A Toolkit for Modelling and Simulation of Data Grids with Integration of Data Storage, Replication and Analysis

Anthony Sulistio [a,*], Uroš Čibej [b], Borut Robič [b], and Rajkumar Buyya [a]

[a]*Grid Computing and Distributed Systems Lab*
Dept. of Computer Science and Software Engineering, The University of Melbourne
111 Barry St, Carlton VIC 3053 Australia

[b]*Faculty of Computer and Information Science*
The University of Ljubljana, 25 Trzaska, 1000 Ljubljana, Slovenia

Abstract

Data Grids are an emerging new technology for managing large amounts of distributed data. This technology is highly-anticipated by scientific communities, such as in the area of astronomy, protein simulation and high energy physics. This is because experiments in these fields generate massive amount of data which need to be shared and analysed. Since it is not possible to test many different usage scenarios on real Data Grid testbeds, it is easier to use simulation as a means of studying complex scenarios.

In this paper, we present our work on a Data Grid simulation infrastructure as an extension to GridSim, a well-known Computational Grid simulator. The extension that we have developed provides essential building blocks for simulating Data Grid scenarios. It provides the ability to define resources with heterogeneous storage components, and the flexibility to implement various data management strategies. It also allows authorized users to share, remove and copy files to resources. Moreover, users and/or resources are able to query to a Replica Catalogue about the location of a particular file. The Replica Catalogue (RC), a core entity of Data Grids, is an information service which provides registry and look up service. Therefore, our work provides the possibility to organize RC entities in different ways, such as in centralized or hierarchical model.

Key words: Data grid, grid computing, grid simulation.

* Corresponding author. Tel.: +61 3 8344 1360; fax: +61 3 9348 1184
Email address: `anthony@csse.unimelb.edu.au` (Anthony Sulistio).

1 Introduction

Grid computing is an emerging new technology, which focuses on sharing dispersed heterogeneous resources among their collaborators, and aggregating them for solving large-scale parallel applications in science, engineering and commerce [15]. Initially, the Grid community focused on compute-intensive applications that contain many parallel and independent tasks. Projects such as Legion [19], Globus [14], Nimrod-G [6], Apples [8], SETI@home [1], and Condor-G [16] are mainly concentrated on development of technologies for Computational Grids. However, in recent years, applications are targeting on the use of shared and distributed data to be analyzed among collaborators. In addition, these applications can produce large data sets in the order of Gigabytes (GB) and possibly Terabytes (TB) or Petabytes (PB). Many of these applications are in the area of astronomy [2], protein simulation [4], and high energy physics [12]. Hence, Data Grids are becoming an important research area [22].

Data Grids are Grids that provide a scalable storage and access to data sets in different locations [9]. The data sets must eventually be evaluated and analyzed by scientists and other collaborators from different research groups, who may be located around the globe. Hence, efficient, reliable and secure access to the data sets and their replication across several sites are primary concerns in Data Grids.

Different scenarios need to be evaluated to ensure the effectiveness of access and replication strategies. Given the inherent heterogeneity of a Grid environment, it is difficult to produce performance evaluation in a *repeatable* and *controlled* manner. In addition, Grid testbeds are limited and creating an adequately-sized testbed is expensive and time consuming. Therefore, it is easier to use simulation as a means of studying complex scenarios.

To address the above issues, we have developed a Data Grid simulation infrastructure, as an extension to GridSim [29]. We opted to work on GridSim because it has a complete set of features for simulating realistic Grid testbeds. Such features are modelling heterogeneous computational resources of variable performance, differentiated network service, and workload trace-based simulation from a real supercomputer. 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.

The rest of this paper is organized as follows: Section 2 discusses related work. Section 3 mentions the GridSim architecture. Section 4 describes an overview of the

Table 1
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
overall network features	good	simple	good	good	good	best
differentiated network QoS	yes	no	no	no	no	no
generate background network traffic	yes	yes	no	no	yes	yes

model, whereas Section 5 explains some of the Data Grid operations incorporated into GridSim. Section 6 describes a construction of a complex simulation using the basic building blocks in GridSim. Finally, Section 7 concludes the paper and suggests further work.

2 Related Work

There are some tools available, apart from GridSim, for application scheduling simulation in Grid computing environments, such as OptorSim [3], Monarc [11]. ChicSim [10], SimGrid [20], and MicroGrid [25]. We evaluate 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 1.

From Table 1, 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.

OptorSim has been developed as part of the EU DataGrid project [13]. It aims to study the effectiveness of data replication strategies. In addition, it incorporates some auction protocols 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 5), CPU reservation and differentiated network Quality of Service (QoS). With network QoS, high priority packets are transferred faster than normal ones under a heavy load [17,29]. 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 GridSim Architecture

GridSim is based on SimJava2 [24], a general purpose discrete-event simulation package implemented in Java. We designed GridSim as a multi-layer architecture for extensibility. This allows new components or layers to be added and integrated into GridSim easily. In addition, the GridSim layer archicture captures the model of grid computing environment. The GridSim architecture with the new DataGrid layer can be shown in Figure 1.

The first layer at the bottom of Figure 1 is managed by SimJava2 for handling the interaction or events among GridSim components. The second layer is dealt with the GridSim infrastructure components, such as network and resource hardware. The third and fourth layer are concerned with modelling and simulation of Computational Grids and Data Grids respectively. GridSim components such as Grid Information Service and Job Management are extended from the third layer to incorporate new requirements of running Data Grids. The fifth and sixth layer are allocated to users wanting to write their own code in GridSim.

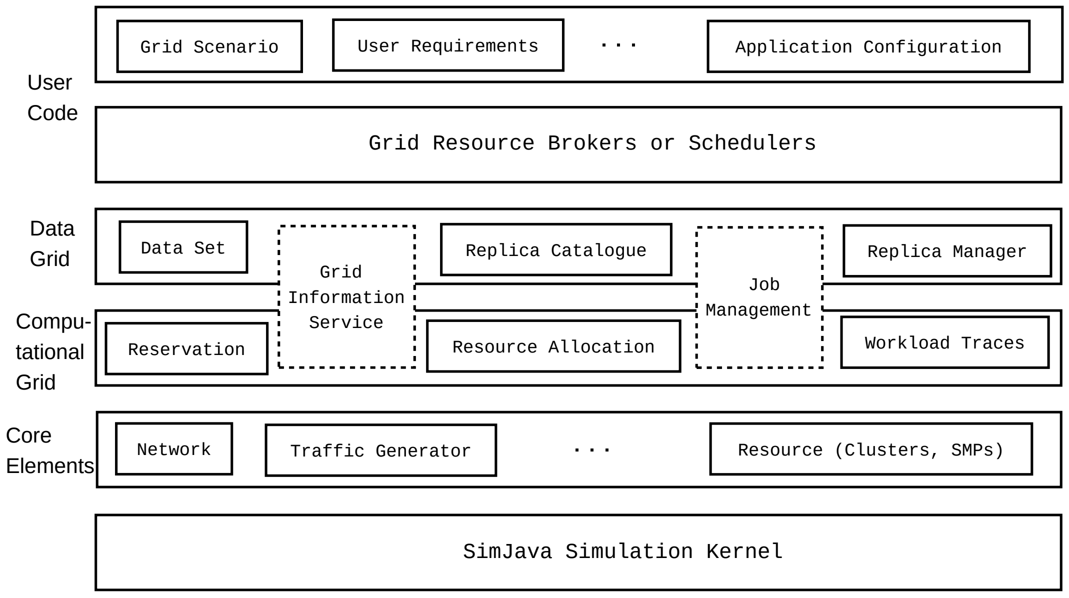


Fig. 1. GridSim architecture

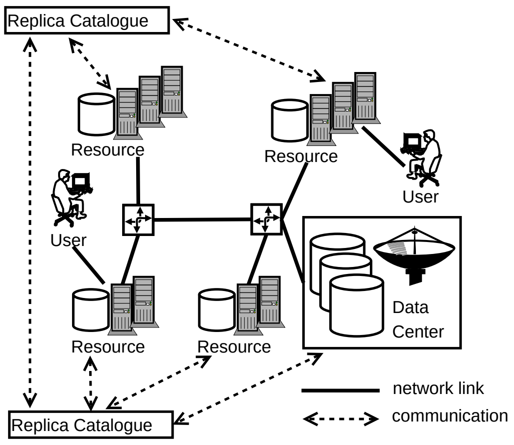


Fig. 2. A high-level overview of Data Grid

4 Description of the DataGrid Model

A high-level overview of Data Grid is shown in Figure 2. A scientific instrument (a satellite dish in this example), generates large data sets which are stored in the Data Center. The Data Center then notifies a Replica Catalogue about the list of available data sets in the center. This approach allows resources to request for file copies of the data sets when a user submits his/her jobs.

To model realistic Data Grid scenarios like in the above description, new components such as Replica Catalogue, and concepts like data sets and file copies are introduced into GridSim.

A data set is typically grouped into one or more files. Therefore, we define a file as a basic unit of information in GridSim. In addition, GridSim supports a feature called *data replication*. With data replication, it is possible to make several copies of the same file on different resources. This approach enhances fault tolerance characteristics of the system and improves access to the file [27].

In GridSim, there are two types of file: master and replica. A *master file* 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.

In GridSim, `File` class represents the physical (master or replica) file, and each file has a `FileAttribute` class representing the information regarding this file. The information stored in the `FileAttribute` class can be the owner name, checksum, file size and last modified time. The information contained in the `FileAttribute` object is the one being sent by a resource to a Replica Catalogue.

4.1 Replica Catalogue

Replica Catalogue (RC) is a core component of every Data Grid system. The function of a RC is to store the information (metadata) about files and to provide mapping between a filename and its physical location(s).

The RC does not have to be a single entity in a Data Grid system. It can also be composed of several distributed components, which, by switching the information among them, provide a transparent service to the users and resources. Currently, GridSim allows two possible catalogue models as described below.

4.1.1 Centralized Model

This model is the most simple solution, as there is only one RC component in a Data Grid system that handles all registrations and queries sent by resources and users. Hence, the RC maps a filename to a list of resources that stores this file. An example of a system that follows this model is Napster [23], a peer-to-peer (P2P) music sharing program.

This model is typically used in smaller systems, where queries to the RC are limited. However, as the number of resources and users grows, the RC becomes a

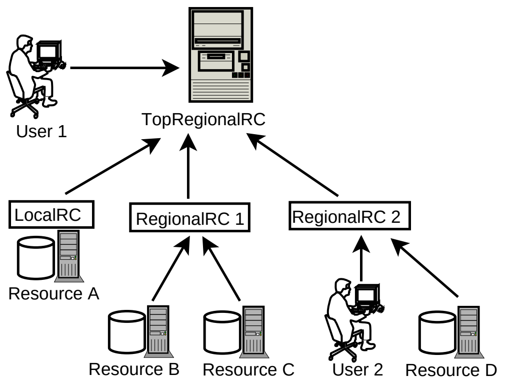


Fig. 3. A Hierarchical Replica Catalogue model

bottleneck since there are too many queries to be processed [27]. In this case, different types of catalogue models are needed. One solution is to use a hierarchical model.

4.1.2 Hierarchical Model

The hierarchical RC model is constructed as a catalogue tree, as depicted in Figure 3. In this model, some information are stored in the root of the catalogue 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 our implementation, we follow an approach described in [27] and used by the EU DataGrid project [13]. 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 2 shows an example of how information are indexed for the hierarchical model in Figure 3.

4.1.3 Implementing a New Model

The hierarchical model described previously deals with improving the performance issue of the RC. However, some issues are not considered, such as reliability or fault tolerance and availability. With the hierarchical model, the root RC can become a single point of failure to the whole system.

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 more advanced RC models. By using this approach, new RC models, such as P2P model can be compared to the existing ones. The P2P model offers better reliability and scalability than the hierarchical model as mentioned in [7].

In GridSim, common functionalities of a RC is encapsulated in `AbstractRC`, an abstract parent class for both `TopRegionalRC` and `RegionalRC`. The `TopRegionalRC`

Table 2
An example of a filename mapping in a hierarchical model

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

LocalRC	
Filename	Location
file1	Resource A
file2	Resource A

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

RegionalRC 2	
Filename	Location
file2	Resource D
file3	Resource D

class acts as a centralized RC or a root RC in a hierarchical model. In constrast, the `RegionalRC` class represents a local RC and/or a leaf RC in a hierarchical model. Therefore, creating a new RC model can be done by extending the `AbstractRC` class and implementing some core functionalities, such as

- adding / deleting a master file or replica(s);
- getting the location / attribute of a file; and
- filtering files based on certain criteria (described in Section 5).

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

4.2 Resource

A resource for Data Grids enables users to run their jobs as well as to gain access to available data sets.

A resource has the following components:

- **Storage.** A resource can have one or more different storage elements, such as

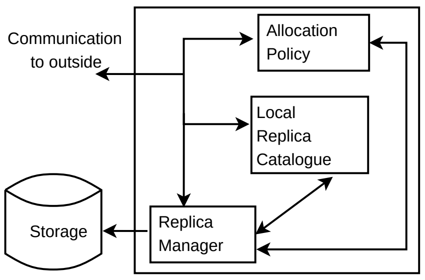


Fig. 4. A Data Resource

harddisks or tape drives. An individual storage is responsible for storing, retrieving and deleting files. GridSim models physical characteristics of a real storage device, such as read and write access delays.

- **Replica Manager.** This component is responsible for handling and managing incoming requests about data sets in a resource for one or more storage elements. It also performs registrations of files stored in the resource to a designated RC.
- **Local Replica Catalogue.** This component is an optional part of a resource. The Local RC is responsible for indexing available files on the resource only. It also handles users' queries. However, the Local RC does not serve as a catalogue server to other resources. This should be done by a leaf or regional RC.
- **Allocation Policy.** This component is responsible for executing user jobs to one or more available nodes in a resource. GridSim currently has two policies, i.e. Space-Shared using a First Come First Serve (FCFS) approach with/without advanced reservation [28] and Time-Shared using a Round Robin approach.

All the resource's components can be extended to implement new ideas and features of the grid. In the next section, we will focus on the extension of the RM, since it is responsible for dealing with the most important functionality of the Data Grid.

4.2.1 Extending Replica Manager

In GridSim, a Replica Manager (RM) is represented as an abstract class `ReplicaManager`. Currently, a concrete implementation of the RM is captured in the `SimpleReplicaManager` class, the functionality of which is described in more detail in Section 5. However, like the RC, the RM also can be extended to test new ideas and approaches to solving various problems that arise in Data Grids. In order to implement a new resource manager, the following user requests have to be handled:

- adding / deleting a master file and replica; and
- sending a file to the user.

Furthermore, the RM has to register and deregister files from the RC when necessary.

```
public class User extends DataGridUser {

  public User(String name, Link l) throws Exception {
    super(name, l);
  }

  public void body() {
    // first get a location of a file
    int loc = super.getReplicaLocation("testFile1");

    // second, request the file from the specified
    // location and make a replica to resource_1
    if (loc != -1) {
      int resId = GridSim.getEntityId("resource_1");
      File file = getFile("testFile1", loc);
      super.replicateFile(file, resId);
    }

    // shut down I/O ports
    super.shutdownUserEntity();
    super.terminateIOEntities();
  }
} // end class
```

Fig. 5. A simple example on how to create a new data grid user

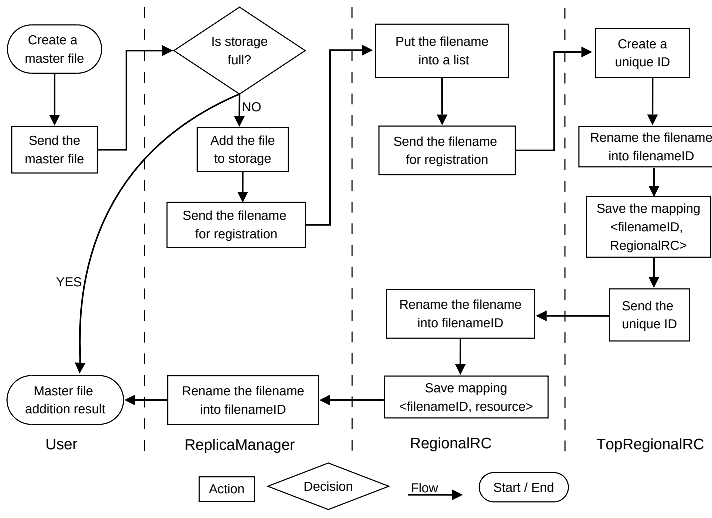
4.3 User

In GridSim, a user can be modelled as an application or a broker performing on behalf of the user, by extending from a `DataGridUser` class. The user, then, can query and request files located on different resources. However, to simulate various user scenarios during an experiment, this class must be extended and its behaviour must also be specified.

Figure 5 shows a simple implementation of a Data Grid user. When the simulation starts, this user simply transfers the file `testFile1`, replicates it to `resource_1`, and then terminates. As it can be seen many different usage scenarios can easily be implemented in this way.

5 Data Grid Operations in GridSim

In the previous section, we defined the entities in GridSim. In this section, we describe in more detail how the most common operations are currently implemented. We use a hierarchical RC model as an example for describing these operations.



Create a master file
Send the master file
Is storage full?
NO
YES
Add the file to storage
Send the filename for registration
Put the filename into a list
Send the filename for registration
Create a unique ID
Rename the filename into filenameID
Save the mapping <filenameID, RegionalRC>
Send the unique ID
Rename the filename into filenameID
Save mapping <filenameID, resource>
Rename the filename into filenameID
Master file addition result
User
ReplicaManager
RegionalRC
TopRegionalRC
Action
Decision
Flow
Start / End

Fig. 6. Flowchart diagram for adding a master file

5.1 *Adding a Master File*

The operation for adding a master file in GridSim requires a four-way approach as summarized in Figure 6. A `TopRegionalRC` entity is responsible for generating a unique ID. This ID is created to ensure that two different files with the same name are treated differently. Then, the ID is appended at the end of the original filename, e.g. `file` with ID is 1 becomes `file1`. This filename changing procedure is reflected not only in the `TopRegionalRC`, but also in the `RegionalRC` and `ReplicaManager`. Furthermore, a tuple `<filenameID, RegionalRC>` is saved to the mapping for future user requests in the `TopRegionalRC`.

The unique ID is incremented every time the `TopRegionalRC` receives a request for a master file addition by its regional or leaf RCs. The combination of filename and its unique ID is sometimes referred to as a *logical file name (lfn)* [27]. 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 locations of a file.

5.2 *Adding a Replica*

The operation for adding a replica in GridSim is summarized in Figure 7. It is important to note that a replica can only be registered when its master file has already been mapped by the `TopRegionalRC`. Otherwise, the replica registration would be denied by both `TopRegionalRC` and `RegionalRC` entities.

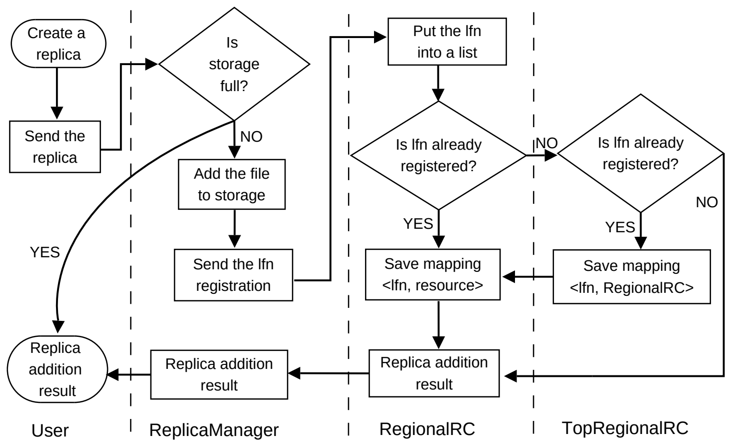


Fig. 7. Flowchart diagram for adding a replica

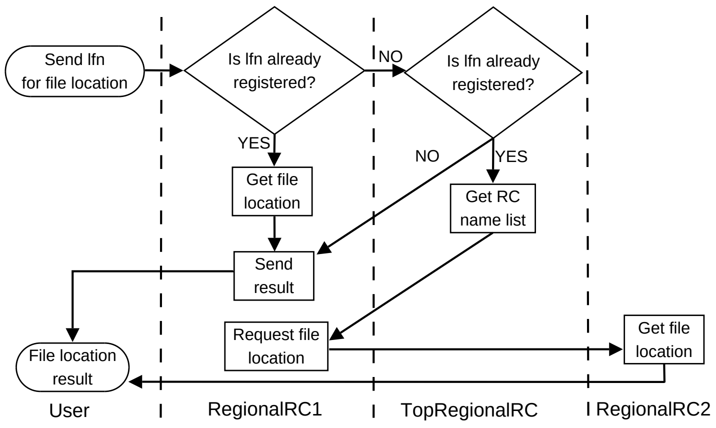


Fig. 8. Flowchart diagram for getting locations of a file

5.3 *Getting Locations of a File*

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`.

5.4 *Filtering a File*

File filtering is a mechanism that allows a user to perform complex data queries to a regional RC. This regional RC will then pass this request to the `TopRegionalRC`. Therefore, the user can 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

```
public interface Filter {
    public boolean match(FileAttribute attr);
} // end class


// create a new custom filter
public class CustomFilter implements Filter {
    private int size;
    private String name;

    public CustomFilter(int s, String lfn) {
        size = s;
        name = lfn;
    }

    // select a file based on its attributes
    public boolean match(FileAttribute attr) {
        if (attr.getSize() >= size &&
            attr.getName().startsWith(name))
        {
            return true;
        }
        else {
            return false;
        }
    }
} // end class
```

Fig. 9. A customized filter example

method, which evaluates all `FileAttribute` objects listed in the `TopRegionalRC`. An example on how to find files based on their file size and partial file name is demonstrated in Figure 9.

5.5 Deleting a Master File or Replica

The user sends a delete request to a resource. If the resource has the specified file in one of its storage elements, a request for file deregistration will be sent to the RC. If the operation is successful, the file will then be deleted from the storage and an acknowledgment message will be sent to the user. Note that a master file can only be deleted if no replicas exist in the Grid.

6 Constructing a Complex Simulation

In this section, we demonstrate how to construct a data grid simulation using the building blocks described in the previous two sections. In addition, we show on how to build complex scenarios and we illustrate the types of experiments that can be simulated on GridSim.

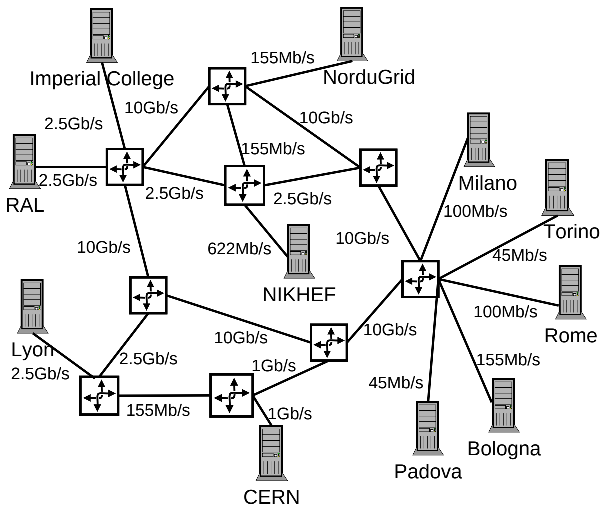


Fig. 10. The simulated topology of EU DataGrid TestBed 1

6.1 Simulation framework

In this simulation, we introduce data-intensive jobs in order to show how a demand-driven replication strategy works in a Data Grid. 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.

To handle the running of a `DataGridlet` on a resource, we modified a resource's RM to find and to transfer required files of the job before its execution. The following section gives an explanation on how the RM provides the required files to the `DataGridlet`.

6.1.1 Managing a Data-intensive Job

After receiving an incoming job or `DataGridlet`, the RM checks a list of required files for executing this job. If all the files are currently available on a resource's local storage, the `DataGridlet` is sent to a resource's scheduler (Allocation Policy) for execution. Otherwise, the RM sends a request for obtaining the needed files from other resources. When all the requested files have been transferred and stored on the local storage, then the `DataGridlet` is executed by the scheduler.

Table 3
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

6.2 Simulation Aims

We created an experiment based on EU DataGrid TestBed 1 [13], which has been used for evaluation of various data replication strategies in [3]. The network topology of the testbed is shown in Figure 10. 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-splittble unit of information, to simulate the already processed raw data of the LHC experiment.

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.

6.3 Simulation Setups

6.3.0.1 Resource Setups. Table 3 summarizes all the resource relevant information. In GridSim, total processing capability of a resource's CPU is modelled in the form of MIPS (Million Instructions Per Second) as devised by Standard Performance Evaluation Corporation (SPEC) [26].

The resource settings were obtained from the current characteristics of the real LHC testbed [21]. 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;
- 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.

6.3.0.2 Files Setups. For this experiment we defined 2000 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 [18].

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 runtime.

6.3.0.3 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 [5]. 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.

6.3.0.4 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.

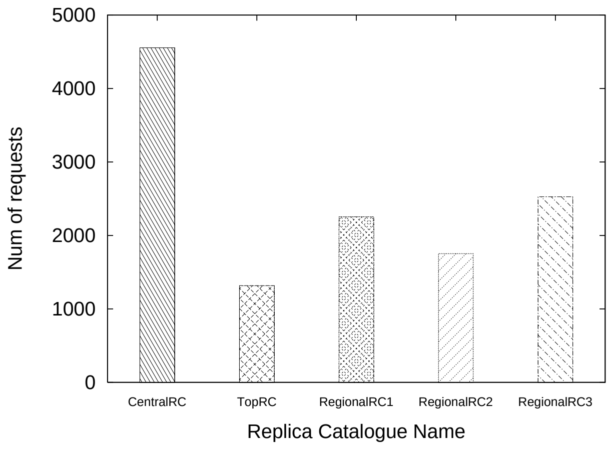


Fig. 11. Centralized vs hierarchical replica catalogue model

6.3.0.5 Users. We simulated 200 users in total, where each resource is assigned a certain number of users as depicted in Table 3. 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.

6.3.0.6 Realistic 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, which are given as input in Standard Workload format. We took 3 workload logs from the Parallel Workload Archive [30] of the DAS2 5-Cluster Grid. Since the workload logs are several months long, we took only one day from each log and simulated it on CERN, RAL and Bologna.

6.4 Simulation Results

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

In a case of a hierarchical RC model, three regional RC entities are introduced, 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 11 shows the number of requests that have to be served by each RC entity.

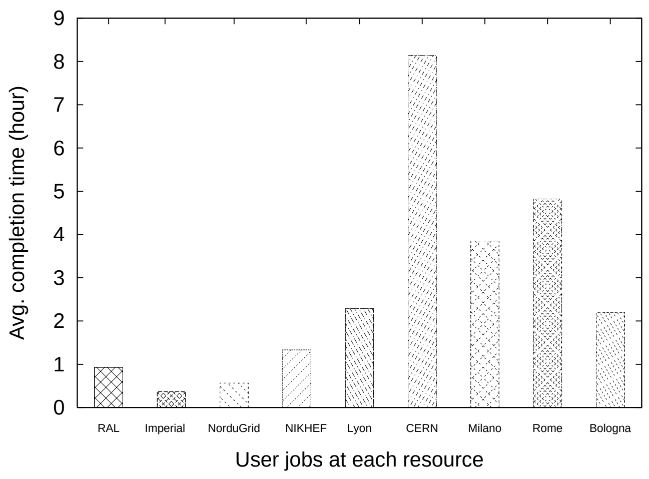


Fig. 12. Average execution time for user jobs on each resource - excluding Padova and Torino

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 Replica Catalogue 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.

6.4.0.8 Execution Time Test The average execution time for jobs on each resource is depicted in Figure 12. 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 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.

6.4.0.9 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

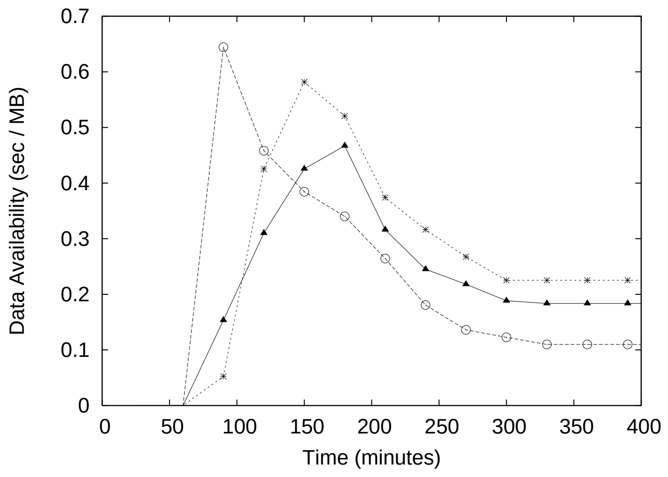


Fig. 13. Average availability in time for three resources

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 13 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.

7 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.

Furthermore, we demonstrate how these building blocks can be used to construct complex simulation scenarios, such as the simulation of data-intensive jobs and evaluation of demand-driven replication strategies.

We also show how GridSim can be used to simulate a comprehensive Data Grid platform. The conducted experiment has shown how a hierarchical model for Replica Catalogue (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 our simulator. We believe this simulation infrastructure 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 also intend to add various replica catalogue models into the framework.

References

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

[2] International Virtual Observatory Alliance. http://www.ivoa.net, 2005.

[3] W. H. Bell, D. G. Cameron, L. Capozza, A. 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*. Springer Verlag, Lecture Notes in Computer Science, November 2002.

[4] BioGrid Project. http://www.biogrid.jp, 2005.

[5] 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, 1999.

[6] R. Buyya, D. Abramson, and J. Giddy. Nimrod-G: An architecture for a resource management and scheduling system in a global computational grid. In *Proc. of the 4th International Conference and Exhibition on High Performance Computing in Asia-Pacific Region*, Beijing, China, May 14–17 2000.

[7] M. Cai, A. Chervenak, and M. Frank. A peer-to-peer replica location service based on a distributed hash table. In *Proceedings of the SC2004 Conference (SC2004)*, 2004.

[8] H. Casanova, G. Obertelli, F. Berman, and R. Wolski. The AppLeS parameter sweep template: User-level middleware for the grid. In *Proc. of IEEE Supercomputing Conference*, Dallas, USA, November 4–10 2000.

[9] 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.

[10] ChicSim – the Chicago Grid Simulator.
http://people.cs.uchicago.edu/∼krangana/ChicSim.html, 2005.

[11] C. Mihai Dobre and C. Stratan. Monarc simulation framework. In *Proc. of the RoEduNet International Conference*, Timisoara, Romania, May 27–28 2004.

[12] The Enabling Grids for E-sciencE (EGEE) Project. http://public.eu-egee.org, 2005.

[13] The European DataGrid project homepage.
http://eu-datagrid.web.cern.ch/eu-datagrid, 2005.

[14] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, 11(2):115–128, 1997.

[15] I. Foster and C. Kesselman. *The Grid: Blueprint for a new computing infrastructure*. Morgan Kaufmann Publishers, San Francisco, 1999.

[16] J. Frey, T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke. Condor-G: A computation management agent for multi-institutional grids. In *Proc. of the 10th Symposium on High Performance Distributed Computing*, San Francisco, USA, August 7–9 2001.

[17] S. Jamaloddin Golestani. A self-clocked fair queueing scheme for broadband applications. In *Proc. of IEEE INFOCOM'94*, pages 636–646, Toronto, Canada, June 12–16 1994.

[18] W. Gong, Y. Liu, V. Misra, and D. Towsley. On the tails of web file size distributions. In *Proc. of the 39-th Allerton Conference on Communication, Control, and Computing*, November 2001.

[19] A. S. Grimshaw and W. A. Wulf. The legion vision of a worldwide virtual computer. *Communications of the ACM*, 40(1):39–45, 1997.

[20] 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.

[21] LCG computing fabric.
http://lcg-computing-fabric.web.cern.ch, 2005.

[22] R. Moore, A. Rajaseka, and M. Wan. Data Grids, Digital Libraries and Persistent Archives: An integrated approach to publishing, sharing and archiving datas. *Proc. of the IEEE (Special Issue on Grid Computing)*, 93(3), 2005.

[23] Napster. http://www.napster.com, 2005.

[24] C. Simatos. Making SimJava count. MSc. Project report, The University of Edinburgh, September 12 2002.

[25] H. J. Song, X. Liu, D. Jakobsen, R. Bhagwan, X. Zhang, K. Taura, and A. Chien. The MicroGrid: A scientific tool for modelling computational grids. In *Proc. of IEEE Supercomputing Conference*, Dallas, USA, November 4–10 2000.

[26] Standard Performance Evaluation Corporation (SPEC). http://www.spec.org, 2005.

[27] H. Stockinger. *Database Replication in World-wide Distributed Data Grids*. PhD thesis, Fakultät für Wirtschaftswissenschaften und Informatik, Universität Wien, 2001.

[28] 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.

[29] 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.

[30] Parallel workload archive.
http://www.cs.huji.ac.il/labs/parallel/workload, 2005.