A Taxonomy of Data Grids for Distributed Data Sharing, Management, and Processing

SRIKUMAR VENUGOPAL, RAJKUMAR BUYYA, AND
KOTAGIRI RAMAMOHANARAO

University of Melbourne, Australia

Data Grids have been adopted as the next generation platform by many scientific communities that need to share, access, transport, process, and manage large data collections distributed worldwide. They combine high-end computing technologies with high-performance networking and wide-area storage management techniques. In this article, we discuss the key concepts behind Data Grids and compare them with other data sharing and distribution paradigms such as content delivery networks, peer-to-peer networks, and distributed databases. We then provide comprehensive taxonomies that cover various aspects of architecture, data transportation, data replication and resource allocation, and scheduling. Finally, we map the proposed taxonomy to various Data Grid systems not only to validate the taxonomy but also to identify areas for future exploration.

Categories and Subject Descriptors: H.3.4 [**Information Storage and Retrieval**]: Systems and Software—*Distributed systems*; C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Client/server*; *distributed applications*; J.2 [**Physical Sciences and Engineering**]; J.3 [**Life and Medical Sciences**]

General Terms: Design, Management

Additional Key Words and Phrases: Grid computing, data-intensive applications, virtual organizations, replica management

1. INTRODUCTION

The next generation of scientific applications in domains as diverse as high energy physics, molecular modeling, and earth sciences involve the production of large datasets from simulations or from 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 have come to be known as part of

This work is partially supported through the Australian Research Council (ARC) Discovery Project grant and Storage Technology Corporation sponsorship of Grid Fellowship.

Authors' address: R. Buyya, Grid Computing and Distributed Sytems Laboratory, Department of Computer Science and Software Engineering, University of Melbourne, VIC 3010, Australia; email: rbuyya@unimelb.edu.au.

e-Science[1] [Hey and Trefethen 2002], a discipline that envisages using high-end computing, storage, networking, and Web technologies together to facilitate collaborative, data-intensive scientific research. However, this requires new paradigms in Internet computing that address issues such as multidomain applications, cooperation and coordination of resource owners, and blurring of system boundaries. Grid computing [Foster and Kesselman 1999] is one such paradigm that proposes aggregating geographically-distributed, heterogeneous computing, storage, and network resources to provide unified, secure, and pervasive access to their combined capabilities. Such aggregations are also called Grids.

Data Grids [Chervenak et al. 2000; Hoschek et al. 2000] 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. For users to derive maximum benefits out of the infrastructure, the following capabilities are needed: (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 as short a time as possible, (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. Content delivery networks, peer-to-peer file-sharing networks, and distributed databases are some of the other paradigms with similar requirements for supporting a distributed data-intensive infrastructure. In the next section, we provide a general overview and systematic characterization of Data Grids and a thorough examination of their differences from the distributed data-intensive mechanisms mentioned.

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 breaks down the overall research in Data Grids into specialized areas and categorizes each of them in turn. Section 4 then surveys some representative projects and publications and classifies them according to the taxonomy.

A few studies have investigated and surveyed Grid research in the recent past. Krauter et al. [2002] present a taxonomy of various Grid resource management systems that focuses on the general resource management architectures and scheduling policies. Specifically for Data Grids, Bunn and Newman [2003] provide an extensive survey of projects in High Energy Physics, while Qin and Jiang [2003] produce a compilation that concentrates more on the constituent technologies. Moore and Merzky [2002] identify functional requirements (features and capabilities) and components of a persistent archival system. In contrast to these articles, Finkelstein et al. [2004] spell out requirements for Data Grids from a software engineering perspective and elaborate on the impact that these have on architectural choices. A similar characterisation has been performed by Mattmann et al. [2005]. The work in this article, however, concentrates on issues pertaining to all data-intensive application environments including Data Grids. It provides a more detailed and complete understanding of Data Grids and its underlying technologies through multiple perspectives including resource allocation, data management, and user requirements.

The main objective of this article, therefore, is to delineate very clearly the uniqueness of Data Grids from other similar paradigms and provide a basis for categorizing present and future developments in this area. This article also aims to provide readers with

[1]Also known as e-Research with the inclusion of digital libraries and the humanities community.

an understanding of the essential concepts of this rapidly changing research area and help them identify important and outstanding issues for further investigation.

2. OVERVIEW

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 [Moore et al. 1998]. 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 allow them to 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 article and unless otherwise stated, the term resource means hardware such as computers or storage systems. Resources are also *nodes* in the network and hence, we use these terms 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, reliable data transfer mechanism, and a scalable replica discovery and management mechanism [Chervenak et al. 2000]. 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 the maintenance of 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 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 [Moore et al. 2005].

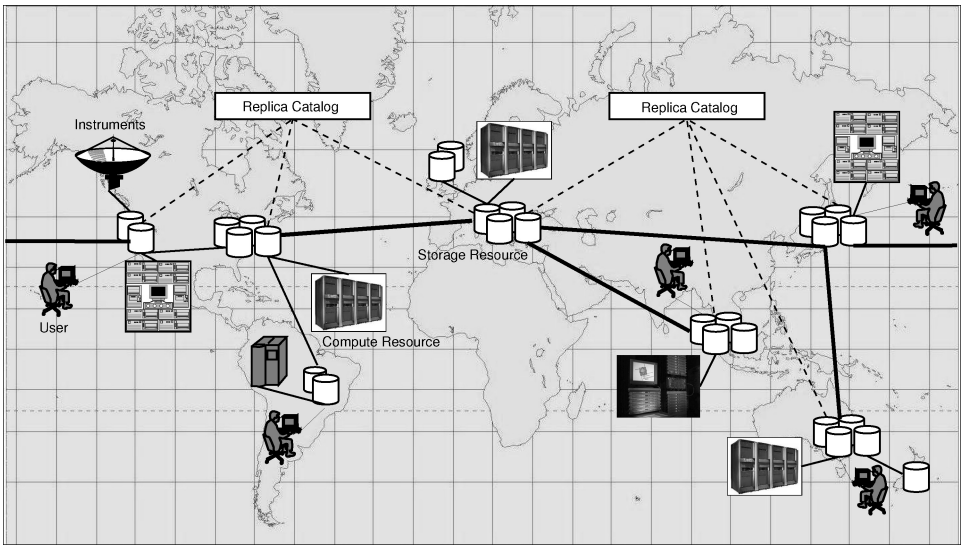


Fig. 1. A high-level view of a Data Grid.

Figure 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 centers, and the thinner lines are lower capacity networks that connect the latter to their subsidiary centers. 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 visualization 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 applications on remote data. It promotes a rich environment for users to analyze 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 analyzing 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) [Lebrun 1999] 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 analyzing 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 here.

—*Massive Datasets*. Data-intensive applications are characterized 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 [Holtman et al. 2001]. 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 characteriztics 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 et al. [2001] 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 organizations 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 can be used. Resources here are 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 organization and retrieval, and centralized scheduling mechanisms for resource management.

The existence of VOs impacts the design of Data Grid architectures in many ways. For example, a VO might be standalone or composed of a hierarchy of regional, national, and international VOs. In the latter case, the underlying Data Grid might have a corresponding hierarchy of repositories, and the replica discovery and management system 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. In subsequent sections, we will look at how Data Grids are differentiated by such design choices and how these affect underlying technologies.

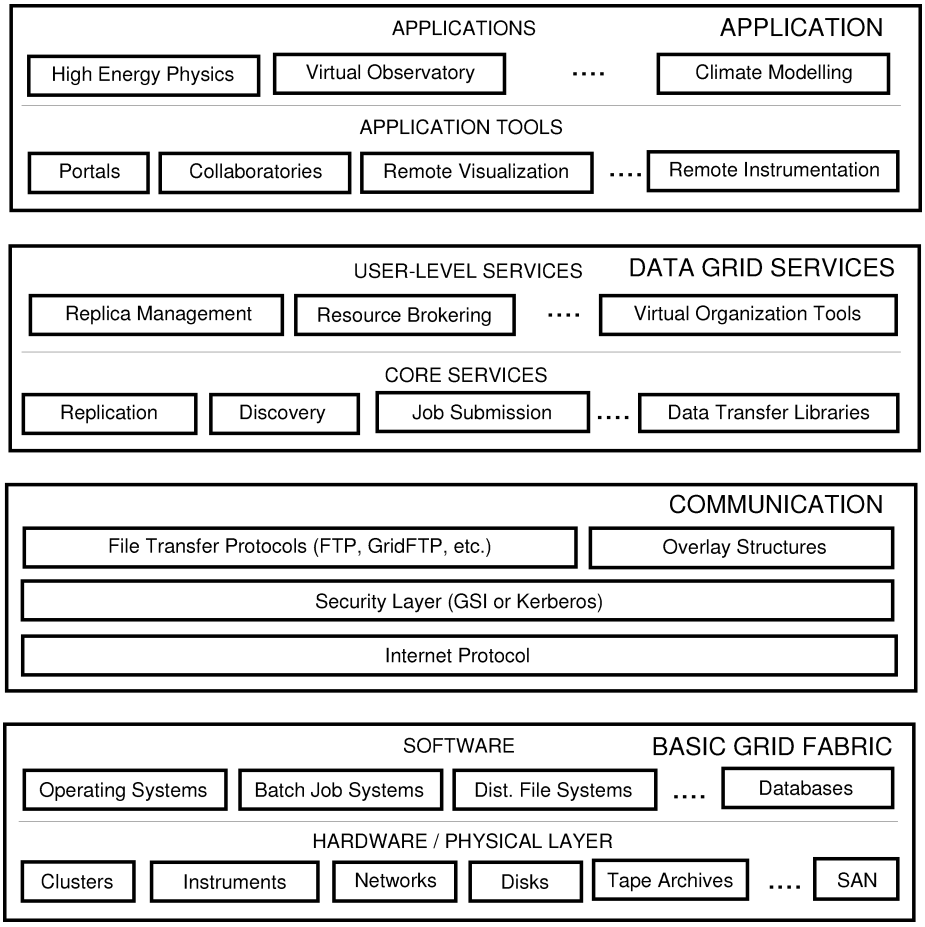


Fig. 2. A layered architecture.

2.3. Layered Architecture

The components of a Data Grid can be organized in a layered architecture as shown in Figure 2. This architecture follows from similar definitions given by Foster et al. [2001] and Baker et al. [2002]. Each layer builds on the services offered by the lower layer in addition to interacting and cooperating with components on the same level (e.g., Resource broker invoking VO tools). We can describe the layers from bottom to top as follows.

(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) [Foster et al. 1998]. 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* provide 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 (selection of resources for a user based on his requirements) and replica management provide mechanisms that allow for efficient resource management hidden behind inituitive commands and APIs (Application Programming Interfaces). VO tools provide an easy way to perform functions such as adding new resources, to a VO, querying the existing resources, and managing users' access rights.

(4) *Applications* are specific services that cater to users by invoking services provided by the layers below and customizing them to suit the target domains such as high-energy physics, biology and climate modeling. Each domain provides a familiar interface and access to services such as visualization. Portals are Web interfaces that provide single-point access to available VO services and domain-specific applications and tools. Collaboratories [Kouzes et al. 1996] 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 with 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 a 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) [Foster et al. 2002] that is based on the emerging *Web services* paradigm. Web services are self-contained, stateless components that use standard mechanisms for the representation and exchange of data. OGSA builds on Web service properties such as vendor and platform neutral service definitions using XML (eXtensible Markup Language) [Bray et al. 2004] 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 [Foster et al. 2003] 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 sets of operations and data attributes. This abstract view of data created by a data service is termed *data virtualization*. Subsequent efforts, through the Data Access and Integration Services Working Group (DAIS-WG) at GGF, have produced a set of more concrete standards [Antonioletti et al. 2005] for representing data through services. While the standards provide the consumers of the services the

advantage of being isolated from the inner workings of Data Grids, the actual work of transferring and managing data is done by the underlying or core mechanisms such as data transport, data replication, and resource management. The taxonomy section focuses on these core mechanisms as they define the capabilities of a Data Grid.

2.4. Related Data-Intensive Research Paradigms

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

2.4.1. Content Delivery Network. A Content Delivery Network (CDN) [Davison 2001; Dilley et al. 2002] consists of a "collection of (nonorigin) servers that attempt to offload work from origin servers by delivering content on their behalf" [Krishnamurthy et al. 2001]. 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 [Dilley et al. 2002]. 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. who have built dedicated infrastructures to serve multiple clients. However, CDNs haven't gained wide acceptance for data distribution because of the restricted model that they follow. Also, current CDN infrastructures are proprietary in nature and owned completely by the providers.

2.4.2. Peer-to-Peer Network. Peer-to-peer (P2P) networks [Oram 2001] are formed by ad hoc aggregation of resources to form a decentralized 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 centralized 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 [Anderson et al. 2002], Compute Power Market [Buyya and Vazhkudai 2001]), content and file sharing (Napster, Gnutella, Kazaa [Choon-Hoong et al. 2005]), and collaborative applications such as instant messengers (Jabber [Jabber Project 2005]). Milojicic et al. [2002] present a detailed taxonomy and survey of peer-to-peer systems. Here we are concerned mostly with 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 by demand for highly popular files. Currently, major P2P content sharing networks do not provide an integrated computation and data distribution environment.

2.4.3. Distributed Databases. A distributed database (DDB) [Ceri and Pelagatti 1984; Ozsu and Valduriez 1999] is a logically organized 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) [Sheth and Larson 1990]. 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 organizations which need to remove the need for a centralized computer center, to interconnect existing databases, to replicate databases to increase reliability, and to add new databases as new organizational units are added. This technology is very robust. It 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 envisioned for Data Grids as they have strong requirements for ACID (Atomicity, Consistency, Isolation, and Durability) properties [Gray and Reuter 1993] to ensure that the state of the database remains consistent and deterministic.

2.5. 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 the 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 based on the differing properties of the latter. These properties are summarised in Table I and are explained here.

Purpose. Considering the purpose of the network, it is generally agreed 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 for 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 supporting 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 the 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 organization. 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 the introduction or removal of resources and services.

Table I. 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
Organization	Centralized, two-level hierarchy, flat	Hierarchical	Centralized, federation	Hierarchical, federation, bottom up or 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, High-speed data movement, Optimal selection of data source and sink
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 Organization	Single Organization	VO
Security Requirements	Anonymity	Data Integrity	Authentication, Authorisation, Data Integrity	Authentication, Authorisation, Data Integrity

Organization. The organization 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 organization of P2P content sharing networks, and these are linked to the searching methods for files within the network. Within Napster, a peer has to connect to a centralized server and search for an available peer that has the required file. The two peers then communicate directly with each other. Gnutella avoids the centralized directory by having a peer broadcast its request to its neighbors 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 [Clarke et al. 2001] uses content-based hashing in which a file is assigned a hash based on its contents, and nearest neighbor search is used to identify the required document. Thus, three different models of organization, centralized, two-level hierarchy, and flat (structured and unstructured) can be seen in the examples presented. Distributed databases provide a relational database management interface and are, therefore, organized 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 organization present in a Data Grid, namely, 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 [Stoica et al. 2003], CAN [Ratnasamy et al. 2001], Pastry [Rowstron and Druschel 2001], and Tapestry [Zhao et al. 2001]. CDNs fetch data which has been requested by a browser through HTTP (Hyper Text Transfer Protocol). DDBs are organized 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 organized into catalog which map the logical description of data to the actual physical representation. One form of these catalog is the replica catalog 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 catalog 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 catalog offer another means for querying for data. In such cases, metadata has to be curated properly, otherwise it would affect the efficiency and accuracy of data discovery. We will look at the role of metadata and catalog in detail in later sections.

Latency Management and 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 prestaging the data before the application starts executing. Replication is different from caching in that 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 [Saroiu et al. 2002]. While several replication strategies have been suggested for a CDN, Karlsson and Mahalingam [2002] 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 [Kossmann 2000].

In Data Grids, all of the techniques mentioned are implemented in one form or another. However, additionally, Data Grids are differentiated by their requirement for the 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, the 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 the 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. [2001] which discusses a consistency service for replication in Data Grids. In P2P networks, Oceanstore [Kubiatowicz et al. 2000] 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 [Pitoura and Bhargava 1999], 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 synchronized. There are two methods for updating that are followed [Gray et al. 1996]: *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 [Transaction Management Research Group (GGF) 2005].

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 optimization 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), where turnaround 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 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 [Sheth and Larson 1990; Alonso and Barbara 1989]. *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 what share they will contribute to the network. CDNs are dedicated networks, and 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 [2005] 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 organized 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 categorizing the data.

It can be seen from the definitions of the data-intensive networks that the same classification 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 the 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. Without this interface, they would be required to manipulate the data by applying transformations or conducting analysis, to view its results, and use these results 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 from 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, we have already discussed the concept of VO. 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 organization. Likewise, DDBs are also maintained by single organizations 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 the 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 operations.

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 (authorization). 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 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 to guard against malicious operations on data while in transit. Also, more elaborate access controls than those required in Grids are needed for safeguarding confidential data in Data Grids.

Thus, it can be seen that though Data Grids share many characteriztics with other types of data-intensive network computing technologies, they are differentiated by heavy computational requirements, wider heterogeneity and autonomy, and the presence of VOs. Most of the current Data Grid implementations focus on scientific

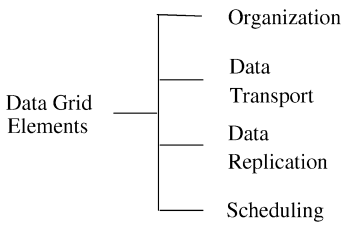


Fig. 3. Data Grid elements.

applications. Recent approaches have, however, explored the integration of the 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 [2003] 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. [2003] 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 [Fox and Pallickara 2002] have used P2P methods for delivering a scalable event service.

3. TAXONOMY

This section details a taxonomy that covers various aspects of Data Grids. As Data Grids consist of several elements, our taxonomy covers each one of them in depth. This taxonomy is split into four subtaxonomies as shown in Figure 3. The first subtaxonomy is from the point of view of Data Grid organization. This classifies ongoing scientific Data Grid efforts worldwide. The next subtaxonomy deals with the transport technologies used within Data Grids. This not only covers well-known file transfer protocols but 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 subtaxonomy presented next takes into account concerns of Grid environments such as metadata and the nature of data transfer mechanisms used. The last subtaxonomy 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 article, these are studied from the point of view of the specific requirements of Data Grid environments that have been provided in previous sections.

3.1. Data Grid Organization

Figure 4 shows a taxonomy based on the various organizational characteriztics of Data Grid projects. These characteriztics are central to any Data Grid and are manifesed in different ways in different systems.

Model. The model is the manner in which data sources are organized 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 5 and

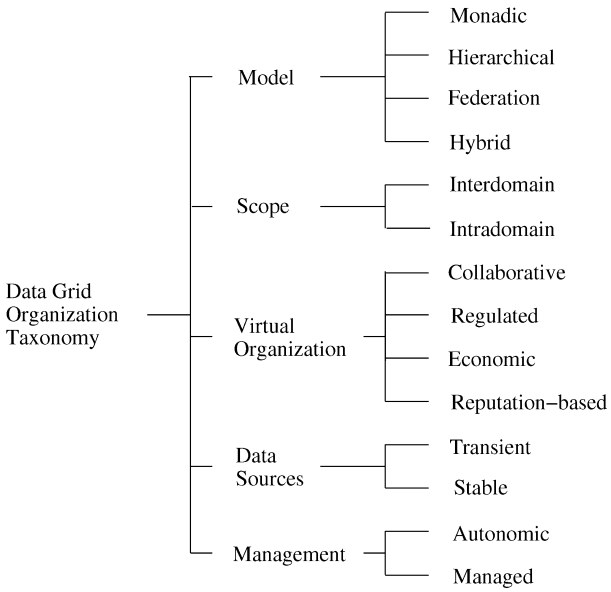
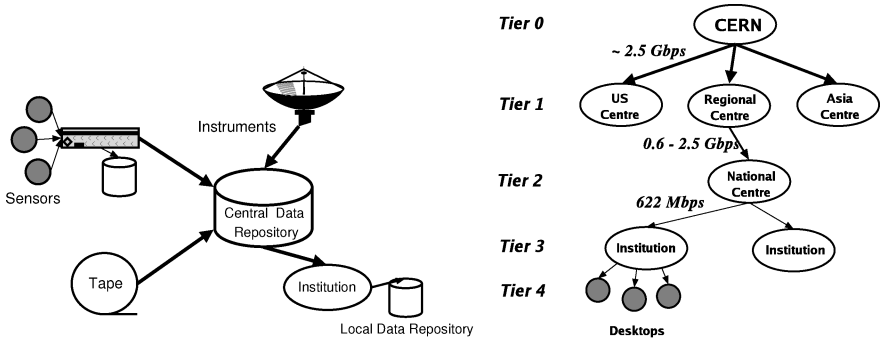
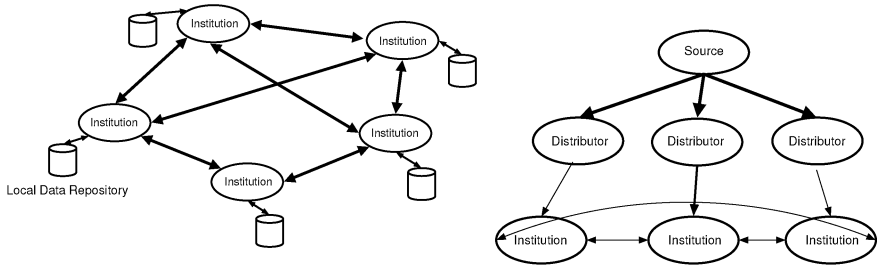


Fig. 4. Data Grid organization taxonomy.



(a) Monadic (b) Hierarchy



(c) Federation (d) Hybrid

Fig. 5. Possible models for organization of Data Grids.

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 senor networks and is made available through a centralized interface such as a Web portal which also verifies users and checks for authorization. This model is shown in Figure 5(a) and has been applied in the NEESgrid Project (Network for Earthquake Engineering Simulation) [2004] in the United States.

The difference between this and other models of Data Grids 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 for by an increase in efficiency of data access such as in the case where all accesses to a particular region are local.

(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 [Aderholz et al. 2000]. This model is presented in Figure 5(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 worldwide called Regional Centers (RCs). From the RCs, the data is then passed downstream to the national and institutional centers and finally on to the physicists. A Tier 1 or a Tier 2 center 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 identified by querying, using metadata. One advantage of this model is that maintenance of consistency is much simpler as there is only one source for the data.

(3) *Federation*. The federation model [Rajasekar et al. 2004] is presented in Figure 5(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) [2005] 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. [2004] discuss about 10 different types of federations that are possible using the Storage Resource Broker (SRB) [Baru et al. 1998] 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 (1)–(3) 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 5(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 can 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. In such cases, service-level agreements dictate the rights of each of the participants. A *reputation-based* VO can 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 we will see later, most of the current Data Grid implementations have always-on data sources such as mass storage systems or production databases. In the future, with diversification, Data Grids are expected to handle transient data sources also.

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 [Parashar and Hariri 2004; Ardaiz et al. 2003] or self-organizing, self-governing systems whose techniques may find applications in future Data Grids.

3.2. 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 6.

Functions. Data transport in Grids can be modeled as a three-tier structure that is similar to the networking stacks such as the 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) [Postel and Reynolds 1985] and GridFTP [Allcock 2003]. 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 [Andersen et al. 2001]. 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-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 when accessing or transferring files to ensure proper authentication of users, file integrity, and confidentiality. Transport

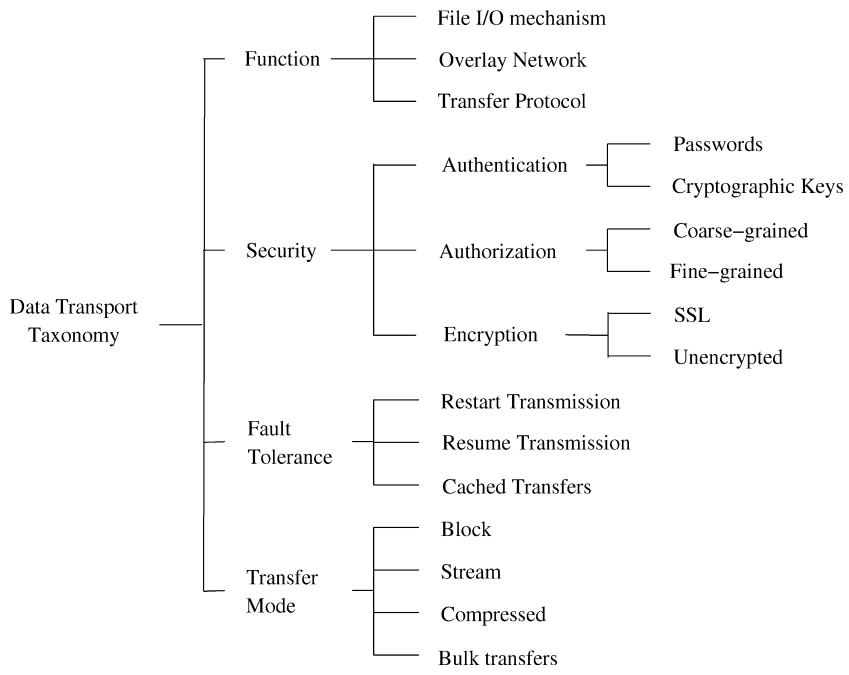


Fig. 6. Data Transport Taxonomy.

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 the Kerberos [Neuman and Ts'o 1994] or the X.509 [Housley et al. 2002] mechanisms. 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 authorized users, delegating read and write access rights to particular files or collections, and flexible ownership of data [Moore et al. 2004]. 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 [Sandhu et al. 1996], and Task-Based Authorization Controls (TBAC) [Thomas and Sandhu 1997]. Data encryption may be present or absent within a transfer mechanism. The most prevalent form of data encryption is through SSL (Secure Sockets Layer) [Wagner and Schneier 1996].

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 [Lee et al. 2001]. Therefore, several optimizations 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.

—*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 the 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 optimizations, while the last one is applied to the transfer mechanism. We group these enhancements under *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.

3.3. 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 7, 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 manager 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

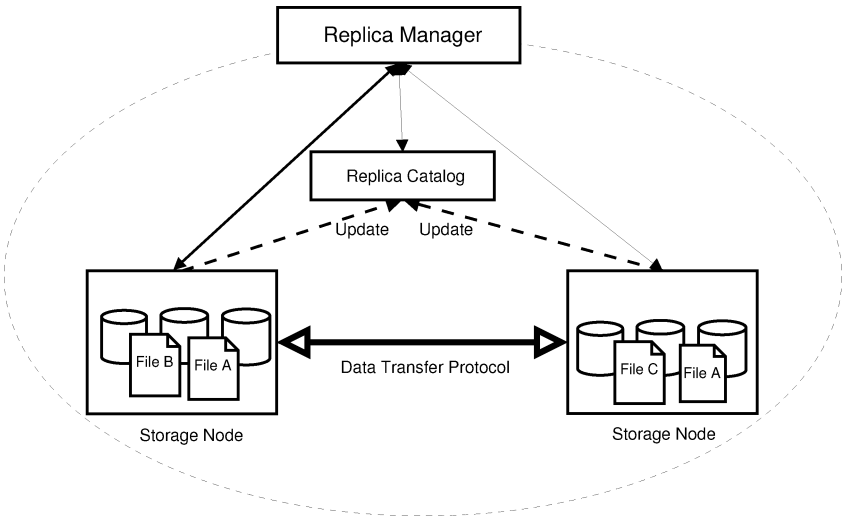


Fig. 7. A replica management architecture.

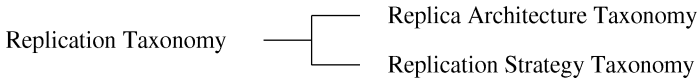


Fig. 8. Replication taxonomy.

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 based on these properties as shown in Figure 8. The architecture of a replication mechanism can be further subdivided into the categories shown in Figure 9.

Model and 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 organized into a variety of topologies which can be grouped chiefly into three types *hierarchy*, *flat* and *hybrid*. Hierarchical topologies have tree-like structure in which updates propogate through definite paths. Flat topologies are found within P2P 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 discussed by Lamehamedi et al. [2002].

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 file system 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

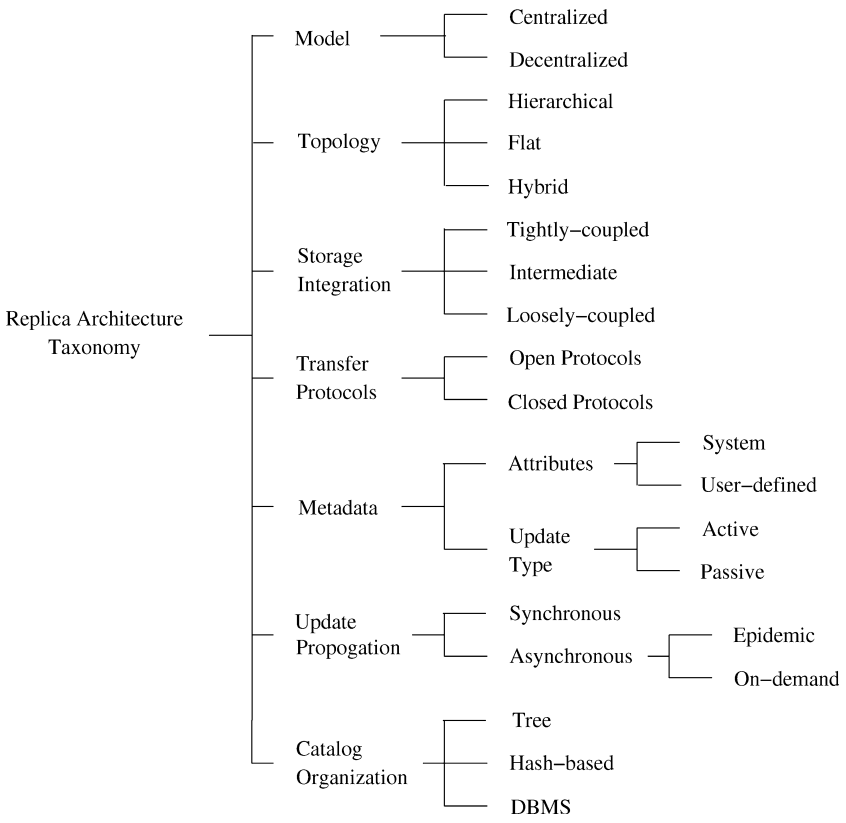


Fig. 9. Replica architecture taxonomy.

System) since they aim to provide transparent access to remote files to applications. An example of such a mechanism is Gfarm [Tatebe et al. 2002]. *Intermediately-coupled* replication systems exert control over the replication mechanism but not over the storage resources. The file systems 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 that is, transparent to users and applications, it is also possible for the latter to direct the mechanism and thereby control the replication process. An example of such a system is the SRB. *Loosely-coupled* replication mechanisms are superimposed over the existing file systems and storage systems. The mechanism exerts no control over the file system. 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 characteriztic. *Open protocols* for data movement such as GridFTP allow clients to transfer data independently 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

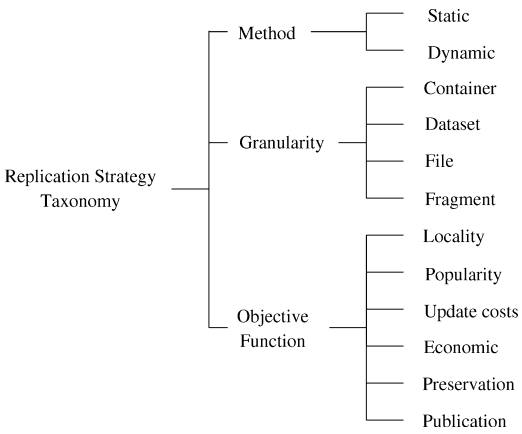


Fig. 10. Replication strategy taxonomy.

data transfer. RLS (Replica Location Service) [Chervenak et al. 2002] and GDMP (Grid Data Mirroring Pilot) [Samar and Stockinger 2001] use GridFTP as their primary transport mechanism. But the flipside 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 outside 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; 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 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 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 [Holliday et al. 2000], 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 GDMP [Stockinger et al. 2001] 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-based (Lightweight Directory Access Protocol) catalogs such as the Globus Replica Catalog [Allcock et al. 2001]. The data can be catalogued on the basis of document hashes as 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, cost of transfer. The replication strategies can be categorized as shown in Figure 10.

Method. The first classification is based on whether the strategies are *static* or *dynamic*. Dynamic strategies adapt to changes in demand, bandwidth and storage availability but induce overhead due to the large number of operations that they undertake since the changes 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 are the result of 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 period of 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*; 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 in order 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.

3.4. 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 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, create a new copy of the data. A taxonomy for scheduling of data-intensive applications is shown in Figure 11. The categories are explained as follows:

Application Model. Scheduling strategies can be classified by the application model toward which they are targeted. Application models are defined in the manner in which the application is composed or distributed for scheduling over global grids. 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 [Foster and Karonis 1998]. *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 [Abramson et al. 2000] wherein a set of tasks is created by running the same program on different inputs. In contrast, a *workflow* is a sequence of tasks

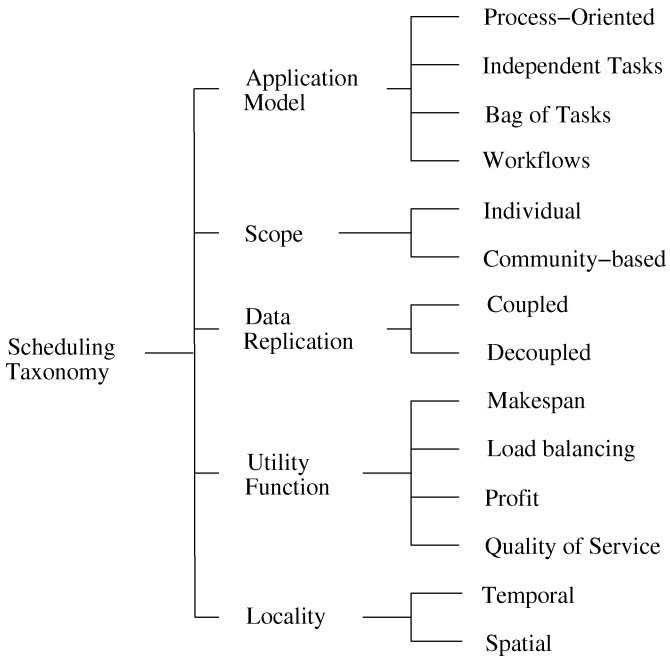


Fig. 11. Data grid scheduling taxonomy.

in which each task is dependent on the results of its predecessor(s). The products of the preceding tasks may be large datasets themselves (e.g., 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 utilize. 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 [Dumitrescu and Foster 2004; Wasson and Humphrey 2003].

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 neighborhood 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 overhead 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 [2002] 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 the architecture of the distributed system at which the algorithm is targeted. Traditionally, scheduling algorithms have aimed at reducing 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 maximum work can be obtained out of the systems. Scheduling algorithms with economic objectives try to maximize the users' economic utility usually expressed as some profit function that takes into account the 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 minimizing the cost of computation, meeting a deadline, meeting stricter security requirements and/or meeting specific resource requirements.

Locality. Exploiting the locality of data has been a tried and tested technique for scheduling and load-balancing in parallel programs [Polychronopoulos and Kuck 1987; Hockauf et al. 1998; McKinley et al. 1996] and in query processing in databases [Shatdal et al. 1994; Stonebraker et al. 1994]. Similarly, data grid scheduling algorithms can be categorized as to 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 be viewed as moving computation to data and temporal locality 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.

4. MAPPING OF TAXONOMY TO VARIOUS DATA GRID SYSTEMS

In this section, we classify various Data Grid research projects according to the taxonomies we developed in Section 3. 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.

4.1. Data Grid Projects

In this space, we study and analyze 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, we will, however, only focus 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 II. These are also classified according to the taxonomy provided in Figure 4

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

Table II. Data Grid Projects Around the World

Name	Domain	Grid Type	Remarks	Country/ Region
LCG [2005]	High Energy Physics	Hierarchical model, Intradomain, Collaborative VO, Stable Sources, Managed	To create and maintain a data movement and analysis infrastructure for the users of LHC.	Global
EGEE [2005]	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 [2005]	Bio-Informatics	Federated model, Intradomain, Collaborative VO, Stable Sources, Managed	To foster collaboration in biomedical science through sharing of data.	United States
NEESgrid [Pearlman et al. 2004]	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 [Avery and Foster 2001]	High Energy Physics	Hierarchical model, Intradomain, Collaborative VO, Stable Sources, Managed	To create an integrated infrastructure that provides computational and storage facilities for high-energy physics experiments.	United States
Grid3 [Gardner et al. 2004]	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 [2005]	Protein Simulation, Brain Activity Analysis	Federated model, Intradomain, Collaborative VO, Stable Sources, Managed	Computational and data infrastructure for medical and biological research.	Japan
Virtual Observatories [Szalay and Gray 2001]	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 [Allcock et al. 2001]	Climate Modelling	Federated model, Intradomain, Collaborative VO, Stable Sources, Managed	Integrating computational, data and analysis resources to create environment for next generation climate research.	United States
GridPP [Huffman et al. 2002]	High Energy Physics	Hierarchical model, Intradomain, Collaborative VO, Stable Sources, Managed	To create computational and storage infrastructure for Particle Physics in the UK.	United Kingdom
eDiaMoND [Brady et al. 2003]	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 [Winton 2003]	High Energy Physics	Hierarchical model, Intradomain, Collaborative VO, Stable Sources, Managed	To create computational and storage infrastructure in Australia for physicists involved in the Belle and ATLAS experiments.	Australia

High Energy Physics (HEP). The computational and storage requirements for HEP experiments have already been covered in previous literature [Bunn and Newman 2003]. 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, managing the infrastructure. Some of them are entering 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 have been 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 [Szalay 2002]. 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 might be required to analyze 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 under construction for the LIGO (Laser Interferometer Gravitational-wave Observatory) [2005] and SDSS (Sloan Digital Sky Survey) [2005] projects.

BioInformatics. The increasing importance of realistic modeling and simulation of biological processes, coupled with the need for accessing existing databases, has led to the adoption of Data Grid solutions by bioinformatics researchers worldwide. These projects involve federating existing databases and providing common data formats for the information exchange. Examples of these projects are the 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 modeling and simulation.

4.2. Data Transport Technologies

In this section, various projects involved in data transport over Grids are discussed and classified according to the taxonomy provided in Section 3.2. The data transport technologies studied here range from protocols such as FTP to over-lay 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 III.

Table III. 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

4.2.1. GASS. Global Access to Secondary Storage (GASS) [Bester et al. 1999] 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 uniform 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 [Czajkowski et al. 1998] 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 a 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.

4.2.2. IBP. Internet Backplane Protocol (IBP) [Plank et al. 1999; Bassi et al. 2002] 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 modeled 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 virtualization 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 it 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 [Plank et al. 2002].

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. IBP is more a low-level storage solution that functions just above the networking layer.

4.2.3. FTP. FTP (File Transfer Protocol) [Postel and Reynolds 1985] 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 the loss of the network 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 [Horowitz and Lunt 1997] and 2389 [Hethmon and Elz 1998] 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) [Galbraith et al. 2006] is a secure file transfer protocol that uses the Secure Shell Protocol (SSH) 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, resumption of interrupted transmissions, or tuning of TCP windows.

4.2.4. GridFTP. GridFTP [Allcock 2003; Allcock et al. 2002] 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 datasets is prone to failures as the network may exhibit transient behavior 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 [Horowitz and Lunt 1997] mechanisms. Parallel and striped data transfers are realized 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 required from the sender side to ensure that the receiver obtains the data correctly. GridFTP implements RFC 2389 [Hethmon and Elz 1998] 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 [Foster and Kesselman 1998]. 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*, built using these libraries functions as a GridFTP client. Another example of a GridFTP client is the UberFTP [NCSA GridFTP Client 2005] client from NCSA.

Evaluation of GridFTP protocols alongside FTP has shown that using the additional features of GridFTP increases performance of data transfer [Ellert et al. 2002]. 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.

4.2.5. Kangaroo. Kangaroo [Thain et al. 2001] 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 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 slowdown 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` (nonblocking read), `put`(nonblocking 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 provides only weak consistency since it is envisioned for grid applications in which data flow is primarily in one direction. 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.

4.2.6. Legion I/O Model. Legion [Chapin et al. 1999] is a object-oriented grid middleware for providing a single system image across a collection of distributed resources. The I/O mechanism within Legion [White et al. 2000] 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 by means of X.509 proxies which are delegated to the file access mechanisms [Ferrari et al. 1999]. Data itself is not encrypted while in transit. Caching and prefetching is implemented for increasing performance and to ensure reliability.

4.2.7. SRB I/O. The Storage Resource Broker (SRB) [Baru et al. 1998] developed at the San Diego Supercomputing Center (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; in this section, we will focus 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 pairs 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 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.

4.3. Data Replication and Storage

In this section, four of the data replication mechanisms used within Data Grids are studied in depth and classified according to the taxonomy given in Section 3.3. These were chosen not only because of their wide usage but also because of the wide variations in design and implementation that they represent. A summary is given in Table IV. Table V 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 section.

4.3.1. Grid DataFarm. Grid Datafarm (Gfarm) [Tatebe et al. 2002] 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 file system, process scheduler, and the parallel I/O APIs.

The Gfarm file system is a parallel file system 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

Table IV. Comparison Between Various Data Replication Mechanisms

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

Table V. Comparison Between Replication Strategies

Project	Method	Granularity	Objective Function
Grid Datafarm	Static	File, Fragment	Locality
RLS	Static	Datasets, File	Popularity, Publication
GDMP [Stockinger et al. 2001]	Static	Datasets, File, Fragment	Popularity, Publication
SRB	Static	Containers, Datasets, File	Preservation, Publication
Lamehamedi et al. [2002, 2003]	Dynamic	File	Update Costs
Bell et al. [2003]	Dynamic	File	Economic
Lee and Weissman [2001]	Dynamic	File	Popularity
Ranganathan et al. [2002]	Dynamic	File	Popularity

throughout the file system as fragments on multiple disks. 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 file system 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 file system 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. In addition, it is 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 [Yamamoto et al. 2004; Tatebe et al. 2004]. 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 interoperating 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 [Tatebe et al. 2004].

4.3.2. RLS. Giggle (GIGa-scale Global Location Engine) [Chervenak et al. 2002] 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 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. [2002]. 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 [Czajkowski et al. 2001].

RLS is aimed at replicating data that is write-once read-many. Data from scientific instruments that needs to be distributed around the world 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 does not handle file transfer or data replication itself. It provides only an index for the replicated data.

4.3.3. GDMP. GDMP [Samar and Stockinger 2001; Stockinger et al. 2001] 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 retransfer. 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.

4.3.4. 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 [Baru et al. 1998; Rajasekar et al. 2003]. SRB also aims to provide a unified view of the data files stored in disparate media and locations by providing the capability to organize 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 organized within a hierarchy of collections and subcollections that is analogous to the UNIX file system 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 collections 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 synchronization 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 [Rajasekar et al. 2002].

4.3.5. Other Replication Strategies. Lamehamedi et al. [2002, 2003] 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.

Table VI. Comparison Between Scheduling Strategies

Work/Project	Application Model	Scope	Data Replication	Utility Function	Locality
Casanova, et al. [2000]	Bag-of-Tasks	Individual	Coupled	Makespan	Temporal
GrADS [Dail et al. 2004]	Process-level	Individual	Decoupled	Makespan	Spatial
Ranganathan & Foster [2002]	Independent Tasks	Individual	Decoupled	Makespan	Spatial
Kim and Weissman 2003	Independent Tasks	Individual	Decoupled	Makespan	Spatial
Takefusa, et. al [2003]	Process-level	Individual	Coupled	Makespan	Temporal
Pegasus [Deelman et al. 2003]	Workflows	Individual	Decoupled	Makespan	Temporal
Thain et al. [2001]	Independent Tasks	Community	Coupled	Makespan	Both
Chameleon [2003]	Independent Tasks	Individual	Decoupled	Makespan	Spatial
SPHINX [In et al. 2003; In et al. 2004]	Workflows	Community	Decoupled	QoS	Spatial
Gridbus Broker [Venugopal and Buyya 2005] and Workflow [Yu and Buyya 2004]	Bag-of-Tasks and Workflows	Individual	Decoupled	QoS	Spatial

Bell et al. [2003] present, a 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 [Vickrey 1961] with descending bids.

Lee and Weissman [2001] 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 entails a performance boost by lessening the load requirements on a particular replica.

Ranganathan et al. [2002] 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.

4.4. Resource Allocation and Scheduling

This section 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 is on features such as adapting to environments with varied data sources and scheduling jobs in order to minimize the movement of data. Table VI summarizes the scheduling strategies surveyed in this section and their classification.

Scheduling strategies for data-intensive applications can be distinguished on the basis of whether they couple data movement to job submission or they don't. As mentioned earlier in Section 3.4, 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. [2000] 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, that is, 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. [2003] have simulated job scheduling and data replication policies for central and tier model organization of Data Grids based on the Grid Datafarm architecture [Tatebe et al. 2002]. Out of the several policies simulated, the authors establish that the combination of *Owner-Computes* 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. [2001] 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 allows 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 from 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 in this section, we will look at another coupled strategy proposed by Phan et al. [2005] 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 [2002] 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 were 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 [Park and Kim 2003] 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. [2003] and the Grid Application Development Software (GrADS) project [Dail et al. 2004] use 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 coordination of all schedules is done through *metascheduling*. Contracts [Vraalsen et al. 2001] are formed to ensure guaranteed execution performance. The mapping and search procedure presented by Dail et al. [2002] 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 [Cooper et al. 2004]. The treatment of data still remains the same.

Casanova et al. [2000] extend three heuristics for reducing makespan—*Min-Min*, *Max-Min*, and *Sufferage* that were introduced by Maheswaran et al. [1999]—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 the 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 [2003] introduce a Genetic Algorithm-based (GA) 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 subgene 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 the 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 favor 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. [2005] 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 previous strategies have concentrated on independent tasks or a BoT model of Grid applications, Pegasus [Deelman et al. 2003] 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) [In et al. 2003] 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 metajob 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 to which he or she belongs. 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 [In et al. 2004] 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.

Data-intensive application scheduling within the Gridbus Broker [Venugopal and Buyya 2005] is carried out on the basis of QoS factors such as deadline and budget. The execution model in this work is that of parameter sweep or Bag of Tasks, each of which depends on multiple data files, each replicated on multiple data resources. The scheduling algorithm tries to minimize the economic objective by incrementally building resource sets consisting of one compute resource for executing the job and one data site each for each file that needs to be accessed by the job. The scheduler itself performs no replication of data in this case. Scheduling of workflows is supported by the Gridbus Workflow Engine [Yu and Buyya 2004] which otherwise has similar properties with respect to the scheduling of data-intensive applications.

5. DISCUSSION

Figures 12–16 pictorially represent the mapping of the systems that were analyzed in Section 4 to the taxonomy. Each of the boxes at the leaves of the taxonomy branches contain 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 12 shows the organizational taxonomy annotated with the Data Grid projects that were studied in Section 4.1. As shown in the figure, current scientific Data Grids mostly follow the hierarchical or the federated models of organization because the data

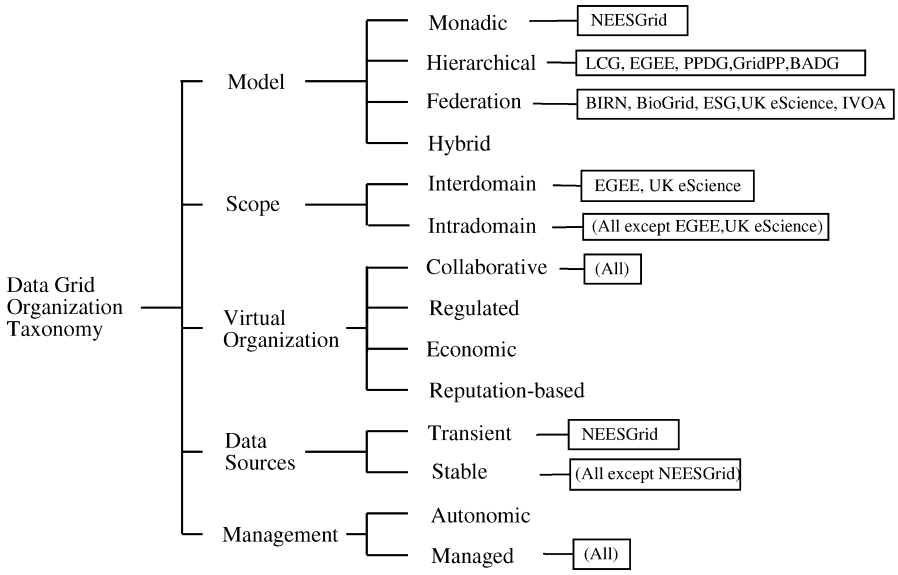


Fig. 12. Mapping of Data Grid organization taxonomy to Data Grid projects.

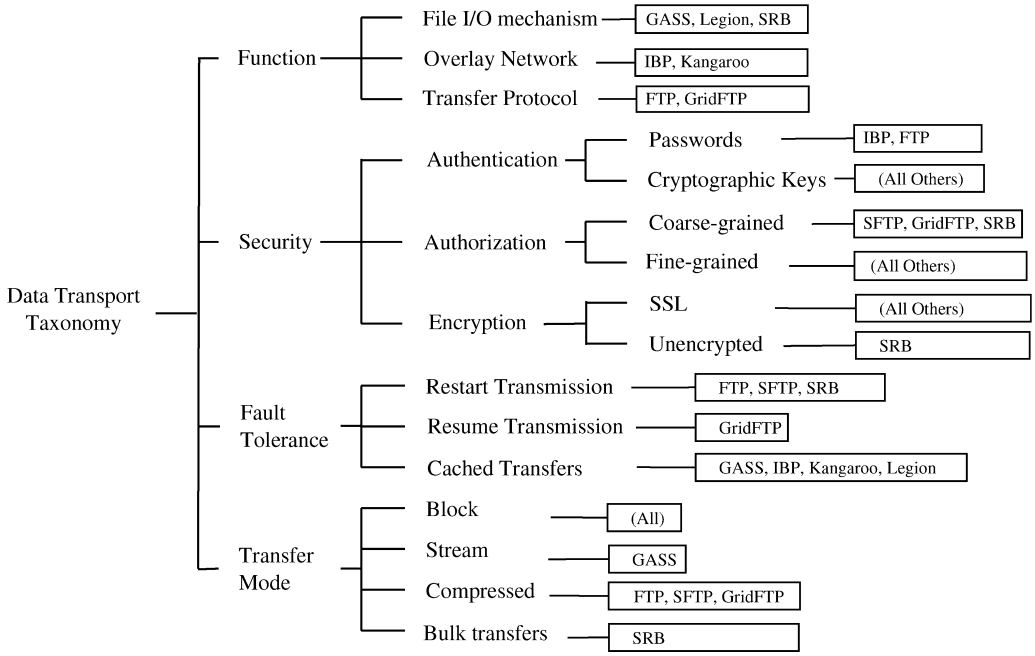


Fig. 13. Mapping of data transport taxonomy to various projects.

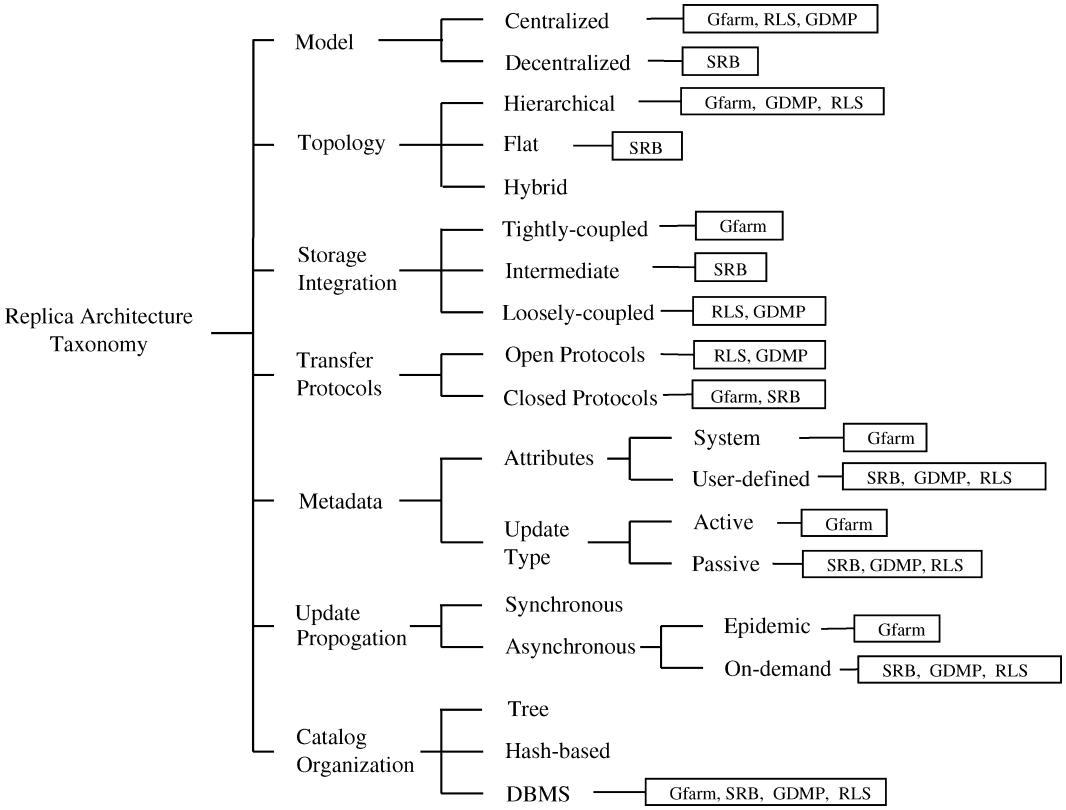


Fig. 14. Mapping of data replication architecture taxonomy to various systems.

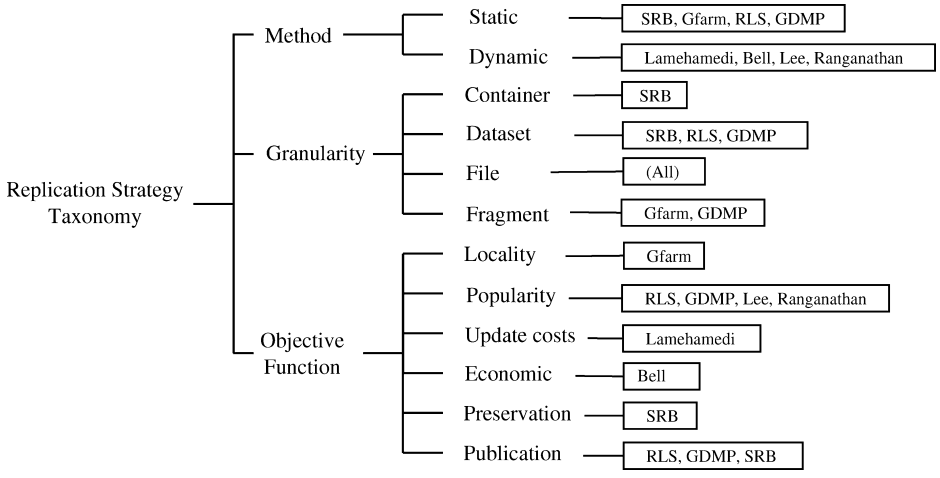


Fig. 15. Mapping of data replication strategy taxonomy to various systems.

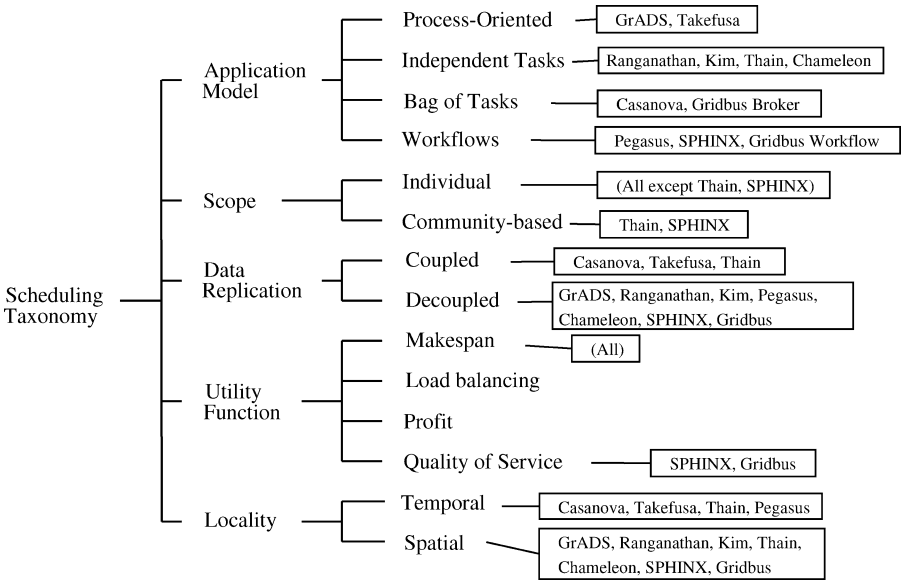


Fig. 16. Mapping of resource allocation and scheduling taxonomy to various systems.

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 4.2 to the proposed taxonomy is shown in Figure 13. 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 14 and 15 show mapping of the data replication systems covered in Sections 4.3 to the replica architecture and strategy taxonomy. The hierarchical model of the HEP experiments in Figure 12 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. Most resource allocation and scheduling efforts, especially those that involve coupling of replication to job submission, follow similar strategies to reduce makespan. This can be inferred from Figure 16 which illustrates mapping of scheduling efforts to the taxonomy.

Table VII. Future Trends and Key Characteristics

Trend	Organization	Transport	Replication	Scheduling
Collaboration	Hybrid models	Fine-grained access	Hybrid topology, Active metadata, Replica Publication	Community
SOA	Autonomic Management	Overlay networks, Fault Tolerance	Open Protocols, Active metadata, Popularity and Economic-based replication	Workflow models, QoS
Market	Interdomain systems, Economic & Reputation-based VOs, Autonomic Management	Fault Tolerance	Decentralized model, Dynamic and Economy-based Replication	Profit, QoS
Enterprise Requirements	Regulated, Economic & Reputation-based VOs	Security	Active metadata, Replica update, Preservation strategy	Workflow models, QoS

Data Grid technologies are only beginning to be employed in production environments and are still evolving to meet present and future requirements. Some of the new developments in areas such as replication and resource allocation and scheduling have already been covered in Section 4. In the next section, we will look at the emerging trends and how these will drive the evolution of Data Grid technologies.

5.1. Future Trends

Four trends that will drive innovation within Data Grids are increased collaboration, service-oriented architectures (SOAs), market mechanisms, and Enterprise requirements. The key properties of each of the constituent technologies, identified within the taxonomy, that are required to realize this innovation is discussed in detail in the following and summarized in Table VII. However, it is important to note that this does not exclude other characteriztics from consideration.

—*Increased Collaboration.* While Data Grids are built around VOs, current technologies do not provide many of the capabilities required for enabling collaboration between participants. For example, the tree structure of many replication mechanisms inhibits direct copying of data between participants that reside on different branches. Replication systems, therefore, will follow hybrid topologies that involve peer-to-peer links between different branches for enhanced collaboration. Exchange of data should be accompanied by enhanced security guarantees. Therefore, this motivates the use of fine-grained access controls throughout the system.

Since communities are formed by the pooling of resources by participants, resource allocation must ensure fair shares to everyone. This requires community-based schedulers that assign quotas to each of the users based on priorities and resource availability. Individual user schedulers could then submit jobs taking into account the assigned quotas and could negotiate with the central scheduler for a quota increase or a change in priorities. They could also be able to swap or reduce quotas in order to gain resource share in the future. Users are able to plan ahead for future resource requirements by advance reservation of resources.

—*Service-Oriented Architecture.* An important element within Web (or Grid) services is the ability for services to be composed of other services by building on standard protocols and invocation mechanisms. This is the key difference between an SOA

[Papazoglou and Georgakopoulos 2003] and a traditional client-server architecture. The high level of transparency within SOAs requires greater reliability guarantees that impact all of the consituent technologies. Service disruptions should be accounted for and quickly from recovered. This requires clean failure models and transparent service migration capabilities that can be realized by implementing autonomic system management in service Grids. Service composition also requires selecting the right services with the required QoS parameters. This impacts both replication and resource allocation and leads to diversification of objective functions and strategies from the current static methods.

As discussed in Section 2, the major focus on the realisation of SOAs in Grids began with the introduction of the OGSA. To realise the requirements of OGSA, the Web Service Resource Framework (WSRF) [Foster et al. 2005] specification has been adopted by the Grid standards community. Globus Toolkit version 4.0 [Foster 2005] and WSRF.NET [Humphrey et al. 2004] are two implementations of the WSRF that provide the basic infrastructure required for Grid services. However, service composition in Grids is currently a work in progress and will only be aided by the ongoing standardisation efforts at the GGF.

—*Market mechanisms.* The increasing popularity of Data Grids as a solution for large-scale computational and storage problems will lead to the entry of commercial resource providers and, therefore, will lead to market-oriented VOs wherein demand-and-supply patterns decide the price and availability of resources. This also provides incentive for content owners to offer their data for consumption outside of specific domains and opens up many interesting new applications. Such VOs are likely to have a broad interdomain scope, and consumers will be able to access domain-specific services by buying them from competing service providers.

From the previous discussion, it is clear that market mechanisms will be based on SOAs. Additionally, resource allocation and replication policies need to be guided by utility functions driven by profit and, at the same time, satisfy user-defined service quality parameters. An example is a dynamic system that takes into account the cost of data movement is presented by Lin [2005].

—*Enterprise requirements.* Enterprises already have production systems in place that handle business functions using distributed data. However, the amount of data that has to be retained and manipulated has been growing by leaps and bounds. Also, with storage devices, even of terabyte capacity, being commoditized, the challenge now is to organize massive volumes of data to enable time-bound extraction of useful information.

Data Grids that provide a solution to these problems also need to take into account the stricter reliability and security requirements in enterprise computing. Support for transaction processing is required to provided consistent computation models in enterprises. Another challenge is to extend the existing Grid mechanisms such as replication, data transfer, and scheduling to work with new data sources such as distributed databases found in businesses [Magowan 2003].

6. SUMMARY AND CONCLUSION

In this article, we have studied, characterized, and categorized 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 a large number of users sharing these resources and wanting to collaborate with each other. We have enumerated several characteriztics where Data Grids share similarities with, and are different from,

other distributed data-intensive paradigms such as content delivery networks, peer-to-peer networks, and distributed databases.

Further on, we focused on the architecture of the Data Grids and the fundamental requirements of data transport mechanisms, data replication systems, and resource allocation and job scheduling. We have developed taxonomies for each of these areas to classify the common approaches and to provide a basis for comparison of Data Grid systems and technologies. We then compared some of the representative systems in each of these areas and categorized them according to the respective taxonomies. In doing so, we have gained an insight into the architectures, strategies, and practices that are currently followed within Data Grids. Also, through our characterization, we have also been able to discover some of the shortcomings and identify gaps in the current architectures and systems. These represent some of the directions that can be taken in the future by researchers in this area. Thus, this article lays down a comprehensive classification framework that, not only serves as a tool for understanding this complex area, but also presents a reference for which future efforts can be mapped.

To conclude, Data Grids are being adopted widely for sharing data and collaboratively managing and executing large-scale scientific applications that process large datasets distributed around the world. However, more research needs to be undertaken in terms of scalability, interoperability, and data maintainability among others, before Data Grids can truly become the preferred infrastructure for such applications. But, solving these problems creates the potential for Data Grids to evolve and become self-organized and self-contained and thus, creating the next generation infrastructure for enabling users to extract maximum utility out of the volumes of available information and data.

ACKNOWLEDGMENTS

We are grateful to the anonymous reviewers for their detailed comments that have helped in improving the quality of this article. We would like to acknowledge the efforts of all the developers of the Grid systems surveyed in this article. We thank our colleagues at the University of Melbourne—Krishna Nadiminti, Tianchi Ma, Sushant Goel, and Chee Shin Yeo- for their comments on this article. We would also like to express our gratitude to Reagan Moore (San Diego Supercomputing Center) for his extensive and thought-provoking comments and suggestions on various aspects of this taxonomy. We also thank Heinz Stockinger (University of Vienna), Chris Mattman (JPL, NASA), and William Allcock (Argonne National Lab) for their instructive comments on this article.

REFERENCES

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'00)*. Cancun, Mexico. IEE Press, Los Alamitos, CA.

CERN, STAFF 2000. Monarc project phase2 report. Tech. rep. (March) CERN.

ALLCOCK, B., BESTER, J., BRESNAHAN, J., CHERVENAK, A., FOSTER, I., KESSELMAN, C., MEDER, S., NEFEDOVA, V., QUESNEL, D., AND TUECKE, S. 2001. Secure, efficient data transport and replica management for high-performance data-intensive computing. In *Proceedings of IEEE Mass Storage Conference*. San Diego. IEEE Press, Los Alamitos, CA.

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. *Parall. Comput. 28*, 5, 749–771.

ALLCOCK, B., FOSTER, I., NEFEDOVA, V., CHERVENAK, A., DEELMAN, E., KESSELMAN, C., LEE, J., SIM, A., SHOSHANI, A., DRACH, B., AND WILLIAMS, D. 2001. High-performance remote access to climate simulation data: a challenge problem for data Grid technologies. In *Proceedings of the ACM/IEEE Conference on Supercomputing (SC'01)* Denver, CO. ACM Press, New York, NY.

ALLCOCK, W. 2003. Gridftp protocol specification. Global Grid Forum Recommendation (GFD.20).

ALONSO, R. AND BARBARA, D. 1989. Negotiating data access in federated database systems. In *Proceedings of the 5th International Conference on Data Engineering*. Los Angeles, CA. IEEE Press, Los Alamitos, CA. 56–65.

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)*. Banff, Alberta, Canada. ACM Press, New York, NY, 131–145.

ANDERSON, D. P., COBB, J., KORPELA, E., LEBOFSKY, M., AND WERTHIMER, D. 2002. Seti@home: An experiment in public-resource computing. *Comm. ACM 45*, 11, 56–61.

ANTONIOLETTI, M., APKINSON, M., KRAUSE, A., LAWS, S., MALAIKA, S., PATON, M. W., PEARSON, D., AND RICCARDI, G. 2005. Web services data access and integration (ws-dai). Tech. rep., (June) GGF DAIS Working Group.

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 Press, Los Alamitos, CA.

AVERY, P. AND FOSTER, I. 2001. The GriPhyN project: Towards petascale virtual-data Grids. Tech. Rep. GriPhyN 2001-14, The GriPhyN Collaboration.

BAKER, M., BUYYA, R., AND LAFORENZA, D. 2002. Grids and grid technologies for wide-area distributed computing. *Softw. Pract. Exper. 32*, 15 (Dec.), 1437–1466. Wiley Publishing, Hoboken, NJ.

BARU, C., MOORE, R., RAJASEKAR, A., AND WAN, M. 1998. The SDSC storage resource broker. In *Proceedings of CASCON'98*. IBM Press, Boston, MA.

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'02)*. Berlin, Germany. IEEE Press, Los Alamitos, CA.

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, 2003 (CCGrid'03)*. Tokyo, Japan. IEEE Press, Los Alamitos, CA.

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, GA. ACM Press, New York, NY.

BIOGRID PROJECT, JAPAN. 2005. `http://www.biogrid.jp/`.

BIOMEDICAL INFORMATICS RESEARCH NETWORK (BIRN). 2005. `http://www.nbirn.net`.

BRADY, M., GAVAGHAN, D., SIMPSON, A., PARADA, M. M., AND HIGHNAM, R. 2003. *Chapter eDiamond: A Grid-Enabled Federated Database of Annotated Mammograms*. Wiley Publishing, Hoboken, NJ, 923–943.

BRAY, T., PAOLI, J., SPERBERG-MCQUEEN, C. M., MALER, E., AND YERGEAU, F. 2004. Extensible markup language (xml) 1.0 3rd ed. W3C Recommendation.

BUNN, J. AND NEWMAN, H. 2003. *Grid Computing: Making the Global Infrastructure a Reality*. Wiley Press, London, UK.

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)*. IEEE Press, Los Alamitos, CA, 574.

CASANOVA, H., LEGRAND, A., ZAGORODNOV, D., AND BERMAN, F. 2000. Heuristics for scheduling parameter sweep applications in grid environments. In *Proceedings of the 9th Heterogeneous Computing Systems Workshop (HCW'00)*. Cancun, Mexico. IEEE Press, Los Alamitos, CA.

CERI, S. AND PELAGATTI, G. 1984. *Distributed Databases: Principles and Systems*. McGraw-Hill, New York, NY.

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*. IEEE Press, Los Alamitos, CA.

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 IEEE/ACM Conference on Supercomputing (SC'02)*. Baltimore, MD.

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. *J. Net. Comput. Appl. 23*, 3, 187–200.

CHOON-HOONG, D., NUTANONG, S., AND BUYYA, R. 2005. *Peer-to-Peer Computing: Evolution of a Disruptive Technology*. Idea Group Publishers, Hershey, PA, 28–65.

CLARKE, I., SANDBERG, O., WILEY, B., AND HONG, T. W. 2001. Freenet: A distributed anonymous information storage and retrieval system. In *International Workshop on Designing Privacy Enhancing Technologies*. Berkeley, CA. Springer-Verlag, Berlin, Germany, 46–66.

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. IEEE Press, Los Alamitos, CA.

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, FL. Springer-Verlag, Berlin, Germany.

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. (HPDC10)*. San Francisco, CA. IEEE Press, Los Alamitos, CA.

DAIL, H., CASANOVA, H., AND BERMAN, F. 2002. A decoupled scheduling approach for the GrADS environment. In *Proceedings of the IEEE/ACM Conference on Supercomputing (SC'02)*. Baltimore, Mb. IEEE Press, Los Alamitos, CA.

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*. Kluwer Academic Publishers, Cambridge, MA, 73–98.

DAVISON, B. D. 2001. A web caching primer. *IEEE Internet Comput. 5*, 4, 38–45.

DEELMAN, E., BLYTHE, J., GIL, Y., AND KESSELMAN, C. 2003. *Grid Resource Management: State of the Art and Future Trends*. Kluwer Academic Publishers, Cambridge, MA, 99–117.

DILLEY, J., MAGGS, B., PARIKH, J., PROKOP, H., SITARAMAN, R., AND WEIHL, B. 2002. Globally distributed content delivery. *IEEE Internet Comput. 6*, 5, 50–58.

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 Press, Los Alamitos, CA.

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. IEEE Press, Los Alamitos, CA.

ELLERT, M., KONSTANTINOV, A., KONYA, B., SMIRNOVA, O., AND WAANANEN, A. 2002. Performance evaluation of GridFTP within the NorduGrid project. Tech. Rep. cs.DC/0205023, (Jan.) NorduGrid Project.

ENABLING GRIDS FOR E-SCIENCE (EGEE). 2005. `http://public.eu-egee.org/`.

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)*. Springer-Verlag, Berlin, Germany, 370–380.

FINKELSTEIN, A., GRYCE, C., AND LEWIS-BOWEN, J. 2004. Relating requirements and architectures: A study of data-grids. *J. Grid Comput. 2*, 3, 207–222.

FOSTER, I. 2005. Globus toolkit version 4: Software for service-oriented systems. Lecture Notes in Computer Science vol. 3779, Springer, verlag, Berlin, Germany, 2–13.

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 (March), 604–612.

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)*. Berkeley, CA. Lecture Notes in Computer Science, vol. 2735. Springer-Verlag, Berlin, Germany, 118–128.

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. IEEE Press, Los Alamitos, CA.

FOSTER, I. AND KESSELMAN, C. 1998. The Globus project: A status report. In *Proceedings of IPPS/SPDP Heterogeneous Computing Workshop*. IEEE Press, Los Alamitos, CA, 4–18.

FOSTER, I. AND KESSELMAN, C. 1999. *The Grid: Blueprint for a Future Computing Infrastructure*. Morgan Kaufmann Publishers, San Francisco, CA.

FOSTER, I., KESSELMAN, C., NICK, J. M., AND TUECKE, S. 2002. Grid services for distributed system integration. *Comput. 35*, 6, 37–46.

FOSTER, I., KESSELMAN, C., TSUDIK, G., AND TUECKE, S. 1998. A security architecture for computational grids. In *Proceedings of the 5th ACM Conference on Computer and Communications Security Conference*. San Francisco, CA. ACM Press, New York, NY.

FOSTER, I., KESSELMAN, C., AND TUECKE, S. 2001. The anatomy of the Grid: Enabling scalable virtual organizations. *Int. J. High Perform. Comput. Appl. 15*, 3, 200–222.

FOSTER, I., TUECKE, S., AND UNGER, J. 2003. OGSA data services. Global Grid Forum 9.

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)*. CSREA Press, Las Vegas, NV, 353–359.

GALBRAITH, J., SAARENMAA, O., YLONEN, T., AND LEHTINEN, S. 2006. SSH File Transfer Protocol (SFTP). Internet Draft. http://www.ietf.org/internet-drafts-ietf-secsh-fliexfer-12.txt.

FOSTER, I. AND COLLEAGUES. 2004. The Grid2003 production grid: Principles and practice. In *Proceedings of the 13th Symposium on High Performance Distributed Computing (HPDC13)*. Honolulu, HI. IEEE Press, Los Alamitos, CA.

GRAY, J., HELLAND, P., O'NEIL, P., AND SHASHA, D. 1996. The dangers of replication and a solution. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'96)*. Montreal, Quebec, Canada. ACM Press, New York, NY, 173–182.

GRAY, J. AND REUTER, A. 1993. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann Publishers, San Mateo, CA.

HETHMON, P. AND ELZ, R. 1998. RFC 2389: Feature negotiation mechanism for the File Transfer Protocol. Proposed Standard.

HEY, T. AND TREFETHEN, A. E. 2002. The UK e-Science Core Programme and the grid. *J. Future Generat. Comput. Syst. 18*, 8, 1017–1031.

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)*. Southhampton, UK. Lecture Notes in Computer Science, vol. 1470. Springer-Verlag, Berlin. Germany, 206–215.

HOLLIDAY, J., AGRAWAL, D., AND ABBADI, A. E. 2000. Database replication using epidemic update. Tech. Rep. TRCS00-01, (Jan.) University of California at Santa Barbara.

HOLTMAN, K. AND COLLEAGUES. 2001. CMS requirements for the Grid. In *Proceedings of Conference on Computing in High Energy Physics (CHEP'01)*. Beijing, China. Science Press.

HOROWITZ, M. AND LUNT, S. 1997. RFC 2228: FTP security extensions. Proposed Standard.

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.

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

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`.

HUMPHREY, M., WASSON, G., MORGAN, M., AND BEEKWILDER, N. 2004. An early evaluation of WSRF and WS-Notification via WSRF.NET. In *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing (GRID'04)*. Pittsburgh, PA. IEEE Press, Los Alamitos, CA, 172–181.

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. Tech. Rep. GriPhyN 2003-17, (May) GriPhyn Grid Physics Network.

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 (IPDPS'04)*. Santa Fe, NM. IEEE Press, Los Alamitos, CA.

JABBER PROJECT. 2005. Jabber protocols. `http://www.jabber.org/protocol/`.

KARLSSON, M. AND MAHALINGAM, M. 2002. Do we need replica placement algorithms in content delivery networks? In *Proceedings of the Web Content Caching and Distribution Conference (WCW'02)*. Boulder, CA. http://www.iwcw.org/.

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 Press, Los Alamitos, CA.

KOSSMANN, D. 2000. The state of the art in distributed query processing. *ACM Comput. Surv. 32*, 4, 422–469.

KOUTRIKA, G. 2005. Heterogeneity in digital libraries: Two sides of the same coin. DELOS Newsletter.

KOUZES, R. T., MYERS, J. D., AND WULF, W. A. 1996. Collaboratories: Doing science on the internet. *IEEE Comput. 29*, 8, 40–46.

KRAUTER, K., BUYYA, R., AND MAHESWARAN, M. 2002. A taxonomy and survey of Grid resource management systems for distributed computing. *Int. J. Softw.: Pract. Exper. 32*, 2, 135–164.

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)*. San Francisco, CA. ACM Press, New York, NY, 169–182.

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)*. Cambridge, MA. ACM Press, New York, NY, 190– 201.

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 Press, Los Alamitos, CA.

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)*. IEEE Press, Los Alamitos, CA.

LASER INTERFEROMETER GRAVITATIONAL WAVE OBSERVATORY. 2005. `http://www.ligo.caltech.edu/`.

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.

LEDLIE, J., SHNEIDMAN, J., SELTZER, M., AND HUTH, J. 2003. Scooped, again. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS2003)*. Berkeley, CA. Lecture Notes in Computer Science, vol. 2735. Springer-Verlag, Berlin, Germany.

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 (HPDC10')*. San Francisco, CA. IEEE Press, Los Alamitos, CA.

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.

LHC COMPUTING GRID. 2005. `http://lcg.web.cern.ch/LCG/`.

LIN, H. 2005. Economy-based data replication broker policies in data Grids. Tech. rep., (Jan.) University of Melbourne, Australia.

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 Press, Los Alamitos, CA.

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. *J. Parall. Distrib. Comput. 59*, 107–131.

MATTMANN, C. A., MEDVIDOVIC, N., RAMIREZ, P., AND JAKOBAC, V. 2005. Unlocking the grid. In *Proceedings of the 8th ACM SIGSOFT Symposium on Component-based Software Engineering (CBSE8)*. St. Louis, MO. ACM Press, New York, NY.

MCKINLEY, K. S., CARR, S., AND TSENG, C.-W. 1996. Improving data locality with loop transformations. In *ACM Trans. Program. Lang. Syst.* Vol. 18. ACM Press, New York, NY, 424–453.

MILOJICIC, D. S., KALOGERAKI, V., LUKOSE, R., NAGARAJA, K., PRUYNE, J., RICHARD, B., ROLLINS, S., AND XU, Z. 2002. Peer-to-peer computing. Tech. Rep. HPL-2002-57, HP Labs, Palo Alto, CA.

MOORE, R., JAGATHEESAN, A., RAJASEKAR, A., WAN, M., AND SCHROEDER, W. 2004. Data Grid management systems. In *Proceedings of the 21st IEEE Conference on Mass Storage Systems and Technologies UMSS'04*, College Park, MD. IEEE Press, Los Alamitos, CA.

MOORE, R. AND MERZKY, A. 2002. Persistent archive basic components. GGF Document Series GFD.26, Global Grid Forum. (July)

MOORE, R., PRINCE, T. A., AND ELLISMAN, M. 1998. Data-intensive computing and digital libraries. *Comm. ACM 41*, 11, 56–62.

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.

NCSA GRIDFTP CLIENT. 2005. `http://dims.ncsa.uiuc.edu/set/uberftp/`.

NEUMAN, B. C. AND TS'O, T. 1994. Kerberos: An authentication service for computer networks. *IEEE Comm. 32*, 9 (Sept.), 33–38.

ORAM, A. 2001. *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O'Reilly & Associates, Inc., Sebastopol, CA.

OZSU, M. T. AND VALDURIEZ, P. 1999. *Principles of Distributed Database Systems*, 2nd Ed. Prentice-Hall, Inc., Upper Saddle River, NJ.

PAPAZOGLOU, M. P. AND GEORGAKOPOULOS, D. 2003. Service-oriented computing. *Comm. ACM 46*, 10.

PARASHAR, M. AND HARIRI, S. 2004. Autonomic Grid computing. In *Proceedings of the International Conference on Autonomic Computing (ICAC'04)*. New York. IEEE Press, Los Alamitos, CA.

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, 2003 (CCGrid'03)*. Tokyo, Japan. IEEE Press, Los Alamitos, CA.

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. IEEE Press, Los Alamitos, CA.

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.

PITOURA, E. AND BHARGAVA, B. 1999. Data consistency in intermittently connected distributed systems. *IEEE Trans. Knowl. Data Engin. 11*, 6, 896–915.

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 Network Storage Symposium (NetStore99)*. Seattle, WA. University of Tennessee, Knoxville, `http://loci.cs.utk.edu/dsi/netstore99/`.

PLANK, J. S., MOORE, T., AND BECK, M. 2002. Scalable sharing of wide area storage resource. Tech. Rep. CS-02-475, (Jan.) Department of Computer Science, University of Tennessee .

POLYCHRONOPOULOS, C. D. AND KUCK, D. J. 1987. Guided self-scheduling: A practical scheduling scheme for parallel supercomputers. *IEEE Trans. Comput. 36*, 12, 1425–1439.

POSTEL, J. AND REYNOLDS, J. K. 1985. RFC 959: File transfer protocol. Standard.

QIN, X. AND JIANG, H. 2003. Data Grids: Supporting data-intensive applications in wide area networks. Tech. Rep. TR-03-05-01, (May) University of Nebraska, Lincoln, NE.

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*. Dubna, Russia, Joint Institute for Nuclear Research, Russia.

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 Conference on Mass Storage Systems and Technologies (MSS'03)*. San Diego, CA. IEEE Press, Los Alamitos, CA.

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'04)*. Las Vegas, NV. CSREA Press, Las Vegas, NV.

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, Scotland. IEEE Press, Los Alamitos, CA.

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 Press, Los Alamitos, CA.

RATNASAMY, S., FRANCIS, P., HANDLEY, M., KARP, R., AND SCHENKER, S. 2001. A scalable content-addressable network. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM'01)*. ACM Press, New York, NY, 161–172.

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'01)*. Heidelberg, Germany. Springer-Verlag, Berlin, Germany, 329–350.

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 (AI'01)*. Innsbruck, Austria. ACTA Press, Calgary, Canada.

SANDHU, R. S., COYNE, E. J., FEINSTEIN, H. L., AND YOUMAN, C. E. 1996. Role-based access control models. *Comput. 29*, 2, 38–47.

SAROIU, S., GUMMADI, K. P., DUNN, R. J., GRIBBLE, S. D., AND LEVY, H. M. 2002. An analysis of internet content delivery systems. *SIGOPS Operat. Syst. Rev. 36*, *Special Issue: Network Behavi.*, 315–327.

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)*. Santiago, Chile. Morgan Kaufmann Publishers Inc., San Francisco, CA, 510–521.

SHETH, A. P. AND LARSON, J. A. 1990. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Comput. Surv. 22*, 3, 183–236.

SLOAN DIGITAL SKY SURVEY. 2005. `http://www.sdss.org/`.

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 (HPDC10)*. San Francisco, CA. IEEE Press, Los Alamitos, CA.

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 Trans. Netw. 11*, 1, 17–32.

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. IEEE Press, Los Alamitos, CA.

SZALAY, A. AND GRAY, J. 2001. The world-wide telescope. *Science 293*, 5537, 2037–2040.

SZALAY, A. S., Ed. 2002. *Proceedings of SPIE Conference on Virtual Observatories*. Waikoloa, HI. Vol. 4846. SPIE.

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(HPDC12)*. Seattle, WA. IEEE Press, Los Alamitos, CA.

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'02)*. Berlin, Germany. IEEE Press, Los Alamitos, CA.

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. 2004. The second trans-pacific Grid datafarm testbed and experiments for SC2003. In *Proceedings of International Symposium on Applications and the Internet—Workshops (SAINT'04)*. Tokyo, Japan. IEEE Press, Los Alamitos, CA.

TATEBE, O., SODA, N., MORITA, Y., MATSUOKA, S., AND SEKIGUCHI, S. 2004. Gfarm v2: A Grid file system that supports high-performance distributed and parallel data computing. In *Proceedings of the Computing in High Energy and Nuclear Physics Conference (CHEP'04)* Interlaken, Switzerland.

THAIN, D., BASNEY, J., SON, S.-C., AND LIVNY, M. 2001. The kangaroo approach to data movement on the Grid. In *Proceedings of the 10th IEEE Symposium on High Performance Distributed Computing (HPDC'10)*. San Francisco, CA. IEEE Press, Los Alamitos, CA.

THAIN, D., BENT, J., ARPACI-DUSSEAU, A., ARPACI-DUSSEAU, R., AND LIVNY, M. 2001. Gathering at the well: Creating communities for Grid I/O. In *Proceedings of Supercomputing*. Denver, CO. IEEE Press, Los Alamitos, CA.

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*. Lake Tahoe, CA. Chapman & Hall, Ltd., London, UK, 166–181.

TRANSACTION MANAGEMENT RESEARCH GROUP (GGF). 2005. `http://www.data-grid.org/tm-rg-charter.html`.

VENUGOPAL, S. AND BUYYA, R. 2005. A deadline and budget constrained scheduling algorithm for e-Science applications on data Grids. In *Proceedings of the 6th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP'05)*. Melbourne, Australia. Lecture Notes in Computer Science, vol. 3719. Springer-Verlag, Berlin, Germany.

VICKREY, W. 1961. Counter-speculation, auctions, and competitive sealed tenders. *J. Finance 16*, 1, 9–37.

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'01)*. Denver, CO. Lecture Notes in Computer Science, vol. 2242. Springer-Verlag, Berlin, Germany.

WAGNER, D. AND SCHNEIER, B. 1996. Analysis of the SSL 3.0 Protocol. In *Proceedings of the 2nd USENIX Workshop on Electronic Commerce*. USENIX Press, Berkeley, CA.

WASSON, G. AND HUMPHREY, M. 2003. Policy and enforcement in virtual organizations. In *Proceedings of the 4th International Workshop on Grid Computing*. Phoenix, AZ. IEEE Press, Los Alamitos, CA.

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, PA. IEEE Press, Los Alamitos, CA.

WINTON, L. 2003. Data Grids and high energy physics—A Melbourne perspective. *Space Science Reviews 107*, 1–2, 523–540.

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'04)*. Pittsburgh, PA. IEEE Press, Los Alamitos, CA.

YU, J. AND BUYYA, R. 2004. A novel architecture for realizing Grid workflow using tuple spaces. In *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing (GRID'04)*. Pittsburgh, PA. IEEE Press, Los Alamitos, CA.

ZHAO, B. Y., KUBIATOWICZ, J. D., AND JOSEPH, A. D. 2001. Tapestry: An infrastructure for fault-tolerant wide-area location and. Tech. Rep. CSD-01-1141, University of California at Berkeley.

Received May 2005; revised November 2005; accepted January 2006