University of Castilla-La Mancha

A publication of the

Computing Systems Department

Extending GridSim to Provide Computing Resource Failures

by

Agustín Caminero, Blanca Caminero, Carmen Carrión

Technical Report **#DIAB-07-01-1** January, 2007

This work has been carried out during a research stay of the first author in the GRIDSLab of the University of Melbourne (Australia), funded by the following projects: Consolider CSD2006-46, CICYT TIN2006-15516-C04-02, PBC-05-007-01, PBC-05-005-01 and José Castillejo grant.

DEPARTAMENTO DE SISTEMAS INFORMÁTICOS
ESCUELA POLITÉCNICA SUPERIOR
UNIVERSIDAD DE CASTILLA-LA MANCHA
Campus Universitario s/n
Albacete - 02071 - Spain
Phone +34.967.599200, Fax +34.967.599224

Extending GridSim to Provide Computing Resource Failures

Agustín Caminero, Blanca Caminero, Carmen Carrión
Computing Systems Department. Escuela Politécnica Superior
Universidad de Castilla-La Mancha. 02071 - Albacete, SPAIN
{agustin, blanca, carmen}@dsi.uclm.es

March 8, 2007

Abstract

Grid technologies are emerging as the next generation of distributed computing, allowing the aggregation of resources that are geographically distributed across different locations. One of the key points of grid computing is that the independence and autonomy of the resources are preserved at all times. This means that the administrator of a resource connected to a grid will make decisions as if the resource were isolated. So he may, e.g., disconnect the resource from the grid. Moreover, resources may suffer failures, and all of this obviously would affect the performance received by the users. In this paper we present an extension to one of the most populars grid simulators, GridSim [11], to support variable resource availability.

1 Introduction and motivation

Grid technologies are emerging as the next generation of distributed computing, allowing the aggregation of resources that are geographically distributed across different

locations. Grid systems are highly variable environments, made of a series of independent organizations that share their resources, creating what is known as *virtual organization, VO*. One of the key points of grid computing is that the independence and autonomy of the resources is preserved at all times. This means that the administrator of a resource connected to a grid will make decisions as if the resource were isolated. So he may, e.g., disconnect the resource from the grid. Moreover, resources may suffer failures, and all of this obviously would affect the performance received by the users.

Simulators are a basic tool to carry out research in many different fields. Up to now, grid simulators could simulate a wide variety of features, but were not able to simulate the variable availability of resources. So, providing computing resources failure simulation is key in order to simulate a real grid system.

The paper is structured as follows: first of all, we enumerate some grid simulators, explaining their main features, and stressing the fact that none of them can simulate the variable availability of resources. One of the simulators explained is GridSim, which is the simulator that has been extended to support this new functionality. After that, we concentrate our attention in this new functionality, explaining the new behavior and the actual implementation. The last, we present a sample output of the GridSim simulator including this new feature, and draw some conclusions.

2 Related Work

A number of simulation tools have been developed in order to carry out research in the field of grid systems, but they do not provide mechanisms to simulate computing resources failure. Some of those simulators will be commented the next, paying special attention to GridSim [11] as it is the simulation tool we use for our developments.

OptorSim is a grid simulator designed to test dynamic replication strategies used in optimizing data location within a grid. Each simulated site contains several storage or computing elements. Simulated jobs run and file accesses may trigger replication [5]. OptorSim is a Data Grid simulator, written in Java, which has been developed in the framework of the EU DataGRID project. The goal of OptorSim is to allow experimentation with and evaluation of various scheduling and replica optimization algorithms. Using a grid configuration and a replica optimizer algorithm as input, OptorSim runs a number of grid jobs on the simulated grid. It also allows a user to visualize the performance of the algorithm [1].

SimGrid [2] is a toolkit that provides core functionalities for the simulation of distributed applications in heterogeneous distributed environments. The specific goal of the project is to facilitate research in the area of distributed and parallel application scheduling on distributed computing platforms ranging from simple network of workstations to Computational Grids.

The MicroGrid [3] provides online simulation of large-scale (20,000 router, thousands of resources) network and grid resources. By creating a virtual grid environment in which existing middleware and applications can be run unchanged, detailed study of complex dynamic behavior such as scaling, failure responses, and other emergent behavior can be explored. The MicroGrid seems to be and ended project, as the last release came out in 2004.

2.1 The GridSim Toolkit

GridSim [11] 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. The first bottom layer is the portable and scalable Java interface and runtime environment called Java Virtual Machine (JVM), whose implementation is available for single and multiprocessor systems including clusters. The second layer is SimJava which provides an event-driven discrete event infrastructure on top of the JVM to drive the simulation for GridSim. The third layer is the GridSim toolkit itself, which provides the modeling and simulation of core grid entities such as resources and information services using the discrete event services defined by the second layer. The fourth layer provides the simulation of resource aggregators called grid brokers or schedulers. The last top layer focuses on application and resource modeling with different scenarios to evaluate scheduling and resource management policies, heuristics, and algorithms [12].

3 New Feature of GridSim: Computing Resource Failure

3.1 The Implementation

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 classes have been developed. These classes are depicted in Figure 2, 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. This is a quite simple class, whose functionality can be summarized as follows:

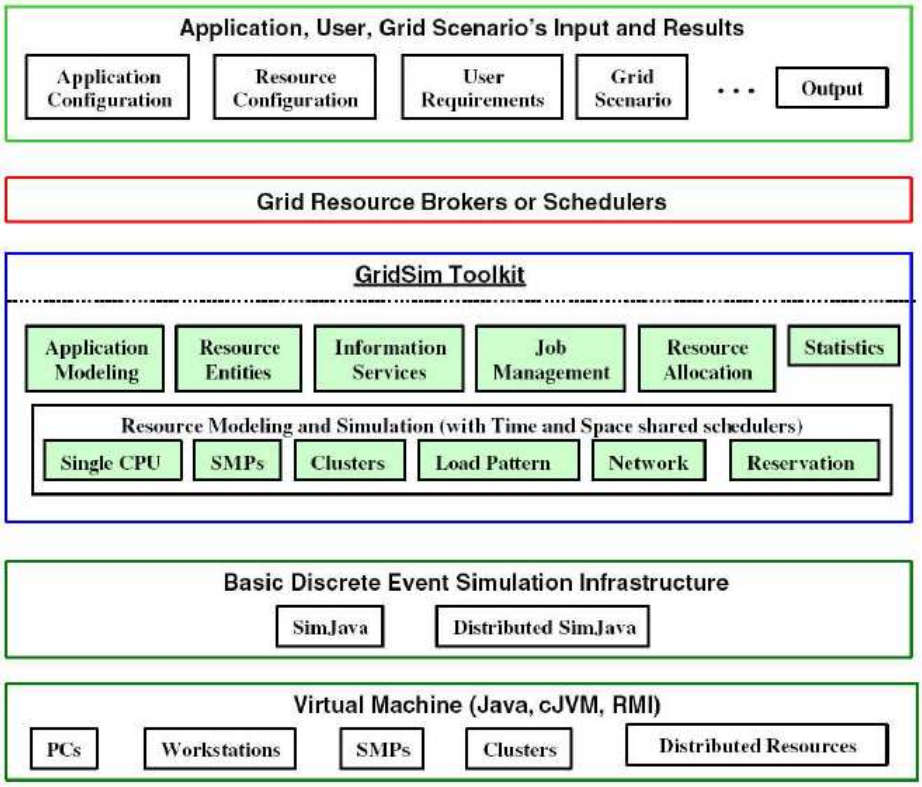


Figure 1: System architecture of GridSim Toolkit [6].

  - Creation of jobs.
  - Submission of jobs to resources, which are chosen at random.
  - On the failure of a gridlet, choose another resource and re-submit the failed gridlet to it.
  - Poll the resources used to run its gridlets, discover the failure of resources, and resubmit the gridlets.
  - Receive succeeded gridlets.

- `AllocPolicyWithFailure`: it is an interface, which provides some functions to deal with resource failures.

- `AllocPolicy`: it is an abstract class, one of the GridSim classes. Implements the general behaviour of the scheduling algorithm of the resources.

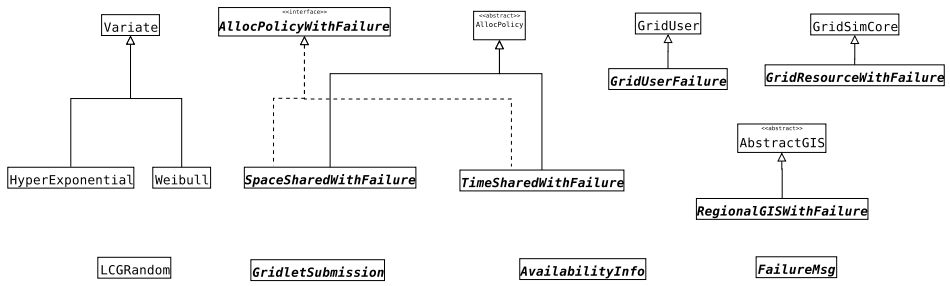


Figure 2: Classes created for the failure functionality.

- `SpaceSharedWithFailure`: this class is based on `SpaceShared` GridSim class, one of the allocation policies already implemented in GridSim. 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 Gridlet to one Processing Element (PE). If a Gridlet requires more than one PE, then this scheduler only assign this Gridlet to one PE. What makes `SpaceSharedWithFailure` different from `SpaceShared` are the functions already mentioned for `AllocPolicyWithFailure`.

- `TimeSharedWithFailure`: this class is based on `TimeShared` GridSim class, other allocation policy already implemented in GridSim. It extends `AllocPolicy` and implements `AllocPolicyWithFailure`. It behaves similar to a round robin algorithm, except that all Gridlets are executed at the same time. This is a basic and simple scheduler that runs each gridlet to one Processing Element (PE). If a gridlet requires more than one PE, then this scheduler only assign this Gridlet to one PE. As the previous class, what makes `TimeSharedWithFailure` different from `TimeShared` are the functions already mentioned for `AllocPolicyWithFailure`.

- `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 or modified:

- `failureNumResPattern_`, `failureResPattern_`, `failureTimePattern_`, `failureLengthPattern_`, 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 `failureNumResPattern_` is reused to choose the number of machines failing at each resource. Also, some functions to get and set these parameters, and get samples have been developed.
- `init(...)`: initialize this entity.
- `GRIDRESOURCE_FAILURE, GRIDRESOURCE_RECOVERY` events, to manage the resource failures.
- `GRIDRESOURCE_POLLING` event, to manage the polling mechanism.
- `processOtherEvent(...)`: as new events have been added, these events should be processed in this method.
- `registerOtherEntity(...)`: needed to start the failure process.
- `processEndSimulation(...)`: clears lists and shutdowns other entities.
- `getResourceCharacteristics(...)`: retrieves the characteristics of a given resource.
- `beginning`: this parameter defines when the simulation has just began, or when it has been running for a while.
- `pollResource(...)`: used to send messages to resources, in order to know which of them are working properly.
- `pollReturn(...)`: used to receive the responses from resources.

- `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.

- `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` commnunicates 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 shoud fail. So, this class is used for that communication.

- `Variate`: This class is for random number generation via an installable random number generator. This class belongs to the JSIM [9] simulation tool.

- `HyperExponential`: As well as the previous class, this class belongs to the JSIM simulation tool. This class generate a hyperexponentially distributed random number mean *mu* and standard deviation *sigma* (sigma ¿ mu) using Morse's two-stage hyperexponential distribution.

- `Weibull`: This class also belongs to the JSIM simulation tool, and it is a Weibull random variate generator.

- `LCGRandom`: Another class from the JSIM simulation tool. It is a basic random number generator. It is a multiplicative Linear Congruential Generator (LCG).

3.2 The Functionality

Basically, this new functionality works as follows:

- At the begining of the simulation, the resources register themselves at one of the RegionalGIS (Grid Information Service) entities.
- After that, at a random moment during the simulations, each of the GIS entity tells one or more resources that some of their machines are going to fail.
- In that moment, depending on the allocation policy that the failing resource is running, and how many machines fail, it will proceed in a different way:
  - Space-shared policy (similar to FCFS): if there are *still* working machines in this resource, only the gridlets 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 gridlets in this resource will be failed and sent back to the user, but with a different 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 gridlets 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 gridlet) that never arrives. So we decided to simulate the real behavior by using that special code.
  - Time-shared policy (similar to round-robin): if there are *still* working machines in this resource, no gridlet will be failed, as gridlets are not tightly allocated to a machine. If there are *no* working machines in this resource, then all the gridlets in this resource will be failed and sent back to the user, with the same special code explained before.

- On the reception of a failed gridlet (with normal code), the user will query the GIS entity for the list of available resources, will choose one of them, and will submit the gridlet to that resource.

- On the reception of a failed gridlet with the special code, the user will schedule the resubmission of that gridlet for a given moment in the future. This moment will be based on the polling time of the user. This is done like this to simulate a polling mechanism, which would allow the user to discover the failure of a resource.

- The GIS entity will, from time to time, poll all the resources. When a resource responds to that poll, then it is working, although some of its machines may be out of order. If all the machines of the resource are out of order, then the resource would respond the poll with an special code. This is done because the simulator, as was mentioned before, does not deal well with entities (in this case, the GIS) waiting for an event (in this case, a poll response) that never arrives. In this case, the resource would wait for a given time before sending the response to the user. This is done to simulate the real behavior of a polling mechanism, where the GIS would send the poll, wait for the response during a given time, and, on the time out, set the resource as unavailable. If a resource is not available, then the GIS will remove that resource from its list of available resources.

- In order to perform the recovery, the GIS entity will tell the failed resources to recover.

- When a resource where all the machines were out of order recovers from that failure, it will proceed to register to the GIS again.

- When all the gridlets have been successfully executed, the simulation will finish.

In the Figure 3 we can see a sequence diagram explaining the behavior of the computing resource failure functionality of GridSim. For the sake of clarity, we have omitted the polling mechanism, which is depicted in Figure 4. A more in depth explanation of that functionality is the following, where the steps refer to the numbers in the Figure 3:

- In the beginning, simulations start running, and the `RegionalGISWithFailure` entity schedules a `GRIDRESOURCE_FAILURE` event to itself for a random moment in the future. The moment when this event takes place will be the starting point of the failure functionality. At the same time, it will schedule a `GRIDRESOURCE_POLLING` event to itself for a given moment in the future to start the polling process. The polling time for the GIS will be twice that of the user. All of this happens in the `registerOtherEntity()` function, and we do not do anything else in this moment because resources still have not registered themselves, so the `RegionalGISWithFailure` entity does not know how many of them there are (step 1).

- Also, users schedule a `SUBMIT_GRIDLET` event for a random noment in the future, so that all the users start submitting their gridlets at a different moment. On the reception of this event, the user will query the GIS for a list of available resources. Then, he will choose one of them and will submit the gridlet (step 2).

- On the reception of the `GRIDRESOURCE_FAILURE` event (function `processOtherEvent(...)`), `RegionalGISWithFailure` starts the failure functionality. What this entity does the next is check the list of available resources and choose how many of them will fail, and when each resource will fail. Then, it will schedule as many `GRIDRESOURCE_FAILURE` events to itself as resources will fail, each event for the moment when the resource will fail (in the Figure 3, only one resource will fail (step 3).

- On the reception of this second `GRIDRESOURCE_FAILURE` event (function `processOtherEvent(...)`), the `RegionalGISWithFailure` entity will choose which resource will fail, among those which are totally available at this moment. This means that resources which have any of their machines failed at this moment will not be chosen. Besides, it will choose how many machines will fail on this resource, and how many hours the failure will last. Then, the GIS entity will forward the `GRIDRESOURCE_FAILURE` event to the resource chosen, including the information mentioned before. The last thing that the GIS entity will do is schedule another event for itself. This time the event is a `GRIDRESOURCE_RECOVERY` and, as its name suggests, it will denote the time when the resource will be back to life again. For the sake of easiness, we have the assumption that all the failed machines of a resource will get failed and be recovered at the same time (step 4).

- When the computing resource (`GRIDRESOURCE_FAILURE` entity) receives the `GRIDRESOURCE_FAILURE` event, then it will set some of its machines as out of order, as many machines as the data coming along with the event say. Then, all the processing elements (PEs) of that machine will be set as out of order as well (step 5). Depending on the allocation policy of the resource, and how many machines fail, several things may happen:

  - **Space-shared allocation policy** (similar to FCFS): if there are *still* working machines in this resource, then the gridlets allocated to the failed PEs will be returned back to the user, and their status set to FAILED. The gridlets which have been allocated to other PEs would go on without problems, as well as the gridlets waiting in the queue. For the following allocations of gridlets to PEs, only the working PEs will be taken into account by the allocation policy. If there are *no* working machines in this resource, then all the gridlets will get their status set to FAILED_RESOURCE_UNAVAILABLE and sent back to the user. The status FAILED_RESOURCE_UNAVAILABLE

is the special code mentioned before, used only to allow simulations to finish properly. Otherwise, the user entity would get jammed, as it is waiting for an event that never arrives.

  - **Time-shared allocation policy** (similar to Round-Robin): if there are *still* working machines in this resource, then no gridlet will fail, as this policy does not allocate tightly gridlets to PEs. So, when a machine (and all its PEs) fails, then that machine becomes no eligible to run more gridlets, but gridlets still run in the other machines. If there are *no* working machines in this resource, then all the gridlets will get their status set to FAILED_RESOURCE_UNAVAILABLE and sent back to the user.

  In the Figure 3, the failed resource has only one machine with one processing element (PE). So, as there are no working machines in the failed resource, the behavior shown is that of both allocation policies.

- When the user (`GridUserFailure` entity) receives the failed gridlets (with the normal code), he will contact the GIS entity in order to get a list of available resources, will choose one of them, and will send the failed gridlets to this new resource. Note that if there are still working machines in the failed resource, then the user may send the gridlets again to the same resource (step 6).

- When the user receives a gridlet with the special code (FAILED_RESOURCE_UNAVAILABLE), he will reschedule the new submission for a given moment in the future. This moment will be decided based on the polling time (T_p) of the user. We do this because we want to simulate the polling mechanism of a real system. In a real system, every T_p seconds, the user would poll the resources used to run his gridlets. If a resource does not respond to the poll, then the user would understand that the resource is not available at this moment, so he would resubmit those gridlets already submitted to the failed

resource. So, in the worst case (the resource fails just after responding a poll), the user would take T_p seconds to discover the failure.

- On the reception of the `GRIDRESOURCE_RECOVERY` event, the GIS will just forward it to the resource, meaning that the failure is over (step 7).

- When a resouce receives a `GRIDRESOURCE_RECOVERY` event, it just sets all it machines as working.

- Every time that the GIS receives a `GRIDRESOURCE_POLLING` event, it will poll all the resources in the list of available resources. Resources where at least one of the machines are still working will respond to the poll. The resources where all the machines are out of order will wait for a given time before sending a response with an special code. This is the way we skip the inability of the GridSim simulator to deal with entities waiting for an event which never comes. On the reception of the response, the GIS will remove that resource from the list of available resources. So the next time an user queries it, the failed resource will not be among the resources returned to the user. After performing the polling mechanism, the GIS will schedule another `GRIDRESOURCE_POLLING` event for the future. The polling mechanism can be seen in the Figure 4.

Once we have expalined in depth the new functionality added to GridSim, we will present a sample output of the simulator running this functionality.

3.3 Sample Output

In this section, we provide a scenario of the new resource failure functionality. We have created an experiment based on EU DataGRID Testbed 1, as shown in Figure 5 [7], which has been used to evaluate data replication strategies in [4].

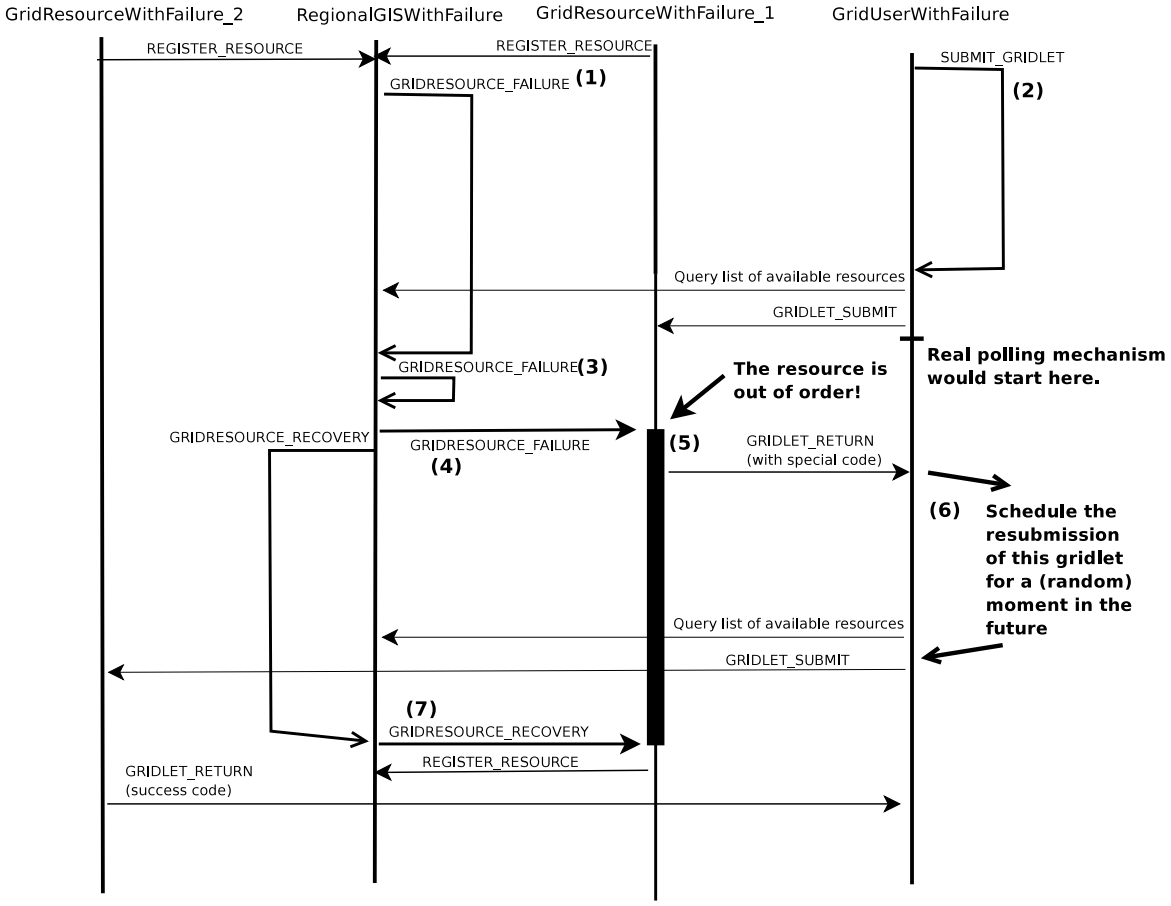


Figure 3: Sequence diagram for a resource failure.

Table 1 summarizes the characteristics of simulated resources, which were obtained from a real LCG testbed [8]. 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.

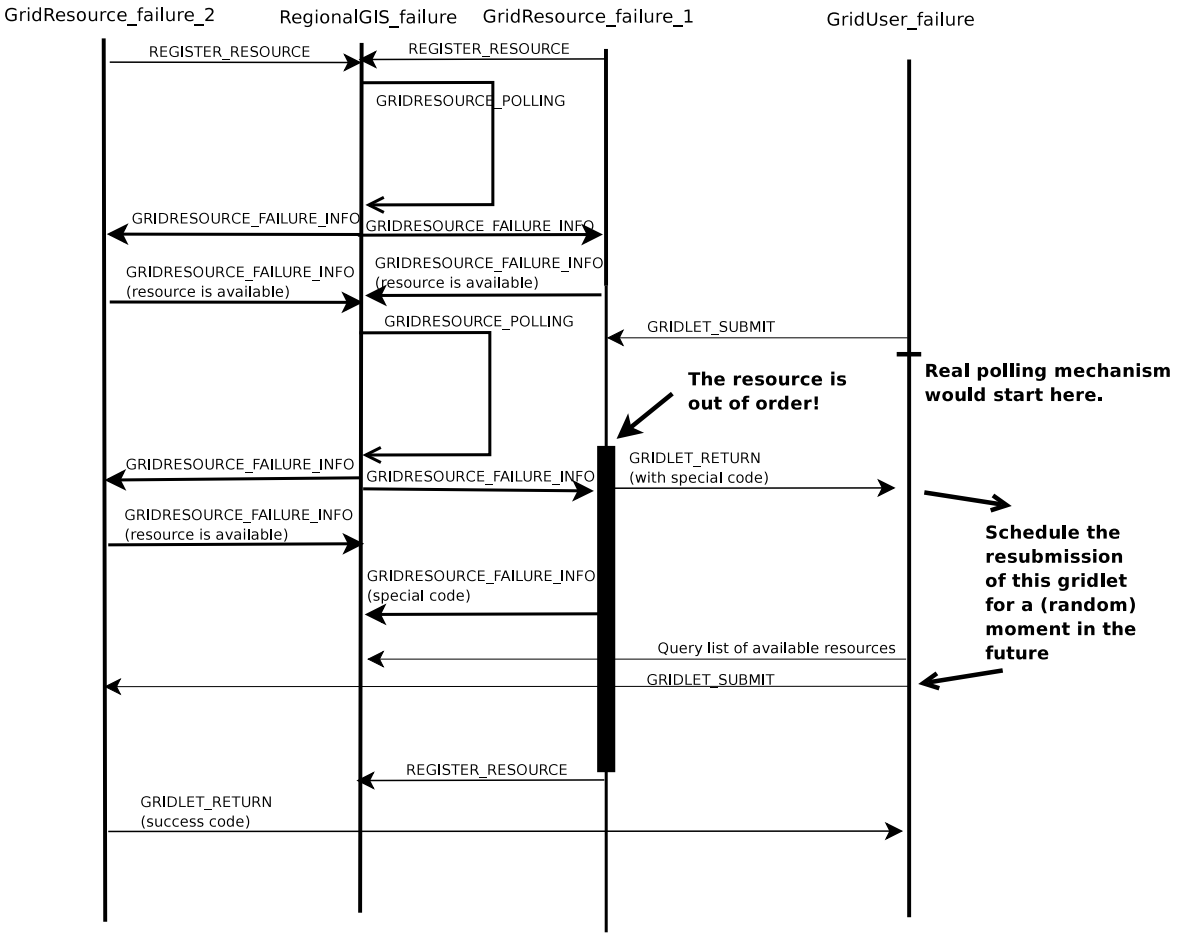


Figure 4: Sequence diagram for the polling mechanism.

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 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, GIS uses a probabilistic distribution on deciding how

Resource Name (Location)	# Nodes	CPU Rating	Policy	# Users	VO
RAL (UK)	41	49,000	Space-Shared	12	2
Imperial College (UK)	52	62,000	Space-Shared	16	2
NorduGrid (Norway)	17	20,000	Space-Shared	4	3
NIKHEF (Netherlands)	18	21,000	Space-Shared	8	3
Lyon (France)	12	14,000	Space-Shared	12	0
CERN (Switzerland)	59	70,000	Space-Shared	24	0
Milano (Italy)	5	70,000	Space-Shared	4	1
Torino (Italy)	2	3,000	Time-Shared	2	1
Rome (Italy)	5	6,000	Space-Shared	4	1
Padova (Italy)	1	1,000	Time-Shared	2	4
Bologna (Italy)	67	80,000	Space-Shared	12	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.

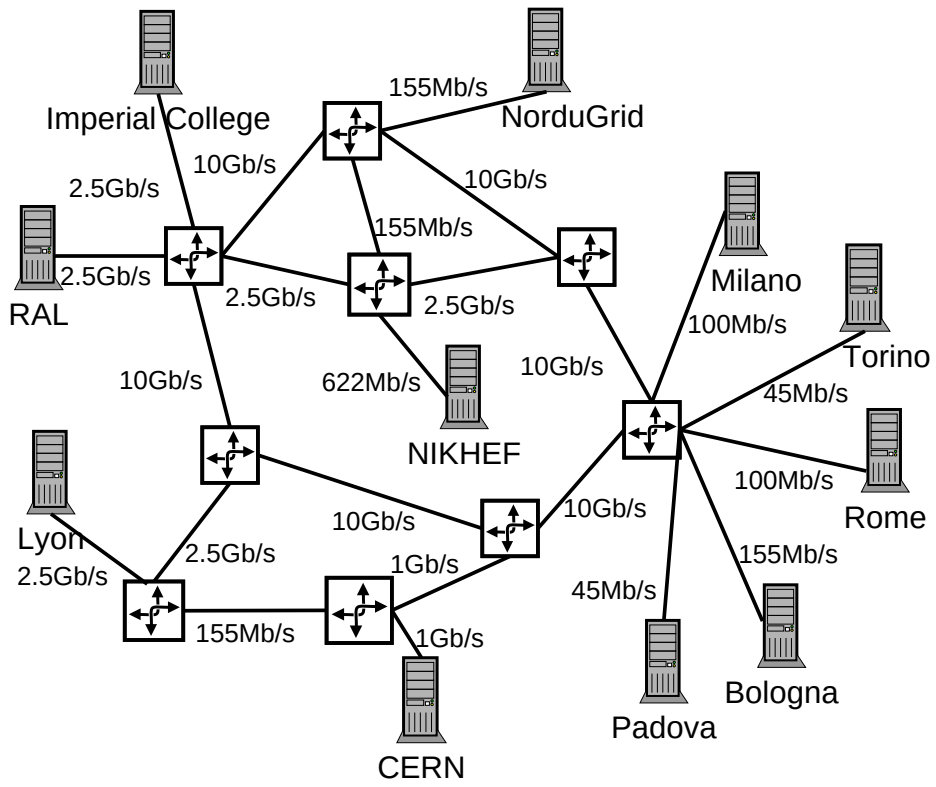


Figure 5: EU DataGRID Testbed 1.

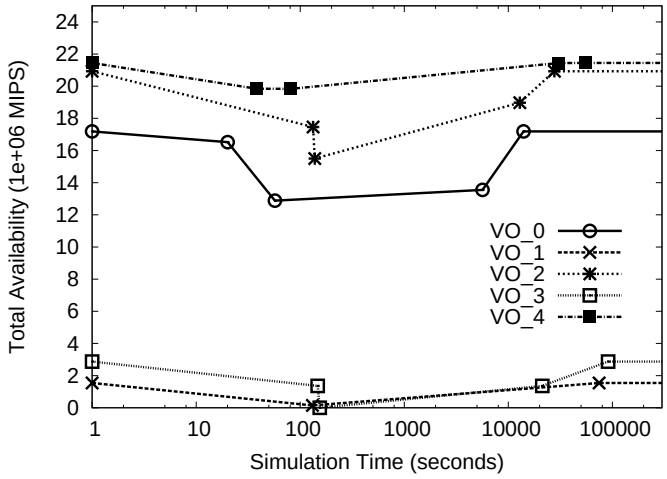
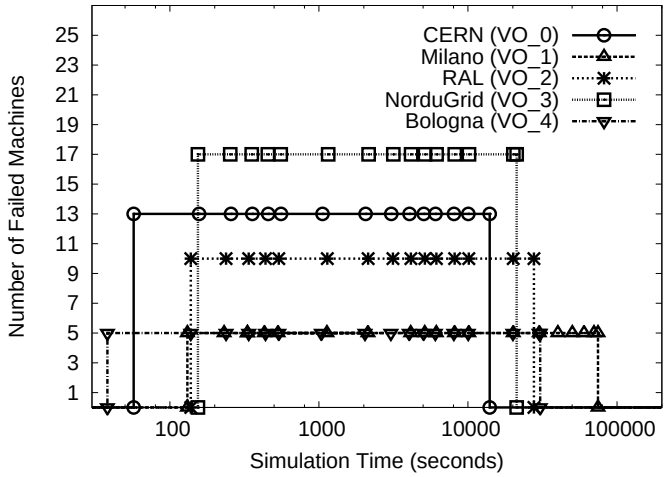
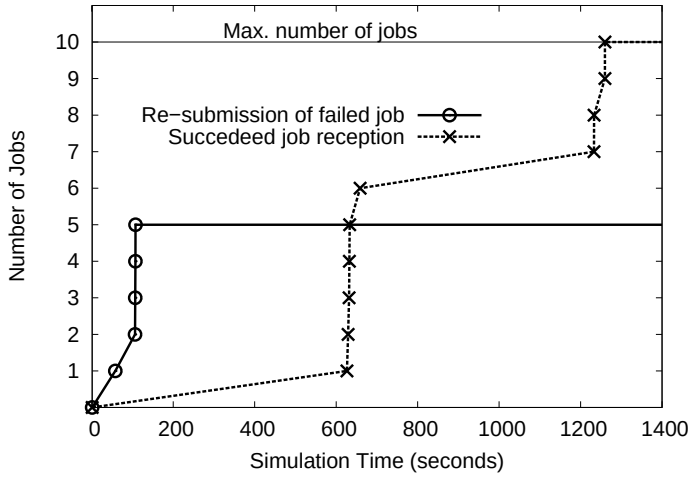


Figure 6: Total availability of each VO domain.

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



(a) Period of a resource failure in each VO



(b) Time-line of `User_0`

Figure 7: Time-lines showing the progress of resources and `User_0`.

computing environments [10]. 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 6. 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.

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

Table 3: Resource failure statistics.

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 6). 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.

4 Conclusions

As grid technologies become more widely used, the need to improve the research in this topic grows as well. Although there are a number of grid simulation tools, none of them cover one of the main features of a grid system: the variable availability of resources. In this report we have explained the new functionality added to one of the most widely used grid simulation tools, GridSim Toolkit.

This new functionality allows researchers to create more real models, which in turn help them to improve their research in different fields of grid computing, such as grid scheduling, network QoS, fault tolerance, resource discovery,

Acknowledgements

This work was partially supported by the following projects: Consolider CSD2006-46, CICYT TIN2006-15516-C04-02, PBC-05-007-01, PBC-05-005-01 and José Castillejo grant. We would like to thank Anthony Sulistio and all the members of the GRIDS Laboratory, University of Melbourne, for their valuable help. This work has been carried out during a research stay at the GRIDS Laboratory, University of Melbourne.

References

[1] OptorSim 2.1: Installation and User Guide, 2006.

[2] The SimGrid Simulation Framework. Web Page, 2006. `http://simgrid.gforge.inria.fr/`.

[3] MicroGrid. Web Page, 2007. http://www-csag.ucsd.edu/projects/grid/microgrid.html.

[4] Bell, W. H., Cameron, D. G., Capozza, L., Millar, A. P., Stockinger, K., and Zini, F. Simulation of dynamic grid replication strategies in OptorSim. In *Grid Computing - GRID 2002, Third International Workshop, Baltimore, MD, USA, November 18, 2002, Proceedings* (2002), M. Parashar, Ed., vol. 2536 of *Lecture Notes in Computer Science*, Springer, pp. 46–57.

[5] Bell, W. H., et al. Optorsim: A Grid simulator for studying dynamic data replication strategies. *The International Journal of High Performance Computing Applications 17*, 4 (Winter 2003), 403–416.

[6] Buyya, R., and Murshed, M. GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing. *Concurrency and Computation: Practice and Experience* (Mayo 2002).

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

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

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

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

[11] Sulistio, A., Poduval, G., Buyya, R., and Tham, C.-K. Constructing a grid simulation with differentiated network service using GridSim. In *6th International Conference on Internet Computing (ICOMP'05)* (June 2005).

[12] Sulistio, A., Yeo, C. S., and Buyya, R. A taxonomy of computer-based simulations and its mapping to parallel and distributed systems simulation tools. *Softw, Pract. Exper 34*, 7 (2004), 653–673.