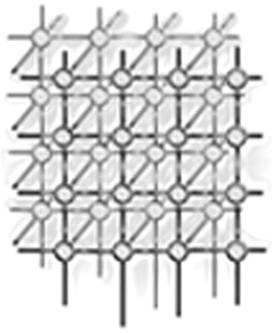
CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE

Concurrency Computat.: Pract. Exper. 0123; **34**:1–2 *Prepared using* cpeauth.cls *[Version: 2002/09/19 v2.02]*

A toolkit for modelling and simulating Data Grids: An extension to GridSim



Anthony Sulistio[1], Uros Cibej[2], Srikumar Venugopal[1], Borut Robic[2], and Rajkumar Buyya[1]

[1] *GRIDS Laboratory, Dept. of Computer Science & Software Engineering, The University of Melbourne, Australia.*
Email: {anthony, srikumar, raj}@csse.unimelb.edu.au
Phone: +61 3 8344-1344; Fax: +61 3 9348-1184.
[2] *Faculty of Computer & Information Science, The University of Ljubljana, 25 Trzaska, 1000 Ljubljana, Slovenia.*
Email: {uros.cibej, borut.robic}@fri.uni-lj.si
Phone: +386 1 4768-867; Fax: +386 1 4264-647.

KEY WORDS: Grid computing; Grid simulation; Data Grids

SUMMARY

Data Grids are an emerging technology for managing large amounts of distributed data. This technology is highly-anticipated by scientific communities, such as in the area of astronomy and high energy physics, because their experiments generate massive amounts of data which need to be shared and analysed. Since it is not feasible to test different usages on real testbeds, it is easier to use simulations as a means of studying complex scenarios. This paper presents our work on incorporating Data Grids features as an extension to GridSim, a Computational Grid simulator. The extension provides essential building blocks for simulating various Data Grids scenarios. Moreover, it is designed to be easily extended. This approach makes it easy to try various strategies and to add functionalities to suit the needs of other communities. This paper also gives a detailed description of the design and usage examples demonstrating the versatility of this tool.

1. Introduction

Grid computing is an emerging technology that focuses on uniformly aggregating and sharing distributed heterogeneous resources for solving large-scale applications in science, engineering and commerce [1]. Data Grids provide infrastructure for whom accessing, transferring, and managing large datasets stored in distributed repositories [2, 3]. Data Grids focus on satisfying requirements of scientific collaborations that have large collections of shared data and where

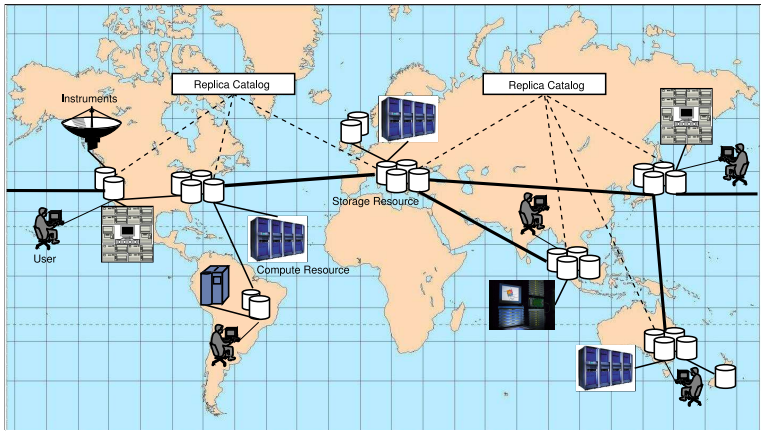


Figure 1. A high-level overview of Data Grid [6]

there is a need for analysing the data and sharing the results among the collaborators. Such applications are commonly found in the area of astronomy [4], climate simulation [5], and high energy physics [3].

There are two basic building blocks for a Data Grid [2]: (i) a high performance data transfer system that enables secure copying of massive datasets; and (ii) a scalable discovery and management system for replicas of datasets. Other services that are required to provide the complete functionality of a Data Grid include management of shared dataset collections, resource allocation for processing, transfer and storage operations, and fine-grained access controls for datasets.

A high-level overview of Data Grid is shown in Figure 1 [6]. A scientific instrument, e.g. a satellite dish, generates large data sets which are stored in a Data Center. The Data Center then notifies a Replica Catalog (RC) about a list of available data sets in the center. Then this RC will synchronize its information with other RCs. This approach allows resources to request for copies of the data sets to the nearest RC when a user submits his/her jobs. The RCs may be arranged in different topologies depending on the requirements of the application domain, the size of the collaboration around the application and its geographical distribution [6].

The LHCGrid [7] project that serves scientific collaborations performing experiments at the Large Hadron Collider (LHC) in CERN, has adopted a *hierarchical* model in which the data is replicated at various levels: from a `tier0` node (at CERN) that acts as a root level to `tierN` nodes (at a leaf level) that respond to requests from regional institutions or organizations in different nations. This approach increases the data availability across all levels and reduces the load on the root node when large number of requests are generated.

On the other hand, the nature of the experiment determines the distribution of data in a project. For example, a scientific project with only a single source of data and limited

geographical distribution of users, such as the Biogrid [9] project, may opt for a simple *monadic* model. A different project that integrates existing independent sources of data, such as the Virtual Observatory [8] project, may decide for a more complex *federated* model.

In a business community, companies such as Sybase and IBM Business Grid provide services and solutions for managing their clients' data into their existing Grid infrastructures and data centers. For example, Avaki EII [10] from Sybase is an enterprise application integration software product that provides data access through a single data layer, and uses a federated approach to deliver data from original sources. DECO [11] from IBM is a Grid meta-scheduler that integrates each job with both computational and data transfer time. Thus, DECO is responsible for deciding which data to replicate, and when to replicate the data to other sites according to users' Quality of Service (QoS) requirements, such as cost and deadline [11].

From these illustrations, Data Grids are shown to be complex, dynamic, and heterogeneous environments. Therefore, it is difficult to evaluate new replication strategies and scheduling algorithms in a *repeatable* and *controlled* manner. In addition, full-scale evaluation on a Data Grid testbed implies interference with processing workloads which may not be encouraged in a production environment. Thus, it is apparent that simulation is required for testing Data Grid strategies in complex scenarios. This need has been recognized before, and has led to the development of various Data Grid simulators such as OptorSim [12], ChicSim [13] and MONARC [14]. However, as we will later argue in Section 2, these systems have various shortcomings that reduce the flexibility of scenarios that researchers can test against.

In this paper, we present an architecture and design of a Data Grid simulation infrastructure developed as an extension to GridSim [15, 16]. GridSim has a complete set of features for simulating realistic Grid testbeds. Such features are modeling heterogeneous computational resources of variable performance, scheduling jobs based on time- or spaced-shared policy, differentiated network service, and workload trace-based simulation from real supercomputers. More importantly, GridSim allows the flexibility and extensibility to incorporate new components into its existing infrastructure.

In this work, GridSim has been extended with the ability to handle: (1) replication of data to several sites; (2) query for location of the replicated data; (3) access to the replicated data; and (4) make complex queries about data attributes. With these new features, GridSim users will be able to do integrated studies of on demand replication strategies with jobs scheduling on available resources. Therefore, the contribution of this paper is the abstraction of Data Grid functionalities integrated into GridSim, thus providing a comprehensive capability and integrated framework for simulating Data Grids. This will benefit a large base of GridSim users, since they can leverage our proposed abstraction without the need to implement their own Data Grids entities.

The rest of this paper is organized as follows: Section 2 discusses related work, whereas Section 3 mentions the GridSim architecture and components of Data Grid realized in GridSim. Section 4 discusses implementation of Data Grid operations, based on the LHC experiment, using GridSim components. Section 5 describes a construction of a complex simulation using the basic building blocks in GridSim. Finally, Section 6 concludes the paper and suggests further work.

Table I. Listing of functionalities and features for each grid simulator

Functionalities	GridSim	OptorSim	Monarc	ChicSim	SimGrid	MicroGrid
data replication	yes	yes	yes	yes	no	no
disk I/O overheads	yes	no	yes	no	no	yes
complex file filtering or data query	yes	no	no	no	no	no
scheduling user jobs	yes	no	yes	yes	yes	yes
CPU reservation of a resource	yes	no	no	no	no	no
workload trace-based simulation	yes	no	no	yes	no	no
differentiated network QoS	yes	no	no	no	no	no
generate background network traffic	yes	yes	no	no	yes	yes

2. Related Work

There are some tools available, apart from GridSim, for application scheduling simulation in Grid computing environments, such as OptorSim [12], Monarc [14], ChicSim [13], SimGrid [17], and MicroGrid [18]. We compare these simulation tools based on three criteria: (1) the ability to handle basic Data Grid functionalities; (2) the ability to schedule compute- and/or data-intensive jobs; and (3) the underlying network infrastructure. Differences among these simulators based on these criteria are summarized in Table I. Note that the new Data Grids features in GridSim are also included in the comparison.

From Table I, it is shown that SimGrid and MicroGrid are mainly targeted as a general-purpose grid simulator for Computational Grids. Hence, they lack features that are core to Data Grids, such as data replication and query for the location of a replica. Although SimGrid has a flow-level simulation of network traffic [19], it does not yet have the capability to simulate differentiated network QoS. MicroGrid that uses a packet-level simulation of network also does not have this feature. This differentiated network functionality is important for modeling QoS compliance for data transfers in wide area network.

OptorSim has been developed as part of the EU DataGrid project [20]. It aims to study the effectiveness of data replication strategies. In addition, it incorporates some auction protocols

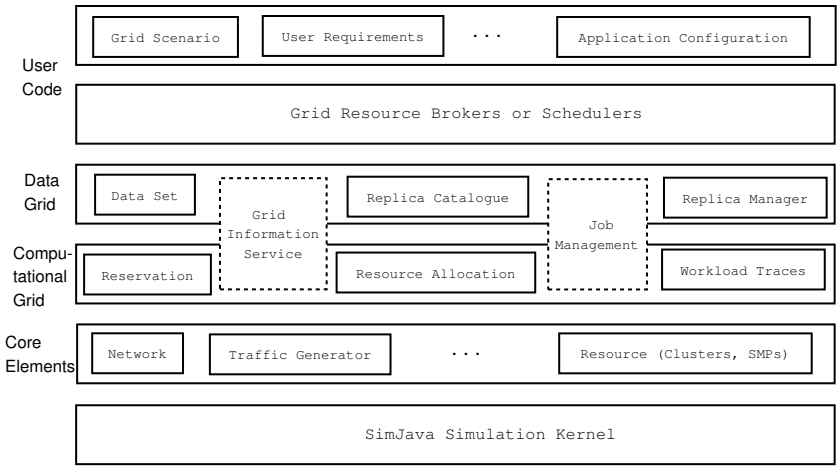


Figure 2. GridSim architecture with the new Data Grids layer

and economic models for replica optimization. In OptorSim, only data transfers are currently being simulated, whereas GridSim is able to run both compute- and data-intensive jobs.

Monarc and ChicSim are grid simulators designed specifically to study different scheduling, replication and performance issues in Data Grid environment. Hence, they provide a general and extensible framework, to implement and evaluate these issues. However, they lack one important feature, i.e. the ability to generate background network traffic. This feature is important because in real-life, networks are shared among users and resources. Hence, congested networks can greatly affect the overall replication and performance issue.

Other features in GridSim that these grid simulators do not have are complex file filtering or data query (will be discussed further in Section 4), CPU reservation and differentiated network QoS. With network QoS, high priority packets are transferred faster than normal ones under a heavy load [16]. Therefore, network QoS is well-suited for applications that require low latency and faster delivery of data generated by scientific instruments, such as in fire or earthquake detection and brain activity analysis application.

3. Mapping Data Grid to GridSim

The GridSim architecture with a new DataGrid layer can be shown in Figure 2. GridSim is based on SimJava2 [21], a general purpose discrete-event simulation package implemented in Java. Therefore, the first layer at the bottom of Figure 2 is managed by SimJava2 for handling the interaction or events among GridSim components.

All components in GridSim communicate with each other through message passing operations defined by SimJava. The second layer models the core elements of the distributed infrastructure, namely Grid resources such as clusters, storage repositories and network links. These core components are absolutely essential to create simulations in GridSim. We will discuss later how new resource types were introduced to model Data Grid components. The third and fourth layers are concerned with modeling and simulation of services specific to computational and Data Grids respectively. Some of the services provide functions common to both types of Grids such as information about available resources and managing job submission. In case of Data Grids, job management also incorporates managing data transfers between computational and storage resources. Replica catalogs, information services for files and data, are also specifically implemented for Data Grids. The fifth layer contains components that aid users in implementing their own schedulers and resource brokers so that they can test their own algorithms and strategies. The layer above this helps users define their own scenarios and configurations for validating their algorithms.

In the following paragraphs, we will discuss in detail the main components of a Data Grid, such as datasets, computing and storage resources, and replica directories (or catalogs), and how these were implemented in GridSim.

3.1. Datasets

A dataset represents a shared collection of data that can be accessed as a single unit. In reality, a dataset may be a collection of files on physical storage or it may just be a single massive file containing records in a particular format. Within actual Data Grids, datasets represent the lowest level of organisation of data. Datasets are replicated to either improve data access performance or reliability or both.

Within GridSim, datasets are modeled using `File` objects. File objects are associated with a `FileAttributes` object that contains information such as owner name, checksum, file size and last modified time. This object can be extended to provide more detailed metadata (or data about data). GridSim supports replication by having two types of `File` objects: master and replica. A *master dataset* is an original instance of the file, whereas all other copies of the file are categorized as *replica* files. This distinction is introduced to identify the purpose of a file copy. The master file is usually generated by a user's program or by a scientific instrument, hence it should not automatically be deleted by resources. On the other hand, a replica file can be created and removed dynamically by a resource based on storage constraints and/or access demands.

3.2. Replica Catalog

A Replica Catalog (RC) provides information about a dataset and its replicas in a Data Grid system. At the very least, the RC indexes the physical locations of the available datasets and replicas in a Data Grid. However, in actual Grids, tools such as the Storage Resource Broker's (SRB's) Metadata CATalog (MCAT) [22] provide more complex information including metadata attributes of the dataset and its parent collection. Also, the RC is generally not a single entity as multiple catalogs are generally networked using different topologies, such as hierarchical, centralized and super peers to improve scalability and fault tolerance.

GridSim provides an RC object which can be configured to be at several levels. For example, a `LocalRC` at the level of resources, and a `RegionalRC` at the level of Grids. Currently, two topologies for networking catalogs have been realized using GridSim: (i) a simple centralized network for Grids with small number of resources and users; and (ii) a hierarchical topology that models the catalog structure used in large Grids such as the LHCGrid project. Currently, there is no support for metadata-based queries in GridSim as its objective is to simulate and not to emulate Data Grids.

3.3. Resource

In Grid computing, any hardware or software component such as a cluster, a supercomputer or a storage repository is called a resource. Computing resources allow users to execute the required application while storage resources allow the users to access datasets and store the results of the computation. In Data Grids, computational resources that allow the files transferred for a job to be stored and accessed for subsequent jobs also function as data resources, thereby fulfilling a dual role. Therefore, the following issues should be considered while modeling Data Grid systems.

- **Storage.** There can be dedicated storage resources available, such as mass storage systems that serve a large number of users or a huge collaboration. It is also possible that a computational resource may be associated with a high performance mass storage resource. For example, a cluster locates in a same network as a large disk array, where both accessible through the same front end node. The media within a storage resource may be disks or tapes, in which case there are differences in read and write times.

- **Resource Management.** Resources in production Grid testbeds are generally high performance computing machines such as clusters and supercomputers that are managed by job management systems or batch schedulers such as Portable Batch System (PBS) [23]. Such schedulers consist of queues that have specific allocation policies. The impact of allocation policies through job waiting times have to be taken into account as well. Along with the computational allocation, data transfer also has to be scheduled as well so that the job does not occupy the resource time waiting for input.

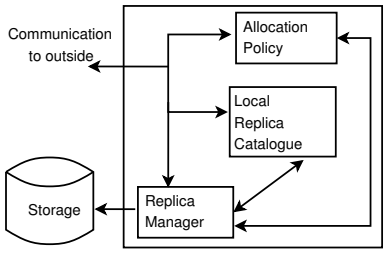


Figure 3. A Data Resource

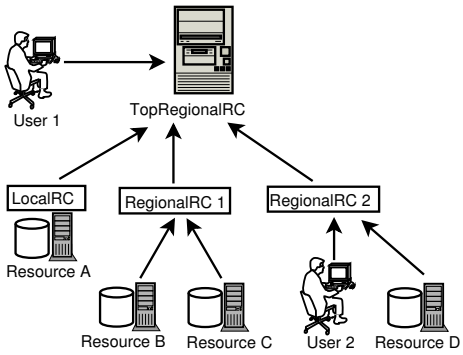


Figure 4. A Hierarchical Replica Catalog model

- **Replica Management.** Resolving requests for datasets involves locating the replicas of the datasets and determing the source from which the dataset has to be accessed from. Also, if the computational resource allows the data to be retained, then a new entry must be created for the replica and subsequent searches must provide the new location to the requesters as well. If the replica is deleted, this has to be propogated through the rest of the system. Therefore, there must be a mechanism for indexing, locating and managing the replicas available in a Data Grid.

GridSim provides the architecture shown in Figure 3 for resource management in Data Grids. A Grid resource is associated with one or more `Storage` objects that can each model either a hard disk-based or a tape-based storage device. The resource has a `ReplicaManager` which handles incoming requests for datasets located on the storage elements. In case a new replica is created, it also registers the replica with the catalog. The replica manager can be extended to incorporate different replica retention or deletion policies. A `LocalRC` object can be optionally associated with the resource to index available files and handle direct user queries about local files. However, other resources cannot query this RC object.

GridSim was initially developed in order to model and simulate computational resources and therefore, provides well-defined abstractions for configuring the resource management on a resource. Each resource is associated with an `AllocationPolicy` object that allocates internal nodes to the user jobs depending on the policy. GridSim currently has two policies, i.e. Space-Shared using a First Come First Serve (FCFS) approach with/without advanced reservation [24] and Time-Shared using a Round Robin approach.

To support Data Grid operations, jobs are allowed to specify, as physical locations, the datasets they require for execution. For each dataset, the resource manager generates a transfer request to the remote storage resource specified in the location. The data transfers can be specified as stream or block operations. In the latter case, the execution of the job is not carried out until an event is received that the complete dataset has arrived at the resource.

The Data Grid version also allows pre-emptive scheduling of data transfers in order to ensure availability of data at the time of execution.

3.4. User

An application or a broker that scheduling jobs on to Grid resources is considered as a user. Such components are able to query and request dataset transfers, submit jobs and register for events. Within GridSim, these are implemented by creating a specific `DataGridUser` object for a particular application or scenario.

4. Implementing Data Grid Operations in GridSim

The previous section described the components of a Data Grid and how they are modeled within GridSim. This section describes the implementation of some of the common operations in Data Grids in GridSim using the entities described previously. The first operation we present is the implementation of a RC structure using the hierarchical topology as an example. We use the hierarchical RC model as a basis for describing other operations such as file addition and deletion. Note that we have opted to implement the hierarchical RC model in this paper, because it has been widely studied [3, 6, 25]. Moreover, this model functions as an interesting test case to demonstrate the utility of our simulator.

4.1. Implementing the Hierarchical Replica Catalog model

The hierarchical RC model is constructed as a catalog tree, as depicted in Figure 4. In this model, some information are stored in the root of the tree, and the rest in the leaves. This approach enables the leaf RCs to process some of the queries from users and resources, thus reducing the load of the root RC. In GridSim, the `TopRegionalRC` class acts as a centralized RC or a root RC in a hierarchical model. In contrast, the `RegionalRC` class represents a local RC and/or a leaf RC in a hierarchical model. The common functionalities of a RC is encapsulated in `AbstractRC`, an abstract parent class for both `TopRegionalRC` and `RegionalRC`.

In our implementation, we follow an approach described in [25] and used by the EU DataGrid project [20]. In this model, the root RC only stores the mapping between a filename and a list of leaf RCs. Each leaf RC in turn stores the mapping between the filename and a list of resources that store this file. Table 5 shows an example of how datasets (in this case files named *file1*, *file2* and *file3*) are indexed for the hierarchical model in Figure 4.

As GridSim is designed to be a research tool, researchers or analysts wishing to test new ideas can easily extend the current component and implement other RC structures such as federated or peer-to-peer or hybrid networks. Creating a new RC model can be done by implementing some core functionalities, such as

TopRegionalRC	
Filename	Location
file1	RegionalRC 1, LocalRC
file2	LocalRC, RegionalRC 2
file3	RegionalRC 1, RegionalRC 2

(a)

RegionalRC 1	
Filename	Location
file1	Resource B, Resource C
file3	Resource B

(b)

RegionalRC 2	
Filename	Location
file2	Resource D
file3	Resource D

(c)

LocalRC	
Filename	Location
file1	Resource A
file2	Resource A

(d)

Figure 5. An example of a filename mapping in a hierarchical model

- adding / deleting entries corresponding to a master dataset or replica(s) when the latter is added / deleted;
- getting the location / attribute of a dataset; and
- filtering dataset queries based on certain attributes.

With the above approach, the RC model and its structure becomes transparent to users and resources. Hence, they are just aware of the location of an RC which they can query but not the type.

4.2. Adding a Master File

Figure 6 depicts the states and events for adding a master file to a Data Grid in GridSim. It involves three entities, i.e. the `ReplicaManager` on the Grid resource in which the file will be stored, the `RegionalRC` which is the leaf RC to which the file must be registered and the `TopRegionalRC` that performs as the root RC for the entire Grid.

The filename is passed by the resource to the `TopRegionalRC` (or root RC) through all the nodes in the hierarchy. The root RC creates a unique identifier which is appended to the

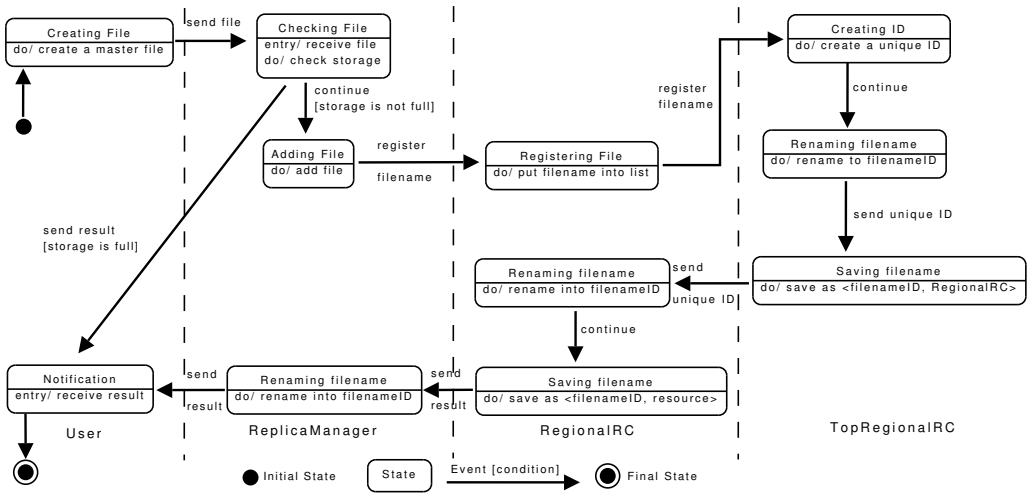


Figure 6. A statechart diagram for adding a master file

filename so that two files with the same name are treated differently. This filename changing procedure is reflected not only in the `TopRegionalRC`, but also in the `RegionalRC` (or leaf RC) and `ReplicaManager`. Furthermore, a tuple `<filenameID, RegionalRC>` is saved within the root RC to serve future user requests. The combination of filename and its unique ID is also considered as a *logical filename (lfn)* [25]. This lfn is a key to query the `TopRegionalRC` and other regional RCs for other Data Grid operations, such as adding a replica or getting location(s) of a file/dataset and its replicas.

4.3. Adding/Deleting a Replica

The operation for creating a replica of an already existing dataset is similar to that of adding a master file and is shown in Figure 7. This operation is invoked when a user or Grid resource wants to explicitly register a copy in the RC structure, i.e. all file copies are not automatically registered within the catalog. This is to support scenarios where copies of datasets are created on the scratch disks of computational resources for specific jobs and then deleted.

The `ReplicaManager` entity in the Grid resource determines whether a file can be retained based on certain policies; in Figure 7, the policy is whether there is enough storage to retain the file. In that case, a registration request with the lfn is sent to the nearest `RegionalRC` (or leaf RC) which is then transmitted to the `TopRegionalRC` for the simulated Data Grid. It is important to note that a replica can only be registered when its master file has already been

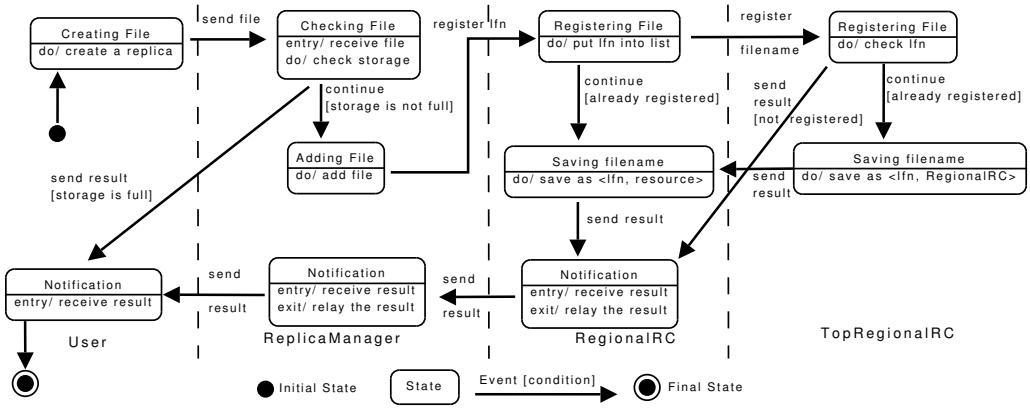


Figure 7. A statechart diagram for adding a replica

mapped by the `TopRegionalRC`. Otherwise, the replica registration would be denied by both `TopRegionalRC` and `RegionalRC` entities.

Deleting a replica works in a similar fashion. The `ReplicaManager` in the resource sends a request to deregister the replica to the nearest `RegionalRC`. If this is the only replica in the `RegionalRC` then the `TopRegionalRC` is requested to disassociate the leaf RC from the file (represented by its lfn). If these operations return successfully, the file is deleted from the resource.

Here, it should be mentioned tha the `ReplicaManager` can be extended to consider different strategies for adding or deleting replicas. One of the possible strategies is that of economy-based creation of replicas presented by Bell et al. [12], that involves the resource determining whether to retain or create a replica based on expected profit.

4.4. Querying a File's Attributes

The operation for getting locations of a file in GridSim is summarized in Figure 8. In this figure, we use two regional RCs as an example. When a lfn does not exist in `RegionalRC1`, then this RC will contact `TopRegionalRC` for a list of RC name that map this lfn. This allows the `RegionalRC1` to request a file location on other RCs, such as `RegionalRC2`.

The user can also find list of available files in the Data Grid system that match certain criteria, such as a combination of filename, size and owner name. This filtering function can be done by extending a `Filter` class and implementing its method, which evaluates all `FileAttribute` objects listed in the `TopRegionalRC`.

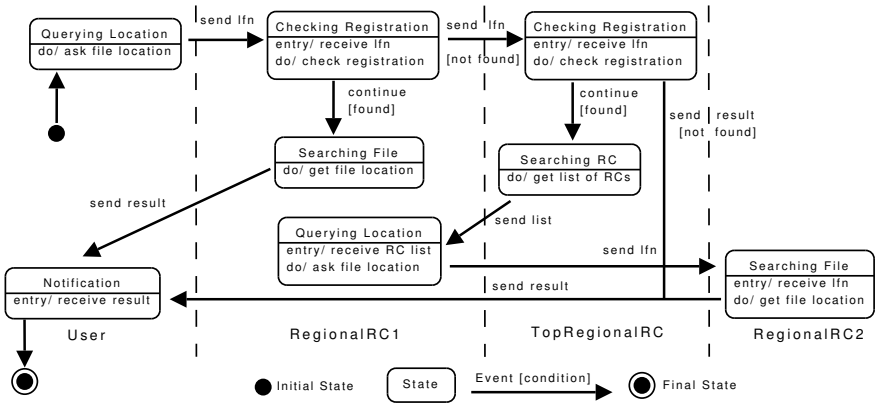


Figure 8. A statechart diagram for getting locations of a file

5. Constructing a Complex Simulation

In this section, we demonstrate how to construct a Data Grid simulation using the building blocks described in the previous sections. As mentioned earlier, there exists several types of Data Grids: hierarchical, monadic and federated. Although this paper demonstrates the usefulness of GridSim for modeling and simulating hierarchical Data Grids (with the LHCGrid as an example), GridSim can be used for dealing with other types of Data Grids. For example, Agarwal et al. [11] has used GridSim to simulate Data Grids in the context business Grid efforts within IBM. Another example is Flahive et al. [26] that use GridSim to simulate the distributed ontology framework in the semantic Grid environment, which requires many data resources to be located as well as large scale processing to take place on the collected data.

We created an experiment based on EU DataGrid TestBed 1 [3]. The network topology of the testbed is shown in Figure 9. In the LHC experiment, for which the EU DataGrid has been constructed, most of the data is read-only. Therefore, to model a realistic experiment, we make these files to be read-only. Furthermore, we assume the atomicity of the files, i.e. a file is a non-splittable unit of information, to simulate the already processed raw data of the LHC experiment.

For experimenting a hierarchical RC model, three regional RC entities are introduced as shown in Figure 9, i.e. *RegionalRC1, RegionalRC2*, and *RegionalRC3*. *RegionalRC1* is responsible for mapping master file locations and communicating with CERN, Lyon and NIKHEF. *RegionalRC2* is responsible for NorduGrid, RAL and Imperial College, and *RegionalRC3* is responsible for Padova, Bologna, Rome, Torino and Milano. Finally, *TopRC* oversees all three regional RCs.

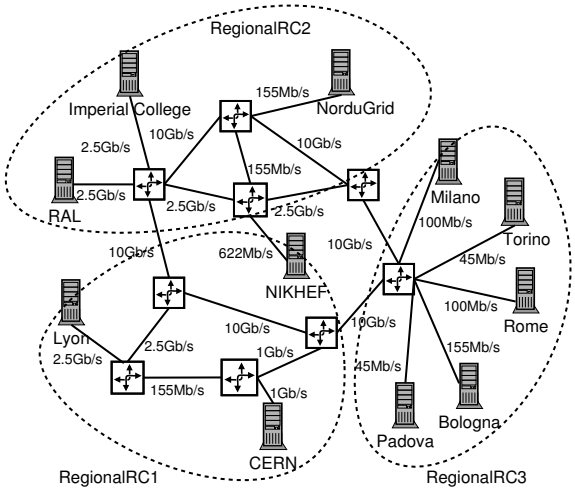


Figure 9. The simulated topology of EU DataGrid TestBed 1

In this experiment, we are trying to look at:

- how a hierarchical RC model can reduce the load of a single (centralized) RC;
- how data replication improves the execution time of data-intensive jobs; and
- how existing GridSim features, such as job allocation policy and the simulation of realistic workloads can be utilized to make this experiment more comprehensive.

5.1. Simulation Setup

Resources. Table II summarizes all the resource relevant information. In GridSim, total processing capability of a resource's CPU or CPU rating is modeled in the form of MIPS (Million Instructions Per Second) as devised by Standard Performance Evaluation Corporation (SPEC) [27].

The resource settings were obtained from the current characteristics of the real LHC testbed [7]. We took the data about the resources and scaled them down. The computing capacities were scaled down by 10 and the storage capacities by 20. The scaling was done primarily for two reasons:

- real storage capacities are very big, hence these resources could store all replicas of files that the jobs require. Since we are trying to demonstrate how a replication strategy works, which deletes some files to make space for new ones, we made the storage capacities smaller;

Table II. Resource specifications

Resource Name (Location)	Storage (TB)	# Nodes	CPU Rating	Policy	# Users
RAL (UK)	2.75	41	49,000	Space-Shared	24
Imperial College (UK)	1.80	52	62,000	Space-Shared	32
NorduGrid (Norway)	1.00	17	20,000	Space-Shared	8
NIKHEF (Netherlands)	0.50	18	21,000	Space-Shared	16
Lyon (France)	1.35	12	14,000	Space-Shared	24
CERN (Switzerland)	2.50	59	70,000	Space-Shared	48
Milano (Italy)	0.35	5	7,000	Space-Shared	8
Torino (Italy)	0.10	2	3,000	Time-Shared	4
Rome (Italy)	0.25	5	6,000	Space-Shared	8
Padova (Italy)	0.05	1	1,000	Time-Shared	4
Bologna (Italy)	5.00	67	80,000	Space-Shared	24

- the simulation of the real computing capacities is not possible because of memory limitation of the computer we ran the simulation on. The complete simulation would require more than 2GB of memory.

Some parameters are identical for all network links, i.e. the Maximum Transmission Unit (MTU) is 1,500 bytes and the latency of links is 10 milliseconds.

Users. We simulated 200 users in total, where each resource is assigned a certain number of users as depicted in Table II. The users arrive with a Poisson distribution; four random users start to submit their jobs every approx. 5 minutes. Each user has between 20 and 40 jobs.

Files. For this experiment we defined 2,000 files. The average file size is 1GB and the file size follows a power-law (Pareto) distribution, which is reported to model a realistic file size distribution [28]. At the beginning of the experiment all master files are placed on the CERN storage. Then copies of the files will be replicated among resources as required during run-time.

Replication Strategy. In this simulation, each Replica Manager (RM) of a resource uses the same strategy, i.e. to delete the least-frequently used replicas when the storage limit capacity for storing new ones is full. However, master files on CERN can not be deleted nor modified during the experiment. This is due to insufficient storage size in other resources to store all of these replicas.

Data-intensive Jobs. For this experiment, we created 500 types of data-intensive jobs. Each job requires between 2 and 9 files to be executed. To model a realistic access the required files are chosen with another power-law distribution, a Zipf-like distribution [29]. This means that the i-th most popular file is chosen with a probability of

$$\frac{1}{i^{\alpha}},$$

in our case $\alpha = 0.6$. The execution size for each job is approximately 84000 kMI $\pm$ 30%, which is around 20 minutes if it is run on the CERN resource.

GridSim already has the ability to schedule compute-intensive jobs, which are represented by a `Gridlet` class. Therefore, we extended this class and implemented a `DataGridlet` class to represent a data-intensive job. As a result, each data-intensive job has a certain execution size (expressed in Millions Instructions – MI) and requires a list of files that are needed for execution. This execution size (in MI) will be used by a resource to determine how much simulation time are required.

Workload Simulation. Grid resources are shared by other (i.e. non-grid) types of application and jobs. To include this factor into our simulation, we added a simulation of a realistic workload on a subset of resources. GridSim makes it possible to simulate realistic workloads. Hence, we took 3 workload logs from the Parallel Workload Archive [30] of the DAS2 5-Cluster Grid in this experiment. Since the workload logs are several months long, we took only one day from each log and simulated it on CERN, RAL and Bologna.

5.2. Simulation Results

Replica Catalog Model Test. In this test, we compare how well a centralized RC model performs in comparison to a hierarchical one as used in the LHC experiment. We use the same configuration as mentioned earlier with the only difference is the RC model.

Figure 10 shows the number of requests that have to be served by each RC entity. We can see that there is a significant reduction in requests (around 60%), when comparing *TopRC* to *CentralRC* from a centralized model. Therefore, the hierarchical model decreases the load of the centralized RC and it is a good solution for Grids with many queries. However, further improvements can be made, such as periodically update and cache replica locations from other regional RCs, or increase the number of regional RCs in proportion to number of resources and users.

Execution Time Test The average execution time for jobs on each resource is depicted in Figure 11. Because of the Time-shared allocation policy, low bandwidth and low CPU power, Padova and Torino have a substantially larger average execution time (80 and 15 hours respectively) and are left out of this figure.

Since the simulated jobs are data-intensive, but they also require a lot of computing power, many parameters influence the speed of job execution. We can see that Imperial has the fastest job execution, since it has a lot of computing power and also a large bandwidth to CERN where

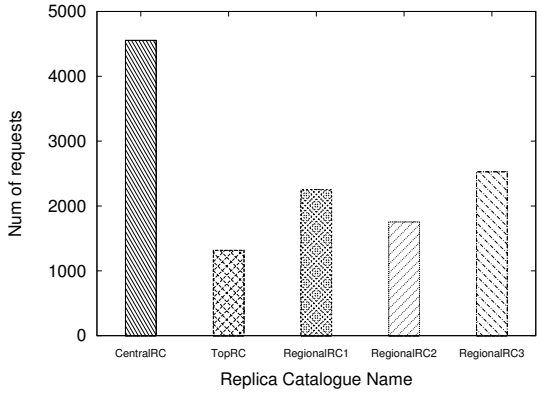


Figure 10. Centralized vs hierarchical RC model

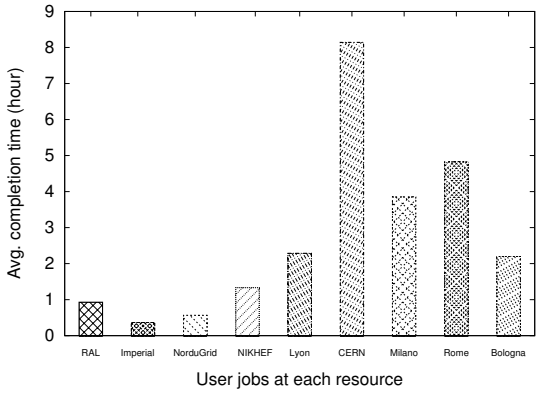


Figure 11. Average execution time

it gets the needed files. The most surprising result is that CERN has a very high computing power and all the data available, but the average execution time is very high. The reason for this is that CERN is running many compute-intensive jobs (defined by the realistic workload) so the jobs have to wait for the execution.

Data Availability Test. To demonstrate how the availability of data can be monitored on each resource we measured how much time does a job require to obtain a unit of data when getting the needed files. This measure will tell us how "close" the resource is to the required data. More precisely, the availability of data for job i on resource j is computed as

$$avail_{ij} = \frac{t_{ij}}{d_i},$$

where d_i is the amount of data required by job i (e.g. in MB) and t_{ij} is the time needed for job i to get all data on the resource j (e.g. in seconds).

The "quality" of a resource to execute data-intensive jobs can be defined as the average availability over all jobs that were executed on the resource. This can be written as

$$avgAvail_j = \frac{\sum_{i \in Jobs_j} avail_{ij}}{|Jobs_j|},$$

where $Jobs_j$ is the set of jobs executed on resource j.

However, because of data replication and different conditions of the network, data availability changes over time. Figure 12 shows the availability change during the simulation on three different resources: Lyon, NIKHEF, and Bologna. Because the behaviour is similar on other resources we omitted them from the figure. In the first minutes we have no information about the data availability, since it is calculated when a job retrieves all the required files. The availability gets initially worse on some nodes, since the jobs that finish first have obviously a better access to data. After this increase, the availability starts to improve significantly because of the replication strategy.

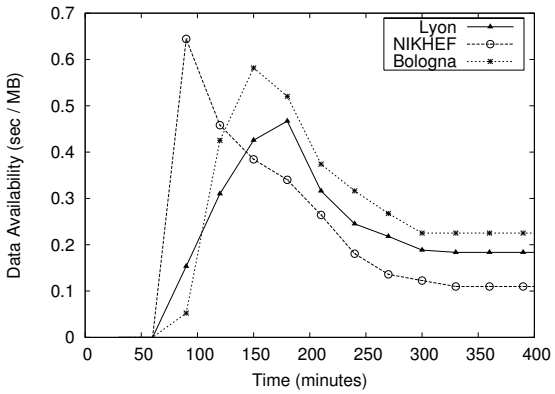


Figure 12. Average availability in time for three resources

6. Conclusion and Further Work

In this work, we present a Data Grid simulation infrastructure as an extension to GridSim, a well-known Computational Grid simulator. With the addition of this extension, GridSim has the ability to handle core Data Grid functionalities, such as replication of data to several sites, query for location of the replicated data, access to the replicated data and make complex queries about data attributes. To give a better insight into the new features, we describe the entities and their basic functions in detail.

Integrating Data Grid functionalities into GridSim provides a rich set of features and a wide range of possibilities for users to explore. We demonstrate this by constructing a complex simulation scenario based on the LHCGrid project, such as the simulation of data-intensive jobs with compute-intensive workloads and the evaluation of demand-driven replication strategies. We also show how GridSim is able to simulate a comprehensive Data Grid platform. The conducted experiment has shown how a hierarchical model for Replica Catalog (RC) provides a better scalability than a centralized one.

In addition, we tested the average execution times on different resources and the data availability which improved substantially because of a simple data replication strategy. The results shown in these experiments are not very surprising, but the described simulation was used as an example of different functionalities of the simulator. We believe this work can help researchers make important findings and help resolve problems arising in Data Grids.

In the future, we are planning to incorporate new functionalities of Data Grids, such as reservation of storage to store future data requests, and automatic synchronization of data sets among resources. We are also intending to add various replica catalog models into the framework. By working with user communities, we will develop some more usage models illustrating applicability of GridSim for simulating various classes of Data application scenarios.

Software Availability

The latest GridSim toolkit with source code and examples can be downloaded from the following website:

`http://www.gridbus.org/gridsim/`

Acknowledgements

This work is partially supported by research grants from the Australian Research Council (ARC) and Australian Department of Education, Science and Training (DEST). We would like to thank Chee Shin Yeo, Marco Netto and anonymous reviewers for their comments on the paper.

REFERENCES

1. I. Foster and C. Kesselman. *The Grid: Blueprint for a new computing infrastructure*. Morgan Kaufmann Publishers, San Francisco, 1999.
2. A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke. The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets. *Journal of Network and Computer Applications*, 23:187–200, 2001.
3. W. Hoschek, F. Janez, A. Samar, H. Stockinger and K. Stockinger. Data Management in an International Data Grid Project. In *Proc. of the 1st International Workshop on Grid Computing (GRID)*, Bangalore, India, December 17, 2000.
4. J. C. Jacob, D. S. Katz, T. Prince, G. B. Berriman, J. C. Good, A. C. Laity, E. Deelman, G. Singh, and M.-H. Su. The Montage architecture for grid-enabled science processing of large, distributed datasets. In *Proc. of NASA's Earth Science Technology Conference, Palo Alto, CA*, Jet Propulsion Laboratory, National Aeronautics and Space Administration, Pasadena, CA, June 2004.
5. M. Mineter, C. Jarvis, and S. Dowers, From stand-alone programs towards grid-aware services and components: a case study in agricultural modelling with interpolated climate data. *Environmental Modelling & Software*, 18(4), pp. 397–391, Elsevier Press, April 2003.
6. S. Venugopal, R. Buyya, and K. Ramamohanarao. A Taxonomy of Data Grids for Distributed Data Sharing, Management and Processing. *ACM Computing Surveys*, 38(1):1–53, March 2006, ACM Press.
7. LCG computing fabric. http://lcg-computing-fabric.web.cern.ch, 5 October 2007.
8. International Virtual Observatory Alliance. http://www.ivoa.net, 5 October 2007.
9. Biogrid Project. http://www.biogrid.jp/e/project/index.html, 5 October 2007.
10. Avaki EII - Enterprise Data Integration Sofware. http://www.sybase.com/products/allproductsa-z/avakieii, 5 October 2007.
11. V. Agarwal, G. Dasgupta, K. Dasgupta, A. Purohit, and B. Viswanathan. DECO: Data replication and Execution CO-scheduling for Utility Grids. In *Proc. of International Conference on Service Oriented Computing (ICSOC)*, Chicago, USA, Dec 4–7 2006.
12. W. Bell, D. Cameron, L. Capozza, P. Millar, K. Stockinger, and F. Zini Simulation of Dynamic Grid Replication Strategies in OptorSim. In *Proc. of the 3rd International Workshop on Grid Computing (GRID)*, Baltimore, USA, 18 November, 2002.
13. ChicSim – the Chicago Grid Simulator. http://people.cs.uchicago.edu/∼krangana/ChicSim.html, 5 October 2007.
14. C. Mihai Dobre and C. Stratan. Monarc simulation framework. In *Proc. of the RoEduNet International Conference*, Timisoara, Romania, May 27–28 2004.

15. R. Buyya and M. Murshed, GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing. *Concurrency and Computation: Practice and Experience (CCPE)*, 14(13-15):1175–1220, Wiley Press, Nov.-Dec., 2002.
16. A. Sulistio, G. Poduvaly, R. Buyya, and C. Tham. Constructing a grid simulation with differentiated network service using GridSim. In *Proc. of the 6th International Conference on Internet Computing*, Las Vegas, USA, June 27–30 2005.
17. A. Legrand, L. Marchal, and H. Casanova. Scheduling distributed applications: The SimGrid simulation framework. In *Proc. of the 3rd International Symposium on Cluster Computing and the Grid*, Tokyo, Japan, May 12–15 2003.
18. H. J. Song, X. Liu, D. Jakobsen, R. Bhagwan, X. Zhang, K. Taura, and A. Chien. The MicroGrid: A scientific tool for modeling computational grids. In *Proc. of IEEE Supercomputing Conference*, Dallas, USA, November 4–10 2000.
19. K. Fujiwara and H. Casanova. Speed and Accuracy of Network Simulation in the SimGrid Framework. In *Proc. of the 1st International Workshop on Network Simulation Tools (NSTools)*, Nantes, France, October 2007.
20. The European DataGrid project homepage. http://eu-datagrid.web.cern.ch/eu-datagrid, 5 October 2007.
21. C. Simatos. Making SimJava count. MSc. Project report, The University of Edinburgh, Sept. 12 2002.
22. C. Baru, R. Moore, A. Rajasekar,and M. Wan, The SDSC Storage Resource Broker. In *Proc. of CASCON'98*, Toronto, Canada, 30 Nov. – 3 Dec. 1998, IBM Press, Indiana, USA.
23. A. Bayucan, R. Henderson, C. Lesiak, B. Mann, T. Proett, and D. Tweten, Portable Batch System: External reference specification. Technical Report, MRJ Technology Solutions, Nov. 1999.
24. A. Sulistio and R. Buyya. A grid simulation infrastructure supporting advance reservation. In *Proc. of the 16th International Conference on Parallel and Distributed Computing and Systems*, Cambridge, USA, November 9–11 2004.
25. H. Stockinger. *Database Replication in World-wide Distributed Data Grids*. PhD thesis, Fakultät für Wirtschaftswissenschaften und Informatik, Universität Wien, 2001.
26. A. Flahive, J. Wenny Rahayu, B. O. Apduhan and D. Taniar. Simulating the Distributed Ontology Framework in the Semantic Grid Environment with GridSim. In *Proc. of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, Las Vegas, USA, June 26–29, 2006.
27. Standard Performance Evaluation Corporation (SPEC). http://www.spec.org, 5 October 2007.
28. W. Gong, Y. Liu, V. Misra, and D. Towsley. On the tails of web file size distributions. In *Proc. of the 39th Allerton Conference on Communication, Control, and Computing*, Illinois, USA, 3–5 October 2001.
29. L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and zipf-like distributions: Evidence and implications. In *INFOCOM (1)*, pages 126–134, New York, USA, 21–25 March 1999.
30. Parallel workload archive. http://www.cs.huji.ac.il/labs/parallel/workload, 5 October 2007.