On Incorporating Differentiated Network Service into GridSim

Anthony Sulistio[a*], Gokul Poduval[b], Rajkumar Buyya[a], and Chen-Khong Tham[b]

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

[b] Computer Communication Networks Laboratory
Department of Electrical and Computer Engineering, National University of Singapore

Abstract

Grid computing technologies are increasingly being used to aggregate computing resources that are geographically distributed across different locations. Commercial networks are being used to connect these resources, and thus serve as a fundamental component of grid computing. Since these grid resources are connected over a shared infrastructure, it is essential that we consider their effect during simulation. In this paper, we discuss how new additions to the GridSim simulation toolkit can be used to explore network effects in grids. We also investigate techniques to incorporate differentiated service, background traffic and collecting information from the network during runtime in GridSim. As a result, these features enable GridSim to realistically model grid computing experiments.

Keywords: Grid computing; Grid simulation; Differentiated network service

1. Introduction

Grid computing has emerged as the next-generation parallel and distributed computing methodology that aggregates dispersed heterogeneous resources for solving various kinds of large-scale parallel applications in science, engineering and commerce [1]. In order to evaluate the performance of a grid environment, we need to conduct *repeatable* and *controlled* experiments, which are difficult due to grid's inherent heterogeneity and its dynamic nature. Additionally, grid testbeds are limited and creating an adequately-sized testbed is expensive and time consuming. Moreover, it needs to handle different administration policies at each resource. Due to these reasons, it is easier to use simulation as a means of studying complex scenarios.

The GridSim toolkit [2] has been developed to overcome the above problems. It is a Java-based discrete-event grid simulation package that provides features for application composition, information services for resource discovery, and interfaces for assigning applications to resources. GridSim also has the ability to model heterogeneous computational resources of varied configurations. The GridSim toolkit has been applied successfully to simulate a Nimrod-G [3] like grid resource broker and to evaluate the performance of deadline and budget constrained cost- and time- optimization scheduling algorithms.

Communication networks serve as a fundamental component of grid computing. Resources, connected over commercial networks, share bandwidth with other users. A realistic simulation of grid environments should include the effects of sending data over shared communication lines. Earlier versions of GridSim did not have the ability to specify a network topology, nor the functionality to connect resources through network links in the experiment. Resources and grid users had all-to-all connections with specifiable band-

*Corresponding author. Tel.: +61 3 8344 1360; fax: +61 3 9348 1184

width. Hence, the simulations did not capture the entire details of an actual grid testbed.

In this work, GridSim has been extended to address the above problems with the ability to simulate realistic network models by: (1) allowing users to create a network topology, (2) packetizing a data into smaller chunks for sending it over a network, (3) generating background traffic, and (4) incorporating different level of services for sending packets.

The rest of this paper is organized as follows: Section 2 provides background on GridSim. Section 3 presents the design and implementation of GridSim network, while Section 4 illustrates the use of GridSim for simulating a Grid computing environment. Section 5 mentions related work. Finally, Section 6 concludes the paper and suggests some further work to be done on GridSim network models.

2. Background

There has been a significant work in the past on `GridSim ver3.0` to incorporate more functionality and extensibility into it, such as extending the GridSim infrastructure to support advance reservation as discussed in [4]. This allows resources to have their own schedulers and policies for reservation-based systems. However, no work has been done into improving the existing network model. Therefore, in the newer GridSim release, a new package is incorporated to provide better capabilities for the existing network model. Inside this package, it contains core network components, such as links and routers. Details of these components will be discussed in Section 3. Also, GridSim denotes the latest version of the software throughout.

2.1. Overall GridSim Architecture

GridSim is based on SimJava [6], 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 overall GridSim architecture can be shown in Figure 1.

The first layer at the bottom of Figure 1 is managed by SimJava for handling the interaction or events among GridSim components. The second layer is dealt with the infrastructure components, such as network and resource hardware. The third and fourth layer are concerned with modeling and simulation of Computational Grids and Data Grids respectively. GridSim components such as Grid Information Service (GIS) and Job Description 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.

2.2. Features

Some of the GridSim features are:

- allowing modeling of different resource characteristics and types;
- enabling simulation of workload traces taken from real supercomputers;
- supporting a reservation-based mechanism of a resource;
- allocating incoming jobs based on space- or time-shared mode;
- has the ability to schedule compute- and/or data-intensive jobs as discussed in [5];
- providing clear and well-defined interfaces for implementing a different resource allocation; and
- allowing modeling of several regional GIS components.

2.3. Fundamental Concepts

In SimJava, each simulated system (e.g. resource and user), that interacts with others, is referred to as an entity. An entity runs in parallel in its own thread by inheriting from the class `Sim_entity`, while its desired behavior must be implemented by overriding a `body()` method.

SimJava requires each entity to have two ports for communication: one for sending events to other entities, whereas the other port is used for receiving incoming events. In GridSim, this

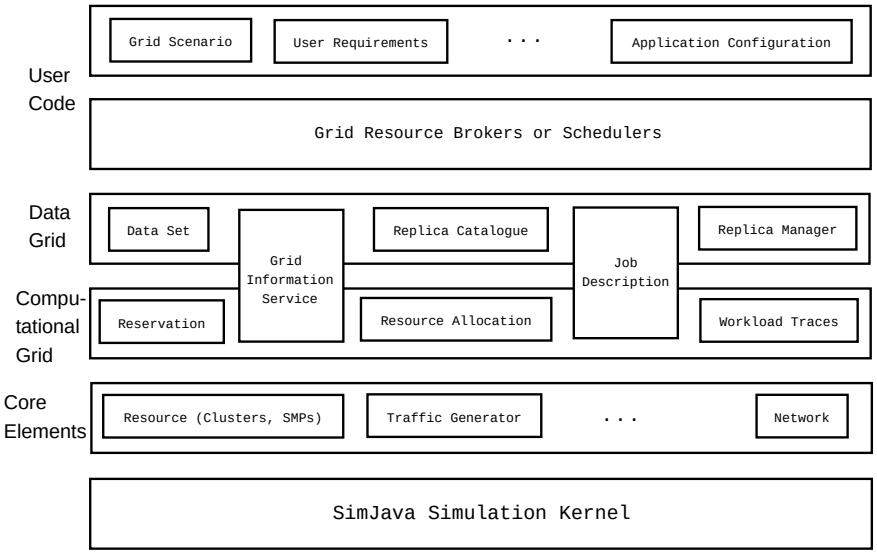


Figure 1. GridSim architecture

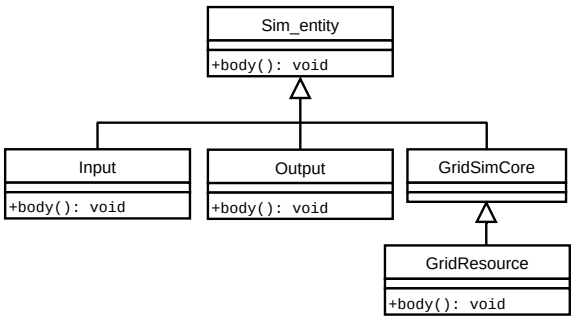


Figure 2. A class diagram showing the relationship between GridSim and SimJava entities

is done via classes `Input` and `Output`. Both classes have their own `body()` method to handle incoming and outgoing events respectively. Similarly, GridSim entities must inherit from the class `GridSimCore` and override a `body()` method. The relationship between `Sim_entity` and GridSim classes is shown in Figure 2. In a class diagram, attributes and methods are prefixed with characters + indicating access modifiers public. Note that the class `GridSimCore` does not have the `body()` method because it is not necessary since its subclass will override the method.

3. Design and Implementation of GridSim Network

The flow of information among GridSim entities happens via their *Input* and *Output* (I/O) entities. Upon creating an entity with a specified bandwidth, GridSim creates a new instance of the `Input` and `Output`, and links them to the new entity. Hence, data sent by an entity goes through

its *Output* entity, and is received by other entities via their *Input* entities

The use of separate entities for I/O provides a simple mechanism for GridSim entities to communicate with each other, and allows modeling of a communications delay [2]. In addition, this existing design provides a clean interface between the network entities and others. Therefore, most of the changes were incorporated into class `Input` and `Output` for transparent and minimal modification to the existing code.

The addition to the existing network architecture allows GridSim entities to be connected using links and routers, with different packet scheduling policies for realistic experiments as shown in Figure 3. Detailed explanation of this figure will be explained later in Section 3.4. The network architecture has also been designed to be extensible and backwards compatible with existing codes written on older GridSim releases.

3.1. Network Components

Important additions to the existing GridSim network architecture are link, router, packet, packet scheduler and background traffic generator components. The relationships among these network components in Unified Modeling Language (UML) notations [7] are depicted in Figure 4. Note that the background traffic generator component will be discussed in Section 3.3.

3.1.1. Link

A link in GridSim is represented as an abstract class `Link` for extensibility. `SimpleLink`, a subclass of `Link` as shown in Figure 4 (a), requires information like the propagation delay, bandwidth and Maximum Transmission Unit (MTU) for packet delivery.

3.1.2. Input and Output

When Gridsim entities want to send or receive data, they use the *Input* and *Output* entities attached to them, as previously mentioned. The *Output* entity is responsible for splitting the data into MTU sized packets, whereas the *Input* entity is accountable to collate the different packets in a stream altogether, and send them as one piece of data to the GridSim entity. In addition, these I/O entities act as a buffer to hold the packets until a link is free.

3.1.3. Router

A router in GridSim is represented as an abstract class `Router` for flexibility as shown in Figure 4 (a). Therefore, this design allows a subclass of `Router` in determining the forwarding table at the start of the simulation, and implementing any routing algorithms.

Routing can be done using static tables or dynamic methods, such as Routing Information Protocol (RIP) [8] and Open Shortest Path First (OSPF) [9]. An implementation of a router in class `FloodingRouter` uses a flooding algorithm to setup its forwarding tables automatically. Since routers and other GridSim entities can not be created and added after the simulation has started, the flooding algorithm is a sufficient method to setup a router's forwarding tables.

3.1.4. Packet

A network packet in GridSim is represented as an interface class `Packet` as shown in Figure 4 (b). Currently, there are two classes that belongs to this category, i.e. `NetPacket` and `InfoPacket`. A `NetPacket` class is used to encapsulate data passing through the network, whereas class `InfoPacket` is devoted to gather network information during runtime which is equivalent to Internet Control Message Protocol (ICMP) [10] in physical networks.

3.1.5. Packet Scheduler

A packet scheduler is responsible for deciding the order in which one or more packets will be sent downlink. Implementing a packet scheduler requires extending from class `PacketScheduler` as depicted in Figure 4 (c).

In Gridsim, three implementations of the packet scheduler are provided i.e. class `FIFOScheduler`, `SCFQScheduler` and `RateControlledScheduler`. The class `FIFOScheduler` uses a simple First In First Out (FIFO) policy, whereas the class `SCFQScheduler` adopts a variation of Weighted Fair Queuing (WFQ) [11], called Self Clocked Fair Queuing (SCFQ) [12] policy. The `RateControlledScheduler` is an implementation of a rate-jitter controlling regulator [32].

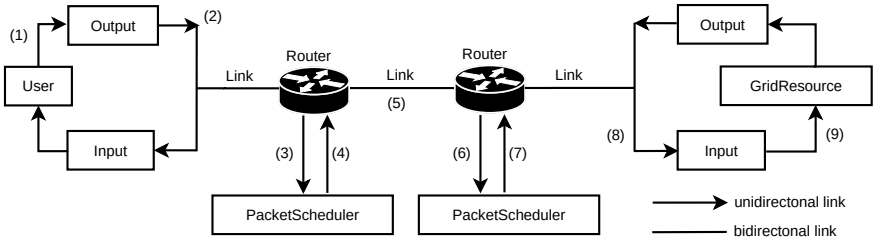


Figure 3. Interaction among Gridsim network components

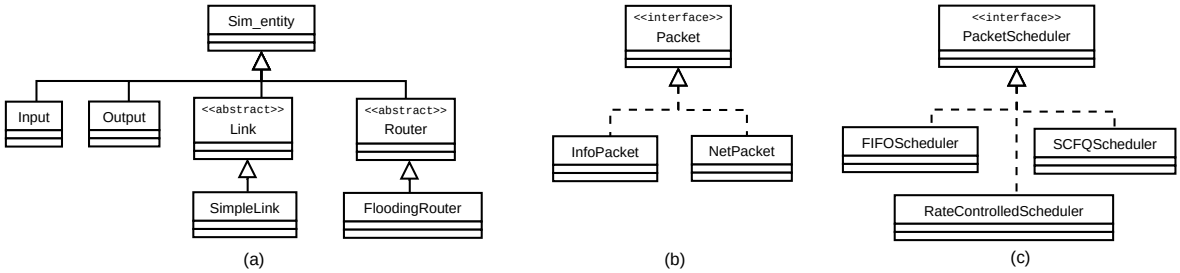


Figure 4. Generalization and realization relationship in UML for GridSim network classes

3.2. Support for Network Quality of Service & Runtime Information

Jobs on grids may have different requirements with respect to bandwidth and latency. Systems like fire or earthquake detection require low latency and reliable delivery. Other jobs like protein folding experiments require high processing power, and may tolerate some network errors. Also, in some cases, grid resource providers may wish to charge for priority access to their resources. Thus grid resource providers need mechanisms to provide users with different Quality of Service (QoS) for using their networks [13]. In order to support this functionality, every packet in GridSim contains a Type of Service (ToS) attribute with a default value of zero weight. This attribute will be used by routers or packet schedulers to provide a differentiated service to heterogeneous links or connections for incoming packets. In GridSim, class `SCFQScheduler` can be configured with different weights. Packets belonging to a class with higher weight receive higher priority according to the SCFQ algorithm. Similarly, `RateControlledScheduler` can be used to control the bandwidth that is assigned to each class of user at a Router. This is a non-work conserving algorithm, which means that the router can remain idle even if there are packets in its queue. Non-work-conserving policies have some benefits like lower buffer space requirements and smoothening of downstream traffic [33]. At a `RateControlledScheduler`, each class of users is assigned to a certain percentage of bandwidth, and the scheduler makes sure that each class remains constrained within its bandwidth limits at all times.

GridSim also supports requesting network status during runtime, such as number of hops to destination, round trip time (RTT), bottleneck bandwidth and all bandwidths that a packet has traversed for current or future simulation time. This feature is similar to an ICMP ping message. The result is captured inside class `InfoPacket`.

To enable this functionality, a GridSim en-

tity must use either blocking or non-blocking method calls from class `GridSimCore`. A blocking call requires to use only a `pingBlockingCall()` method, where it waits for a result to come back while preventing other entity's activities. In contrast, a non-blocking call needs to use a combination of `ping()` and `getPingResult()` methods while doing something else in between. Both `pingBlockingCall()` and `getPingResult()` method return an object of class `InfoPacket`.

3.3. Simulating with Background Traffic

In commercial or even academic networks, users expect to experience network traffic that does not belong to them. In order to capture this real world scenario into a simulation, GridSim supports modeling of background traffic. This can be done by creating an instance of class `TrafficGenerator`, and storing it as a class `Output` attribute. The class `TrafficGenerator` generates inter-arrival time, packet size, and number of packets for each interval according to various distributions that are supported by SimJava [6]. Some of the distributions are Bernoulli, negative exponential, and binomial. Then, these generated values are used by an *Output* entity to send background traffic packets to one or all other entities in the experiment.

3.4. Interaction among GridSim Network Components

When a simulation starts, routers send out advertisement packets to all neighboring routers, advertising any other GridSim entities they are connected to. Later on, the neighboring routers adjust their forwarding tables upon receiving these packets. Then, they forward the packets to all neighboring routers except the source. Depending on the complexity of a network topology and number of GridSim entities created, this process might take a while.

Once the forwarding tables have been completed, a GridSim entity, named *User*, as shown in Figure 3, can start sending jobs to a *GridResource* entity. Each GridSim entity has I/O entities attached to it that act as a buffer. Therefore, when a job is to be sent out by a *User* entity, it is first buffered at the *Output* entity (step 1). Here, the job is split into multiple packets if it is larger than the MTU of a link connected to the *Output* entity. The packets are then given sequence numbers, enqued in a buffer, and sent to link that connects the entity to the neighbouring downstream router. The link takes the packet, delays it by the propagation delay specified, and dequeues it at the other end (step 2).

Routers receive the packet from the link, and decide the packet scheduler that the packet should be sent to (step 3). If the outgoing interface has a MTU less than the packet size, it splits the packet into smaller ones, similar to what *Output* entity does. Next, these packets are enqueued at the packet scheduler. The packet scheduler uses its own algorithm, such as FIFO or WFQ to decide the order in which the packets should be dequeued (step 4). When a link attached to the packet scheduler is free, the router dequeues one packet from the packet scheduler, and sends it down the link (step 5). Similar approach is required if the other end of the link is another router entity (step 6–8).

When the final link is traversed and the packet reaches the *GridResource* entity, all packets in a sequence are collated back together into the job (step 9). This is done by the *Input* entity. The job is then passed to the *GridResource* entity for processing. Once processing is complete, the *GridResource* entity passes the completed job to its *Output* entity, which follows a similar path until it reaches the *Input* entity that created this job.

The current protocol used for sending packets is a datagram oriented protocol, which is similar to User Datagram Protocol (UDP). There is no support for acknowledging each packets and packet reordering. Since there is no support for recovering lost packets, I/O buffers are considered to be unlimited in order to ensure no packets are lost.

4. Experiments and Results

4.1. Experiment Aim

The main aim of this experiment is to show GridSim's ability to simulate an adequate-size grid testbed. Therefore, we create a network

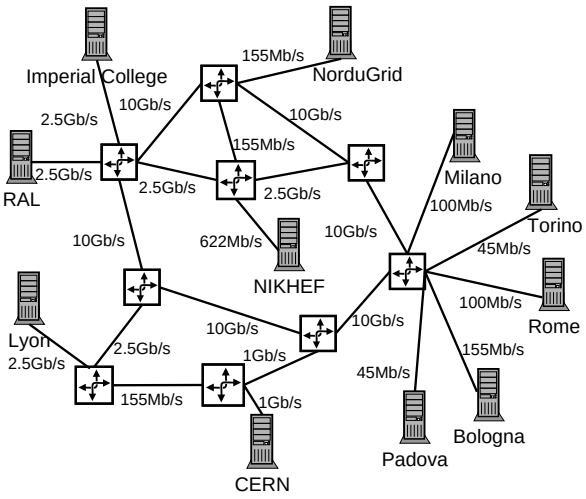


Figure 5. Network topology

topology based on EU DataGrid TestBed I [22].

For this experiment, our main concern is the network behavior in a grid environment. Hence, we are trying to look at:

- how background traffic might affect network loads and overall packets execution time; and
- how differentiated QoS for packets might help in a heavy load situation.

4.2. Experiment Setups

Table 1 summarizes all the resource relevant information. Each node of a resource is assumed to be a 2Ghz AMD Opteron processor. We took the data about the resources and scaled them down. The number of nodes and its CPU rating were scaled down by 10. The scaling was done primarily to reduce the run time of the experiment and the memory consumption. The complete simulation would require more than 2GB of memory.

In GridSim, total processing capability of a resource's CPU is modelled in the form of Million Instructions Per Second (MIPS) rating as per SPEC (Standard Performance Evaluation Corporation) CPU (INT) 2000 [16] benchmarks. A space shared policy or First Come First Served (FCFS) algorithm is used to compute incoming jobs in all resources. In addition, all links share same characteristics, i.e. MTU size of 1,500 bytes and latency of 10 milliseconds.

Table 1
EU DataGrid testbed simulated using GridSim

Resource Name (Location)	# Nodes	CPU Rating
RAL (UK)	41	49,000
Imperial College (UK)	52	62,000
NorduGrid (Norway)	17	20,000
NIKHEF (Netherlands)	18	21,000
Lyon (France)	12	14,000
CERN (Switzerland)	59	70,000
Milano (Italy)	5	7,000
Torino (Italy)	2	3,000
Rome (Italy)	5	6,000
Padova (Italy)	1	1,000
Bologna (Italy)	67	80,000

There are four users located on each of the resource (with a total of 44 users), sharing the same characteristics:

- bandwidth: 100 Mbps connected to a router
- total number of jobs: 30 each
- job length: approximately 42000k Million Instructions (MI) $\pm$ 30%, which is around 10 minutes if it is run on the CERN resource
- job data size: 15 MB each
- job submission: 20 jobs are submitted to CERN, while the rest are uniformly distributed among other resources as mentioned in Table 1
- arrival time: uses a Poisson distribution, with four random users submit all their jobs every approximately 5 minutes

To incorporate a background traffic functionality into this experiment, selected users are chosen as the source to generate these background packets. A Poisson distribution is used with an inter-arrival time of 1 minute. In addition, total number of packets for each interval is uniformly distributed in [500...1000], with the size of each packet is 1,500 bytes.

To investigate the advantage of having network QoS, two users from each site are chosen with a higher ToS weight or rate. For the experiment using a SCFQ packet scheduler, high priority users' jobs are assigned a weight of 2 and normal users' jobs are assigned a weight of 1. Background traffic receives a weight of 0. For the experiment using a Rate Controlled scheduler, high and normal priority jobs are assigned 55

4.3. Building an Experiment with GridSim

Creating an experiment in GridSim always requires the following steps:

1. Initialize the GridSim package by using a `GridSim.init()` method. This should be done before creating any GridSim entities in order to start the SimJava simulation kernel.

2. Create one or more grid resource entities. Each resource must have number of processors, speed of processing and internal process scheduling policy. Currently, two implementations of scheduling policy are provided, i.e. time-shared and space-shared. However, a user can implement a different scheduling policy easily as described in [4] because of well-defined interfaces between grid resource and its scheduling policy entity.

3. Create one or more grid user entities. A grid user is responsible for sending jobs to one or more resources. Other complex functionalities are open for implementation based on user's needs and requirements. This can be done by extending `GridSimCore` class and write the necessary code inside a `body()` method.

4. Build a network topology by connecting grid user and resource entities. At the moment, connecting those entities need to be done manually by first creating network objects such as `Router` and `Link`. Then, connect the entities to a `Router` object using an `attachHost()` method. An `attachRouter()` method can be used to link one or more routers. For experiments with a large network topology, this process can be tedious and error-prone. Hence, building a network topology automatically from a file is also supported in GridSim.

5. Finally, run the experiment by calling a `GridSim.startGridSimulation()` method.

The GridSim toolkit contains documentation and few simple tutorial examples that illustrate the above steps.

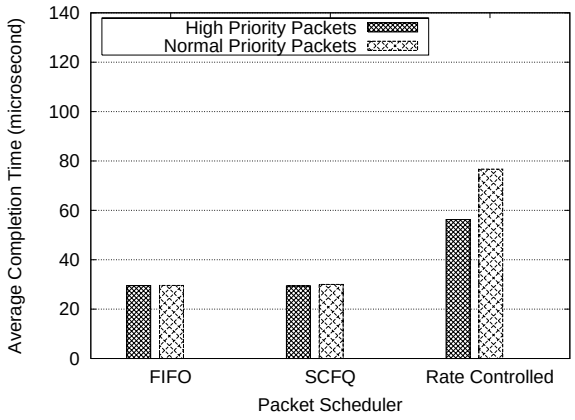
4.4. Analysis

The results displayed in Figure 6 show the average amount of time spent by each packet in a router's queue, in this case the router located in CERN. This router is chosen because the resource at CERN receives many jobs for execution, hence it routes a substantial amount of incoming and outgoing traffic.

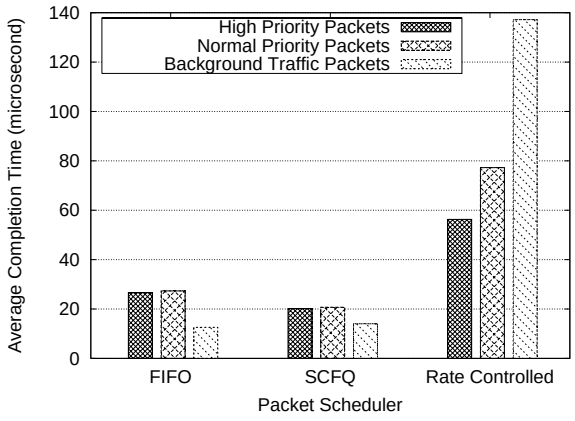
As mentioned previously, we compare two type of users, one of whom has been set to a high priority, while the other sends packets at a normal priority. It can be seen that high priority packets are dequeued faster than normal packets, except for the FIFO experiment, thus providing better QoS to high priority users.

For the FIFO experiment shown in Figure 6, all packets are treated based on the arrival time. Hence, there are no prioritization for these packets. On the other hand, for the SCFQ experiment as shown in Figure 6, high priority packets are dequeued faster than normal packets by more than 2%. An interesting observation in the SCFQ experiment of Figure 6(b) is that the background packets are dequeued faster than other packets. This is because these packets are being sent at a continuous rate, while other packets are sent in an interval or burst mode. As a result, the background packets utilized the whole bandwidth during times at which other packets are not there.

For the Rate Controlled experiment displayed in Figure 6, high priority packets are dequeued faster than normal packets by approximately 36%. The main reason is because, as mentioned earlier, each class of users is assigned to a certain

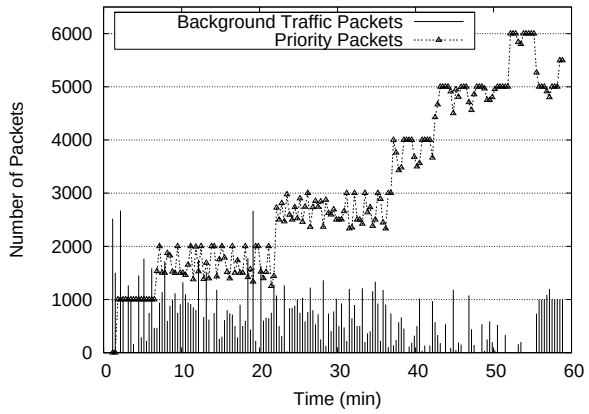


(a) without background traffic

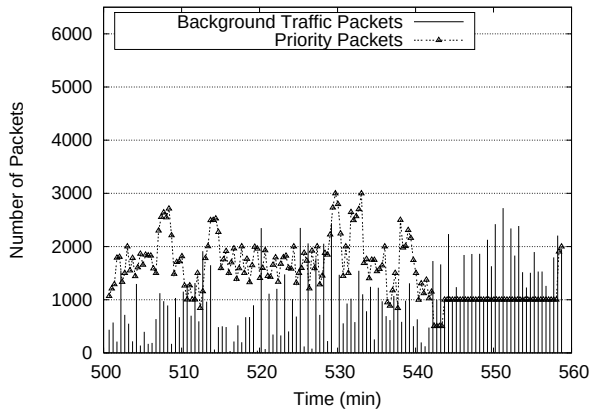


(b) with background traffic

Figure 6. Average packet lifetime at the CERN router (lower is better)



(a) in the beginning of the experiment



(b) in the middle of the experiment

Figure 7. Number of packets passing through the CERN router during the Rate Controlled experiment

percentage of bandwidth, and each class does not use more than the allocated percentage. Hence, there is not much of a difference for the experiment with and without background traffic stated in Figure 6(a) and Figure 6(b) respectively.

As expected, high priority packets spent less time using the SCFQ scheduler rather than the FIFO one. However, it is also interesting to note that the Rate Controlled scheduler dequeued these packets the slowest of all schedulers as shown in Figure 6. This is because, as stated earlier, the Rate Controlled scheduler does not utilize the whole bandwidth in comparison to other schedulers.

In the real world, Rate Controlled scheduling is useful when absolute guarantees are required from the network sub-system. For example, Voice over Internet Protocol (VoIP) or Internet Protocol Television (IPTV) applications might require a certain minimum bandwidth in order to perform well. The drawback of using Rate Controlled scheduling is that it can lead to wastage of bandwidth. If 10% of the bandwidth is reserved for a certain application, and the application is well below its limit, then the additional bandwidth is being wasted. It is possible to implement schedulers which detect this wastage, and send other kinds of traffic in its place, but this adds to the complexity of the implementation. Higher complexity leads to increase in memory and processing requirements, hence higher costs. When prioritization rather than guarantees are required, SCFQ should be used. SCFQ is also a simpler algorithm to implement than Rate Controlled schedulers.

The effect of the background traffic in the experiment is shown in Figure 7. In the beginning of the Rate Controlled experiment, as shown in Figure 7(a), there are many (high/normal) priority packets entering the router heading towards the CERN resource for job submission. However, as the experiment progresses, only few completed jobs are being sent back to users. Hence, there are times where background packets have a higher number than priority packets as shown in Figure 7(b). On average, the background packets accounted for 36% of total packets passed by the CERN router as shown in Figure 8.

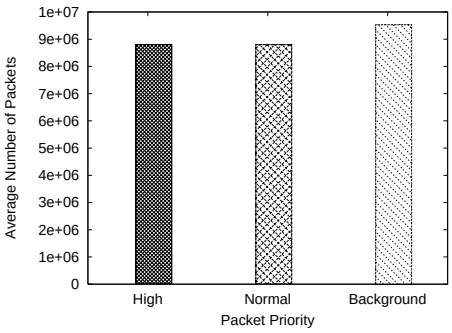


Figure 8. Average number of packets passing through the CERN router for all experiments

5. Related Work

Simulation is very much used in the networking research area. Examples of such simulators include NS–2 [18], DaSSF [19], OMNET++ [20] and J–Sim [21]. Though their support for network protocols is extensive, they are not targeted at studying grid computing. This is because simulating grids requires modeling the effects of scheduling algorithms on grid resources and investigating user's QoS requirements for application processes. In addition, we believe simulating TCP and UDP connections are sufficient to model a real world behaviour, because grid users are mostly interested in finding out RTT and available bandwidth of a host. Therefore, these network simulators perform other complex functionalities which are not needed in simulating a grid computing environment.

There are some tools available, apart from GridSim, for application scheduling simulation in Grid computing environments, such as Bricks [23], MicroGrid [24] [25], SimGrid [26] [27], and OptorSim [28]. All of these simulators also have an underlying network infrastructure, with the ability to simulate realistic experiments by using background traffic. Differences among the grid simulators, except for Bricks, in terms of network functionalities and features are highlighted in Table 2. Note that for Routing Table Entry column, an automatic entry means filling in

Table 2
Listing of network functionalities and features for each grid simulator

Functionalities	GridSim	MicroGrid	SimGrid	OptorSim
Routing Table Entry	Automatic	Automatic	Manual	Manual
Type of Transport Protocol	a datagram oriented protocol similar to UDP	TCP and UDP	TCP	Not supported
Data Packetization	Supported	Supported	Not supported	Not supported
Runtime Network Status	Supported	Supported	Supported	Not supported
Network QoS	Supported	Not supported	Not supported	Not supported

a router's forwarding table automatically during runtime. In contrast, a manual entry means filling in the forwarding table by reading from an external file that defines a router's connection with others, or by manually entering the information into the table.

Bricks [23] is able to specify a network topology, bandwidth, throughput and variance of the throughput over time. The background traffic functionality is modeled by using a probabilistic distribution, which is similar to GridSim. However, at the time this article is being written, this package is not available to download from its website [29]. As a result, we are not able to compare it with our work in more details. Therefore, it is not included in Table 2.

MicroGrid [24] [25] allows complex network modeling, such as transport and routing protocols, and large-scale experiments since it is based on DaSSF [19]. Hence, in terms of network capabilities, MicroGrid is the most complete of all grid simulators. However, it is actually an emulator, meaning that actual application code is executed on the virtual grid modeled after Globus [30].

SimGrid [26] [27] has a good network infrastructure that supports Transmission Control Protocol (TCP) transport protocol for a reliable service. It also models background traffic by reading from a trace file generated by Network Weather Service (NWS) [31]. NWS is used to monitor current available bandwidth between two machines over the network. However, SimGrid does not make any distinction between a job computation and a data transfer, since they are modeled as a resource performing a specific task. Therefore, it does not support data packetization. In addition, requesting network status functionalities during runtime in SimGrid are limited to latency and bandwidth of a link. In contrast, GridSim reports more network information than SimGrid, such as number of hops to a destination and RTT as mentioned in Section 3.2.

OptorSim [28] has a very simple network infrastructure compared to other simulation tools, since it does not support routing and transport protocol nor data packetization. The background traffic functionality is modeled by using a Landau distribution only. In addition, simulating with background traffic requires a configuration file that describes a network topology in a matrix format.

From the above discussion and Table 2, GridSim incorporated QoS into a network for scheduling packets, which are not supported by other grid simulators. In addition, GridSim provides a good set of network functionalities and features, which some of them are not supported in the other grid simulators.

6. Conclusion and Further Work

Network serves as a fundamental component in grid computing since resources and users are connected over a network topology with shared bandwidth. Previously, GridSim does not have the ability to specify a network topology nor the functionality to connect resources through network links in the experiment. In this work, modifications into an existing network architecture have been incorporated into GridSim to address

the above problems.

With the addition of this network functionality, users can study the effects that both the network topology and grid resources can have on their jobs. This paper explores the various types of network elements in GridSim like routers, links, packet schedulers; and how they can be extended to add more functionalities. Moreover, GridSim has new exciting features such as generating background traffic during an experiment, requesting network information during runtime and providing differentiated service for packets based on users' Quality of Service (QoS) requirements. We believe these features help make GridSim a comprehensive package to simulate a realistic grid environment.

Our experiment has shown how GridSim can be used to simulate a medium-sized grid testbed. It has shown how schedulers, which provide differentiated service, can help high priority users achieve better QoS than normal users. However, providing differentiated service at the network level only may not be enough. Grid resources will also be required to support it in order to achieve end-to-end QoS.

In the future, we are planning to incorporate additional features into GridSim, such as having different types of routing algorithms, schedulers and reservation of network resources. We are also planning to add other type of network building blocks like switches and domain gateways. Support will be added for non work-conserving routers.

Acknowledgement
We thank Uros Cibej for his work on implementing the functionality of creating a network topology from a file. We also thank CS Yeo for his comments on the paper.

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

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

REFERENCES

1. I. Foster and C. Kesselman, Eds., *The Grid: Blueprint for a Future Computing Infrastructure*. Morgan Kaufmann Publishers, 1999.
2. R. Buyya and M. Murshed, "Gridsim: A toolkit for the modeling and simulation of distributed management and scheduling for grid computing," *The Journal of Concurrency and Computation: Practice and Experience*, vol. 14, pp. 13–15, 2002.
3. 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 (HPC Asia'00)*, Beijing, China, May 14–17 2000.
4. 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 (PDCS'04)*, Cambridge, USA, November 9–11 2004.
5. A. Sulistio, U. Cibej, R. Buyya, and B. Robic, "A toolkit for modelling and simulation of data grids with integration of data storage, replication and analysis," Technical Report, GRIDS-TR-2005-13, GRIDS Lab, University of Melbourne, Australia, Nov. 8, 2005.
6. C. Simatos, "Making simjava count," MSc. Project report, The University of Edinburgh, September 12 2002.
7. M. Priestley, *Practical Object-Oriented Design with UML*. McGraw-Hill, 2000.
8. G. Malkin, "RFC 2453: RIP version 2," November 1998. [Online]. Available: http://www.apps.ietf.org/rfc/rfc2453.html
9. J. Moy, "RFC 2328: OSPF version 2," April 1998. [Online]. Available: http://www.ietf.org/rfc/rfc2328.txt
10. J. Postel, "Internet control message protocol: Darpa internet program protocol specification," September 1981. [Online]. Available: http://www.ietf.org/rfc/rfc0792.txt
11. A. J. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing

algorithm," in *Proc. of the ACM Symposium on Communications Architectures and Protocols (SIGCOMM'89)*, Austin, USA, September 19–22 1989, pp. 1–12.
12. S. J. Golestani, "A self-clocked fair queueing scheme for broadband applications," in *Proc. of IEEE INFOCOM'94*, Toronto, Canada, June 12–16 1994, pp. 636–646.
13. S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "RFC 2475: An architecture for differentiated service," December 1998. [Online]. Available: http://www.ietf.org/rfc/rfc2475.txt
14. L. Winton, "Data grids and high energy physics: A melbourne perspective," *Space Science Reviews*, vol. 107, no. 1–2, pp. 523–540, 2003.
15. "Uk e-science programme." [Online]. Available: www.escience-grid.org.uk
16. "Spec – standard performance evaluation corporation." [Online]. Available: http://www.spec.org/
17. "Grangenet – grid and next generation network." [Online]. Available: http://www.grangenet.net/
18. "The network simulator – ns-2." [Online]. Available: http://www.isi.edu/nsnam/ns/
19. J. Liu and D. M. Nicol, *DaSSF 3.1 User's Manual*, Dartmouth College, April 2001.
20. A. Varga, "The omnet++ discrete event simulation system," in *Proc. of the European Simulation Multiconference (ESM'01)*, Prague, Czech Republic, June 6–9 2001.
21. "J–sim." [Online]. Available: http://www.j-sim.org/
22. The European DataGrid project homepage. http://eu-datagrid.web.cern.ch/eu-datagrid, 2005.
23. K. Aida, A. Takefusa, H. Nakada, S. Matsuoka, S. Sekiguchi, and U. Nagashima, "Performance evaluation model for scheduling in a global computing system," *The International Journal of High Performance Computing Applications*, vol. 14, no. 3, pp. 268–279, 2000.
24. H. J. Song, X. Liu, D. Jakobsen, R. Bhagwan, X. Zhang, K. Taura, and A. Chien, "The microgrid: a scientific tool for modeling computational grids," in *Proc. of IEEE Supercomputing 2000*, Dallas, USA, November 4–10 2000.
25. X. Liu and A. Chien, "Realistic large-scale online network simulation," in *Proc. of IEEE Supercomputing 2004*, Pittsburgh, USA, November 6–12 2004.
26. H. Casanova, "Simgrid: A toolkit for the simulation of application scheduling," in *Proc. of the First IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid'01)*, Brisbane, Australia, May 15–18 2001.
27. 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.
28. W. H. Bell, D. G. Cameron, L. Capozza, A. P. Millar, K. Stockinger, and F. Zini, "Optorsim – a grid simulator for studying dynamic data replication strategies," *The Intl. Journal of High Performance Computing Applications*, vol. 7, no. 4, pp. 403–416, 2003.
29. "Bricks: A performance evaluation system for grid computing scheduling algorithms." [Online]. Available: http://www.is.ocha.ac.jp/ takefusa/bricks/index.shtml
30. I. Foster and C. Kesselman, "Globus: A metacomputing infrastructure toolkit," *The International Journal of Supercomputer Applications and High Performance Computing*, vol. 11, no. 2, pp. 115–128, 1997.
31. R. Wolski, N. Spring, and J. Hayes, "The network weather service: A distributed resource performance forecasting service for metacomputing," *The Journal of Future Generation Computing Systems*, vol. 15, no. 5–6, pp. 757–768, 1999.
32. H. Zhang, and D. Ferrari, "Rate-Controlled Static-Priority Queueing", *INFOCOM*, pp. 227–236", 1993
33. H. Zhang, and S. Keshav, "Comparison of Rate-Based Service Disciplines", *SIGCOMM*, pp. 113–121, 1991