Extending GridSim with an Architecture for Failure Detection

Agustín Caminero [1], Anthony Sulistio [2], Blanca Caminero [1],
Carmen Carrión [1], Rajkumar Buyya [2]

[1]Department of Computing Systems,
The University of Castilla La Mancha, Spain
{agustin, blanca, carmen}@dsi.uclm.es

[2]GRIDS Laboratory,
Dept. of Computer Science and Software Engineering,
The University of Melbourne, Australia.
{anthony, raj}@csse.unimelb.edu.au

Abstract

Grid technologies are emerging as the next generation of distributed computing, allowing the aggregation of resources that are geographically distributed across different locations. However, these resources are independent and managed separately by different organizations with different policies. This will have a major impact to users who submit their jobs to the Grid, as they have to deal with issues such as policy heterogeneity, security and fault tolerance. Moreover, the changes of Grid conditions, such as resources that may become unavailable for a period of time due to maintenance and/or suffer failures, would significantly affect the Quality of Service (QoS) requirements of users. Therefore, it is essential for users to take into account the effects of resource failures during jobs execution.

In this paper, we present our work on introducing resource failures and failure detection into the GridSim simulation toolkit. As we need to conduct repeatable and controlled experiments, it is easier to use simulation as a means of studying complex scenarios. We also give a detailed description of the overall design and a use case scenario demonstrating the conditions of resources varied over time.

1 Introduction

Grid systems are the next generation of distributed computing systems. They are highly heterogeneous environments and made of a series of independent organizations sharing their resources, creating what is known as *Virtual Organization* (VO) [7]. In Grid systems, each organization or administrative domain keeps its own independence and autonomy, since it is an essential issue for the creation of a real Grid. Therefore, each organization can decide its own policy and whether to join/leave a VO at any time. Such decisions will certainly affect users in submitting and executing their jobs.

Another important issue affecting users is that the number of resources can fluctuate significantly over time. The availability of resources may vary due to changes in their working conditions, such as network congestion, partial failures or even the connection or disconnection of computing resources. With many resources in a Grid, the resource or network failures are the rule rather than the exception. Hence, they should be taken into account in order to provide a reliable service [10].

Supporting fault tolerance is one of the main technical challenges in designing Grid environments. This is because production Grid systems must be able to tolerate resource failures, while at the same time effectively exploiting the resources in a scalable and transparent manner. Therefore, in order to cope with these challenges, from the fault tolerance point of view, the system must have failure detection and recovery schemes. Such a scenario can be described in Figure 1. A failure detector monitors the components of the system (step 1), and notifies to a local Grid Information Service (GIS) entity when a failure occurs in any of them (step 2). Then, a recovery scheme is being applied in order to restart a failed job to another computational resource dynamically. An example of such a scheme is the failure monitor notifies users that resource `Res2` is out of order (step 3). Hence, users can resubmit their jobs to other resources . As demonstrated in this example, both detection and recovery schemes must be an integral part of the Grid computing infrastructure [15][11]. Therefore, it is important to conduct
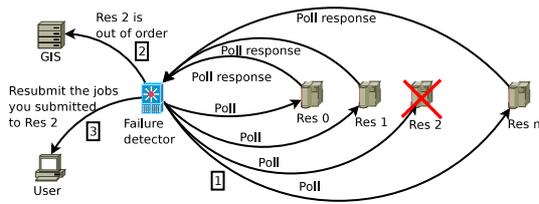
Figure 1. An example of resource failure scenario.

thorough study and evaluation of new reliability models, error detection and recovery techniques before they can be deployed in production Grid environments.

To test new detection and recovery schemes in a Grid environment like the above scenario, a lot of work is required to set up the testbeds on many distributed sites. Even if automated tools exist to do this work, it would still be very difficult to produce performance evaluation in a *repeatable* and *controlled* manner, due to the inherent heterogeneity of the Grid. 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 incorporated failure detection and recovery scheme into GridSim [2]. We opted to work on GridSim because it 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 [21]. More importantly, GridSim allows the flexibility and extensibility to incorporate new components into its existing infrastructure.

The main contribution of this work is the implementation of an extension to GridSim. This extension allows GridSim to simulate the failure of computing resources, and includes a failure detection mechanism, essential to provide a complete simulation environment. Most of the parameters of this extension are configurable, allowing researchers to simulate a wide variety of failure patterns. To evaluate our design, we simulate a failure scenario by constructing an EU DataGRID testbed into the experiment [5].

The rest of this paper is organized as follows: Section 2 briefly mentions existing methods for failure detections and several Grid simulation tools. Section 3 explains the GridSim toolkit, which is a simulation tool that has been extended to provide the failure functionality. Section 4 describes the failure functionality, including the actual implementation using GridSim. Section 5 shows a use case scenario, where we demonstrate the usefulness of our work.

Finally, Section 6 concludes the paper and suggests some guidelines for future work.

2 Related Work

2.1 Existing Resource Failure Detections

In general, computing resource failure refer to hardware, operating systems, and Grid middleware components, as well as network connections. On the failure of a resource, rescheduling and migration of jobs submitted to the failed resource should be done [10]. According to the work in [8], there are two methods for detecting resource failures, i.e. *push* and *pull*.

The *push* method uses some form of "heartbeat" messages to renew a soft-state availability registration [6]. Each monitored resource periodically sends a message to a central server indicating its availability. Missing a heartbeat after a certain time interval (timeout) T indicates that this resource has failed. Although this method is robust when the central server is running on a highly available system, it is inextricably convolving network failure and host failure. That is, a missing heartbeat or set of heartbeats can either be interpreted as the failure in the monitored resource or the loss of network connection. A real implementation using the heartbeat method can be found at [20].

The *pull* method works by sending a message or a polling request to the monitored resources. On the reception of these messages, the resources will send the message back. When the sender receives the messages back, it will understand that the resource is alive. However, if T expires before the reply message comes back, it will understand that this resource is not available at this moment. Hence, the sender will keep an up-to-date list of available resources. The *pull* method has been implemented in the Gridbus Grid Service Broker [23] and the GridWay [10] metascheduler.

2.2 Simulation Tools

As we mentioned previously, simulations are essential for carrying out research experiments in Grid systems. A number of simulation tools for Grids exist, such as GridSim [2], OptorSim [1], SimGrid [13] and MicroGrid [14]. These tools will be briefly mentioned next, except for GridSim that will be discussed in Section 3.

OptorSim has been developed as part of the EU DataGrid project [5], and it aims to study the effectiveness of data replication strategies. SimGrid is an event driven simulator, which provides functionality to simulate infrastructures and applications based on their features. Finally, MicroGrid provides on-line emulation of large-scale network and Grid resources. However, MicroGrid is actually an em-
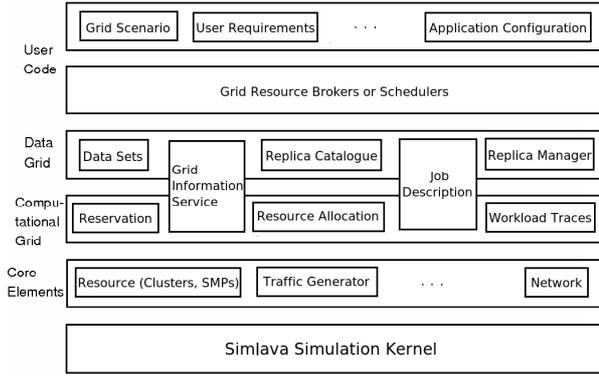
Figure 2. Architecture of GridSim.

Figure 3. Interactions among entities.

ulator, meaning that actual application code is executed on the virtual Grid modeled after Globus.

To the best of our knowledge the above tools do not provide mechanisms to simulate computing resources failure.

3 The GridSim Toolkit

The GridSim toolkit [2][22] is one of the most widely used Grid simulation tools. It has been used for simulating and evaluating VO-based resource allocation [4], workflow scheduling [18], and dynamic resource provisioning techniques[19] in Global Grids.

It supports modeling and simulation of heterogeneous Grid resources (both time- and space-shared), users, applications, brokers and schedulers in a Grid computing environment. It provides primitives for creation of application tasks, mapping of tasks to resources, and their management so that resource schedulers can be simulated to study the involved scheduling algorithms. GridSim adopts a multi-layered design architecture, as shown in Figure 2 [21].

GridSim is based on SimJava [9], a general purpose discrete-event simulation package implemented in Java. Therefore, the first layer at the bottom of Figure 2 is managed by SimJava 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. 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 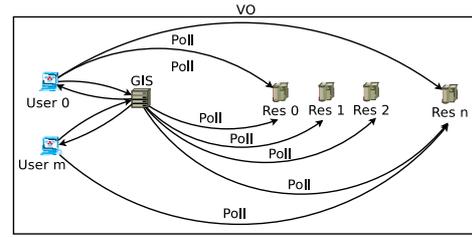management also incorporates managing data transfers between computational and storage resources. Replica catalogues are information services specifically implemented for Data Grids. The fifth layer contains components that aid users in implementing their own schedulers and resource brokers so that they can test their own algorithms and strategies. The layer above this helps users define their own scenarios and configurations for validating their algorithms.

In this paper, we only concentrate on incorporating failure detections to all layers of the GridSim architecture, except for the first layer (SimJava kernel) and the third layer (Data Grids). Work on introducing resource failures for Data Grids components will be considered as a future work. The discussion related to the failure architecture will be presented next.

4 Designing and Implementing Resource Failures into GridSim

4.1 Designing Resource Failures

In our model, we use the *pull* method, and the interactions among entities are depicted in Figure 3. These entities are briefly described as follows:

Algorithm 1 Resource Failure Detection Algorithm used by the Users.

---

repeat
    poll the resources which are running my jobs
    if a resource does not respond **then**
        ask the GIS for a list of resources
        choose one of them, based on its features and my job specification
        resubmit the jobs originally submitted to the failed resource
    end if
    wait for $T_{polling}^{user}$ seconds
until all my jobs have been successfully executed

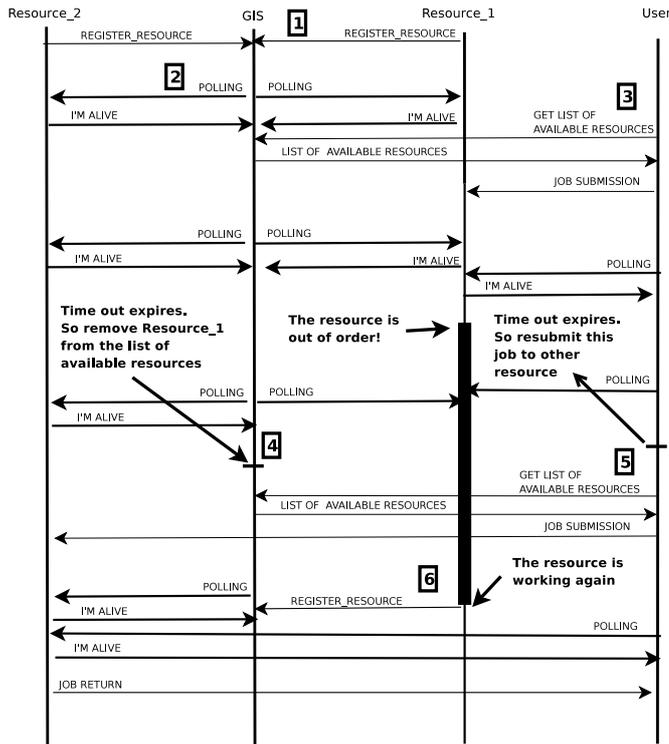- Computing resources: they are responsible for executing users' jobs.

Figure 4. A sequence diagram showing a scenario of a failure detection.

- Users: in order to know where to run jobs, users have to contact a GIS entity (explained next) for a list of available resources. The functionality of this entity can be summarized in Algorithm 1.

- GIS: is responsible for maintaining an up-to-date list of available resources. GIS entities of the same VO domain can interact each other for exchanging the information of available resources. This process can be summarized in Algorithm 2.

For enabling an efficient polling mechanism, User Datagram Protocol (UDP) is used by these entities. This is due to the fact that UDP requires a less significant network latency in comparison with a Transmission Control Protocol (TCP).

Figure 4 shows a scenario of a user that has a job in execution prior to a resource failure. The sequential steps are shown in a box with a number inside. In the initial stage, *Resource_1* and *Resource_2* register to *GIS* (step 1). Then, *GIS* creates a list of available resources. In order to keep that list up-to-date, *GIS* polls the resources periodically (step 2).

When *User* wants to run a job, he/she contacts *GIS* in order to get a list of available resources (step 3). Upon receiving the user's request, *GIS* returns its list. In that moment, *User* will choose *Resource_1* for example, based on the features of the resource and the job specification. When *User* has chosen the resource, he/she submits the job to *Resource_1* and starts a regular polling mechanism.

In the event of a failure affecting *Resource_1*, *GIS* is able to detect this problem due to the polling mechanism in place (step 4). Hence, *GIS* removes the failed resource from the list. During a routine poll, *User* discovers that *Resource_1* has failed. As a result, *User* ask *GIS* for a list of resources (step 5). When *Resource_1* recovers, it registers itself again to *GIS*(step 6). With this approach, *GIS* is able to maintain an up-to-date list of available resources.

If the failure only affects some of the machines in a resource, what happens next depends on the allocation policy of this resource. If the resource runs a *space-shared* (first come first serve) allocation policy, the jobs that are currently running on the failed machines will be terminated and sent back to users. However, when the resource runs a *time-shared* (round-robin) allocation policy, no jobs will be failed, as their execution will continue in the remaining machines of the resource. For both allocation policies, the remaining machines are responsible for responding to polling requests from users and GIS. Moreover, they are required to inform the GIS about such failure. This way, the GIS can have accurate information on the current status of the resource.

4.2 Implementing Resource Failures into GridSim

We have implemented the computing resource failure functionality on GridSim 4.0, available to download at `http://www.gridbus.org/gridsim/`. In order to provide GridSim with this new functionality, several new classes have been developed. The new classes are depicted in Figure 5 in italic-bold font. We will explain them next:

- `GridUserFailure`: as its name suggests, this class implements the behavior of the users of our grid environment. Its functionality can be summarized as fol-
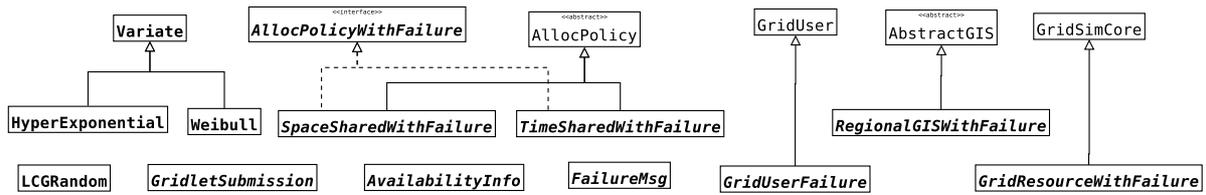
Figure 5. Classes created for the failure functionality.

lows: (1) creation of jobs; (2) submission of jobs to resources; (3) poll the resources used to run its jobs; (4) on the failure of a job, choose another resource and resubmit the failed job to it; (5) receive succeeded jobs.

- `GridResourceWithFailure`: based on `GridResource` GridSim class, this class interacts with `RegionalGISWithFailure` to set machines as failed/ working. Also it interacts with classes implementing `AllocPolicyWithFailure` to set jobs as failed.

- `AllocPolicyWithFailure`: it is an interface, which provides some functions to deal with resource failures. Each allocation pollicy implementing this interface will have a different behavior with regard to the failures.

- `SpaceSharedWithFailure`: this class is based on `SpaceShared` GridSim class, one of the allocation policies already implemented in Grid-Sim. It extends `AllocPolicy` and implements `AllocPolicyWithFailure`. It behaves exactly like First Come First Serve (FCFS). This is a basic and simple scheduler that runs each job to one Processing Element (PE). On the event of a failure, if there are *still* working machines in this resource, only the jobs which are being executed in this moment in the failing machines will be failed, and sent back to the user. If there are *no* working machines in this resource, then all the jobs in this resource will be failed and sent back to the user, but with an special code, so that the user can make out that this resource is out of order. The most realistic behavior, in a real grid, would be not sending the jobs back to the user when all the machines of the resource are out of order. But the simulator does not deal well with entities (in this case, users) waiting for an event (in this case, a job) that never arrives. So we decided to simulate the real behavior by using that special code.

- `TimeSharedWithFailure`: this class is based on `TimeShared` GridSim class, other allocation policy already implemented in Grid-Sim. It extends `AllocPolicy` and implements `AllocPolicyWithFailure`. It behaves similar to a round robin algorithm, except that all jobs are executed at the same time. This is a basic and simple scheduler that runs each job to one Processing Element (PE). On the event of a failure, if there are *still* working machines in this resource, no job will be failed, as jobs are not tightly allocated to a machine. If there are *no* working machines in this resource, then its behavior will be the same as explained for the previous class.

- `RegionalGISWithFailure`: this class is based on `RegionalGIS` GridSim class. The difference between these classes is that `RegionalGISWithFailure` provides support for resource failures. To do that, several functions and parameters have been added.

 The main new parameters are `NumResPattern_`, `ResPattern_`, `TimePattern_` and `LengthPattern_`, all of them in discrete, continuous and variate versions. This parameters allow random number generators based on continuous distributions (like Uniform distribution), discrete distributions (like Poisson distribution) and variate distributions (like HyperExponential distribution). These parameters are used to choose the number of resources that will fail in a simulation, which resource will fail, when and how long the failure will be. The `NumResPattern_` is reused to choose the number of machines failing at each resource. This parameters allow a flexible configuration of the simulation tool, so that a wide variety of failure patterns can be studied.

- `AvailabilityInfo`: This class is used to implement the polling mechanism. The user and GIS send objects of this class to the resources, which in turn send it back, as was explained before. When the resource still has some working machines, it will send this object back with no delay, but when all the machines of the resource are out of order, it does it with some delay and an special code. This is done to simulate a real situation, in which a time out defines when a resource is not available if it has not replied to the poll. We do it
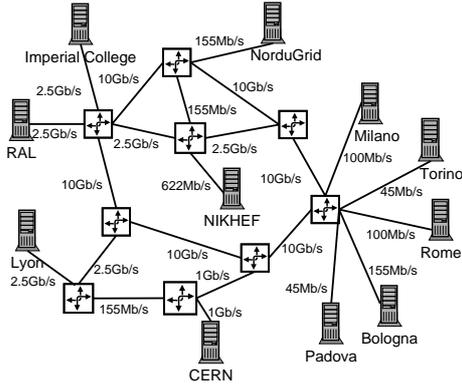
Figure 6. EU DataGRID Testbed 1.

Resource Name (Location)	# Nodes	CPU Rating	Policy	VO
RAL (UK)	41	49,000	Space-Shared	2
Imp. College (UK)	52	62,000	Space-Shared	2
NorduGrid (Norway)	17	20,000	Space-Shared	3
NIKHEF (Netherlands)	18	21,000	Space-Shared	3
Lyon (France)	12	14,000	Space-Shared	0
CERN (Switzerland)	59	70,000	Space-Shared	0
Milano (Italy)	5	70,000	Space-Shared	1
Torino (Italy)	2	3,000	Time-Shared	1
Rome (Italy)	5	6,000	Space-Shared	1
Padova (Italy)	1	1,000	Time-Shared	4
Bologna (Italy)	67	80,000	Space-Shared	4

Table 1. Resource specifications.

User Location	# Users	Primary VO	Secondary VO
RAL (UK)	12	2	4
Imperial College (UK)	16	2	0
NorduGrid (Norway)	4	3	2
NIKHEF (Netherlands)	8	3	4
Lyon (France)	12	0	1
CERN (Switzerland)	24	0	1
Milano (Italy)	4	1	2
Torino (Italy)	2	1	3
Rome (Italy)	4	1	4
Padova (Italy)	2	4	3
Bologna (Italy)	12	4	0

Table 2. The allocation of VO domains to users.

this way because the simulator does not deal well with entities waiting for an event that never arrives. Hence, if an entity keeps waiting for an event that never arrives, it makes simulations run with no end.

- `GridletSubmission`: This class is used to keep track of each job, so that we know whether a job has already been submitted or not.

- `FailureMsg`: This class implements the way that the `RegionalGISWithFailure` communicates with the `GridResourcesWithFailure` to simulate a resource failure. Recall that in this implementation, it is the GIS the entity which tells the resource when, how and for how long they should fail. So, this class is used for that communication.

- `Variate` and `LCGRandom`: These classes are for random number generation, and belong to the JSIM [16] simulation tool.

- `HyperExponential` and `Weibull`: As the previous classes, these ones belong to the JSIM simulation tool. They generate random number based on hyperexponential and Weibull distributions, respectively.

5 Use Case Scenario

In this section, we provide a scenario of the new resource failure functionality. We have created an experiment based on the EU DataGRID Testbed 1, as shown in Figure 6 [5], which has been used to evaluate data replication strategies in [1].

Table 1 summarizes the characteristics of simulated resources, which were obtained from a real LCG testbed [12]. The parameters regarding to a CPU rating is defined in the form of MIPS (Million Instructions Per Second) as per SPEC (Standard Performance Evaluation Corporation)

benchmark. Moreover, the number of nodes for each resource have been scaled down by 10, because of memory limitation on the computer we ran the experiments on. The complete experiments would require more than 2GB of memory. Finally, each resource node has four CPUs.

For this experiment, we have five VO domains and each resource belongs to one of them as shown in Table 1. The VO mapping is done by taking into account a geographical proximity between the resources.

For this experiment, we created 100 users and distributed them among the VO domains, as shown in Table 2. Each user has 10 jobs and each job takes about 10 minutes if it is run on the CERN resource. Each user belongs to two different VOs and submits jobs to resources from the primary VO. The secondary VO is chosen at random and it is used only when all of resources from the primary VO have failed.

To simplify the experiment set-up, some parameters are

VO	# CPUs	# Failed CPUs	# Jobs	# Failed Jobs	Mean Failure Time
VO_0	71	25	360	219	2.76 h.
VO_1	12	5	100	20	103.36 h.
VO_2	93	24	280	96	5.25 h.
VO_3	35	35	120	120	15.82 h.
VO_4	68	6	140	96	9.5 h.

Table 3. Resource failure statistics.

identical for all network elements, such as the maximum transfer unit (MTU) of links is 1,500 bytes and the latency is 10 milliseconds.

As mentioned previously, the GIS uses a probabilistic distribution on deciding how many resources fail. Therefore, we use a hyperexponential distribution for generating a failure model, because it is suitable for representing availability of resources in different computing environments [17]. We chose the mean of this distribution to be a half of the total CPUs of each VO domain. We assume that each VO contains only one GIS entity.

Once we have explained the experiment setups used in our simulations, we will show the simulation results. These results depict that we have been able to efficiently simulate the failure of computing resources in Grids.

We have modeled the dynamic behavior of the Grid system, as shown in Figure 7(a). This figure depicts the total availability for each VO domain varied throughout the simulation time due to resource failures. VO_0 and VO_1 suffered a big drop compared to others due to a fact that powerful CPUs suffered a failure.

In Figure 7, we can see a time-line of one resource from each VO, and another time-line of User_0. These time-lines display more interesting views on both resources and the user. Figure 7(a) shows the period of a resource failure on each VO. For simplicity, we assume that failed machines in the same resource have the same start and finish period. In addition, each resource has given a failure notice only once.

Figure 7(b) shows an event from User_0 of VO_0. Initially, the user submits 10 jobs to resources from VO_0. Around 100 seconds of simulation time, a failure is happening at CERN and the user detects this problem. Unfortunately, one of the failed machines is running the user's job. Hence, this job is being migrated to another machine. The same scenario applies to other four jobs. In the end, all jobs are completed successfully, where some of them finished at time 600 second and the rests at time 1200 seconds. The time difference is because the last four jobs were resubmitted to a busy resource, hence they were enqueued.

Table 3 presents statistics regarding to the number of failed machines, mean failure time, and how many jobs have failed because of that. For VO_4, there is a big amount

of failed jobs compared to the number of failed machines. This is due to a failure of the whole resource that belongs to this VO. More precisely, all the machines in Padova failed, hence all the jobs waiting or being executed in Padova failed also. Similarly, we can see that on an exact period of time, all NorduGrid and NIKHEF machines failed in VO_3 (also shown in Figure 7(a)). Hence, users of VO_3 have to submit all of their affected jobs to their secondary VO. At the end, all jobs were successfully executed.

6 Conclusion and Future Work

Grid systems is a hot topic in distributed systems research at this moment. In order to carry out this research efficiently, simulations are absolutely essential. Hence, simulation tools should cover the main features of real Grid systems, but up to now it is not easy to find a simulation tool covering resource failures and detection mechanisms.

In this paper, we have presented an extension to Grid-Sim, which is one of the most widely used simulation tools. Failure models are created by means of probabilistic distributions with fully configurable parameters, so that researchers will be able to decide the way how these failures take place. To support the usefulness of our work, we have presented simulation results based on a real Grid testbed.

By means of this new functionality, researchers will be able to create more realistic Grid models. In turn, this will help them exploring their research projects to different fields of Grid computing, such as Grid scheduling, fault tolerance, and resource discovery.

As for future work, we are planning to use the improved simulation tool to carry out research aimed at providing network QoS in Grid. This will be done by integrating this functionality into the network broker outlined in [3]. Also, in order to make our research more realistic, new extensions regarding network link failures and finite buffers will be added to GridSim.

References

[1] W. H. Bell, D. G. Cameron, L. Capozza, A. P. Millar, K. Stockinger, and F. Zini. Simulation of dynamic grid repli-

(a) Total availability of each VO domain   (b) Period of a resource failure in each VO   (c) Time-line of User_0
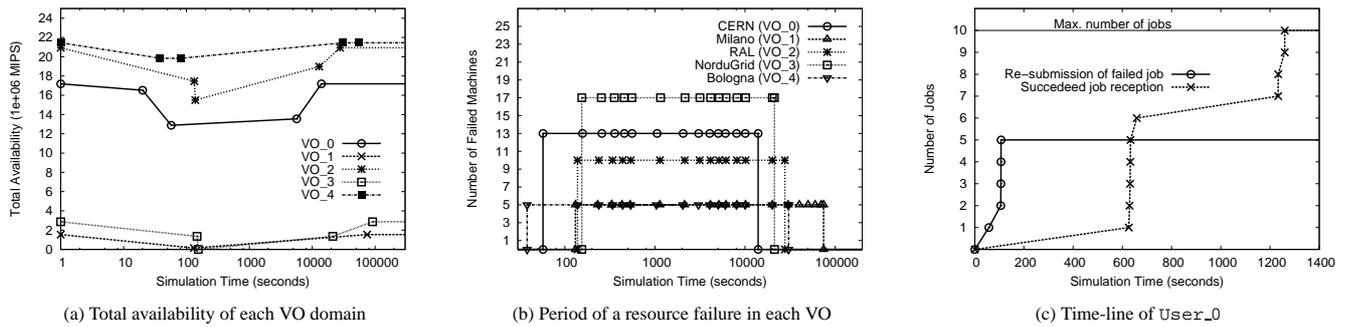
Figure 7. Time-lines showing the progress of resources and User_0.

cation strategies in OptorSim. In *GRID '02: Proceedings of the Third Intl. Workshop on Grid Computing*, pages 46–57, London, UK, 2002. Springer-Verlag.

[2] 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*, 14:1175–1220, Nov-Dec 2002.

[3] A. Caminero, C. Carrion, and B. Caminero. On the improvement of the network QoS in a grid environment. In *Proceedings of the 4th Intl. Workshop on Middleware for Grid Computing - MGC 2006*. ACM, 2006.

[4] E. Elmroth and P. Gardfjäll. Design and evaluation of a decentralized system for grid-wide fairshare scheduling. In *eScience*, pages 221–229. IEEE Computer Society, 2005.

[5] European DataGrid Project. http://eu-datagrid.web.cern.ch/eu-datagrid.

[6] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. *Grid Computing: Making the Global Infrastructure a Reality*, chapter The Physiology of the Grid. Wiley InterScience, 2003.

[7] I. T. Foster. The anatomy of the Grid: Enabling scalable virtual organizations. In *CCGRID*, pages 6–7. IEEE Computer Society, 2001.

[8] N. Hayashibara, A. Cherif, and T. Katayama. Failure detectors for large-scale distributed systems. In *The 21th IEEE Symposium on Reliable Distributed Systems, (SRDS '02)*, pages 404–409, Washington - Brussels - Tokyo, Oct. 2002. IEEE.

[9] F. Howell and R. McNab. SimJava: A discrete event simulation library for Java. In *International Conference on Web-Based Modeling and Simulation*. Society for Computer Simulation Intl. (SCS), 1998.

[10] E. Huedo, R. S. Montero, and I. M. Llorente. A framework for adaptive execution in grids. *Softw. Pract. Exper.*, 34(7):631–651, 2004.

[11] G. Kola, T. Kosar, and M. Livny. Phoenix: Making data-intensive grid applications fault-tolerant. In R. Buyya, editor, *GRID*, pages 251–258. IEEE Computer Society, 2004.

[12] LCG Computing Fabric Area. http://lcg-computing-fabric.web.cern.ch.

[13] A. Legrand, L. Marchal, and H. Casanova. Scheduling distributed applications: The SimGrid simulation framework. In *Proc. of the Third IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid'03)*, Tokyo, Japan, May 12–15 2003.

[14] X. Liu. *Scalable Online Simulation for Modeling Grid Dynamics*. PhD thesis, Univ. of California at San Diego, 2004.

[15] R. Medeiros, W. Cirne, F. V. Brasileiro, and J. P. Sauvé. Faults in grids: Why are they so bad and what can be done about it? In H. Stockinger, editor, *GRID*, pages 18–24. IEEE Computer Society, 2003.

[16] J. A. Miller, R. S. Nair, Z. Zhang, and H. Zhao. JSIM: A JAVA-based simulation and animation environment. In *30th Annual Simulation Symposium (SS '97)*, pages 31–42. IEEE Computer Society, 1997.

[17] D. Nurmi, J. Brevik, and R. Wolski. Modeling machine availability in enterprise and wide-area distributed computing environments. In *Proceedings of 11th Intl. Euro-Par Conference*, volume 3648 of *Lecture Notes in Computer Science*, pages 432–441. Springer, 2005.

[18] A. Ramakrishnan, G. Singh, H. Zhao, E. Deelman, R. Sakellariou, K. Vahi, K. Blackburn, D. Meyers, and M. Samidi. Scheduling data-intensiveworkflows onto storage-constrained distributed resources. *CCGRID*, 0:401–409, 2007.

[19] G. Singh, C. Kesselman, and E. Deelman. A provisioning model and its comparison with best-effort for performance-cost optimization in grids. *IEEE Intl. Symposium on High Performance Distributed Computing (HPDC)*, 2007.

[20] P. Stelling, C. DeMatteis, I. T. Foster, C. Kesselman, C. A. Lee, and G. von Laszewski. A fault detection service for wide area distributed computations. *Cluster Computing*, 2(2):117–128, 1999.

[21] A. Sulistio, U. Cibej, R. Buyya, and B. Robic. A tool for modelling and simulation of data grids with integration of data storage, replication and analysis. Technical Report GRIDS-TR-2005-13, GRIDS Laboratory, The University of Melbourne, Australia, 2005.

[22] A. Sulistio, G. Poduval, R. Buyya, and C.-K. Tham. On incorporating differentiated levels of network service into GridSim. *Future Generation Computer Systems*, 23(4):606–615, May 2007.

[23] S. Venugopal, R. Buyya, and L. J. Winton. A Grid service broker for scheduling e-Science applications on global data Grids. *Concurrency and Computation: Practice and Experience*, 18(6):685–699, May 2006.