j_1, j_2, \ldots, j_N\}$, without interdependencies. Typically, $N \gg M$, the number of compute resources. Also, a job is the smallest unit of computation, that is, it is not possible to divide a job into smaller sub-units. It is also associated with a set of K' datasets, $F = \{f_1, f_2, \ldots, f_{K'}\}$, which are distributed on members of D. Specifically, for a dataset $f_k \in F$, $D_{f_k} \subseteq D$ is the set of data hosts on which f_k is replicated and from which it is available. Also, D_{f_1} and D_{f_2} need not be pairwise disjoint for every $f_1, f_2 \in F$. In other words, a data host can serve multiple datasets at a time.

A job $j \in J$ processes a subset of F of size K denoted by F^j. Each job requires one processor in a compute resource for executing the job and one data host each for accessing each of the K datasets required by the job. The compute resource and the data hosts thus selected are collectively referred to as the *resource set* associated with the job and is denoted by $S^j = \{R^j, D^j\}$ where $R^j \subseteq R$ is a singleton representing the compute resource selected for executing the job and D^j is an L-sized set of data hosts chosen for accessing the datasets required by the job. Therefore, $R^j = \{r\}, r \in R$ and $D^j = \bigcup D_f, f \in F^j$. Since multiple datasets can be retrieved from a single data host, $L \leq K$, the number of datasets required for the job. Figure 5.4 shows an example of such

a job j that requires resources shown in Figure 5.3.

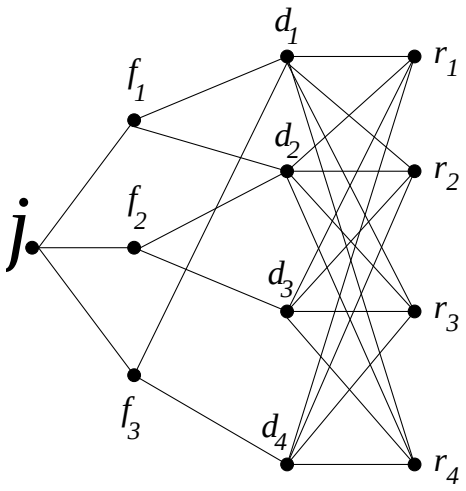


Figure 5.4: Job Model.

Job Execution Time Model

The job execution time model followed here is extended from that presented by Maheswaran, et. al [142]. Consider a job j that has been submitted for execution to a compute resource r. The time spent in waiting in the queue on the compute resource is denoted by $T_w(j,r)$ and the expected execution time of the job is given by $T_e(j,r)$. T_w increases with increasing load on the resource. Likewise, T_e is the time spent in purely computational operations and depends on the processing speed of the individual nodes within the compute resource. For each dataset $f \in F^j$, the time required to transfer f from d_f to r is given by

$$T_t(f, d_f, r) = Response\_time(d_f) + Size(f)/BW(Link(d_f, r))$$

$Response\_time(d_f)$ is the difference between the time when the request was made to d_f and the time when the first byte of the dataset f is received at r. This is a measure of the latency of the response and is therefore, an increasing function of the load on the data host. The *estimated completion time* for the job, $T_{ct}(j)$, is the wallclock time taken for the job from submission till eventual completion and is a function of these three times. Figure 5.5 shows two examples of data-intensive jobs with times involved in various stages shown

along a horizontal time-axis. In this figure, for convenience, the time for transferring $f_1, f_2, \ldots, f_k$ is denoted by $T_{f_1}, T_{f_2}, \ldots, T_{f_k}$ respectively.

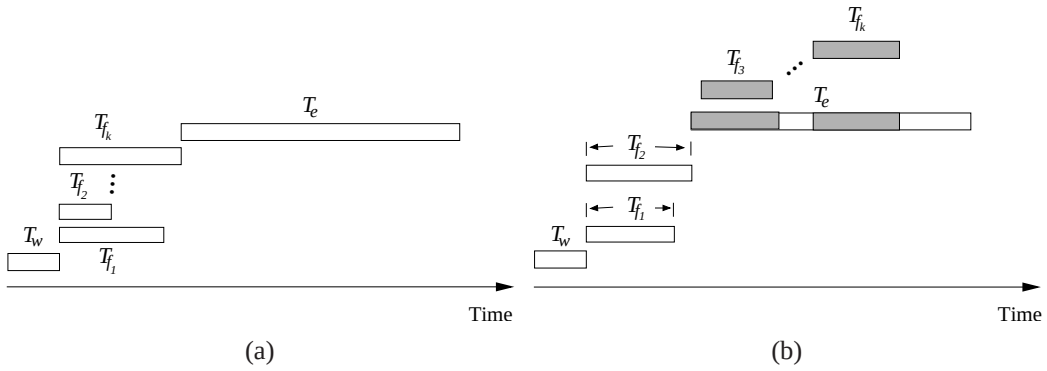


Figure 5.5: Job Execution Stages and Times (Gray areas denote overlaps between the computation and data operations).

The impact of the transfer time of the datasets is dependent on the manner in which the dataset is processed by the job. For example, Figure 5.5(a) shows a common scenario in which Grid applications request and receive the required datasets in parallel before starting computation. In this case,

$$T_{ct}(j) = T_w(j,r) + \max_{f \in F^j}(T_t(f, d_f, r)) + T_e(j,r)$$

However, the number of simultaneous transfers on a link determines the bandwidth available for each transfer and therefore, the T_t.

Figure 5.5(b) shows a more generic data processing approach in which some of the datasets are transferred completely prior to execution and the rest are accessed as streams during the execution. The grey areas show the overlap of computation and communication. A restricted form of this model was applied in the case study of the Belle application in the previous chapter where only one file was accessed as a stream during the execution of the application. In this case, the transfer time of the streamed data is masked by the computation time of the application. However, data access still affects the performance of the application. If there is a latency associated with accessing the data, the application may still have to wait until the first byte of the data is received at the compute resource.

The thesis focuses on the application models of the first type, that is, applications that require all the datasets to be transferred to the actual compute resource (or its associated

data host) before execution. This is the most common model followed by data-intensive applications [173]. Also, the impact of data transfer time is the highest in this model. However, it is possible that lessons learnt from scheduling these type of applications may also be applicable to the other types of data-intensive applications.

Economic Cost Model

In an economy-based system, there are costs associated with the access, transfer and processing of data. The processing cost is levied upon by the computational service provider, while the transfer cost comes on account of the access cost for the data host and the cost of transferring datasets from the data host to the compute resource through the network.

The economic cost of executing the job j on the compute resource r is denoted by $C_e(j,r)$ and the cost of transferring the dataset $f \in F^j$ from $d_f \in D_f$ to r by $C_t(f,d_f,r)$ where

$$C_t(f,d_f,r) = Access\_cost(d_f) + Size(f) \times Cost(Link(d_f,r))$$

Here, $Access\_cost(d_f)$ is the cost of requesting a dataset which is levied by the data host. It can be an increasing function on either the size of the requested dataset or the load on the data host or both. This cost regulates the size of the dataset being requested and the load which the data host can handle. $Cost(Link(d_f,r))$ is the cost of transferring a unit size (eg. 1 MB or GB) of the requested dataset through the network link between the data host and the compute resource. The cost of the link may increase with the Quality of Service (QoS) being provided by the network. For example, in a network supporting different channels with different Quality of Services as described by Hui, et al. [109], the channel with a higher QoS may be more expensive but the data may be transferred faster. Hence, the file is transferred faster but at a higher expense. All traffic within a Local Area Network (LAN) is considered to be essentially free, that is, no cost is levied upon them. Therefore, the total execution cost for job j, $C(j)$ is given by

$$C(j) = C_e(j,r) + \sum_{f \in F^j} C_t(f,d_f,r)$$

The notations that have been presented till now are summarised in Table 5.1.

Table 5.1: Notations.

Symbol	Definition
$R = \{r_m\}_{m=1}^{M}$	Set of M compute resources
$D = \{d_p\}_{p=1}^{P}$	Set of P data hosts
$Link(r, d)$	Logical link between $r \in R$ and $d \in D$
$BW(Link(r, d))$	Available bandwidth of $Link(r, d)$. It is the bottleneck bandwidth of the actual physical network between r and d
$Cost(Link(r, d))$	Price of moving a unit size of data (in MB or GB) through $Link(r, d)$
$F = \{f_k\}_{k=1}^{K'}$	Set of K' datasets required by the application
D_f	Subset of D on which f is replicated
$J = \{j_n\}_{n=1}^{N}$	Set of N jobs created for the application
F^j	Set of K datasets required by $j \in J$
R^j	Singleton set representing the compute resource executing $j \in J$
D^j	For a job j, the set of L data hosts from which the K datasets are retrieved, $L \leq K$
S^j	Resource set associated with $j \in J$
$T_w(j, r)$	Expected waiting time for job j in the batch queue at r
$T_t(f, d_f, r)$	Expected time for transferring $f \in F^j$ from $d_f \in D_f$ to r
$T_e(j, r)$	Expected execution time for job j on resource r
$T_{ct}(j)$	Expected completion time for job j
$C_e(j, r)$	Expected execution cost for job j on r
$C_t(f, d_f, r)$	Expected cost of transferring dataset $f \in F^j$ from $d_f \in D_f$ to $r \in S^j$
$C(j)$	Expected total cost of executing j

5.2.3 A Generic Scheduling Algorithm

The scheduling paradigm followed in this thesis is that of *offline* or *batch mode* scheduling of a set of independent tasks [142]. (Note: Since each task is translated into a job, tasks and jobs are used interchangeably throughout the rest of this thesis). The general problem of creating a schedule for a set of jobs to run on distributed resources is called *list scheduling* and is considered to be *NP*-complete [38]. Many approximate heuristics have been devised for this problem and a short survey of these have been presented by Braun, et al. [38]. Figure 5.6 shows a general scheduling algorithm for batch mode scheduling of a set of jobs based on the skeleton presented by Casanova, et al. [51].

As described in Chapter 4, the resource broker is able to identify resources that meet

```
while there exists unsubmitted jobs do
    Update the resource performance data based on job scheduled in previous
    intervals
    Update network data between resources based on current conditions
    foreach unsubmitted job do
        Match the job to a resource set to satisfy the objective function at the job
        level
        Order the jobs depending on the overall objective
    end
    repeat
        Assign mapped jobs to each compute resource heuristically
    until all jobs are submitted or no more jobs can be submitted
    Wait until the next scheduling event
end
```

Figure 5.6: A Generic Scheduling Algorithm.

minimum requirements of the application such as architecture (instruction set), operating system, storage threshold and data access permissions and present these as suitable candidates for job execution to the scheduler. The scheduling is carried out at time intervals called *scheduling events* [141]. These events can be determined to either run at regular intervals (*poll-based*) or in response to certain conditions (*event-based*).

There are two parts in a scheduling strategy: mapping and dispatching. The jobs have to be *matched* to a set of resources and ordered depending on the objective function (*mapping*) and then sent to remote resources for execution (*dispatching*). Each of the parts can be implemented independently of each other and therefore, many strategies are possible. The rest of this thesis focuses on the mapping problem. In particular, this and the next chapter introduce heuristics for matching jobs to distributed resources where the selection of computational and data resources are interdependent on each other.

5.3 Cost-based Scheduling for Data-Intensive Applications

The previous section outlined the economic costs involved in executing a data-intensive job on Grid resources that have prices associated with their usage. Additionally, a user may expect some specific QoS conditions to be fulfilled for the overall application execution. Previous work in computational Grid scheduling by Buyya [42] introduced two QoS constraints that have to be fulfilled simultaneously: one, to finish the application

execution by a user-specified *deadline* and the other, to keep the cost of execution within the maximum *budget* that the user has for the execution. Based on the model presented so far, the deadline for the application (denoted by $T_{Deadline}$) can be expressed in terms of job execution time as $\max_{j \in J} T_{ct}(j) \leq T_{Deadline}$. The budget constraint can be expressed as $\sum_J C(j) \leq Budget$.

The pricing of goods in a real world economy is determined by the laws of supply and demand [143]. In a Grid based on computational economy, it is expected that the same mechanism will apply to pricing of Grid resources [44]. Resources that are in demand due to their higher capabilities are expected to be more expensive than others. It follows that using cheaper (and less capable) resources will minimise the cost while using more expensive (and more capable) resources will result in faster application execution. Evaluations have shown the applicability of these assumptions to computational Grid resources [45] where the resources can be compared on the basis of CPU share that they provide to the jobs.

For a data-intensive application, however, the selection of resources must take into account requirements for transferring and processing large datasets as well. This means that mapping a job to a computational resource must not only consider the cost and time for executing a job on that resource but also the cost and time of transferring data to that resource from the selected compute resources as well. The following sections present an algorithm that takes into account these factors for scheduling data-intensive jobs in a Data Grid environment with resource costs.

5.3.1 Objective Functions

Depending on the user-provided deadline, budget and scheduling preference, two objective functions can be defined, viz:

- **Cost minimisation:** The objective is to produce a schedule that causes least expense while keeping the execution time within the deadline provided.
- **Time minimisation:** Here, the jobs are executed in the fastest time possible with the budget for the execution acting as the constraint.

Here, the same heuristic is applied to achieve either of the objective functions but while considering the appropriate variables.

5.3.2 Cost and Time Minimisation Algorithms

Figure 5.7 lists the algorithm for cost minimisation scheduling of data-intensive applications. The scheduling loop is invoked at regular polling intervals until all the jobs are completed or until either the deadline or the budget is exceeded. At every polling interval, the performance data of the compute resources is updated by taking into account status of the jobs allocated to and those completed by the resource in the previous intervals and information from external performance monitors, if any (line 3). This is used to calculate the limit of allocation (number of available job slots) of the resource for the current polling interval. Also queried are market information services for latest information on instantaneous resource prices. For each data resource, the cost and available bandwidth between itself and the computational resources is refreshed by querying the network information services (line 6). Then, for each data host, a sorted list of available compute resources is created based on the cost of transmitting a unit of data between the data host and the compute resource (line 7). This is followed by the mapping loop (lines 8-24) wherein each job is mapped to a set of resources. After the jobs are mapped, the dispatch function is invoked (line 25) and the jobs are submitted to the selected resources while taking into consideration deadline and budget constraints specified by the user.

Mapping: The aim of the mapping loop is to match each job to a resource set and then assign the jobs to the selected resources. For each job, the loop starts off with an empty resource set S^j which itself is a set of the empty singleton R^j and the empty set of datahosts D^j. For each dataset associated with the job, another set U is created consisting of ordered pairs, each of which has one data host that contains the dataset and a compute resource such that the cost of transfer for that dataset is minimum (line 13). The compute resource is the first element of the sorted set of compute resources (R_d) that has been created for each data host in line 7. The ordered pair (d_f, r) that provides the smallest cost is then selected out of all the pairs in U. The compute resource from the ordered pair is then assigned to R^j while the corresponding data host is added to D^j. R^j_{temp} is another

1. **while** $J \neq \phi \ OR \ T_{current} < T_{Deadline} \ OR \ Budget\_spent < Budget$ **do**
2. **foreach** $r \in R$ **do**
3. Calculate performance data on the basis of resource performance in the previous polling interval
4. **end**
5. **foreach** $d \in D$ **do**
6. Update the network information
7. Let $R_d \leftarrow \{r_m | r_m \prec r_{m+1}$ if $Cost(Link(d, r_m)) < Cost(Link(d, r_{m+1}))$ $\forall \, r_m \in R, 1 \leq m \leq M\}$
8. **end**
   //Mapping Begins
9. **foreach** $j \in J$ **do**
10. Let $S^j \leftarrow \{R^j, D^j\}, \ R^j \leftarrow \phi, \ D^j \leftarrow \phi$
11. Let $R^j_{temp} \leftarrow \phi$ //A temporary variable
12. **foreach** $f \in F^j$ **do**
13. Let $U \leftarrow \{(d_f, r)\}_{d_f \in D_f}$ where r is the first element of ordered set R_{d_f}
14. Find (d_f, r) such that $C_t(f, d_f, r) + C_e(j, r)$ is minimum over U
15. **if** $S^j = \{\phi, \phi\}$ **then**
16. $R^j \leftarrow \{r\}, D^j \leftarrow \{d_f\}, R^j_{temp} \leftarrow \{r\}$
17. **end**
18. **else**
19. $R^j \leftarrow \{r\}, D^j \cup \{d_f\}$
20. **end**
21. $S^j \leftarrow \min\{\{R^j, D^j\}, \{R^j_{temp}, D^j\}\}$
22. $R^j_{temp} \leftarrow R^j$
23. **end**
24. **end**
    //Mapping Ends
25. `Dispatch(`$J, T_{Deadline}, Budget$`)`
26. Wait until next polling interval
27. Update $Budget\_spent$ by taking into account jobs completed in the last interval
28. **end**

Figure 5.7: An Algorithm for Minimising Cost of Scheduling of Data Intensive Applications.

singleton which has the compute resource selected in the previous iteration of the loop. A comparison is then made between the resource set with the current compute resource ($\{R^j, D^j\}$) and the one with the previous compute resource ($\{R^j_{temp}, D^j\}$) and the one which provides the least cost is then selected as the resource set for the next iteration of the dataset loop.

The matching heuristic is therefore, essentially a greedy strategy with a choice step to improve the resource set being selected in every iteration. For a job that requires a

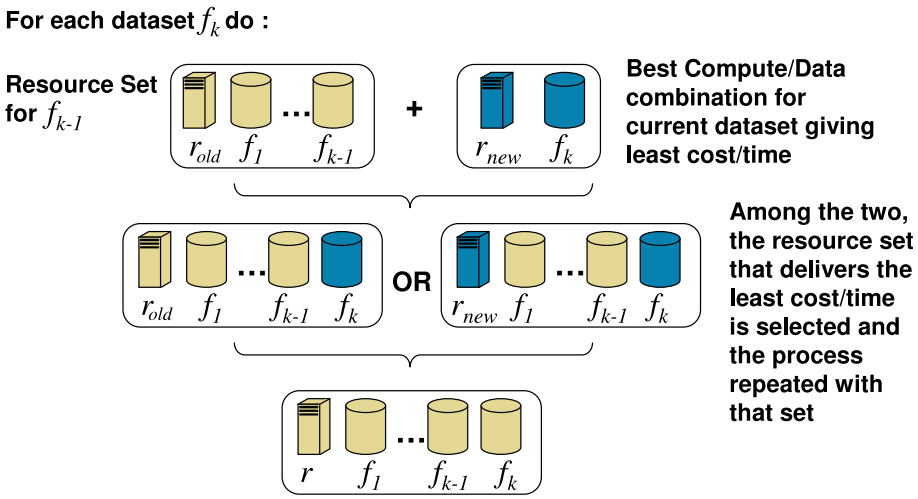


Figure 5.8: The Greedy Matching Heuristic.

single dataset, this is a straightforward greedy choice. For a job with multiple datasets, the process involved is shown pictorially in Figure 5.8. For each dataset, a pair of compute resource and data host is selected such that it ensures the best metrics for the job if only that dataset were involved. This is then merged with the resource set that has been built up previously to derive two resource sets, one with the compute resource selected in the previous iteration and the other with the current compute resource. The one that provides the least cost is then selected as the input for the next iteration. The idea behind this heuristic is, therefore, to ensure that adding every pair of a compute resource and a data host produces a better resource set at the end of each iteration than that was produced by the previous iteration.

Dispatching: The job dispatch function is listed in Figure 5.9. The allocated jobs are sorted in the ascending order of their expected costs for their respective resource sets. Then, starting with the job with the least cost, each job is submitted to its compute resource selected in the mapping step if the allocation for that resource has not been exhausted by previous assignments. For cost minimisation, it is determined whether the deadline is violated by checking whether the current time ($T_{Current}$) plus the expected completion time exceeds $T_{Deadline}$ (line 7). If so, the job goes back into the unsubmitted list in the expectation that the next iteration of the mapping loop will produce a better resource set for that job. If $Budget$ is exceeded by the current job, then the dispatching is halted and

```
1.  Dispatch(J, T_Deadline, Budget, Min)
2.  Sort J in the ascending order of C(j), ∀j ∈ J
3.  Expected_Budget ← Budget_spent
4.  foreach j ∈ J do
5.      Take the next job j ∈ J in sorted order
6.      if r ∈ R^j can be allocated more jobs then
7.          if (T_Current + T_ct(j) < T_Deadline then
8.              if (Expected_Budget + C(j)) ≤ Budget then
9.              submit j to r
10.             else stop dispatching and exit to main loop
11.         end
12.         Expected_Budget = Expected_Budget + e_j
13.         Remove j from J
14.     end
15. end
```

Figure 5.9: Deadline and Budget Constrained Job Dispatch.

the functions returns to the main loop as the rest of the jobs in the list will have a higher cost than the current job (line 8). If these two constraints are not violated, then the job is submitted to the compute resource and removed from the list of unsubmitted jobs.

Time minimisation can be achieved with the same algorithm but with time-specific variables as shown in Figure 5.10. The mapping function sorts the compute resources for each data host based on the time for transferring unit data. That is, the term $Cost(Link(r,d))$ in line 7 in Figure 5.7 is replaced by $1/BW(Link(r,d))$. Line 14 will have $T_t(f,d_f,r) + T_e(j,r)$ instead of the cost metric and line 21 selects a resource set based on total completion time instead of cost. The dispatch function also changes as the deadline and budget checks are swapped between lines 7 and 8. For time minimisation, if the budget spent (including the budget for all the jobs previously submitted in current iteration) plus the budget for the current job exceeds $Budget$ then the function proceeds to the next job. However, if the deadline is violated by the current job, then the dispatch function returns to the main loop.

5.4 Experiments and Results

The cost-aware deadline and budget-constrained scheduling algorithm presented in the previous section, was implemented in the Gridbus broker and was evaluated on a testbed

1. **while** $J \neq \phi \ OR \ T_{current} < T_{Deadline} \ OR \ Budget\_spent < Budget$ **do**
2. **foreach** $r \in R$ **do**
3. Calculate performance data on the basis of resource performance in the previous polling interval
4. **end**
5. **foreach** $d \in D$ **do**
6. Update the network information
7. Let $R_d \leftarrow \{r_m | r_m \prec r_{m+1}$ if $1/BW(Link(d, r_m)) < 1/BW(Link(d, r_{m+1}))$ $\forall \, r_m \in R, 1 \leq m \leq M\}$
8. **end**
   //Mapping Begins
9. **foreach** $j \in J$ **do**
10. Let $S^j \leftarrow \{R^j, D^j\}, \ R^j \leftarrow \phi, \ D^j \leftarrow \phi$
11. Let $R^j_{temp} \leftarrow \phi$ //A temporary variable
12. **foreach** $f \in F^j$ **do**
13. Let $U \leftarrow \{(d_f, r)\}_{d_f \in D_f}$ where r is the first element of ordered set R_{d_f}
14. Find (d_f, r) such that $T_t(f, d_f, r) + T_e(j, r)$ is minimum over U
15. **if** $S^j = \{\phi, \phi\}$ **then**
16. $R^j \leftarrow \{r\}, D^j \leftarrow \{d_f\}, R^j_{temp} \leftarrow \{r\}$
17. **end**
18. **else**
19. $R^j \leftarrow \{r\}, D^j \cup \{d_f\}$
20. **end**
21. $S^j \leftarrow \min\{\{R^j, D^j\}, \{R^j_{temp}, D^j\}\}$
22. $R^j_{temp} \leftarrow R^j$
23. **end**
24. **end**
    //Mapping Ends
25. `Dispatch(`$J, T_{Deadline}, Budget$`)`
26. Wait until next polling interval
27. Update $Budget\_spent$ by taking into account jobs completed in the last interval
28. **end**

Figure 5.10: An Algorithm for Minimising Execution Time.

slightly extended from the Belle testbed used in the case study in the previous chapter. Details of the resources including configuration, role and price are provided in Table 5.2. A new resource from the Victorian Partnership for Advanced Computing (VPAC), Melbourne was added to the testbed for this evaluation. Also, the machines in School of Physics, University of Melbourne, and Computer Science, University of Adelaide were only used as data sources (data hosts) and no jobs were executed on them. The machines functioning as compute resources were assigned rates for executing jobs in Grid Dollars (G$) [45] per CPU second used. Grid Dollars is a synthetic unit of currency that mod-

els the role of actual currencies such as Australian dollars in real world economies.The resources functioning as pure data hosts were not assigned any prices.

Table 5.2: Resources within Belle testbed used for evaluation.

Organisation	**Resource details**	**Role**	**Rate (G$)**	**Total Jobs**	
				Time	**Cost**
Computer Science, University of Melbourne	*belle.cs.mu.oz.au* 4 Intel 2.6 GHz CPU, 2 GB RAM, 70 GB HD, Linux	Broker Host, Data Host, Compute resource, NWS Server	6	94	2
School of Physics, University of Melbourne	*fleagle.ph.unimelb.edu.au* 1 Intel 2.6 Ghz CPU, 512 MB RAM, 70 GB HD, Linux	Replica Catalog host, Data host, NWS sensor	N.A.*	–	–
Computer Science, University of Adelaide	*belle.cs.adelaide.edu.au* 4 Intel 2.6 GHz CPU, 2 GB RAM, 70 GB HD, Linux	Data host, NWS sensor	N.A. *	–	–
Australian National University, Canberra	*belle.anu.edu.au* 4 Intel 2.6 GHz CPU, 2 GB RAM, 70 GB HD, Linux	Data Host, Compute resource, NWS sensor	6	2	4
Dept. of Physics, University of Sydney	*belle.physics.usyd.edu.au* 4 Intel 2.6 GHz CPU(1 avail), 2 GB RAM, 70 GB HD, Linux	Data Host, Compute resource, NWS sensor	2	2	119
VPAC, Melbourne	*brecca-2.vpac.org* 180 node cluster (only head node utilised)	Compute resource, NWS sensor	4	27	0

* N.A. - Not Applicable. Resource not used as a compute resource but only as a data host

Information about the network conditions were obtained through the same Network Weather Service (NWS) set up used in the previous chapter. An NWS sensor was also started on the VPAC resource which was added to the testbed clique. In this evaluation, however, data transmission costs were also assigned to the network links between the resources. These costs were in the form of G$ per MB (MegaByte) of data transmitted. The

available bandwidth, reported as an average of the measurements throughout the evaluation, is given in Table 5.3. Alongside the bandwidth data, the cost assigned to the network is also given inside the parentheses. The network between a compute resource and a data host located at the same site is assigned a high available bandwidth (1000 Mbps, not shown in the table) and zero cost.

In this evaluation, the costs have been artificially assigned to the resources. However, these can be linked to real world costs that will occur once the economic paradigm is adopted by all the participants within the Grid. Network users already pay Internet Service Providers (ISPs) for usage based on volume of data or as a regular subscription fee. Computational services are being offered by corporations such as Sun Microsystems as utilities that are charged on the basis of time of usage [195].

Table 5.3: Avg. Available Bandwidth between Data Hosts and Compute Resources as reported by NWS(in Mbps) and Network Costs between Data Hosts and Compute Resources (G$/MB) in parentheses.

	Compute Resources			
Data Hosts	UniMelb CS	ANU	UniSyd	VPAC
ANU	6.99 (34.0)	1000 (0)	10.24 (31.0)	6.33 (38.0)
Adelaide	3.45 (36.0)	1.68 (34.0)	2.29 (31.0)	6.05 (33.0)
UniMelb Physics	41.05 (40.0)	6.53 (32.0)	2.65 (39.0)	20.57 (35.0)
UniMelb CS	1000 (0)	6.96 (30.0)	4.77 (36.0)	36.03 (33.0)
UniSyd	4.78 (33.0)	12.57 (35.0)	1000 (0)	2.98 (37.0)

A synthetic data-intensive program was created for the purpose of evaluating different data-intensive applications. This program would request K datasets located on distributed data sources and process them to produce a small output file (of the order of KiloBytes (KB)). The bag-of-task data-intensive application being evaluated here is a parameter-sweep application consisting of 125 jobs, each job being an instance of the program requiring 3 files (that is, $K = 3$ for all the jobs in this evaluation). The execution times for the jobs (excluding data transfer times) were randomly distributed within 60-120 seconds. Each of the jobs would request 3 files at random from the set of 100 files (distributed equally among the data hosts listed in Table 5.2) that was used in the Belle case study presented in the previous chapter. Each of these files are 30 MB is size.

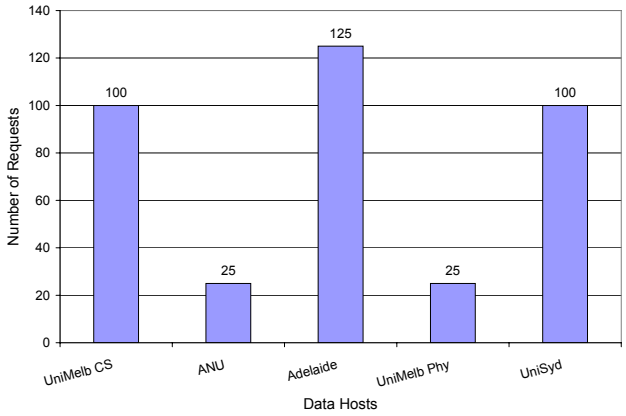


Figure 5.11: Distribution of file access.

These files are specified as Logical File Names (LFNs) and resolved to the actual physical locations by the broker at runtime by querying the Globus replica catalog located at the UniMelb Physics resource (*fleagle.ph.unimelb.edu.au*). Figure 5.11 gives the distribution of the number of requests for data made by the total set of jobs against each data host.

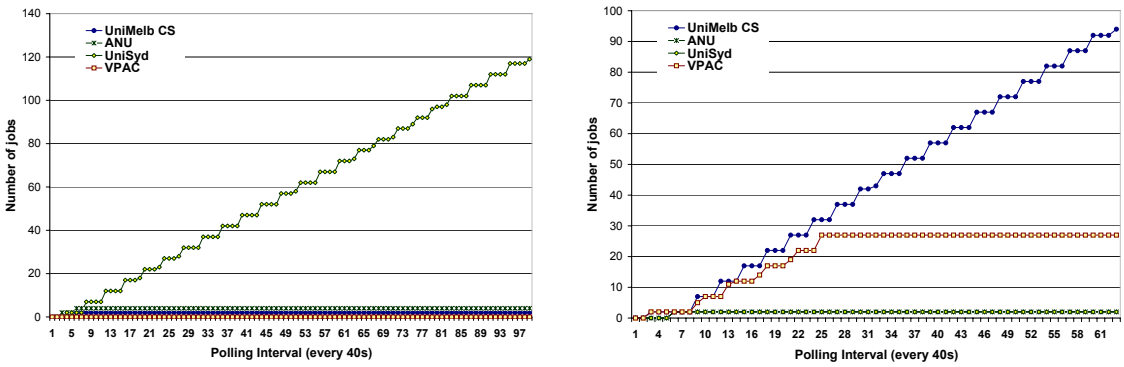
The distributed application was run under both cost and time minimisation. The minimisation algorithms were evaluated and compared against two measures of performance: the first is the relative usage of the computational resources under cost and time minimisation which indicates the impact of the choice of minimisation criteria on resource selection, and the second is the distribution of jobs with respect to the computational and data transfer costs and times incurred within each minimisation, which indicates the effectiveness of the algorithm in producing the cheapest or the fastest schedule. The experiments were carried out on 29th November 2004 between 6:00 p.m. and 10:00 p.m. AEDT. The deadline and budget values for both cost and time minimization were 2 hours and 500,000 G$ respectively. Table 5.4 shows the summary of the results that were obtained. The total time is the wall clock time taken from the start of the scheduling procedure up to the completion of the last job. All the jobs completed successfully in both the experiments. The average costs per job incurred during cost and time minimisation are 562.6 G$ and 959 G$ with standard deviations of 113 and 115 respectively. Mean wall clock time taken per job (including computation and data transfer time) was 167 secs. for cost minimisation and 135 secs for time minimisation with standard deviations 16.7 and 19 respectively.

As expected, cost minimisation scheduling produces minimum computation and data transfer expenses whereas time minimisation completes the experiments in the least time.

Table 5.4: Summary of Evaluation Results.

Minimization	Total Time (min.)	Compute Cost (G$)	Data Cost (G$)	Total Cost (G$)
Cost	80	31198.27	39126.65	70324.93
Time	54	76054.90	43821.64	119876.55

The graphs in Figures 5.12(a) and 5.12(b) show the number of jobs completed versus time for the two scheduling strategies presented. Since the computation time was dominant, within cost minimisation, the jobs were executed on the least economically expensive compute resource. This can be seen in Figure 5.12(a) where the compute resource with the least cost per second, the resource at University of Sydney, was chosen to execute 95% of the jobs. Since a very relaxed deadline was given, no other compute resource was engaged by the scheduler as it was confident that the least expensive resource alone would be able to complete the jobs within the given time.

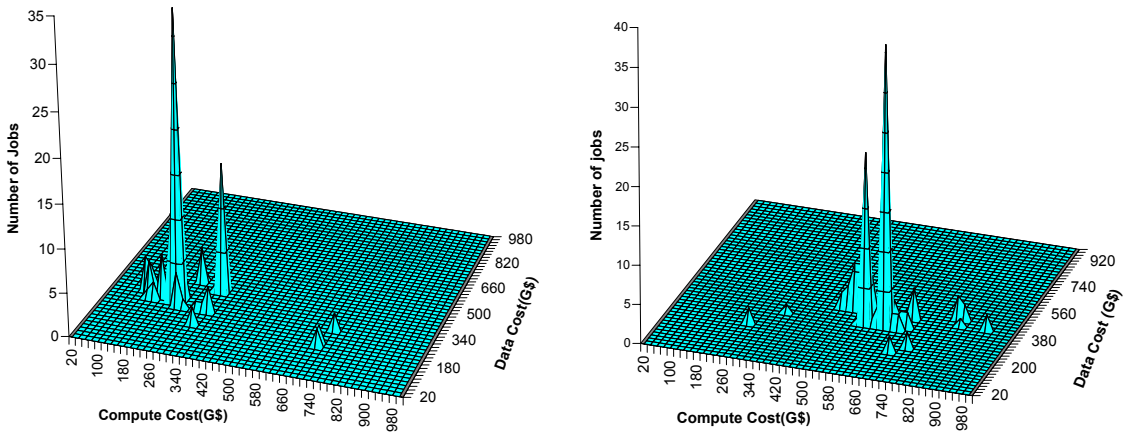


(a) cost minimisation scheduling (b) time minimisation scheduling

Figure 5.12: Cumulative number of jobs completed vs time for cost and time minimisation scheduling.

Within time minimisation, the jobs were dispatched to the compute resources which promised the least execution time even if they were expensive as long as the expected cost for the job was less than the budget per job. Initially, the scheduler utilised two of the faster resources, the University of Melbourne Computer Science (UniMelb CS) resource and the VPAC resource (Figure 5.12(a)). However, as seen from Figure 5.11, 26.67% of the requests for datasets were directed to the UniMelb CS resource. A further

6.67% were directed to the resource in UniMelb Physics. Hence, any jobs requiring one of the datasets located on either of the above resources were scheduled at the UniMelb CS resource because of the resultant low transfer time. Also, the UniMelb CS resource had more processors. Hence, a majority of the jobs were dispatched to it within time minimization.

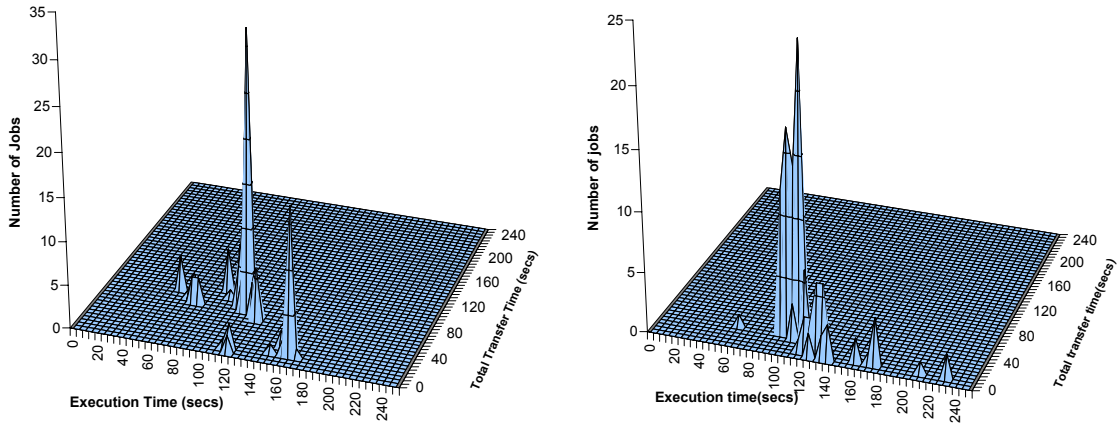


(a) cost minimisation scheduling (b) time minimisation scheduling

Figure 5.13: Distribution of jobs against compute and data costs.

Figures 5.13(a) and 5.13(b) show the distribution of the jobs with respect to the compute and data costs respectively. For cost minimisation, 95% of the jobs have compute costs less than or equal to 400 G$ and data costs between 250 G$ to 350 G$. In contrast, within time minimization, 91% of the jobs are in the region of compute costs between 500 G$ to 700 G$ and data costs between 300 G$ to 400 G$. Hence, in time minimization, more jobs are in the region of high compute costs and medium data costs. Thus, it can be inferred that the broker utilized the more expensive compute and network resources to transfer data and execute the jobs within time minimization.

Figures 5.14(a) and 5.14(b) show the distribution of the jobs with respect to the total execution time and the total data transfer time for cost minimisation and time minimisation respectively. The execution time excludes the time taken for data transfer. It can be seen that within time minimisation (Figure 5.14(b)), the maximum data transfer time was 35s as compared to 75s for cost minimisation. Also, there are more jobs within time minimisation that have had transfer time less than 10s which implies that the jobs were scheduled close to the source of the data.



(a) cost minimisation scheduling (b) time minimisation scheduling

Figure 5.14: Distribution of jobs against execution time and data transfer time.

The results of the empirical evaluation show that the algorithms presented in this chapter are able to minimize the objective function for upto 90% of the input set of jobs. Also, cost minimization gave more preference to reducing cost of computation than the cost of data transfer. This can be seen by the smaller difference between the data costs for both cost and time minimization as compared to the compute costs in Table 5.4. Time minimization, however, attempted to reduce both the execution and the data transfer times as can be seen by comparing Figures 5.14(a) and 5.14(b). This, however, lead to an increase of only 11% in the data costs over cost minimization (Table 5.4).

5.5 Summary

Typical Data Grid environments consist of heterogeneous computational, storage and networking resources that are shared among the users and may have expenses associated with their usage. A scheduler operating in such environments must not only take into account the variations of availabilities, capabilities and costs among the resources but also should consider application requirements that may include multiple large-sized datasets, each replicated on multiple resources. This chapter models this problem formally and applies it to cost-based scheduling of distributed data-intensive applications.

An interesting result gathered from the empirical evaluation is that time minimization preferred scheduling jobs close to the source of data. That is, it exploited the locality

of datasets more effectively. This result is used as the basis for the investigation in the next chapter, where the problem of resource selection is modelled using graph theoretic concepts. A heuristic based on an instance of the Set Covering Problem is then proposed to exploit access locality of jobs. The proposed heuristic is evaluated against others including the Greedy matching heuristic presented in this chapter through simulation. This allows investigation of impact of different variables such as degree of replication, size of datasets and larger number of resources on the performance of the heuristics.

Chapter 6

A Set Coverage-based Scheduling Algorithm

The previous chapter defined the *mapping* problem of matching a set of resources to a job and ordering the set of jobs for allocation. The mapping must minimise the objective function not only for a single job but also for the overall set of jobs that constitutes the entire application. Previous work in scheduling of distributed data-intensive Grid applications (see Chapter 3, Section 3.2) and evaluations in Chapters 4 and 5 have provided various solutions for successful mapping of such jobs to Grid resources. Some of them are listed below:

1. Scheduling job execution "close" to the point of location of data or exploiting the spatial locality of data access.
2. Reusing existing data replicas or exploiting the temporal locality.
3. Giving weightage to the computational requirements of the jobs while creating data replicas if necessary.
4. Meeting users' requirements such as shortest makespan, expense minimisation within deadline or time minimisation within budget.

These solutions may seem to conflict with each other, especially in the case of jobs that require multiple datasets that are each available from multiple sources. However, it is also possible for them to complement one another as well. For example, a job may be

scheduled to a compute resource that is closest to the data host that contains the maximum number of the set of datasets required by the job. Other datasets may be staged to that data host and may be used as replica sources for later jobs. This also reduces the number of remote data transfers thereby reducing both the time and cost of total data movement.

The selection of the computational resource, however, should not only be based on the proximity of the data but also on its availability and performance as well. In case of data-intensive jobs that are computationally heavy as well, this choice may have a higher impact on the realisation of the objective than the selection of data resources. Therefore, the selection of resources depends on the interrelationship between the computational and data components of the performance metrics.

This chapter focuses on the resource selection or the *matching* problem for jobs which require multiple datasets that are each replicated on multiple data hosts. First, the problem is modelled as an instance of the well-known Set Covering Problem. Based on this, a tree search heuristic for the matching problem is detailed. This heuristic, along with other known heuristics, is then evaluated through extensive simulations.

6.1 A Graph-based Approach to the Matching Problem

This chapter follows the same notations that were provided in Table 5.1 in the previous chapter. For a job $j \in J$, consider a graph $G^j = (V, E)$ where $V = (\bigcup_{f \in F^j} \{D_f\}) \cup F^j$ and E is the set of all directed edges $\{d, f\}$ such that $d \in D_f$. Figure 6.1(a) shows an example of a job j that requires 3 datasets f_1, f_2 and f_3 that are replicated on data host sets $\{d_1, d_2\}$, $\{d_2, d_3\}$ and $\{d_1, d_4\}$ respectively. The graph of data sets and data resources for job j is shown in Figure 6.1(b).

As mentioned before, it is required to find the minimum number of data hosts that can serve the required datasets to minimise the amount of data transfer involved. In terms of the graph model presented, this can be considered as the minimal set H of data hosts such that there exists an edge from a member of H to f for every $f \in F^j$ in G^j. Figure 6.1(c) shows a possible minimal set for the graph of datasets and data hosts shown in Figure 6.1(b). However, it is possible that more than one minimal set of data hosts exists for a graph. Also, one minimal set of data hosts can be combined with each compute

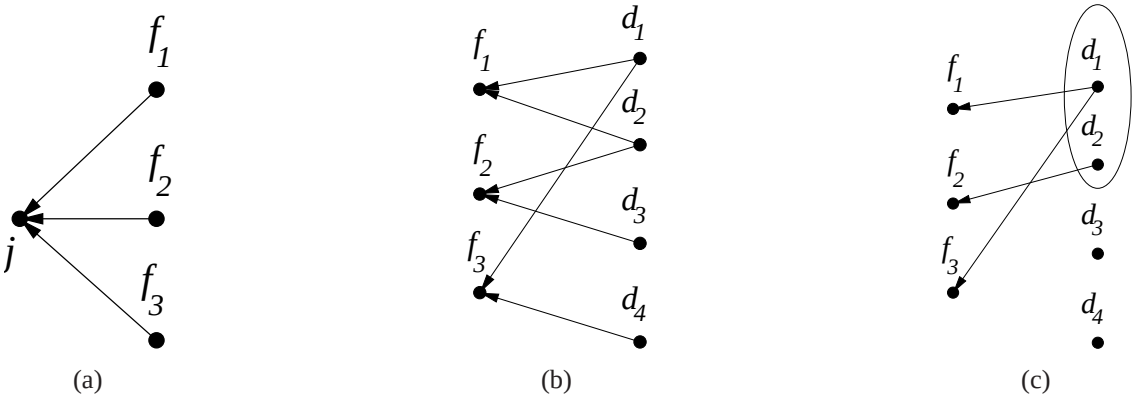


Figure 6.1: Graph-based approach to the matching problem. (a)Job j dependent on 3 datasets. (b) Directed graph of data resources and data sets for job j. (c) A minimal set for the data graph.

resource in R to produce M resource sets (where M is the number of resources) with different values of total completion time or execution cost. The goal here is, therefore, to find a combination of a minimal set of data hosts and a compute resource such that the total completion time or execution cost for j is minimised. This problem is defined and referred to hereafter as the **M**inimum **R**esource **S**et (MRS) problem.

The following section presents a heuristic for the MRS problem based on an algorithm to solve the Set Covering Problem [24]. The heuristic generates set covers of data hosts and combines them with the list of compute resources to produce candidate resource sets that are then compared to produce the resource set giving the smallest value of the objective function for a job. The term *minimal sets* are used hereafter to refer to minimal sets of data hosts.

6.1.1 Modelling the Minimum Resource Set as a Set Cover

For a graph G^j such as that shown in Figure 6.1(b), a reduced adjacency matrix $A = [a_{ik}], 1 \leq i \leq P, 1 \leq k \leq K$ can be constructed wherein $a_{ik} = 1$ if data host $d_i \in D_{f_k}$ for a dataset f_k. Such an adjacency matrix is shown in Figure 6.2. The rows that contain a 1 in a particular column are said to "cover" the column. The problem of finding the minimal set of data hosts for G^j is now equivalent to finding the sets of the least number of rows such that every column is covered, that is, every column contains an entry of 1 in at least one of the rows. In other words, if each data host can be considered as a set

of datasets, then finding the minimal set of data hosts is equivalent to finding the least number of such sets of datasets such that all datasets are covered. This problem has been studied extensively as the *Set Covering Problem (SCP)* [24].

$$\begin{array}{c} \\ d_1 \\ d_2 \\ d_3 \\ d_4 \end{array} \begin{array}{c} \begin{array}{ccc} f_1 & f_2 & f_3 \end{array} \\ \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \end{array}$$

Figure 6.2: Adjacency Matrix for the job example.

The SCP is an NP-*complete* problem and the most common approximation algorithm applied to the SCP is the greedy strategy [62]. It is possible to derive a set cover for the datasets by following the greedy strategy as outlined below:

Step 1. Repeat until all the datasets have been covered

Step 2. $\hookrightarrow$ Pick the data host that has the maximum number of uncovered datasets and add it to the current candidate set.

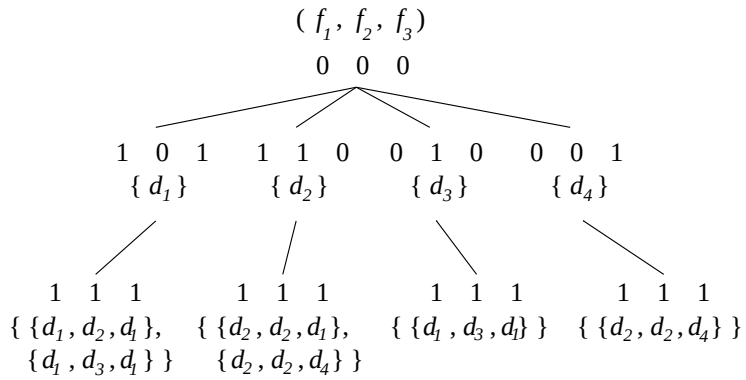


Figure 6.3: Solution Tree.

It is also possible to arrive at a depth first search procedure that generates all the covers by repeating the same greedy strategy with every data host (not just the ones with the maximum number of datasets). The possible minimal sets for the graph in Figure 6.1(b) can be enumerated in this manner and presented as a tree structure shown in Figure 6.3. It can be seen that the greedy set covering strategy will produce only one of the possible minimal sets. For example, starting off with d_1, the greedy strategy is likely to end up with $\{d_1, d_2, d_1\}$ as the solution. This, however, excludes the other candidate sets from

consideration. On the other hand, doing a depth-first search on the entire solution tree is going to be computationally-intensive and can lead to repeat generation of some of the candidate sets. This can be seen in the branches for d_3 and d_4 in Figure 6.3.

The next section details a heuristic that identifies the region of the tree where the solution is most likely to be found and then augments the greedy search through a depth-first search in that region. But, before applying that algorithm, it is possible to reduce the size of the problem by taking advantage of the nature of the SCP. These reductions are:

- If a dataset required for a job is present on only one data host, then that data host is part of any solution. Therefore, the problem can be reduced by assigning the dataset to that data host and removing the dataset from later consideration.
- For $f_1, f_2 \in F^j$, if $D_{f_1} \subseteq D_{f_2}$, then f_2 can be removed from consideration as any solution that covers f_1 must also cover f_2.

6.1.2 The SCP Tree Search Heuristic

This heuristic is listed in Figure 6.4 and is based on the approximate tree search algorithm provided by Christofides [58] for the SCP. There are three distinct phases in this heuristic: initialisation, execution and termination. These are described in the following paragraphs.

Initialisation (Lines 1-3)

The initialisation starts off with the creation of the adjacency matrix A for a job. The rows of this matrix (that is, the data hosts) are then sorted in the descending order of number of 1's per column (or, the number of datasets contained). This sorted matrix is used to create an augmented matrix that is henceforth referred to as the *tableau* and is shown in Figure 6.5. The tableau T consists of K blocks of rows, where K is the size of F^j and the $k^{th}(1 \leq k \leq K)$ block consists of rows corresponding to data hosts that contain $f_k, f_k \in F^j$. The tableau is constructed in such a manner that the rows within each block are in the same sorted order as the rows in the sorted adjacency matrix. At any stage of execution, the set of data hosts B keeps track of the current solution set of datahosts, the set E contains the datasets already covered by the solution set and the variable z keeps track of the minimum value of the objective function offered by the current solution set.

Begin Main

1. For a job j, create the adjacency matrix A with data hosts forming the rows and datasets forming the columns.
2. Sort the rows of A in the descending order of the number of 1's in a row.
3. Create the tableau T from sorted A and begin with initial solution set $B_{final} = \phi$, $B = \phi$, $E = \phi$ and $z = \infty$
4. `Search(`B_{final}, *B, T, E, z*`)`
5. $S^j \leftarrow \{\{r\}, B_{final}\}$ where $r \in R$ such that `MinVal` (B_{final}) is minimum

End Main

`Search(`B_{final}, *B, T, E, z*`)`

6. Find the minimum k, such that $f_k \notin E$. Let T_k be the block of rows in T corresponding to f_k. Set a pointer q to the top of T_k.
7. **while** *q does not reach the end of T_k* **do**
8. $\quad F_T \leftarrow \{f_i | t_{qi} = 1, 1 \leq i \leq K\}$
9. $\quad B \leftarrow B \cup \{d_q^k\}, E \leftarrow E \cup F_T$
10. $\quad$ **if** $E = F^j$ **then**
11. $\quad\quad$ **if** $z >$ `MinVal` *(B)* **then**
12. $\quad\quad\quad B_{final} \leftarrow B, z \leftarrow$ `MinVal` (B)
13. $\quad$ **else** `Search(`B_{final}, *B, T, E, z*`)`
14. $\quad B \leftarrow B - \{d_q^k\}, E \leftarrow E - F_T$
15. $\quad$ Increment q
16. **end**

`MinVal(`*B*`)`

17. Find $r \in R$ such that the value of the objective function is minimum for the resource set $S^j = \{\{r\}, B\}$ and return value

Figure 6.4: Listing of the SCP Tree Search Heuristic for the MRS problem.

The final solution set is the stored in B_{final}. The procedure begins with the partial solution set $B = \phi$, $E = \phi$, $z = \infty$.

Execution (Lines 6-16)

During execution, the blocks are searched sequentially starting from the k^{th} block in T where k is the smallest index, $1 \leq k \leq K$ such that $f_k \notin E$. Within the k^{th} block, let d_q^k mark the data host under consideration where q is a row pointer within block k. The data host d_q^k is added to B and all the datasets for which the corresponding row contains 1 are added to E as they are already covered by d_q^k. These datasets are removed from consideration and the process then moves to the next uncovered block until $E = F^j$, that is, all the datasets have been covered. At this point, B represents the corresponding minimal set of data hosts that covers all the datasets. The function `MinVal`(B) computes

$$
\begin{array}{c}
 \\
d_1 \\
d_2 \\
- \\
d_2 \\
d_3 \\
- \\
d_1 \\
d_4
\end{array}
\begin{array}{c}
\begin{array}{ccc} f_1 & f_2 & f_3 \end{array} \\
\left(\begin{array}{ccc}
1 & 0 & 1 \\
1 & 1 & 0 \\
- & - & - \\
1 & 1 & 0 \\
0 & 1 & 0 \\
- & - & - \\
1 & 0 & 1 \\
0 & 0 & 1
\end{array}\right)
\end{array}
$$

Figure 6.5: Tableau.

the expected value of the objective function for each compute resource combined with B and returns with the minimum of the values so found. If this is lower than the existing value in z, then the solution set is replaced with the current minimal set and z is assigned the returned value.

Whenever the heuristic enters a block that is not yet covered, it branches out within the block by a recursive call that passes along the incomplete solution set (line 13). The final solution set is returned in the variable B_{final} through normal pass-by-reference methods. At the end of each loop, the heuristic backtracks to try the next data host in the block and repeat the branching with that host (line 14).

Termination (Line 5)

Through the recursive procedure outlined in the listing, the heuristic then backtracks and discovers other minimal sets. The solution set that guarantees minimum makespan is then chosen as the final minimal set. The compute resource that provides the minimum value of objective function is then combined with the minimal set to obtain the minimal resource set for the job.

To reduce the scope of the tree traversal, the heuristic terminates when the first block is exhausted. The data hosts with the maximum number of datasets appear at the top of the tableau due to the initialisation process. Therefore, most of the minimal sets will be covered by the search function by starting at the rows in the first block.

6.2 Other Approaches to the Matching Problem

Compute-First - In this mapping strategy, shown in Figure 6.6, a compute resource that ensures minimum value of the computational component of the objective function is selected first for the job. For example, in case of reducing the completion time (T_{ct}) for a job, the compute resource that gives the least execution time is selected first. This step is followed by choosing data hosts such that the data component of the objective function is reduced. For the example of reducing the completion time, this would be selecting the data hosts that have the highest bandwidths (and therefore, the lowest transfer times) to the selected compute resource. The running time of this heuristic is $O(MKP)$.

1. **foreach** $j \in J$ **do**
2. Let $S^j \leftarrow \{R^j, D^j\},\ R^j \leftarrow \phi,\ D^j \leftarrow \phi$
3. Let $R^j \leftarrow \{r_{final}\}$ such that $T_e(j, r_{final})$ is minimum for all $r \in R$
4. **foreach** $f \in F^j$ **do**
5. $D^j \leftarrow D^j \cup \{d_f\}$ where $T_t(f, d_f, r_{final})$ is minimum for all $d_f \in D_f$
6. **end**
7. **end**

Figure 6.6: The Compute-First Matching Heuristic.

Exhaustive Search - In this case, all the possible resource sets for a particular job are generated and the one guaranteeing the least value of the objective function is chosen for the job. While this heuristic guarantees that the resource set selected will be the best for the job, it searches through MP^K resource sets at a time. This leads to unreasonably large search spaces for higher values of K. For example, for a job requiring 5 datasets with 20 possible data hosts and 20 available compute resources, the search space will consist of $(20 * 20^5) = 64 * 10^6$ resource sets. This algorithm is listed in Figure 6.7.

1. **foreach** $j \in J$ **do**
2. Let $S^j \leftarrow \{R^j, D^j\},\ R^j \leftarrow \phi,\ D^j \leftarrow \phi$
3. Let $U \leftarrow R \times D_{f_1} \times D_{f_2} \times \ldots \times D_{f_K}$ where $f_1, f_2, \ldots, f_K \in F^j$
4. Find $u \in U$ such that $T_{ct}(j)$ is minimum
5. **end**

Figure 6.7: The Exhaustive Search Matching Heuristic.

Greedy - This is the heuristic that was presented in the previous chapter for deadline and budget constrained cost and time minimisation scheduling of data-intensive applica-

tions. This heuristic builds the resource set by iterating through the list of datasets and making a greedy choice for the data host for accessing each dataset, followed by choosing the best compute resource for that data host. At the end of each iteration, it checks whether the compute resource so selected is better than the one selected in previous iteration when the data hosts selected in previous iterations are considered. The running time of this heuristic is $O(MKP)$.

6.3 Scheduling Heuristics

The mapping heuristic finds a resource set such that the objective function is minimised for a single job. However, the goal here is to produce a schedule such that the objective function is minimised over the entire set of jobs. Many algorithms have been proposed for the problem of scheduling a set of independent jobs [38] and two well-known heuristics are the *MinMin* and the *Sufferage* heuristics proposed by Maheswaran, et al. [142] for dynamic scheduling of jobs on heterogeneous computing resources. These are extended to take into account the distributed data requirements of the target application model.

The extended MinMin scheduling heuristic is listed in Figure 6.8. The basic idea of this heuristic is to find the job that has the minimum value of the objective function and allocate it to the resource set that achieves it. The intuition behind this is that such an allocation over all the jobs will minimize the overall objective function. The term J_U denotes the set of jobs that have not been allocated to any resource set yet. In the beginning, it matches all the jobs to a resource set that guarantees minimum value of the objective function for that job (line 4). This is produced through matching heuristics such as the SCP Tree Search, Greedy, Compute-First or Exhaustive Search, that have been presented in previous sections. Then, the job that has achieved the minimum value of the objective function in the present allocation, is allocated to its chosen resource set (line 7). Allocation means that the job is mapped to an available processor or a queue slot on the remote computational node. If the available slots on the resource have already been allocated to previous jobs, the job is assigned provisionally to the compute resource by storing it in a local queue corresponding to that resource. This is done even if there were other available resources as the matching function and MinMin would have taken those

into consideration while mapping the job. This job is then removed from the unallocated job set. As allocation changes the availability of the resource with respect to the number of available slots, the resource information is updated and the process is repeated until all the jobs in J_U have been allocated to some resource set.

1. **repeat**
   Begin Mapping
2. **repeat**
3. **foreach** $j \in J_U$ **do**
4. Find the least value of the objective function for j and find the resource set that achieves the value
5. **end**
6. Find the job $j \in J_U$ with the minimum value of the objective function
7. Allocate j to its resource set that was selected previously
8. Remove j from J_U
9. Update the resource availability based on the allocation performed in the previous step
10. **until** J_U *is empty*
    End Mapping
11. Dispatch the mapped jobs to the selected resources such that the job allocation limit of each resource is not exceeded
12. Wait until the next scheduling event
13. **foreach** *job completed in the previous interval* **do**
14. For each dataset that has been transferred from a remote data host for the job, add its eventual destination (compute resource) as a future source of the dataset for the jobs remaining in J_U
15. **end**
16. For each resource, revise its capability estimates (job allocation limit or available queue slots) depending on various information sources such as external performance monitors or the jobs completed in the previous interval
17. **until** *all jobs are completed*

Figure 6.8: The MinMin Scheduling Heuristic extended for distributed data-intensive applications.

The dispatching function cycles through the set of compute resources and submits the jobs that were allocated to available slots on the remote resource. The jobs that were stored on the local queues are returned back to the unallocated jobs list. The scheduler then waits for the specified polling interval or for a specific event to resume.

When a job is scheduled for execution on a compute resource, all the datasets that are required for the job and are not available local to the resource, are transferred to the resource prior to execution. These datasets become replicas that can be used by following jobs. Here, this is taken into account by registering the compute resource in question (or

its associated data host) as a source of the transferred datasets for succeeding allocation loops (line 14). This enables the exploitation of both temporal and spatial locality of data access.

Begin Mapping
1. **repeat**
2. **foreach** $j \in J_U$ **do**
3. Find the best (least) value of the objective function for j and find the resource set that achieves the value
4. Find the second best value of the objective function for j
5. sufferage value = second best value - best value
6. **end**
7. Find the job $j \in J_U$ with the maximum sufferage value
8. Assign j to the resource set that was selected for it originally
9. Remove j from J_U
10. Update the resource availability based on the allocation performed in the previous step
11. **until** J_U *is empty*

End Mapping

Figure 6.9: Sufferage Algorithm.

The motivation behind the Sufferage heuristic (listed in Figure 6.9) is to allocate a resource set to a job that would be disadvantaged the most (or "suffer" the most) if that resource set were not allocated to it. This is determined through a sufferage value computed as the difference between the second best and the best value of the objective function for the job.

For each job, the resource set that offers the least value of the objective function is determined through the same mechanisms as that in MinMin. Then the compute resource in that resource set is removed from consideration and the matching function is rerun to provide another minimal resource set with the next best value for the objective function. The selection of the compute resource determines both the execution metrics and the data transfer metrics. Therefore, removing it from consideration will produce the maximum impact on the value of the objective function. After determining the sufferage value for each job, the job with the largest sufferage value is then selected and assigned to its chosen resource set. The rest of the heuristic including dispatching and updating of compute resource and data host information proceeds in the same manner as MinMin.

6.4 Evaluation of Scheduling Algorithms

Effective evaluation of scheduling algorithms requires the study of their performance under different scenarios such as different user inputs and varying resource conditions. Within Grid environments, resource loads and the number of users vary continuously and the spread of resources among different administrative domains makes it nearly impossible to control the environment to provide a stable configuration for evaluation. Furthermore, the network plays a large role in the performance of scheduling algorithms for data-intensive applications and it is impossible to create consistent conditions over public networks. The scale of the evaluation is also limited by the number of Grid resources that can be accessed.

Therefore, it was decided to evaluate the performance of algorithms on a simulated Grid environment to ensure a stable and repeatable configuration. Simulation has been used extensively for modelling and evaluation of distributed computing systems and the popularity of this methodology for evaluation of Grid scheduling algorithms have led to the availability of several Grid simulation packages [194]. Some of the simulation systems available for data-intensive computing environments such as Data Grids include GridSim [46], MONARC simulator [135], OptorSim [29], ChicSim [174] and SimGrid [49]. GridSim enables modelling and simulation of heterogeneous Grid resources with time-shared and space-shared node allocation and different economic costs; Grid networks with different routing topologies and QoS classes [193]; and Data Grid replica catalogs that can be connected in different configurations [192]. Also, it presents itself as a toolkit that allows creation of different applications such as resource brokers having scheduling algorithms with different objectives. Most importantly, the Grid model followed by GridSim is the closest, among all others, to that followed in this thesis. Hence, GridSim was used as the simulation system for evaluating the scheduling algorithms for distributed data-intensive applications.

Evaluation of the scheduling algorithms in GridSim required modelling of Grid resources, their interconnections and the data-intensive applications. The sections that follow describe in detail how each of these were modelled.

6.4.1 Simulated Resources

The testbed modelled in this evaluation is shown in Figure 6.10 and is based on a subset of the European Union DataGrid Testbed [92]. The modelled testbed contains 11 resources spread across 6 countries connected via high capacity network links. Each resource, except the one at CERN (Geneva), was used both as a compute resource and as a data host. The resource at CERN was used as a pure data source (data host) in the evaluation and therefore, no jobs were submitted to it for execution.

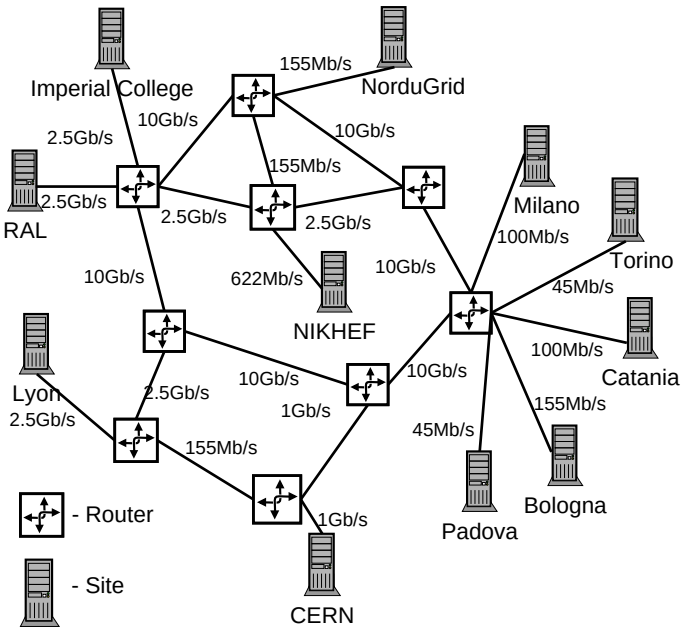


Figure 6.10: European Union DataGrid testbed model used in simulation.

The resources in the actual testbed have gone through several configuration changes, not all of which are publicly available, and hence it was impossible to model their layout and CPU capability accurately. Instead, it was decided to create a configuration for each resource such that the modelled testbed, in whole, would reflect the heterogeneity of platforms and capabilities that is normally the characteristic of Grids. All the resources were simulated as clusters of single CPU nodes or Processing Elements (PEs) with a batch job management system using space-shared policy. This modelled real world Grid resources that are generally high performance clusters in which each job is allocated to a processing node through a job submission queue. The processing capabilities of the PEs were rated in terms of Million Instructions Per Sec (MIPS) so that the application requirements can

Table 6.1: Resources within EDG testbed used for evaluation.

Resource Name (Location)	**No. of Nodes**	**Single PE Rating (MIPS)**	**Storage (TB)**	**Mean Load**
RAL (UK)	41	1140	2.75	0.9
Imperial College (UK)	52	1330	1.80	0.95
NorduGrid (Norway)	17	1176	1.00	0.9
NIKHEF (Netherlands)	18	1166	0.50	0.9
Lyon (France)	12	1320	1.35	0.8
CERN (Switzerland)	–	–	12	–
Milano (Italy)	7	1000	0.35	0.5
Torino (Italy)	4	1330	0.10	0.5
Catania (Italy)	5	1200	0.25	0.6
Padova (Italy)	13	1000	0.05	0.4
Bologna (Italy)	20	1140	5.00	0.8

be modelled in Million Instructions (MI). The configuration assigned to the resources in the testbed for the simulation are listed in Table 6.1.

To model resource contention caused by multiple users submitting jobs simultaneously and the resultant variation in resource availability, a *load factor* was associated with each resource. The load factor is simply the ratio of the number of PEs that are occupied to the total number of PEs available in a resource. During simulation, the instantaneous load (or number of PEs occupied) for each resource was derived from a Gaussian distribution centered around its mean load factor shown in Table 6.1.

Storage at the resources was modelled as the total disk capacity available at the site. Site access latencies such as disk read time were ignored as these are less than the network delays by an order of magnitude. The network between the resources were modelled as the set of routers and links shown in Figure 6.10. Variations of the available network bandwidth are simulated by associating a link load factor, which is the ratio of the available bandwidth to the total bandwidth for a network link. During simulation, the instantaneous measure of the link load is derived from another Gaussian distribution centered around a mean load assigned at random, at the start of the simulation, to each of the links.

It was possible to keep track of the various load variations through information services built into the simulation entities. For example, it was possible to query the instantaneous bandwidth of the network link between any two resources. It was also possible to determine resource availability information by querying the resource for its instantaneous load and number of PEs available.

6.4.2 Distribution of Data

A universal set of 1000 datasets was used for this evaluation. Studies of similar environments [159] have shown that the size of the datasets follow a heavy-tailed distribution in which there are larger numbers of smaller size files and vice versa. Therefore, the set of datasets are generated with sizes distributed according to the logarithmic distribution in the interval $[1GB, 6GB]$. The distribution of datasets in a Data Grid depends on many factors including variations in popularity, the replication strategy employed and the nature of the Grid fabric. To model this distribution, at the start of the simulation, each of the datasets were replicated on one or more of the data hosts according to a preset pattern of file distribution. Two common patterns of file distribution considered in this evaluation are given below:

- *Uniform :* Here, the distribution of datasets is modelled on a uniform random probability distribution. Here, each file is equally likely to be replicated at any site.
- *Zipf :* Zipf-like distributions follow a power law model in which the probability of occurrence of the i^{th} ranked file in a list of files is inversely proportional to i^{-a} where $a \leq 1$. In other words, a few files are distributed widely whereas most of files are found in one or two places. This models a scenario where the files are replicated on the basis of popularity. It has been shown that Zipf-like distributions holds true in cases such as requests for pages in World Wide Web where a few of the sites are visited the most [40]. This scenario has been evaluated for a Data Grid environment in related publications [48].

Henceforth, the distribution applied is described by the variable $Dist$. The distribution of datasets was also controlled through a parameter called the *degree of replication* which is the maximum possible number of replicas of any dataset present in the Data Grid at the

beginning of the simulation. For example, a degree of replication of 3 means there can be up to 3 copies of any dataset on the Grid resources. However, not all datasets are replicated to the limit of the degree of replication. In a uniform distribution, a higher percentage of the datasets are replicated up to the maximum limit than in the Zipf distribution. The degree of replication in this evaluation is 5.

6.4.3 Application and Jobs

The simulated application models a Bag-of-Task application that can be converted into a set of independent jobs. The size of the application was determined by the number of jobs in the set (or N). Each job translates to a Gridlet object which is the smallest unit of execution in GridSim. The computational size of a job or the job length, described by the term $Size$, is expressed in terms of the time taken to run the job on a standard PE with a MIPS rating of 1000. That is, a job with length 100,000 MI runs for 100 seconds on a standard resource. Each job requires as input, a pre-determined number of datasets (or K datasets) selected at random from the universal set of datasets. For the purpose of comparison, K is kept a constant among all the jobs in a set although this is not a condition imposed on the heuristic itself.

An experiment is an execution of the all the heuristics for an application while keeping the values for these parameters constant, and is therefore described by the tuple $(N, K, Size, Dist)$. At the beginning of each experiment, the set of datasets, their distribution among the resources, and the set of jobs are generated. This configuration is then kept constant while each of the scheduling heuristics are evaluated in turn. To keep the resource and network conditions repeatable among evaluations, a random number generator is used with a constant seed. The evaluation is conducted with different values for $N, K, Size$ and $Dist$ to study the performance under different input conditions.

6.5 Experimental Results

6.5.1 Comparison between the Matching Heuristics

The performances of the matching heuristics discussed in the previous section were compared with each other by pairing each of them with the MinMin heuristic and conducting 50 simulation experiments with different values for N, K, $Size$ and $Dist$. Throughout this section, *SCP* refers to the SCP Tree Search heuristic presented in the previous section. The objective of this evaluation was to reduce the *makespan* [142] of the application which is the total wallclock time between the submission of the first job to the completion of the last job in the set.

Table 6.2: Summary of Simulation Results.

Mapping Heuristic	**Geometric Mean**	**Avg. deg. (SD)**	**Avg. rank (SD)**
Compute-First	37593.71	69.01 (19.4)	3.63 (0.48)
Greedy	36927.44	71.86 (50.55)	3.23 (0.71)
SCP	24011.17	7.68 (10.42)	1.67 (0.6)
Exhaustive Search	23218.49	3.87 (6.46)	1.47 (0.58)

The results of the experiments are summarised in Table 6.2 and are based on the methodology provided by Casanova, et. al [51]. For each matching heuristic, the table contains three values:

1. *Geometric Mean* of the makespans: The geometric mean is used as the makespans vary in orders of magnitude depending on parameters such as number of jobs per application set, number of files per job and the size of each job. The lower the geometric mean, the better the performance of the heuristic.

2. Average degradation (*Avg. deg.*) from the best heuristic: In an experiment, the degradation of a heuristic is the difference between its makespan and the makespan of the best heuristic for that experiment and is expressed as a percentage of the latter measure. The average degradation is computed as an arithmetic mean over all experiments and the standard deviation of the population is given in the parentheses

next to the means in the table. This is a measure of how far a heuristic is away from the best heuristic for an experiment. A lower number for a heuristic certainly means that on an average that heuristic is better than the others.

3. Average rank (*Avg. rank*) of each heuristic in an experiment: The ranking is in the ascending order of makespans produced by the heuristics for each experiment, that is, the lower the makespan, the lower the rank of the heuristic. The average rank is calculated over all the experiments and the standard deviation is provided alongside the averages in parantheses.

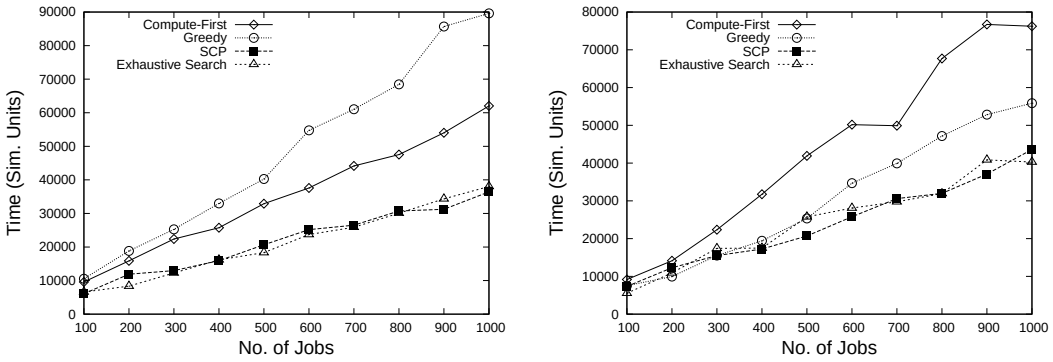
The three values together provide a consolidated view of the performance of each heuristic. For example, it can be seen that on average Compute-First and Greedy both perform worse than either SCP or Exhaustive Search. However, the standard deviation of the population is much higher in the case of Greedy than that of Compute-First. Therefore, Compute-First can be expected to perform as the worst heuristic most of time. Indeed, in a few of the experiments, Greedy performed as good or even better than SCP while Compute-First never came close to the performance of the other heuristics.

As expected, between SCP and Exhaustive Search, the latter provides the better results by having a consistently lower score than the former. However, the nature of Exhaustive Search means that as the number of datasets per job increases, the number of resource sets that need to be considered by the heuristic increases dramatically. The geometric mean and average rank of SCP is close to that of Exhaustive Search heuristic. The average rank is less than 2 for both heuristics which implies that in many scenarios, SCP provides a better performance than Exhaustive Search.

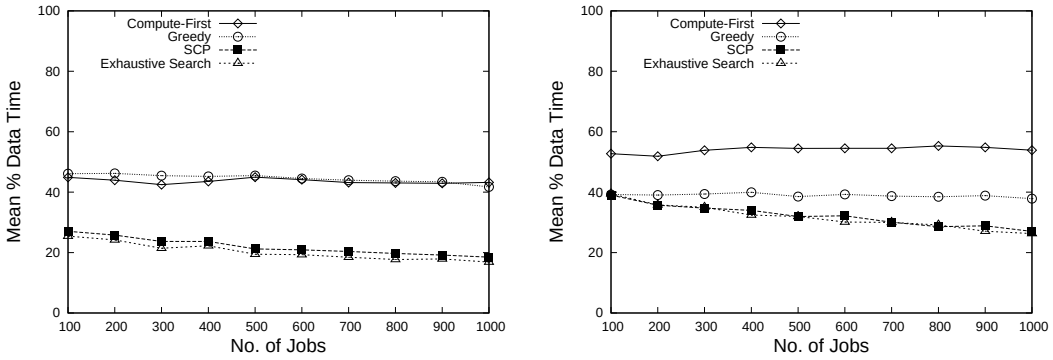
Impact of Data Transfer on Performance

Figures 6.11-6.13 show a more fine-grained view of the experimental evaluation by showing the effect of varying one of the variables (N, K, $Size$, $Dist$), all others kept constant. Essentially, these are snapshots of the experimental results that contributed to the summary data in Table 6.2. Along with the makespan, two more measures of performance are considered within these figures. These are:

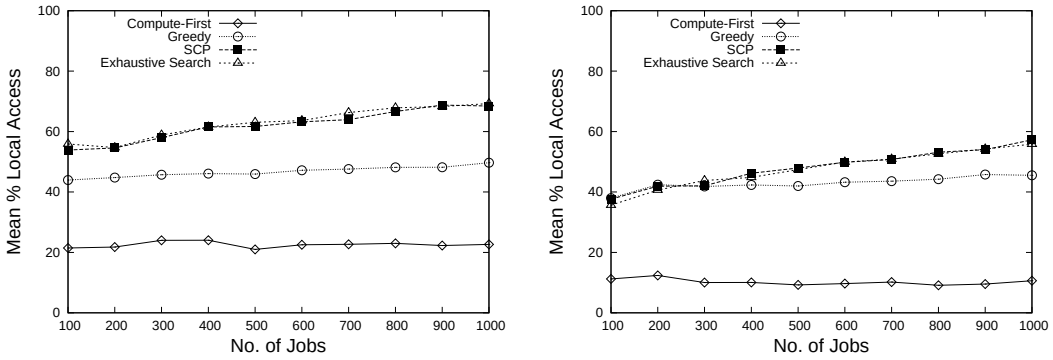
1. *Mean percentage of data time*: For each job in an experiment, the share of the data



(a) Makespan vs. No. of Jobs



(b) Data Time vs. No. of Jobs



(c) Locality vs. No. of Jobs

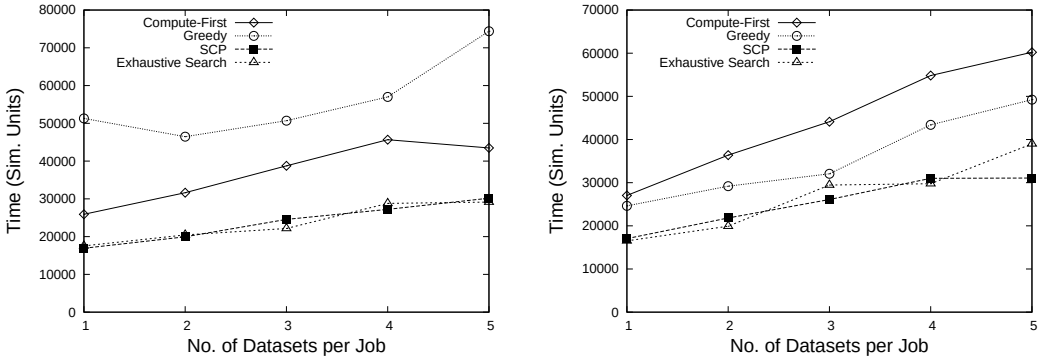
Figure 6.11: Evaluation with increasing number of jobs ($Size$=300000 MI, K=3, Left: $Dist$=Uniform, Right: $Dist$=Zipf).

transfer time is calculated as a percentage of the total execution time for that job. The average of this measure over all the jobs then represents the mean impact of the data transfer time on the set of jobs or the application as a whole. A lower number is better as one of the aims of the scheduling algorithms presented so far has been to reduce the data transfer time.

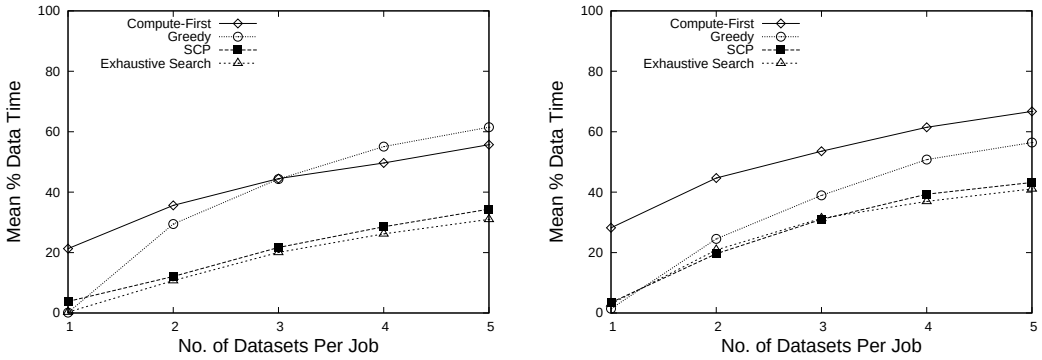
2. *Mean locality of access*: For each job, the ratio of the number of datasets accessed from the local disk storage of the compute resource to the total number of datasets accessed by the job from all resources is calculated as a percentage of the latter and is termed as the *local access ratio*. Since by design, each of the jobs in an experiment accessed the same number of datasets, the average of the local access ratio over all the jobs becomes a measure of locality exploited by each of the algorithms. In this case, a higher number is better as increased local access decreases the impact of remote data transfer on the performance.

These two measures represent two slightly different perspectives on the data access performed by the jobs. Consider a job that requires one dataset of size 6 GB and two datasets of size 1 GB each. The job may be scheduled such that the larger-sized dataset is accessed locally, whereas the smaller-sized datasets may be accessed from remote data hosts. In this case, the data transfer component is small but the locality of access is low as well. However, when the sizes of the datasets are more or less equal, the locality of access becomes an important factor. These two measures, therefore, give an indication of the importance given by the algorithms to the location of data. These can be correlated with the makespan to judge the impact of the selection made by an algorithm on its performance.

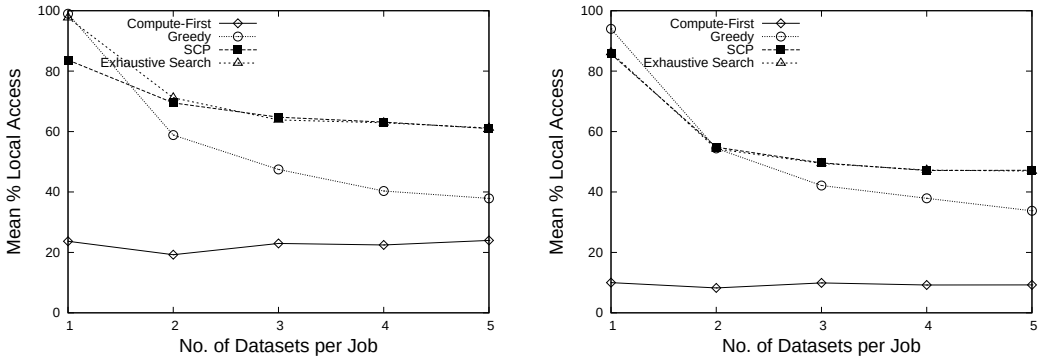
Figure 6.11 shows the impact of the number of jobs on the performance of the algorithm. It can be seen that as the number of jobs increases, the makespan of Compute-First and Greedy heuristic rise more steeply than the other two. The impact of data time is lower for SCP and Exhaustive Search than it is for Compute First and is a factor in their improved performance. Locality of access is also higher for the former two algorithms and it increases as the number of jobs in the set increases. This is because the probability of datasets being shared increases with more jobs accessing the same global set of datasets as was the case in this evaluation. This means that there is a greater chance for transferred



(a) Makespan vs. No. of Jobs

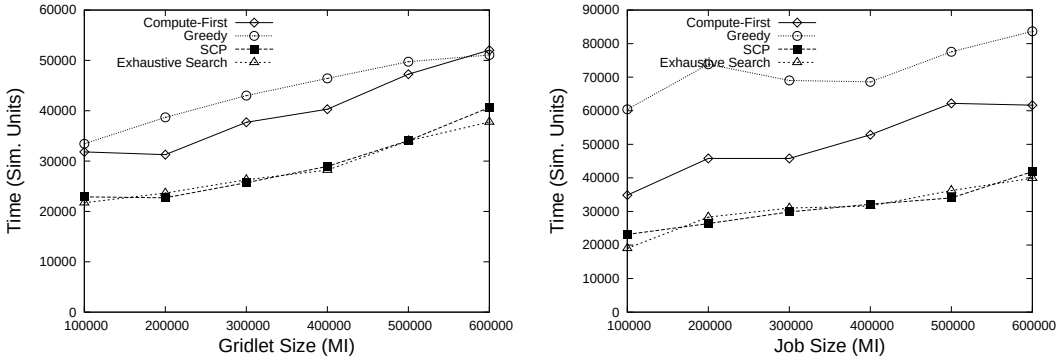


(b) Data Time vs. No. of Jobs

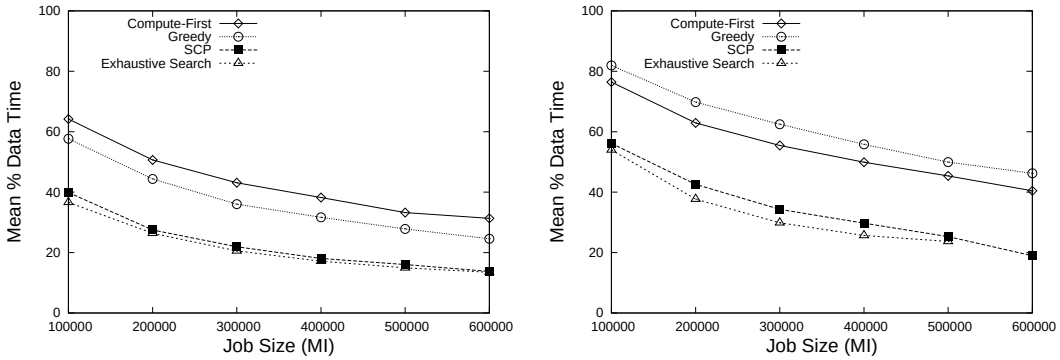


(c) Locality vs. No. of Jobs

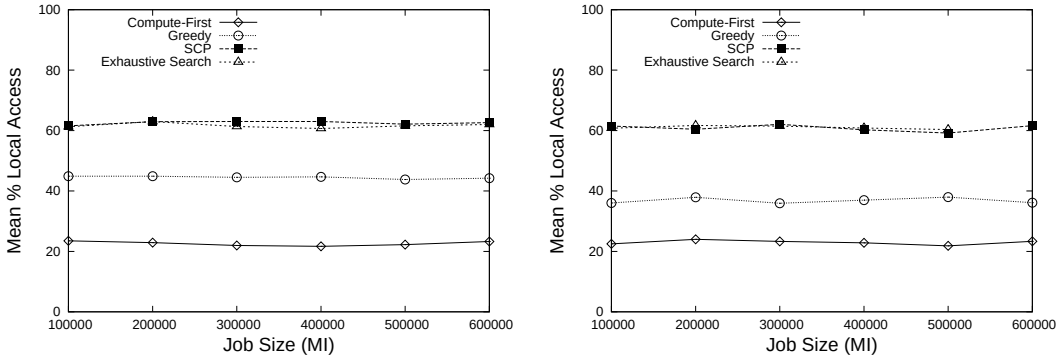
Figure 6.12: Evaluation with increasing number of datasets per job (N=600, $Size$=300000 MI, Left: $Dist$=Uniform, Right: $Dist$=Zipf).



(a) Makespan vs. No. of Jobs



(b) Data Time vs. No. of Jobs



(c) Locality vs. No. of Jobs

Figure 6.13: Evaluation with increasing computational size (N=600, $Dist$=Uniform, Left: K=3, Right: K=5).

datasets to be reused with a higher number of jobs. In case of Zipf distribution (right column), the locality is lower than in the case of Uniform distribution which means that a job submitted to a compute resource is less likely to find its required datasets locally. This can be attributed to the rarer availability of datasets in Zipf distribution than in the Uniform distribution.

An interesting result here is that even with a high locality of access, the Greedy heuristic performs significantly worse than Compute-First for Uniform distribution (left column) while it performs better than the latter when the datasets are replicated according to Zipf distribution. In the second case, there is a lower number of choices than in the first and thus, the greedy strategy has a better probability of forming minimal resource sets. In this case, it can be seen that the performance of Greedy comes close to or in some cases, becomes as competitive as SCP mirroring the results of Table 6.2. With a higher number of choices, the greedy strategy has a lower probability of arriving at the best compute resource for a job and its performance is degraded.

Figure 6.12 shows the impact of changing only the number of datasets per job. Some of the trends in the previous graphs are also reflected here. With only one dataset per job, all algorithms except for Compute-First are able to produce schedules with zero data time and full locality of access. With the jobs per dataset increasing, the impact of data transfer time increases at a faster rate for Greedy than for SCP and Exhaustive Search. Also, the locality reduces more steeply in the Zipf distribution than in the Uniform distribution, because there are fewer data hosts for each file. Finally, Figure 6.13 shows the impact of the computation time on the performance of data-oriented scheduling algorithms. The locality remains almost constant throughout the experiments. However, as expected, the impact of data transfer is steadily reduced with increasing size of computation.

An interesting result here is that the performance of Exhaustive Search is worse than that of SCP in certain cases. This runs contrary to expectations that Exhaustive Search will produce the best results in every case. This is due to the fact that MinMin itself is not guaranteed to give the best schedules in every case [142]. The assignment of resources to a job impacts the selection of resources for jobs that are yet to be assigned. This leads to variations in performance of the Exhaustive Search algorithm.

6.5.2 Comparison between MinMin and Sufferage

Table 6.3: Summary of Comparison between MinMin and Sufferage.

Heuristic	Geometric Mean	Avg. deg	Avg. rank
MinMin			
Compute-First	19604.73	18.7 (12.84)	4.93 (1.0)
Greedy	25782.28	57.93 (28.51)	6.33 (1.45)
SCP	17353.87	5.2 (13.58)	1.73 (1.44)
Exhaustive Search	18481.26	11.83 (11.39)	3.47 (1.41)
Sufferage			
Compute-First	60631.56	269.31 (57.81)	8.0 (0)
Greedy	18558.61	12.06 (8.45)	4.2 (1.72)
SCP	17353.87	5.2 (13.58)	1.73 (1.44)
Exhaustive Search	18584.88	12.47 (11.53)	3.67 (1.53)

Each of the matching heuristics were paired with both MinMin and Sufferage scheduling algorithms and evaluated to determine if the latter provided a better performance than the former. The results of the experiments carried out within this evaluation is summarised using the same metrics as in the previous section and are listed in Table 6.3. It can be seen that there is little difference in the performance of both SCP and Exhaustive Search heuristics when coupled with either MinMin or Sufferage scheduling algorithms. Also, there is only a slight improvement in the performance for Greedy when coupled with the Sufferage algorithm. However, the performance for Compute-First is significantly degraded by coupling it with the Sufferage algorithm. On average, it is about 2 1/2 times as worse as the best heuristic in any experiment. Also, the Compute-First-Sufferage pair is ranked 8th in terms of performance in all experiments (standard deviation is zero). In other words, it gives the worst performance in every case.

6.6 Related Work

On the basis of the taxonomy presented in Chapter 3, the algorithms presented in this thesis can be classified as follows: *Bag-of-Tasks* application model, *individual* in scope,

decoupled from replication, with *makespan* and *QoS* utility functions and exploiting both *temporal* and *spatial* locality. Chapter 3, Section 3.2.4 provided a brief survey of some of the related scheduling algorithms for data intensive applications on Grid resources.

Some of the publications [29, 160, 173] that were surveyed in Chapter 3, Section 3.2.4 tackle the problem of replicating the data for a single job depending on the site where the job is scheduled. However, the application model applied in this thesis is closer to that of Casanova, et.al [51] who investigate scheduling algorithms for a set of independent tasks that share files. They extend the MinMin and Sufferage algorithms to consider data requirements of the tasks and introduce the XSufferage algorithm to take advantage of file locality. However, in their article, the source of all the files for the tasks is the resource that dispatches the jobs. This work is extended by Giersch, et. al [97] to consider the general problem of scheduling tasks that share multiple files, each available from multiple sources. They focus on developing routing algorithms for staging the input files through the network links on to data resources, close to the selected compute resources, such that the total execution time is minimised. Khanna, et al. [115] propose a hypergraph-based approach for scheduling a set of independent tasks with a view to minimise the I/O overhead by considering the sharing of files between the tasks. However, they do not take into account the aspect of data replication as the files have only single sources.

The scheduling model considered in this thesis is distinct from those mentioned previously because it considers: a) the problem of selecting a resource set for a job requiring multiple datasets in an environment where the data is available from multiple sources due to prior replication and b) the selection of computational and data resources in such a resource set to be interconnected. This chapter also extends MinMin and Sufferage algorithms similar to that done by Casanova, et al. [51] and Giersch, et al. [97]. However, in the algorithms presented in this thesis, the focus of the effort remains on matching or selection of resources which is not given adequate weightage in related work. The matching algorithms aim to select a resource set such that both the computational and data transfer components of the execution time are reduced simultaneously. This is different from the approach, followed by most of the Data Grid scheduling algorithms studied in Chapter 3, Section 3.2.4, of scheduling the jobs onto a compute resource based on minimum computation time, and then replicating the data to minimise the access time. The latter approach

was generalised and extended to support the multiple datasets model in the previous sections, and was evaluated as the Compute-First heuristic. Simulation results show that Compute-First produces worse schedules when compared to a strategy giving weightage to both computational and data factors such as the SCP Tree Search algorithm.

Mohamed and Epema [146] present a Close-to-Files algorithm for a similar application model, though restricted to one dataset per job, that searches the entire solution space for a combination of computational and storage resources to minimise execution time. This strategy, extended to support multiple datasets per job and evaluated as Exhaustive Search in the previous section, produces good schedules but becomes unmanageable for large solution spaces that occur when more than one dataset is considered per job.

Jain, et al. [113] proposed a set of heuristics for scheduling I/O operations so as to avoid transfer bottlenecks in parallel systems. However, these heuristics do not consider the problem of scheduling computational operations and also, the problem of selecting data sources in case of data replication. Other publications in parallel I/O optimisation [3, 179, 204] pay attention to improving performance through techniques such as interleaving and disk striping. However, such optimisation techniques are not the focus of this thesis.

6.7 Summary

A crucial step in the scheduling of jobs to distributed resources is that of matching the jobs to appropriate resources. This chapter models the problem of matching distributed data-intensive jobs to computational and data resources as an instance of the SCP and proposes a tree-search heuristic based on a solution to the SCP. This is then combined with the MinMin and Sufferage algorithms for scheduling sets of independent jobs and evaluated through simulation against other matching heuristics such as Compute-First, Greedy and Exhaustive Search. Experiments show that the SCP Tree Search and the Exhaustive Search heuristics provide the best performance among all the four heuristics mainly because they exploit the locality of datasets, and thereby reduce the amount of data transferred during execution. However, the high computational complexity of Exhaustive Search means that it will search through large spaces that may become infeasible for jobs requiring large number of datasets. Also, there is no gain in performance by applying the Sufferage heuristic in place of MinMin for scheduling the entire set of jobs.

Chapter 7

Conclusion

This thesis began by studying, characterising and categorising several aspects of Data Grid systems. Data Grids have several unique features such as presence of applications with heavy computing requirements, geographically-distributed and heterogeneous resources under different administrative domains, and large number of users sharing these resources and wanting to collaborate with each other. This thesis then enumerated several characteristics where Data Grids are similar to and are different from other distributed data-intensive paradigms such as content delivery networks, peer-to-peer networks and distributed databases.

Further on, the thesis focused on the architecture of the Data Grids and the fundamental requirements of data transport mechanism, data replication systems, and resource allocation and job scheduling. Taxonomies for each of these areas were developed to classify the common approaches and to provide a basis for comparison of Data Grid systems and technologies. Then, some of the representative systems in each of these areas were compared and categorised according to the respective taxonomies. This exercise presented an insight into the architectures, strategies and practises that are currently adopted within Data Grids. Thus, the taxonomy chapter laid down a comprehensive classification framework that not only serves as a tool to understanding this complex area but also presents a reference to which future efforts can be mapped.

The lessons learnt from the study of Data Grid environments provided the basis for the design of the Gridbus Grid resource broker. The requirements of the broker were to

provide a software framework that: (a) abstracted the heterogeneity of the environment, (b) supported multiple application types, (c) allowed different types of user objectives, (d) supported multiple user interfaces, and (e) handled Grid characteristics such as job failure and dynamic availability. The architectural separation of interface and core layers enabled support for multiple user interfaces such as command line interfaces and web portals. The separation of core and execution layers allowed the broker to support different implementations of Grid services in a standard manner as is shown by the support for a large number of computational and data Grid middleware. The design of the core layer as a collection of passive entities enabled the creation of different application models that implemented different active logical components to manipulate the same entities in different ways. Fault-tolerance on the broker side is provided by a persistent database to which the state of the passive components is saved periodically.

The broker allowed the creation of schedulers that can have a variety of objectives and can take into account various factors such as presence of data and costs of resource usage. Data aware scheduling was demonstrated through a case study of Grid-enabling a data-intensive analysis application for the Belle particle physics experiment. The case study discussed the motivation for using Grid techniques in the Belle experiment and the methodology adopted for Grid-enabling the analysis application. It also evaluated the deployment of the application on a set of Grid resources within Australia. The empirical results indicated that considering both the presence of data and the availability of computational resources led to an improvement in the performance of the application scheduling by improving the job turnaround time.

This exercise motivated further research into the scheduling of distributed data intensive applications on Grid resources. This thesis introduced a generic model of a data intensive application that consists of a set of independent tasks, each of which required one or more datasets available from one or more storage repositories or data hosts in a Grid. For each task, and depending on the objective function, the scheduler is required to select a resource set consisting of one compute resource to execute the task and one data host each for each dataset that needs to be accessed for the task. The model also took into account the economic costs of using the Grid resources. This model was applied to present a greedy algorithm for deadline and budget constrained cost and time minimisation-based

scheduling of the target distributed data-intensive applications. Empirical results of evaluating this algorithm on a set of Grid resources show that the algorithm is able to reduce either the execution and the data transfer time or the cost of computation and data transfer depending on the chosen objective.

Further on, this thesis concentrates on the matching of jobs to resources and models it as an instance of the well-known Set Covering Problem. An approximate heuristic based on tree search is presented and evaluated via simulation against the popular Compute-First strategy, the Greedy strategy proposed previously, and the Exhaustive Search strategy that returns the best match for any job. The results show that the proposed heuristic is better than Compute-First and Greedy approaches and leads to schedules that are competitive with the Exhaustive Search.

7.1 Future Work

This thesis improves the understanding of data intensive Grid computing environments and advances the state-of the art through its contributions. Its investigation has revealed areas in Data Grids where much work remains to be done. Also, the contributions of this thesis have led to new questions that need to be addressed through further research. This section briefly describes some of these questions within each of the areas explored in this thesis.

7.1.1 Brokering of Grid Services

The Gridbus broker has been shown to be effective for executing scientific applications that are sets of independent tasks such as Bag-of-Task or parameter sweep applications, on Grid resources. The broker is currently under heavy development and much of the work is devoted to extending it to support other application models such as process-oriented applications [188]. However, there is still the question of whether newer application models that will emerge in the future can be accommodated by the current architecture of the broker.

Chapter 2, Section 2.2 discussed the Open Grid Service Architecture (OGSA) and its vision of enabling a service-oriented architecture for Grid computing. The Grid commu-

nity has recently standardised on the Web Services Resource Framework (WSRF) [80] to realise the OGSA. To be able to function in a service-oriented environment, the Grid broker has to be able to compose services based on their attributes and create service aggregations to achieve users' utility functions. Grid services present a highly abstracted view of the underlying infrastructure and disruption of an invoked service should be managed by quickly switching to similar services in order to maintain a transparent view of infrastructure. These requirements place demands for intelligent fault management within the broker and motivate the development of new scheduling mechanisms.

7.1.2 Scheduling of Distributed Data-Intensive Workflows

This thesis has explored the scheduling of applications that require multiple datasets each replicated on multiple data repositories on the Grid. The scheduling algorithms proposed in this thesis explicitly take into account the availability of data replicas on distributed resources. The taxonomy in Chapter 3, Section 3.1 introduced four bulk transfer modes - parallel transfers, striped transfers, auto-resizing of buffers and container operations - that may be adopted by Data Grid applications for optimal utilisation of available network capacity. While container operations and parallel transfers can be accommodated in the single location transfer model, it remains an open question whether striped transfers, which require accessing the same file from multiple nodes at the same time, can be handled by the scheduling algorithms proposed in this thesis.

The scheduling algorithms proposed in this thesis apply to the Bag of Task model of applications. However, this thesis has only explored the MinMin and Sufferage scheduling algorithms within the space of scheduling algorithms for sets of independent tasks. It would be interesting to explore the applicability of the matching heuristics proposed in this thesis within some of the other known scheduling techniques such as Genetic Algorithms. It would also be interesting to investigate the applicability of the matching heuristics to other task models such as Directed Acyclic Graphs (DAGs) which are used to model workflows [220] and process-oriented parallel applications. An immediate follow-up work would be to implement the matching heuristics within well-known DAG scheduling algorithms such as the Dynamic Critical Path (DCP) [126] algorithm.

7.1.3 Economic Mechanisms in Data Grids

This thesis has investigated the properties that are unique to Data Grids. Currently, the utility of Data Grids is limited to scientific collaborations that need to manage volumes of shared data. However, some of the tools developed within Data Grids may find applicability to areas outside of scientific computing such as in enterprises with similar requirements for resource sharing and data access. This would require taking into account more strict reliability and security requirements. Another challenge would be to extend existing Data Grid techniques to work with technologies within enterprises such as databases [139].

Present-day Data Grids are based on the notion of sharing resources within virtual organisations. However, as the dependence on Data Grids increases, there will be higher demands for reliability and resource share. Service providers may not be able to fulfil these without investing economically in the infrastructure and would expect returns on their investment. Service consumers will require quality of service guarantees enforced through Service Level Agreements (SLAs). Therefore, a wider exploration of economic aspects of Data Grid computing requires investigation of the utility functions of the participants, SLAs and market mechanisms.

Appendix A

List of Articles Published during the Candidature

Book Chapters

1. P. Asadzadeh, R. Buyya, C. L. Kei, D. Nayar, and **S. Venugopal**, Global Grids and Software Toolkits: A Study of Four Grid Middleware Technologies, *High Performance Computing: Paradigm and Infrastructure*, L. Yang and M. Guo (eds), Wiley Press, USA, June 2005.
2. A. Luther, R. Buyya, R. Ranjan, and **S. Venugopal**, Peer-to-Peer Grid Computing and a .NET-based Alchemi Framework, *High Performance Computing: Paradigm and Infrastructure*, L. Yang and M. Guo (eds), Wiley Press, USA, June 2005.
3. C. S. Yeo, M. Dias de Assuno, J. Yu, A. Sulistio, **S. Venugopal**, M. Placek, and R. Buyya, *Utility Computing and Global Grids*, Hossein Bidgoli (ed), The Handbook of Computer Networks, Wiley Press, USA, (accepted in April 2006 and in print).

Journals

1. **S. Venugopal**, R. Buyya, and K. Ramamohanarao, "A Taxonomy of Data Grids for Distributed Data Sharing, Management and Processing", *ACM Computing Surveys*, 38(1):1-53, ACM Press.
2. **S. Venugopal**, R. Buyya and L. Winton, "A Grid Service Broker for Scheduling e-Science Applications on Global Data Grids", *Concurrency and Computation: Practice and Experience*, 18(6):685-699, Wiley Press, UK.
3. J. Yu, **S. Venugopal** and R. Buyya, "A Market-Oriented Grid Directory Service for Publication and Discovery of Grid Service Providers and their Services", *Journal of Supercomputing*,36(1):17-31, Springer Science+Business Media, Berlin, Germany.
4. R. Buyya, S. Date, Y. Mizuno-Matsumoto, **S. Venugopal** and D. Abramson, "Neuroscience Instrumentation and Distributed Analysis of Brain Activity Data: A Case for

eScience on Global Grids", *Concurrency and Computation: Practice and Experience (CCPE)*, 17(15):1783 - 1798, Wiley Press, U.K.

5. R. Buyya, M. Murshed, D. Abramson and **S. Venugopal**, "Scheduling Parameter Sweep Applications on Global Grids: A Deadline and Budget Constrained Cost-Time Optimisation Algorithm", *Software: Practice and Experience (SPE)* , 35(5):491 - 512, Wiley Press, UK.
6. R. Buyya, D. Abramson and **S. Venugopal**,"The Grid Economy", *Proceedings of the IEEE*, M. Parashar and C. Lee (editors), 93(3):698-714, IEEE Press, USA.

Refereed Conference Papers

1. **S. Venugopal** and R. Buyya, "A Set Coverage-based Mapping Heuristic for Scheduling Distributed Data-Intensive Applications on Global Grids", *Proceedings of the 7th International Conference on Grid Computing (GRID 06)*, Barcelona, Spain, Sept. 28-29, 2006, IEEE CS Press, Los Alamitos, CA, USA.
2. M. Dias de Assuncao, K. Nadiminti, **S. Venugopal**, T. Ma, and R. Buyya, An Integration of Global and Enterprise Grid Computing: Gridbus Broker and Xgrid Perspective, *Proceedings of the 4th International Conference on Grid and Cooperative Computing (GCC 2005)*, Beijing, China, Dec. 2005, Springer-Verlag, Germany.
3. **S. Venugopal** and R. Buyya, "A Deadline and Budget Constrained Scheduling Algorithm for eScience Applications on Data Grids", *Proceedings of the 6th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP-2005)*, Oct. 2005, Melbourne, Australia, Springer-Verlag, Berlin, Germany.
4. N. Muthuvelu, J. Liu, N. L. Soe, **S. Venugopal**, A. Sulistio and R. Buyya, "A Dynamic Job Grouping-Based Scheduling for Deploying Applications with Fine-Grained Tasks on Global Grids", *Proceedings of the 3rd Australasian Workshop on Grid Computing and e-Research (AusGrid 2005)*, Feb. 2005, Newcastle, Australia, Australian Computing Society (ACS).
5. B. Beeson, S. Melnikoff, **S. Venugopal** and D.G. Barnes, "A Portal for Grid-enabled Physics", *Proceedings of the 3rd Australasian Workshop on Grid Computing and e-Research (AusGrid2005)*, Feb. 2005, Newcastle, Australia, ACS.
6. B. Hughes, **S. Venugopal** and R. Buyya, "Grid-based Indexing of a Newswire Corpus", *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing (GRID 2004)*, Nov. 2004, Pittsburgh, USA, IEEE CS Press, USA.
7. **S. Venugopal**, R. Buyya, and L. Winton, "A Grid Service Broker for Scheduling Distributed Data-Oriented Applications on Global Grids", *Proceedings of the 2nd International Workshop on Middleware for Grid Computing (MGC 04)*, Oct. 2004, Toronto, Canada, ACM Press, USA.
8. R. Buyya and **S. Venugopal**, "The Gridbus Toolkit for Service Oriented Grid and Utility Computing: An Overview and Status Report", *Proceedings of the 1st IEEE International Workshop on Grid Economics and Business Models (GECON 04)*, Apr. 2004, Seoul, Korea, IEEE Press, USA.
9. R. Buyya, S. Date, Y. Mizuno-Matsumoto, **S. Venugopal**, and D. Abramson, "Composition of Distributed Brain Activity Analysis and its On-Demand Deployment on

Global Grids", *Proceedings of the 10th International Conference on High Performance Computing (HiPC 2003) Workshops,* Dec. 2003, Hyderabad, India.

Posters and Magazine Articles

1. **S. Venugopal** and R. Buyya, "Cost-based Scheduling for Data-Intensive Applications on Global Grids",*Proceedings of the 14th IEEE International Symposium on High Performance Distributed Computing (HPDC-14)*, Poster, July 2005, Research Triangle Park, North Carolina, USA. IEEE CS Press, USA.
2. R. Buyya and **S. Venugopal**,"A Gentle Introduction to Grid Computing and Technologies", *CSI Communications*, Vol.29, No.1, pp9-19, Computer Society of India (CSI) Publication, July 2005.

REFERENCES

[1] Abramson, D., Giddy, J., and Kotler, L. (2000). High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid? In *Proceedings of the 14th International Parallel and Distributed Processing Symposium (IPDPS 2000)*, Cancun, Mexico. IEEE CS Press, Los Alamitos, CA, USA.

[2] Abramson, D., Sosic, R., Giddy, J., and Hall, B. (1995). Nimrod: a tool for performing parametrised simulations using distributed workstations. In *Proceedings of the Fourth IEEE International Symposium on High Performance Distributed Computing (HPDC '95)*, Pentagon City, VA, USA. IEEE CS Press , Los Alamitos, CA, USA.

[3] Acharya, A., Uysal, M., Bennett, R., Mendelson, A., Beynon, M., Hollingsworth, J., Saltz, J., and Sussman, A. (1996). Tuning the performance of i/o-intensive parallel applications. In *Proceedings of the fourth workshop on I/O in parallel and distributed systems (IOPADS '96)*, Philadelphia, PA, USA. ACM Press.

[4] Adachi, I. et al. (2004). Belle computing system. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 534(1-2):53–58.

[5] Aderholz, M. et al. (2000). MONARC Project Phase2 Report. Technical report, CERN.

[6] Alfieri, R. et al. (2005). From gridmap-file to VOMS: managing authorization in a Grid environment. *Future Gener. Comput. Syst.*, 21(4):549–558.

[7] Allcock, B., Bester, J., Bresnahan, J., Chervenak, A., Foster, I., Kesselman, C., Meder, S., Nefedova, V., Quesnel, D., and Tuecke, S. (2001a). Secure, efficient data transport and replica management for high-performance data-intensive computing. In *Proceedings of IEEE Mass Storage Conference*, San Diego, USA. IEEE CS Press, Los Alamitos, CA, USA.

[8] Allcock, B., Bester, J., Bresnahan, J., Chervenak, A. L., Foster, I., Kesselman, C., Meder, S., Nefedova, V., Quesnel, D., and Tuecke, S. (2002). Data management and transfer in high-performance computational grid environments. *Parallel Computing*, 28(5):749–771.

[9] Allcock, B., Foster, I., Nefedova, V., Chervenak, A., Deelman, E., Kesselman, C., Lee, J., Sim, A., Shoshani, A., Drach, B., and Williams, D. (2001b). High-performance remote access to climate simulation data: a challenge problem for data grid technologies. In *Proceedings of the 2001 ACM/IEEE conference on Supercomputing (SC '01)*, Denver, CO, USA. ACM Press, New York, NY, USA.

[10] Allcock, W. (2003). GridFTP Protocol Specification. (Global Grid Forum Recommendation GFD.20).

[11] Allen, G., Benger, W., Goodale, T., Hege, H.-C., Lanfermann, G., Merzky, A., Radke, T., Seidel, E., and Shalf, J. (2000). The Cactus Code: A Problem Solving Environment for the Grid. In *Proceedings of the 9th International Symposium on High Performance Distributed Computing (HPDC-9)*, Pittsburgh, USA. IEEE CS Press, Los Alamitos, CA, USA.

[12] Aloisio, G. and Cafaro, M. (2002). Web-based access to the Grid using the Grid Resource Broker portal. *Concurrency and Computation: Practice and Experience*, 14(13-15):1145–1160.

[13] Alonso, R. and Barbara, D. (1989). Negotiating data access in federated database systems. In *Proceedings of the 5th International Conference on Data Engineering*, pages 56–65, Los Angeles, CA, USA. IEEE CS Press, Los Alamitos, CA, USA.

[14] Andersen, D., Balakrishnan, H., Kaashoek, F., and Morris, R. (2001). Resilient overlay networks. In *Proceedings of the 18th ACM symposium on Operating systems principles(SOSP '01)*, pages 131–145, Banff, Alberta, Canada. ACM Press, New York, NY, USA.

[15] Anderson, D. P., Cobb, J., Korpela, E., Lebofsky, M., and Werthimer, D. (2002). SETI@home: an experiment in public-resource computing. *Communications of the ACM*, 45(11):56–61.

[16] Anglano, C. et al. (2001). Integrating GRID tools to build a computing resource broker: activities of DataGrid WP1. In *Proceedings of the 2001 International Conference on Computing in High Energy and Nuclear Physics (CHEP)*, Beijing, China.

[17] Anjomshoaa, A., Brisard, F., Drescher, M., Fellows, D., Ly, A., McGough, S., Pulsipher, D., and Savva, A. (2005). Job Submission Description Language (JSDL) Specification, Version 1.0. Technical report, Global Grid Forum.

[18] Antonioletti, M. et al. (2005). Web Services Data Access and Integration (WS-DAI). Technical report, GGF DAIS Working Group. Informational Document.

[19] Apache Software Foundation (2006). Apache Tomcat. `http://tomcat.apache.org/`. Accessed Jun 2006.

[20] Ardaiz, O., Artigas, P., Eymann, T., Freitag, F., Navarro, L., and Reinicke, M. (2003). Self-organizing resource allocation for autonomic networks. In *Proceedings of the 1st International Workshop on Autonomic Computing Systems*, Prague, Czech Republic. IEEE CS Press, Los Alamitos, CA, USA.

[21] Arnold, D., Agrawal, S., Blackford, S., Dongarra, J., Miller, M., Seymour, K., Sagi, K., Shi, Z., and Vadhiyar, S. (2002). Users' Guide to NetSolve V1.4.1. Innovative Computing Dept. Technical Report ICL-UT-02-05, University of Tennessee, Knoxville, TN.

[22] Avery, P. and Foster, I. (2001). The GriPhyN Project: Towards Petascale Virtual-Data Grids. Technical Report GriPhyN 2001-14, The GriPhyN Collaboration.

[23] Baker, M., Buyya, R., and Laforenza, D. (2002). Grids and Grid Technologies for Wide-Area Distributed Computing. *Software: Practice and Experience*, 32(15):1437–1466. Wiley Press, USA.

[24] Balas, E. and Padberg, M. W. (1972). On the Set-Covering Problem. *Operations Research*, 20(6):1152–1161.

[25] Barmouta, A. and Buyya, R. (2003). GridBank: A Grid Accounting Services Architecture (GASA) for Distributed Systems Sharing and Integration. In *Proceedings of Workshop on Internet Computing and E-Commerce : 17th Annual International Parallel and Distributed Processing Symposium (IPDPS 2003)*, Nice, France. IEEE CS Press, Los Alamitos, CA, USA.

[26] Baru, C., Moore, R., Rajasekar, A., and Wan, M. (1998). The SDSC Storage Resource Broker. In *Proceedings of CASCON'98*, Toronto, Canada. IBM Press.

[27] Bassi, A., Beck, M., Fagg, G., Moore, T., Plank, J., Swany, M., and Wolski, R. (2002). The Internet Backplane Protocol: A Study in Resource Sharing. In *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID 2002)*, Berlin, Germany. IEEE CS Press, Los Alamitos, CA, USA.

[28] Bayucan, A., Henderson, R. L., Lesiak, C., Mann, B., Proett, T., and Tweten, D. (1999). Portable Batch System: External reference specification. Technical report, MRJ Technology Solutions.

[29] Bell, W. H., Cameron, D. G., Capozza, L., Millar, A. P., Stockinger, K., and Zini, F. (2002). Simulation of Dynamic Grid Replication Strategies in OptorSim. In *Proceedings of the 3rd International Workshop on Grid Computing(GRID 02)*, pages 46–57, Baltimore,MD,USA. Springer-Verlag, Berlin, Germany.

[30] Bell, W. H., Cameron, D. G., Carvajal-Schiaffino, R., Millar, A. P., Stockinger, K., and Zini, F. (2003). Evaluation of an Economy-Based File Replication Strategy for a Data Grid. In *Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2003)*, Tokyo, Japan. IEEE CS Press, Los Alamitos, CA, USA.

[31] Berman, F. et al. (2001). The grads project: Software support for high-level grid application development. *Int. J. High Perform. Comput. Appl.*, 15(4):327–344.

[32] Berman, F. and Wolski, R. (1997). The AppLeS Project: A Status Report. In *Proceedings of the 8th NEC Research Symposium*, Berlin, Germany.

[33] Bester, J., Foster, I., Kesselman, C., Tedesco, J., and Tuecke, S. (1999). GASS: A Data Movement and Access Service for Wide Area Computing Systems. In *Proceedings of the 6th Workshop on I/O in Parallel and Distributed Systems*, Atlanta, USA. ACM Press, New York, NY, USA.

[34] BioGrid Project, Japan (2005). `http://www.biogrid.jp/`.

[35] Biomedical Informatics Research Network (BIRN) (2005). `http://www.nbirn.net`.

[36] Björkander, M. and Kobryn, C. (2003). Architecting Systems with UML 2.0. *IEEE Software*, 20(4):57–61.

[37] Brady, M., Gavaghan, D., Simpson, A., Parada, M. M., and Highnam, R. (2003). *Grid Computing: Making the Global Infrastructure a Reality*, chapter eDiamond: A Grid-Enabled Federated Database of Annotated Mammograms, pages 923–943. Wiley Press, London, UK.

[38] Braun, T. D., Siegel, H. J., Beck, N., Bölöni, L. L., Maheswaran, M., Reuther, A. I., Robertson, J. P., Theys, M. D., Yao, B., Hensgen, D., and Freund, R. F. (2001). A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing*, 61(6):810–837.

[39] Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., and Yergeau, F. (2004). Extensible Markup Language (XML) 1.0 (3rd Edition). W3C Recommendation.

[40] Breslau, L., Cao, P., Fan, L., Phillips, G., and Shenker, S. (1999). Web caching and zipf-like distributions: evidence and implications. In *Proceedings of the 18th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '99.)*, New York, NY, USA.

[41] Bunn, J. and Newman, H. (2003). *Grid Computing: Making the Global Infrastructure a Reality*, chapter Data Intensive Grids for High Energy Physics. Wiley Press, London, UK.

[42] Buyya, R. (2002). *Economic-based Distributed Resource Management and Scheduling for Grid Computing*. PhD thesis, Monash University, Australia.

[43] Buyya, R., Abramson, D., and Giddy, J. (2000a). Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid. In *Proceedings of the 4th International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia 2000)*, Beijing, China. IEEE Computer Society Press, USA.

[44] Buyya, R., Abramson, D., Giddy, J., and Stockinger, H. (2002). Economic models for resource management and scheduling in grid computing. *The Journal of Concurrency and Computation: Practice and Experience (CCPE)*, 14(13-15):1507–1542.

[45] Buyya, R., Giddy, J., and Abramson, D. (2000b). An Evaluation of Economy-based Resource Trading and Scheduling on Computational Power Grids for Parameter Sweep Applications. In *Proceedings of the 2nd Workshop on Active Middleware Services (AMS 2000)*, Pittsburgh, USA. IEEE CS Press , Los Alamitos, CA, USA.

[46] 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)*, 14(13-15):1175–1220.

[47] Buyya, R. and Vazhkudai, S. (2001). Compute Power Market: Towards a Market-Oriented Grid. In *Proceedings of the 1st International Symposium on Cluster Computing and the Grid (CCGRID '01)*, page 574, Brisbane, Australia. IEEE CS Press, Los Alamitos, CA, USA.

[48] Cameron, D. G., Carvajal-Schiaffino, R., Millar, A. P., Nicholson, C., Stockinger, K., and Zini, F. (2003). Evaluating Scheduling and Replica Optimisation Strategies in OptorSim. In *Proceedings of the 4th International Workshop on Grid Computing (Grid2003)*, Phoenix, AZ, USA. IEEE CS Press, Los Alamitos, CA, USA.

[49] Casanova, H. (2001). Simgrid: A Toolkit for the Simulation of Application Scheduling. In *Proceedings of the 1st International Symposium on Cluster Computing and the Grid (CCGRID '01)*, Brisbane, Australia. IEEE CS Press , Los Alamitos, CA, USA.

[50] Casanova, H. and Berman, F. (2003). *Grid Computing*, chapter Parameter Sweeps on the Grid with APST, pages 773–787. Wiley Press, London, UK.

[51] Casanova, H., Legrand, A., Zagorodnov, D., and Berman, F. (2000a). Heuristics for Scheduling Parameter Sweep Applications in Grid environments. In *Proceedings of the 9th Heterogeneous Computing Systems Workshop (HCW 2000)*, Cancun, Mexico. IEEE CS Press, Los Alamitos, CA, USA.

[52] Casanova, H., Obertelli, G., Berman, F., and Wolski, R. (2000b). The AppLeS parameter sweep template: User-level middleware for the grid. In *Proceedings of the 2000 ACM/IEEE conference on Supercomputing (SC'00)*, Dallas, TX, USA. IEEE Computer Society.

[53] Ceri, S. and Pelagatti, G. (1984). *Distributed databases : principles and systems*. McGraw-Hill, New York, USA.

[54] Chapin, S., Karpovich, J., and Grimshaw, A. (1999). The Legion resource management system. In *Proceedings of the 5th Workshop on Job Scheduling Strategies for Parallel Processing*, San Juan, Puerto Rico. IEEE CS Press, Los Alamitos, CA, USA.

[55] Chervenak, A., Deelman, E., Foster, I., Guy, L., Hoschek, W., Iamnitchi, A., Kesselman, C., Kunst, P., Ripeanu, M., Schwartzkopf, B., Stockinger, H., Stockinger, K., and Tierney, B. (2002). Giggle: A framework for constructing scalable replica location services. In *Proceedings of the 2002 IEEE/ACM Conference on Supercomputing (SC '02)*, Baltimore,USA.

[56] Chervenak, A., Foster, I., Kesselman, C., Salisbury, C., and Tuecke, S. (2000). The Data Grid: Towards an architecture for the distributed management and analysis of large scientific datasets. *Journal of Network and Computer Applications*, 23(3):187–200.

[57] Choon-Hoong, D., Nutanong, S., and Buyya, R. (2005). *Peer-to-Peer Computing: Evolution of a Disruptive Technology*, chapter Peer-to-Peer Networks for Content Sharing, pages 28–65. Idea Group Publishers, Hershey, PA, USA.

[58] Christofides, N. (1975). *Graph Theory: An Algorithmic Approach*, chapter Independent and Dominating Sets – The Set Covering Problem, pages 30 – 57. Academic Publishers, London, UK. ISBN 012 1743350 0.

[59] Clarke, I., Sandberg, O., Wiley, B., and Hong, T. W. (2001). Freenet: a distributed anonymous information storage and retrieval system. In *Proceedings of International Workshop on Design Issues in Anonymity and Unobservability*, Berkeley, CA, USA. Springer-Verlag, Berlin, Germany.

[60] Cooke, A. et al. (2003). R-GMA: An Information Integration System for Grid Monitoring. *Lecture Notes in Computer Science*, 2888:462 – 481.

[61] Cooper, K., Dasgupata, A., Kennedy, K., Koelbel, C., Mandal, A., Marin, G., Mazina, M., Mellor-Crummey, J., Berman, F., Casanova, H., Chien, A., Dail, H., Liu, X., Olugbile, A., Sievert, O., Xia, H., Johnsson, L., Liu, B., Patel, M., Reed, D., Deng, W., Mendes, C., Shi, Z., YarKhan, A., and Dongarra, J. (2004). New Grid Scheduling and Rescheduling Methods in the GrADS Project. In *Proceedings of NSF Next Generation Software Workshop:International Parallel and Distributed Processing Symposium*, Santa Fe, NM, USA. IEEE CS Press, Los Alamitos, CA, USA.

[62] Cormen, T. H., Stein, C., Rivest, R. L., and Leiserson, C. E. (2001). *Introduction to Algorithms*. McGraw-Hill Higher Education.

[63] Czajkowski, K., Foster, I., and Kesselman, C. (1999). Resource co-allocation in computational grids. In *Proceedings of the Eighth IEEE International Symposium on High Performance Distributed Computing (HPDC '99)*, Redondo Beach, CA, USA. IEEE CS Press , Los Alamitos, CA, USA.

[64] Czajkowski, K., Foster, I. T., Karonis, N. T., Kesselman, C., Martin, S., Smith, W., and Tuecke, S. (1998). A Resource Management Architecture for Metacomputing Systems. In *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing(IPPS/SPDP '98)*, Orlando, Florida, USA. Springer-Verlag, Berlin, Germany.

[65] Czajkowski, K., Kesselman, C., Fitzgerald, S., and Foster, I. (2001). Grid information services for distributed resource sharing. In *Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10)*, San Francisco, CA. IEEE CS Press, Los Alamitos, CA, USA.

[66] Dail, H., Casanova, H., and Berman, F. (2002). A Decoupled Scheduling Approach for the GrADS Environment. In *Proceedings of the 2002 IEEE/ACM Conference on Supercomputing (SC'02)*, Baltimore, USA. IEEE CS Press, Los Alamitos, CA, USA.

[67] Dail, H., Sievert, O., Berman, F., Casanova, H., YarKhan, A., Vadhiyar, S., Dongarra, J., Liu, C., Yang, L., Angulo, D., and Foster, I. (2004). *Grid resource management: state of the art and future trends*, chapter Scheduling in the Grid application development software project, pages 73–98. Kluwer Academic Publishers, Cambridge, MA, USA.

[68] Davison, B. D. (2001). A web caching primer. *IEEE Internet Computing*, 5(4):38–45.

[69] Deelman, E., Blythe, J., Gil, Y., and Kesselman, C. (2003). *Grid Resource Management: State of the Art and Future Trends*, chapter Workflow Management in GriPhyN, pages 99–117. Kluwer Academic Publishers, Cambridge, MA, USA.

[70] Dias de Assuncao, M., Nadiminti, K., Venugopal, S., Ma, T., and Buyya, R. (2005). An integration of global and enterprise grid computing: Gridbus broker and xgrid perspective. In *Proceedings of the 4th International Conference on Grid and Cooperative Computing (GCC 2005)*, LNCS, Beijing, China. Springer-Verlag, Berlin, Germany.

[71] Dilley, J., Maggs, B., Parikh, J., Prokop, H., Sitaraman, R., and Weihl, B. (2002). Globally distributed content delivery. *IEEE Internet Computing*, 6(5):50– 58.

[72] Dullmann, D., Hoschek, W., Jaen-Martinez, J., Segal, B., Samar, A., Stockinger, H., and Stockinger, K. (2001). Models for Replica Synchronisation and Consistency in a Data Grid. In *Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10)*, San Francisco, CA. IEEE CS Press, Los Alamitos, CA, USA.

[73] Dumitrescu, C. and Foster, I. (2004). Usage Policy-Based CPU Sharing in Virtual Organizations. In *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing (GRID'04)*, Pittsburgh, PA, USA. IEEE CS Press, Los Alamitos, CA, USA.

[74] Dumitrescu, C., Raicu, I., and Foster, I. (2005). Di-gruber: A distributed approach to grid resource brokering. In *Proceedings of the 2005 ACM/IEEE conference on Supercomputing (SC'05)*, page 38, Seattle, WA, USA. IEEE CS Press,Los Alamitos, CA, USA.

[75] Ellert, M., Konstantinov, A., Konya, B., Smirnova, O., and Waananen, A. (2002). Performance Evaluation of GridFTP within the NorduGrid Project. Technical Report cs.DC/0205023, NorduGrid Project.

[76] Enabling Grids for E-SciencE (EGEE) (2005). `http://public.eu-egee.org/`.

[77] Eriksson, H.-E. and Penker, M. (2000). *Business modeling with UML : business patterns at work*. Wiley Press, New York, USA.

[78] Erwin, D. W. and Snelling, D. F. (2001). UNICORE: A Grid Computing Environment. In *Proceedings of the 7th International Euro-Par Conference on Parallel Processing (Euro-Par '01)*, Manchester, UK. Springer-Verlag, Berlin, Germany.

[79] Ferrari, A., Knabe, F., Humphrey, M., Chapin, S. J., and Grimshaw, A. S. (1999). A Flexible Security System for Metacomputing Environments. In *Proceedings of the 7th International Conference on High-Performance Computing and Networking (HPCN '99)*, pages 370–380, Amsterdam, The Netherlands. Springer-Verlag, Berlin, Germany.

[80] Foster, I., Czajkowski, K., Ferguson, D., Frey, J., Graham, S., Maguire, T., Snelling, D., and Tuecke, S. (2005). Modeling and managing State in distributed systems: the role of OGSI and WSRF. *Proceedings of the IEEE*, 93(3):604– 612.

[81] Foster, I. and Iamnitchi, A. (2003). On death, taxes, and the convergence of peer-to-peer and grid computing. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS)*, volume 2735 of *Lecture Notes in Computer Science*, pages 118 – 128, Berkeley, CA, USA. Springer-Verlag, Berlin, Germany.

[82] Foster, I. and Karonis, N. (1998). A Grid-Enabled MPI: Message Passing in Heterogeneous Distributed Computing Systems. In *Proceedings of the IEEE/ACM SuperComputing Conference 1998 (SC'98)*, San Jose, CA, USA. IEEE CS Press, Los Alamitos, CA, USA.

[83] Foster, I. and Kesselman, C. (1998). The Globus Project: A Status Report. In *Proceedings of IPPS/SPDP'98 Heterogeneous Computing Workshop*, pages 4–18, Orlando, FL, USA. IEEE CS Press , Los Alamitos, CA, USA.

[84] Foster, I. and Kesselman, C. (1999a). *The Grid: Blueprint for a Future Computing Infrastructure*. Morgan Kaufmann Publishers, San Francisco, USA.

[85] Foster, I. and Kesselman, C. (1999b). *The Grid: Blueprint for a new computing infrastructure*, chapter The Globus toolkit, pages 259–278. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1 edition.

[86] Foster, I., Kesselman, C., Nick, J. M., and Tuecke, S. (2002). Grid services for distributed system integration. *Computer*, 35(6):37–46.

[87] Foster, I., Kesselman, C., Tsudik, G., and Tuecke, S. (1998). A security architecture for computational grids. In *Proc. 5th ACM Conference on Computer and Communications Security Conference*, San Francisco, CA, USA. ACM Press, New York, NY, USA.

[88] Foster, I., Kesselman, C., and Tuecke, S. (2001). The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of High Performance Computing Applications*, 15(3):200–222.

[89] Foster, I., Tuecke, S., and Unger, J. (2003). OGSA Data Services. Global Grid Forum 9.

[90] Fox, G. and Pallickara, S. (2002). The Narada Event Brokering System: Overview and Extensions. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA '02)*, pages 353–359, Las Vegas, USA. CSREA Press.

[91] Frey, J., Tannenbaum, T., Livny, M., Foster, I., and Tuecke, S. (2002). Condor-G: A Computation Management Agent for Multi-Institutional Grids. *Cluster Computing*, 5(3):237–246.

[92] Gagliardi, F., Jones, B., Reale, M., and Burke, S. (2002). European DataGrid Project: Experiences of Deploying a Large Scale Testbed for E-science Applications. In *Proceedings of Performance Evaluation of Complex Systems: Techniques and Tools, Performance 2002, Tutorial Lectures*, number 2459 in Lecture Notes in Computer Science, pages 480–500. Springer-Verlag, Berlin, Germany.

[93] Galbraith, J., Saarenmaa, O., Ylonen, T., and Lehtinen, S. (2005). SSH File Transfer Protocol (SFTP). Internet Draft. Valid upto September 2005.

[94] Gardner, R. et al. (2004). The Grid2003 Production Grid: Principles and Practice. In *Proceedings of the 13th Symposium on High Performance Distributed Computing (HPDC 13)*, Honolulu, HI, USA. IEEE CS Press, Los Alamitos, CA, USA.

[95] Gentzsch, W. (2001). Sun Grid Engine: Towards Creating a Compute Power Grid. In *Proceedings of the 1st International Symposium on Cluster Computing and the Grid (CCGRID '01)*, Brisbane, Australia. IEEE CS Press , Los Alamitos, CA, USA.

[96] Gibbins, H., Nadiminti, K., Beeson, B., Chhabra, R. K., Smith, B., and Buyya, R. (2005). The Australian BioGrid Portal: Empowering the Molecular Docking Research Community. In *Proceedings of the 3rd APAC Conference and Exhibition on Advanced Computing, Grid Applications and eResearch (APAC 2005)*, Gold Coast, Australia.

[97] Giersch, A., Robert, Y., and Vivien, F. (2004). Scheduling tasks sharing files from distributed repositories. In *Proceedings of the 10th International Euro-Par Conference (EuroPar '04)*, Pisa, Italy. Springer-Verlag, Berlin, Germany.

[98] Gray, J., Helland, P., O'Neil, P., and Shasha, D. (1996). The dangers of replication and a solution. In *Proceedings of the 1996 ACM SIGMOD international conference on Management of data (SIGMOD '96)*, pages 173–182, Montreal, Quebec, Canada. ACM Press, New York, NY, USA.

[99] Gray, J. and Reuter, A. (1993). *Transaction processing : concepts and techniques*. Morgan Kaufmann Publishers, San Mateo, Calif.

[100] Hethmon, P. and Elz, R. (1998). RFC 2389: Feature negotiation mechanism for the File Transfer Protocol. Proposed Standard.

[101] Hey, T. and Trefethen, A. E. (2002). The UK e-Science Core Programme and the Grid. *Journal of Future Generation Computer Systems (FGCS)*, 18(8):1017–1031.

[102] Hockauf, R., Karl, W., Leberecht, M., Oberhuber, M., and Wagner, M. (1998). Exploiting Spatial and Temporal Locality of Accesses: A New Hardware-Based Monitoring Approach for DSM Systems. In *Proceedings of the 4th International Euro-Par Conference on Parallel Processing (Euro-Par '98)*, volume 1470 of *Lecture Notes in Computer Science*, pages 206 – 215, Southhampton, UK. Springer-Verlag, Berlin, Germany.

[103] Holliday, J., Agrawal, D., and Abbadi, A. E. (2000). Database replication using epidemic updates. Technical Report TRCS00-01, University of California at Santa Barbara.

[104] Holtman, K. et al. (2001). CMS Requirements for the Grid. In *Proceedings of 2001 Conference on Computing in High Energy Physics (CHEP 2001)*, Beijing, China. Science Press.

[105] Horowitz, M. and Lunt, S. (1997). RFC 2228: FTP security extensions. Proposed Standard.

[106] Hoschek, W., Jaen-Martinez, F. J., Samar, A., Stockinger, H., and Stockinger, K. (2000). Data Management in an International Data Grid Project. In *Proceedings of the 1st IEEE/ACM International Workshop on Grid Computing (GRID '00)*, Bangalore, India. Springer-Verlag, Berlin, Germany.

[107] Housley, R., Polk, W., Ford, W., and Solo, D. (2002). RFC 3280: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List Profile. STANDARD.

[108] Huffman, B. T., McNulty, R., Shears, T., Denis, R. S., and Waters, D. (2002). The CDF/D0 UK GridPP Project. `http://www.gridpp.ac.uk/datamanagement/metadata/SubGroups/UseCases/docs/cdf5858.ps.gz`. CDF Internal Note.

[109] Hui, T. and Tham, C. (2003). Reinforcement learning-based dynamic bandwidth provisioning for quality of service in differentiated services networks. In *Proceedings of the 2003 IEEE International Conference on Networks (ICON 2003)*, Sydney, Australia.

[110] In, J.-U., Arbree, A., Avery, P., Cavanaugh, R., Katageri, S., and Ranka, S. (2003). Sphinx: A Scheduling Middleware for Data Intensive Applications on a Grid. Technical Report GriPhyN 2003-17, GriPhyn (Grid Physics Network).

[111] In, J.-U., Avery, P., Cavanaugh, R., and Ranka, S. (2004). Policy based scheduling for simple quality of service in grid computing. In *Proceedings of the 18th International Parallel and Distributed Processing Symposium 2004 (IPDPS '04).*, Santa Fe, NM, USA. IEEE CS Press, Los Alamitos, CA, USA.

[112] Jabber Project (2005). Jabber Protocols. Available at `http://www.jabber.org/protocol/`.

[113] Jain, R., Somalwar, K., Werth, J., and Browne, J. C. (1997). Heuristics for Scheduling I/O Operations. *IEEE Transactions on Parallel and Distributed Systems*, 8(3):310–320.

[114] Karlsson, M. and Mahalingam, M. (2002). Do we need replica placement algorithms in content delivery networks? In *Proceedings of the 2002 Web Content Caching and Distribution Conference (WCW '02)*, Boulder, Colorado. http://www.iwcw.org/.

[115] Khanna, G., Vydyanathan, N., Kurc, T., Catalyurek, U., Wyckoff, P., Saltz, J., and Sadayappan, P. (2005). A hypergraph partitioning-based approach for scheduling of tasks with batch-shared I/O. In *Proceedings of the 2005 IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2005)*, Cardiff, UK. IEEE CS Press.

[116] Kim, S. and Weissman, J. (2003). A GA-based Approach for Scheduling Decomposable Data Grid Applications. In *Proceedings of the 2004 International Conference on Parallel Processing (ICPP 04)*, Montreal, Canada. IEEE CS Press, Los Alamitos, CA, USA.

[117] Kolbe, S., Ma, T., Liu, W., Soh, W. S., Buyya, R., and Egan, G. (2005). A Platform for Distributed Analysis of Neuroimaging Data on Global Grids. In *Proceedings of the 1st IEEE International Conference on e-Science and Grid Computing (e-Science 2005)*, Melbourne, Australia. IEEE CS Press , Los Alamitos, CA, USA.

[118] Kosar, T. and Livny, M. (2004). Stork: Making data placement a first class citizen in the grid. In *Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04)*, Tokyo, Japan. IEEE CS Press, Los Alamitos, CA, USA.

[119] Kossmann, D. (2000). The state of the art in distributed query processing. *ACM Computing Surveys*, 32(4):422–469.

[120] Koutrika, G. (2005). Heterogeneity in digital libraries: Two sides of the same coin. DELOS Newsletter.

[121] Kouzes, R. T., Myers, J. D., and Wulf, W. A. (1996). Collaboratories: Doing science on the internet. *IEEE Computer*, 29(8):40–46.

[122] Kramer, D. and MacInnis, M. (2004). Utilization of a Local Grid of Mac OS X-Based Computers using Xgrid. In *Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing (HPDC '04)*, Honolulu, HI, USA. IEEE CS Press , Los Alamitos, CA, USA.

[123] 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)*, 32(2):135–164.

[124] Krishnamurthy, B., Wills, C., and Zhang, Y. (2001). On the use and performance of content distribution networks. In *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement (IMW '01)*, pages 169–182, San Francisco, CA, USA. ACM Press, New York, NY, USA.

[125] Kubiatowicz, J., Bindel, D., Chen, Y., Czerwinski, S., Eaton, P., Geels, D., Gummadi, R., Rhea, S., Weatherspoon, H., Wells, C., and Zhao, B. (2000). OceanStore: an architecture for global-scale persistent storage. In *Proceedings of the 9th international conference on Architectural support for programming languages and operating systems (ASPLOS-IX)*, pages 190–201, Cambridge, MA, USA. ACM Press, New York, NY, USA.

[126] Kwok, Y.-K. and Ahmad, I. (1996). Dynamic Critical-Path Scheduling: An Effective Technique for Allocating Task Graphs to Multiprocessors. *IEEE Trans. Parallel Distrib. Syst.*, 7(5):506–521.

[127] La Rosa, M., Moloney, G., and Winton, L. (2005). Towards belle monte carlo production on the apac national grid infrastructure. In *Proceedings of the 2005 APAC Conference and Exhibition on Advanced Computing, Grid Applications and eResearch (APAC 05)*, Gold Coast, QLD, Australia. Australian Partnership for Advanced Computing.

[128] Lamehamedi, H., Shentu, Z., Szymanski, B., and Deelman, E. (2003). Simulation of Dynamic Data Replication Strategies in Data Grids. In *Proceedings of the 17th International Symposium on Parallel and Distributed Processing (IPDPS '03)*, Nice, France. IEEE CS Press, Los Alamitos, CA, USA.

[129] Lamehamedi, H., Szymanski, B., Shentu, Z., and Deelman, E. (2002). Data replication strategies in grid environments. In *Proceedings of the 5th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP'02)*, Beijing, China. IEEE CS Press, Los Alamitos, CA, USA.

[130] Laser Interferometer Gravitational Wave Observatory (2005). `http://www.ligo.caltech.edu/`.

[131] Lebrun, P. (1999). The Large Hadron Collider, A Megascience Project. In *Proceedings of the 38th INFN Eloisatron Project Workshop on Superconducting Materials for High Energy Colliders*, Erice, Italy.

[132] Ledlie, J., Shneidman, J., Seltzer, M., and Huth, J. (2003). Scooped, again. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS 2003)*, volume 2735 of *Lecture Notes in Computer Science*, Berkeley, CA, USA,. Springer-Verlag, Berlin, Germany.

[133] Lee, B.-D. and Weissman, J. B. (2001). Dynamic Replica Management in the Service Grid. In *Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10')*, San Francisco, CA. IEEE CS Press,Los Alamitos, CA, USA.

[134] Lee, J., Gunter, D., Tierney, B., Allcock, B., Bester, J., Bresnahan, J., and Tuecke, S. (2001). Applied techniques for high bandwidth data transfers across wide area networks. In *Proceedings of International Conference on Computing in High Energy and Nuclear Physics*, Beijing, China.

[135] Legrand, I. C. and Newman, H. B. (2000). The MONARC toolset for simulating large network-distributed processing systems. In *Proceedings of the 32nd Winter Simulation Conference (WSC '00)*, Orlando, FL. Society for Computer Simulation International, San Diego, CA.

[136] LHC Computing Grid (2005). `http://lcg.web.cern.ch/LCG/`.

[137] Litzkow, M., Livny, M., and Mutka, M. W. (1988). Condor - a hunter of idle workstations. In *Proceedings of the 8th Int'l Conference of Distributed Computing Systems*, Los Alamitos, CA, USA. IEEE CS Press.

[138] Luther, A., Buyya, R., Ranjan, R., and Venugopal, S. (2005). *High Performance Computing: Paradigm and Infrastructure*, chapter Peer-to-Peer Grid Computing and a .NET-based Alchemi Framework. Wiley Press, USA.

[139] Magowan, J. (2003). A view on relational data on the Grid. In *Proceedings of the 17th International Symposium on Parallel and Distributed Processing(IPDPS '03)*, Nice, France. IEEE CS Press,Los Alamitos, CA, USA.

[140] Mahajan, R., Bellovin, S. M., Floyd, S., Ioannidis, J., Paxson, V., and Shenker, S. (2002). Controlling High Bandwidth Aggregates in the Network. *Computer Communications Review*, 32(3):62–73.

[141] Maheshwaran, M., Ali, S., Siegel, H. J., Hengsen, D., and Freund, R. F. (1999). Dynamic Matching and Scheduling of a Class of Independent Tasks onto Heterogeneous Computing Systems. In *8th Heterogeneous Computing Systems Workshop (HCW '99)*, San Juan, Puerto Rico.

[142] Maheswaran, M., Ali, S., Siegel, H. J., Hensgen, D., and Freund, R. F. (1999). Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems. *Journal of Parallel and Distributed Computing*, 59:107–131.

[143] Marshall, A. (1890). *The Principles of Economics*. History of Economic Thought Books. McMaster University.

[144] McKinley, K. S., Carr, S., and Tseng, C.-W. (1996). Improving data locality with loop transformations. *ACM Transactions on Programmming Languages and Systems*, 18(4):424–453.

[145] Milojicic, D. S., Kalogeraki, V., Lukose, R., Nagaraja, K., Pruyne, J., Richard, B., Rollins, S., and Xu, Z. (2002). Peer-to-peer computing. Technical Report HPL-2002-57, HP Labs, Palo Alto, CA, USA.

[146] Mohamed, H. and Epema, D. (2004). An evaluation of the close-to-files processor and data co-allocation policy in multiclusters. In *Proceedings of the 2004 IEEE International Conference on Cluster Computing*, San Diego, CA, USA. IEEE CS Press,Los Alamitos, CA, USA.

[147] Moloney, G. (2006). Data Transfer Requirements for High Energy Physics. Presentation at the 21st meeting of Asia-Pacific Advanced Network (APAN), Tokyo, Japan. `http://www.apan.net/meetings/tokyo2006/presentation/GlennMoloney_HEPDataTransfers_apan06.pdf`.

[148] Moore, R., Jagatheesan, A., Rajasekar, A., Wan, M., and Schroeder, W. (2004). Data Grid Management Systems. In *Proceedings of the 12th NASA Goddard, 21st IEEE Conference on Mass Storage Systems and Technologies*, College Park, MD, USA. IEEE CS Press, Los Alamitos, CA, USA.

[149] Moore, R., Prince, T. A., and Ellisman, M. (1998). Data-intensive computing and digital libraries. *Communications of the ACM*, 41(11):56–62.

[150] Moore, R., Rajasekar, A., and Wan, M. (2005). Data Grids, Digital Libraries and Persistent Archives: An Integrated Approach to Publishing, Sharing and Archiving Datas. *Proceedings of the IEEE (Special Issue on Grid Computing)*, 93(3).

[151] Nakada, H., Sato, M., and Sekiguchi, S. (1999). Design and implementations of Ninf: Towards a global computing infrastructure. *Future Generation Computing Systems*, 15(5-6):649–658.

[152] NCSA GridFTP Client (2005). `http://dims.ncsa.uiuc.edu/set/uberftp/`.

[153] Neuman, B. C. and Ts'o, T. (1994). Kerberos: An authentication service for computer networks. *IEEE Communications*, 32(9):33–38.

[154] Nielsen, N. R. (1970). The allocation of computer resources: Is pricing the answer? *Communications of the ACM*, 13(8):467–474.

[155] O'Callaghan, J. (2002). An Introduction to GrangeNet. *Telecommunication Journal of Australia*, 52(1):11–15.

[156] Oram, A. (2001). *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O'Reilly & Associates, Inc., Sebastopol,CA,USA.

[157] Ozsu, M. T. and Valduriez, P. (1999). *Principles of distributed database systems*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2nd edition.

[158] Parashar, M. and Hariri, S. (2004). Autonomic grid computing. In *Proceedings of the 2004 International Conference on Autonomic Computing (ICAC '04)*, New York, USA. IEEE CS Press, Los Alamitos, CA, USA. Tutorial.

[159] Park, K., Kim, G., and Crovella, M. (1996). On the relationship between file sizes, transport protocols, and self-similar network traffic. In *Proceedings of the 1996 International Conference on Network Protocols (ICNP '96)*, Atlanta, GA, USA. IEEE CS Press.

[160] Park, S.-M. and Kim, J.-H. (2003). Chameleon: A Resource Scheduler in a Data Grid Environment. In *Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2003)*, Tokyo, Japan. IEEE CS Press, Los Alamitos, CA, USA.

[161] Pearlman, L., Kesselman, C., Gullapalli, S., Spencer Jr., B., Futrelle, J., Kathleen, R., Foster, I., Hubbard, P., and Severance, C. (2004). Distributed hybrid earthquake engineering experiments: Experiences with a ground-shaking grid application. In *Proceedings of the 13th IEEE Symposium on High Performance Distributed Computing (HPDC-13)*, Honolulu, HI, USA. IEEE CS Press, Los Alamitos, CA, USA.

[162] Phan, T., Ranganathan, K., and Sion, R. (2005). Evolving toward the perfect schedule: Co-scheduling job assignments and data replication in wide-area systems using a genetic algorithm. In *Proceedings of the 11th Workshop on Job Scheduling Strategies for Parallel Processing*, Cambridge, MA. Springer-Verlag, Berlin, Germany.

[163] Pitoura, E. and Bhargava, B. (1999). Data consistency in intermittently connected distributed systems. *IEEE Transactions on Knowledge and Data Engineering*, 11(6):896–915.

[164] Plank, J., Beck, M., Elwasif, W. R., Moore, T., Swany, M., and Wolski, R. (1999). The Internet Backplane Protocol: Storage in the Network. In *Proceedings of the 1999 Network Storage Symposium (NetStore99)*, Seattle, WA, USA. University of Tennessee, Knoxville. `http://loci.cs.utk.edu/dsi/netstore99/`.

[165] Plank, J. S., Moore, T., and Beck, M. (2002). Scalable Sharing of Wide Area Storage Resource. Technical Report CS-02-475, University of Tennessee, Knoxville, TN, USA.

[166] Polychronopoulos, C. D. and Kuck, D. J. (1987). Guided self-scheduling: A practical scheduling scheme for parallel supercomputers. *IEEE Transactions on Computers*, 36(12):1425–1439.

[167] Postel, J. and Reynolds, J. K. (1985). RFC 959: File transfer protocol. STANDARD.

[168] Rajasekar, A., Moore, R., Ludascher, B., and Zaslavsky, I. (2002). The GRID Adventures: SDSC'S Storage Resource Broker and Web Services in Digital Library Applications. In *Proceedings of the 4th All-Russian Scientific Conference (RCDL'02) Digital Libraries: Advanced Methods and Technologies, Digital Collections*.

[169] Rajasekar, A., Wan, M., and Moore, R. (2002). MySRB & SRB: Components of a Data Grid. In *Proceedings of the 11 th IEEE International Symposium on High Performance Distributed Computing (HPDC-11)*, Edinburgh, UK. IEEE CS Press,Los Alamitos, CA, USA.

[170] Rajasekar, A., Wan, M., Moore, R., Kremenek, G., and Guptil, T. (2003). Data Grids, Collections, and Grid Bricks. In *Proceedings of the 20 th IEEE/11 th NASA Goddard Conference on Mass Storage Systems and Technologies (MSS'03)*, San Diego, CA, USA. IEEE CS Press, Los Alamitos, CA, USA.

[171] Rajasekar, A., Wan, M., Moore, R., and Schroeder, W. (2004). Data Grid Federation. In *Proceedings of the 11th International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2004)*, Las Vegas, USA. CSREA Press.

[172] Raman, R., Livny, M., and Solomon, M. (1999). Matchmaking: An extensible framework for distributed resource management. *Cluster Computing*, 2(2):129–138.

[173] Ranganathan, K. and Foster, I. (2002). Decoupling Computation and Data Scheduling in Distributed Data-Intensive Applications. In *Proceedings of the 11th IEEE Symposium on High Performance Distributed Computing (HPDC)*, Edinburgh, UK. IEEE CS Press, Los Alamitos, CA, USA.

[174] Ranganathan, K. and Foster, I. (2003). Simulation studies of computation and data scheduling algorithms for data grids. *Journal of Grid Computing*, 1(1):53–62.

[175] Ranganathan, K., Iamnitchi, A., and Foster, I. (2002). Improving data availability through dynamic model-driven replication in large peer-to-peer communities. In *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'02)*, Berlin, Germany. IEEE CS Press, Los Alamitos, CA, USA.

[176] Ratnasamy, S., Francis, P., Handley, M., Karp, R., and Schenker, S. (2001). A scalable content-addressable network. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*

(SIGCOMM '01), pages 161–172, San Diego, CA, USA. ACM Press, New York, NY, USA.

[177] Rowstron, A. I. T. and Druschel, P. (2001). Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)*, pages 329–350, Heidelberg, Germany. Springer-Verlag, London, UK.

[178] Sacerdoti, F., Katz, M., Massie, M., and Culler, D. (2003). Wide area cluster monitoring with Ganglia. In *Proceedings of the 2003 IEEE International Conference on Cluster Computing, 2003 (Cluster '03)*, Hong Kong. IEEE CS Press , Los Alamitos, CA, USA.

[179] Salem, K. and Garcia-Molina, H. (1986). Disk striping. In *Proceedings of the Second International Conference on Data Engineering (ICDE-86)*, Los Angeles, USA. IEEE CS Press , Los Alamitos, CA, USA.

[180] Samar, A. and Stockinger, H. (2001). Grid Data Management Pilot (GDMP): A Tool for Wide Area Replication. In *Proceedings of the IASTED International Conference on Applied Informatics (AI2001)*, Innsbruck, Austria. ACTA Press, Calgary, Canada.

[181] Sandhu, R. S., Coyne, E. J., Feinstein, H. L., and Youman, C. E. (1996). Role-based access control models. *Computer*, 29(2):38–47.

[182] Saroiu, S., Gummadi, K. P., Dunn, R. J., Gribble, S. D., and Levy, H. M. (2002). An analysis of internet content delivery systems. *SIGOPS Operating Systems Review*, 36:315–327.

[183] Seidel, E., Allen, G., Merzky, A., and Nabrzyski, J. (2002). GridLab: a grid application toolkit and testbed. *Future Gener. Comput. Syst.*, 18(8):1143–1153.

[184] Shatdal, A., Kant, C., and Naughton, J. F. (1994). Cache conscious algorithms for relational query processing. In *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB '94)*, pages 510–521, Santiago, Chile. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[185] Sheth, A. P. and Larson, J. A. (1990). Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 22(3):183–236.

[186] Shoshani, A., Sim, A., and Gu, J. (2002). Storage Resource Managers: Middleware Components for Grid Storage. In *Proceedings of the Nineteenth IEEE Symposium on Mass Storage Systems (MSS '02)*.

[187] Sloan Digital Sky Survey (2005). `http://www.sdss.org/`.

[188] Soh, H., Haque, S., Liao, W., Nadiminti, K., and Buyya, R. (2005). GTPE: A thread programming environment for the grid. In *Proceedings of the 13th International Conference on Advanced Computing and Communications (ADCOM 2005)*, Coimbatore, India.

[189] Stockinger, H., Samar, A., Allcock, B., Foster, I., Holtman, K., and Tierney, B. (2001). File and object replication in data grids. In *Proceedings of the 10th IEEE Symposium on High Performance and Distributed Computing (HPDC-10)*, San Francisco, USA. IEEE CS Press, Los Alamitos, CA, USA.

[190] Stoica, I., Morris, R., Liben-Nowell, D., Karger, D. R., Kaashoek, M. F., Dabek, F., and Balakrishnan, H. (2003). Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on Networking*, 11(1):17–32.

[191] 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 Mariposa. In *Proceedings of 3rd International Conference on Parallel and Distributed Information Systems*, Austin, TX, USA. IEEE CS Press, Los Alamitos, CA, USA.

[192] Sulistio, A., Cibej, U., Robic, B., and Buyya, R. (2005). A Tool for Modelling and Simulation of Data Grids with Integration of Data Storage, Replication and Analysis. Technical Report GRIDS-TR-2005-13, University of Melbourne, Australia.

[193] Sulistio, A., Poduval, G., Buyya, R., and Tham, C.-K. (2006). On Incorporating Differentiated Network Service into GridSim. Technical Report GRIDS-TR-2006-5, The University of Melbourne, Australia.

[194] Sulistio, A., Yeo, C. S., and Buyya, R. (2004). A taxonomy of computer-based simulations and its mapping to parallel and distributed systems simulation tools. *Software: Practice and Experience (SPE)*, 34(7):653–673.

[195] Sun Microsystems Inc. (2006). Sun Grid Compute Utility. `http://www.network.com`. Accessed May 2006.

[196] Szalay, A. and Gray, J. (2001). The World-Wide Telescope. *Science*, 293(5537):2037–2040.

[197] Szalay, A. S., editor (2002). *Proceedings of SPIE Conference on Virtual Observatories*, volume 4846, Waikoloa, HI, USA. SPIE.

[198] Takefusa, A., Tatebe, O., Matsuoka, S., and Morita, Y. (2003). Performance Analysis of Scheduling and Replication Algorithms on Grid Datafarm Architecture for High-Energy Physics Applications. In *Proceedings of the 12th IEEE international Symposium on High Performance Distributed Computing (HPDC-12)*, Seattle, USA. IEEE CS Press, Los Alamitos, CA, USA.

[199] Tatebe, O., Morita, Y., Matsuoka, S., Soda, N., and Sekiguchi, S. (2002). Grid Datafarm Architecture for Petascale Data Intensive Computing. In *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2002)*, Berlin, Germany. IEEE CS Press, Los Alamitos, CA, USA.

[200] Tatebe, O., Ogawa, H., Kodama, Y., Kudoh, T., Sekiguchi, S., Matsuoka, S., Aida, K., Boku, T., Sato, M., Morita, Y., Kitatsuji, Y., Williams, J., and Hicks, J. (2004a). The Second Trans-Pacific Grid Datafarm Testbed and Experiments for SC2003. In *Proceedings of 2004 International Symposium on Applications and the Internet - Workshops (SAINT 2004 Workshops)*, Tokyo, Japan. IEEE CS Press, Los Alamitos, CA, USA.

[201] Tatebe, O., Soda, N., Morita, Y., Matsuoka, S., and Sekiguchi, S. (2004b). Gfarm v2: A Grid file system that supports high-performance distributed and parallel data computing. In *Proceedings of the 2004 Computing in High Energy and Nuclear Physics (CHEP04) Conference*, Interlaken, Switzerland.

[202] Thain, D., Basney, J., Son, S.-C., and Livny, M. (2001a). The Kangaroo Approach to Data Movement on the Grid. In *Proc. of the 10th IEEE Symposium on High Performance Distributed Computing (HPDC10)*, San Francisco, CA. IEEE CS Press, Los Alamitos, CA, USA.

[203] Thain, D., Bent, J., Arpaci-Dusseau, A., Arpaci-Dusseau, R., and Livny, M. (2001b). Gathering at the well: Creating communities for grid I/O. In *Proceedings of Supercomputing 2001*, Denver, Colorado. IEEE CS Press, Los Alamitos, CA, USA.

[204] Thakur, R., Choudhary, A., Bordawekar, R., More, S., and Kuditipudi, S. (1996). Passion: Optimized I/O for Parallel Applications. *Computer*, 29(6):70–78.

[205] The Belle Analysis Data Grid (BADG) Project (2006). `http://epp.ph.unimelb.edu.au/epp/grid/badg/about.php3`.

[206] Thomas, R. K. and Sandhu, R. K. (1997). Task-Based Authorization Controls (TBAC): A Family of Models for Active and Enterprise-Oriented Authorization Management. In *Proceedings of the IFIP TC11 WG11.3 11th International Conference on Database Securty XI*, pages 166–181, Lake Tahoe, CA, USA. Chapman & Hall, Ltd., London, UK.

[207] Transaction Management Research Group (GGF) (2005). `http://www.data-grid.org/tm-rg-charter.html`.

[208] Vazhkudai, S., Tuecke, S., and Foster, I. (2001). Replica Selection in the Globus Data Grid. In *Proceedings of the 1st IEEE/ACM International Conference on Cluster Computing and the Grid (CCGRID 2001)*, Brisbane, Australia.

[209] Vickrey, W. (1961). Counter-speculation, auctions, and competitive sealed tenders. *Journal of Finance*, 16(1):9 – 37.

[210] von Laszewski, G., Foster, I., Gawor, J., and Lane, P. (2001). A Java commodity Grid kit. *Concurrency and Computation-Practice and Experience*, 13(8-9):645–662.

[211] Vraalsen, F., Aydt, R., Mendes, C., and Reed, D. (2001). Performance Contracts: Predicting and Monitoring Grid Application Behavior. In *Proceedings of the 2nd International Workshop on Grid Computing (GRID 2001)*, volume 2242 of *Lecture Notes in Computer Science*, Denver, CO. Springer-Verlag, Berlin, Germany.

[212] Wagner, D. and Schneier, B. (1996). Analysis of the SSL 3.0 Protocol. In *Proceedings of the 2nd USENIX Workshop on Electronic Commerce*, Berkeley, CA, USA. USENIX Press.

[213] Waldspurger, C. A., Hogg, T., Huberman, B. A., Kephart, J. O., and Stornetta, W. S. (1992). Spawn: A Distributed Computational Economy. *IEEE Transactions on Software Engineering*, 18(2):103–117.

[214] Wasson, G. and Humphrey, M. (2003). Policy and enforcement in virtual organizations. In *Proceedings of the 4th International Workshop on Grid Computing*, Phoenix, AZ, USA. IEEE CS Press,Los Alamitos, CA, USA.

[215] White, B. S., Grimshaw, A. S., and Nguyen-Tuong, A. (2000). Grid-Based File Access: The Legion I/O Model. In *Proceedings of the 9th IEEE International Symposium on High Performance Distributed Computing (HPDC'00)*, Pittsburgh, USA. IEEE CS Press,Los Alamitos, CA, USA.

[216] Winton, L. (2003). Data grids and high energy physics - A Melbourne perspective. *Space Science Reviews*, 107(1–2):523–540.

[217] Wolski, R., Spring, N., and Hayes, J. (1999). The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. *Journal of Future Generation Computing Systems*, 15:757–768.

[218] Yamamoto, N., Tatebe, O., and Sekiguchi, S. (2004). Parallel and Distributed Astronomical Data Analysis on Grid Datafarm. In *Proceedings of 5th IEEE/ACM International Workshop on Grid Computing (Grid 2004)*, Pittsburgh, USA. IEEE CS Press,Los Alamitos, CA, USA.

[219] Yang, Y. and Casanova, H. (2003). UMR: A Multi-Round Algorithm for Scheduling Divisible Workloads. In *Proceedings of the 17th International Symposium on Parallel and Distributed Processing (IPDPS '03)*, Nice, France. IEEE CS Press , Los Alamitos, CA, USA.

[220] Yu, J. and Buyya, R. (2005). A Taxonomy of Workflow Management Systems for Grid Computing. *Journal of Grid Computing*, 3(3-4):171–200.

[221] Yu, J., Venugopal, S., and Buyya, R. (2006). A market-oriented grid directory service for publication and discovery of grid service providers and their services. *Journal of Supercomputing*, 16(1).

[222] Zhao, B. Y., Kubiatowicz, J. D., and Joseph, A. D. (2001). Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report CSD-01-1141, University of California at Berkeley, USA.