Constructing A Grid Simulation with Differentiated Network Service Using GridSim

Anthony Sulistio*, Gokul Poduval†, Rajkumar Buyya*, and Chen-Khong Tham†

*Grid Computing and Distributed Systems (GRIDS) Laboratory
Department of Computer Science and Software Engineering,
The University of Melbourne, Australia
ICT Building, 111 Barry Street, Carlton, VIC 3053
Email: {anthony, raj}@cs.mu.oz.au

†Computer Communication Networks Laboratory
Department of Electrical and Computer Engineering
National University of Singapore,
Singapore 117576
Email: {gokul, eletck}@nus.edu.sg

***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.**

I. 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 bandwidth. 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 II provides background on GridSim. Section III presents the design and implementation of GridSim network, while Section IV illustrates the use of GridSim for simulating a Grid computing environment. Section V mentions related work. Finally, Section VI concludes the paper and suggests some further work to be done on GridSim network models.

II. 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 latest `GridSim ver3.1` 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 III. Also, GridSim denotes version 3.1 of the software throughout.

GridSim is based on SimJava2 [5], a general purpose discrete-event simulation package implemented in Java. In SimJava, each simulated system (e.g. resource and user), that interacts with others, is referred to as an entity. An

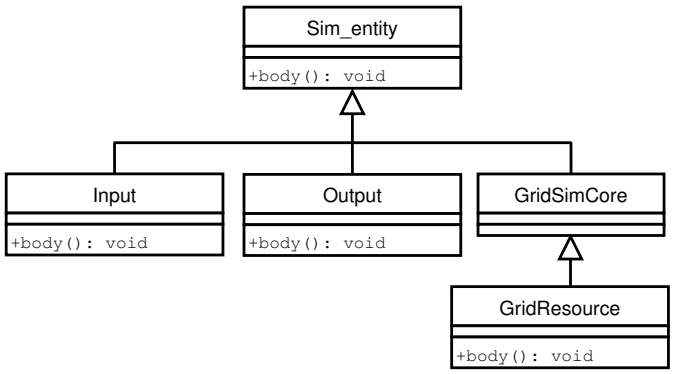

Fig. 1. A class diagram showing the relationship between GridSim and SimJava entities

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 is done via class `Input` and `Output`. Both classes have their own `body()` method to handle incoming and outgoing events respectively. Similar to SimJava, 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 1. 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.

III. Design and Implementation of GridSim Network

The flow of information among GridSim entities happens via their *Input* and *Output* (I/O) entities. Upon creating a GridSim entity with a specified bandwidth, it automatically creates both instances of class `Input` and `Output`, and links them to this entity. Hence, sending a data must go through to a sender's *Output* entity before going into a recipient's *Input* entity for collection.

The use of separate entities for I/O provides a simple mechanism for a GridSim entity 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 new 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 2. Detailed explanation of this figure will be explained later in Section III-D. The network architecture has also been designed to be extensible and backwards compatible with existing codes written on older GridSim releases.

A. Network Components

Important addition 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 [6] are depicted in figure 3 and 4. Note that the background traffic generator component will be discuss in Section III-C.

1) Link: A link in GridSim is represented as an abstract class `Link` for extensibility. `SimpleLink`, a subclass of `Link` as shown in figure 3 (a), requires information like the propagation delay, bandwidth and Maximum Transmission Unit (MTU) for packet delivery.

2) Input and Output: When GridSim entities want to send a data over the network, each of them has *Input* and *Output* (I/O) entities attached to it, 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) Router: A router in GridSim is represented as an abstract class `Router` for flexibility as shown in figure 3 (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) [7] and Open Shortest Path First (OSPF) [8]. 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.

4) Packet: A network packet in GridSim is represented as an interface class `Packet` as shown in figure 3 (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) [9] in physical networks.

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 3 (c).

Two implementations of a packet scheduler are provided in GridSim, i.e. class `FIFOScheduler` and `SCFQScheduler`. The class `FIFOScheduler` uses a simple First In First Out (FIFO) policy, whereas the class `SCFQScheduler` adopts a variation of Weighted Fair Queuing (WFQ) [10], called Self Clocked Fair Queuing (SCFQ) [11] policy, which will be discussed next.

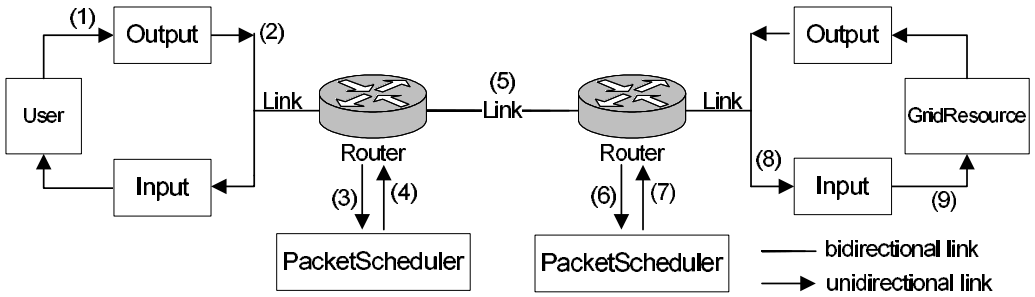

Fig. 2. Interaction among Gridsim network components

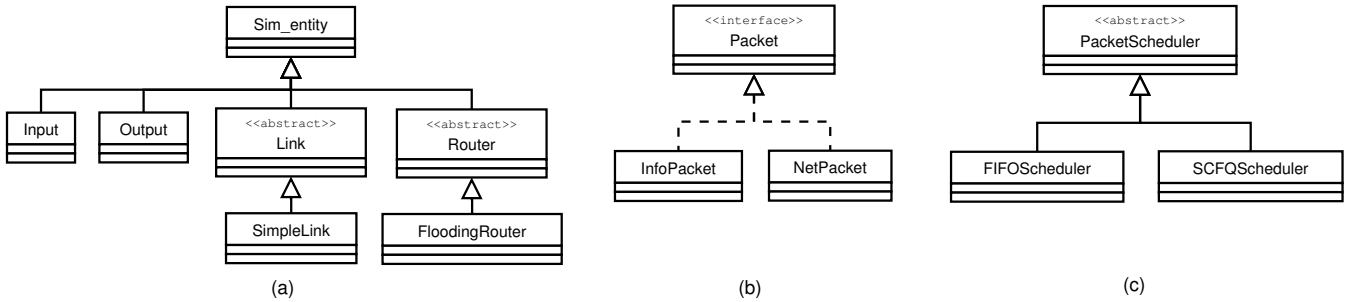

Fig. 3. Generalization and realization relationship in UML for GridSim network classes

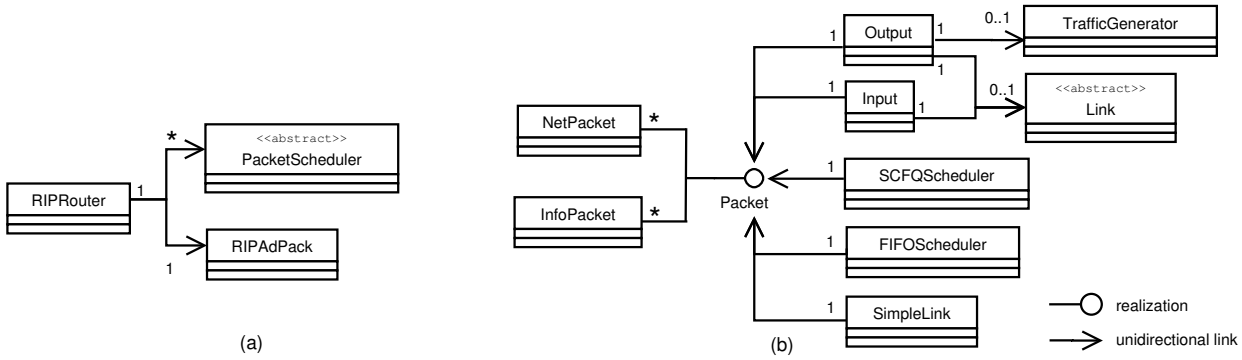

Fig. 4. Association relationship in UML for GridSim network classes

B. 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 [12]. 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.

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 entity 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`.

C. 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, as shown in figure 4 (b). The class `TrafficGenerator` generates inter-arrival time, packet size, and number of packets for each interval according to various distributions that are supported by SimJava2 [5] [13]. 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.

D. Interaction among GridSim Network Components

When a simulation starts, routers send out advertisement packets to all neighboring router, 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* from following an example shown in figure 2, 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, enqueuing in a buffer, and sent to the router down the link one by one. 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.

IV. Experiments and Results

A. 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 topology based on a Belle Analysis Data Grid (BADG) testbed in Australia in collaboration with IBM, as shown in figure 5. The BADG tested is used by scientists to analyze high-energy physics experiment data [14], similar to the grids running under the e-Science programme [15].

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

- how background traffic might affect network loads and overall execution time;
- how differentiated QoS for packets might help in a heavy load situation; and
- how much the costs, in terms of actual time for running the experiment are required to achieve an accurate and realistic simulation.

B. Experiment Setups

Table I summarizes all the resource relevant information. Five resources are created in four different locations: Canberra, Adelaide, Melbourne and Sydney. The processing ability of a resource's CPU is measured 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.

All resources, except for the Adelaide one, are connected via GrangeNet [17], a Gigabit wide-area network within Australia. However, to simplify the experiment setups, all resources and users have the same set of network properties for connection to GrangeNet, as shown in figure 5. In addition, all links share same characteristics, i.e. MTU size of 1,500 bytes and latency of 10 milliseconds.

There are 5 users located on each of the four locations, sharing the same characteristics:

- bandwidth: 100 Mbps connected to a leaf router of each testbed site
- total number of jobs: 20 each
- job data size: 1 MB each
- job processing power: 100 Million Instructions (MI) each
- job submission: uniformly distributed among five resources as mentioned in Table I

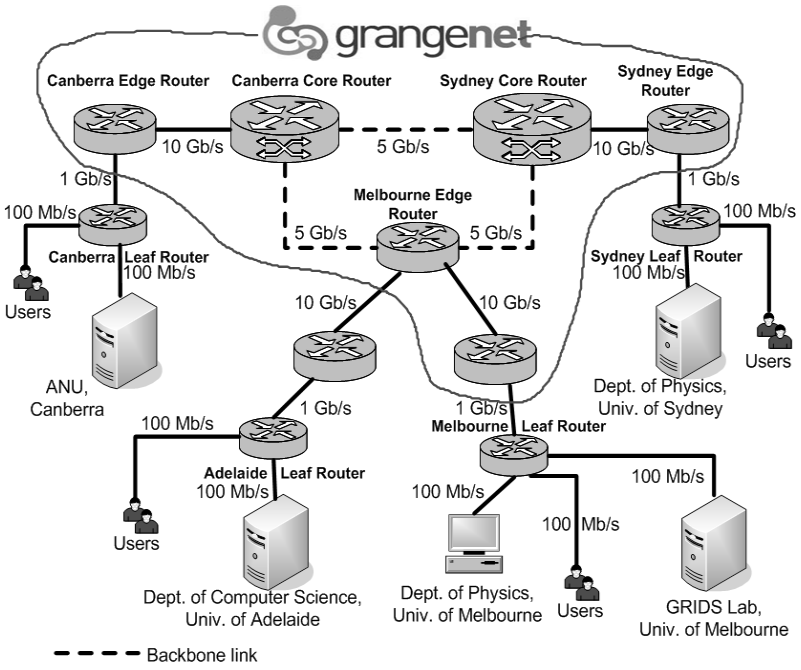

Fig. 5. Network topology

TABLE I

AUSTRALIAN BELLE ANALYSIS DATA GRID TESTBED SIMULATED USING GRIDSIM

Name	Location	Resource Type & Characteristics	Num CPU	A SPEC Rating
R0	Dept. of Physics, Univ. of Melbourne	PC with Intel Pentium 2.0 Ghz, 512 MB RAM	1	684
R1	GRIDS Lab, Univ. of Melbourne	IBM eServer with dual Intel Xeon 2.6 Ghz, 2 GB RAM	4	1050
R2	Dept. of Physics, Univ. of Sydney	IBM eServer with dual Intel Xeon 2.6 Ghz, 2 GB RAM	4	1050
R3	Dept. of Computer Science, Univ. of Adelaide	IBM eServer with dual Intel Xeon 2.6 Ghz, 2 GB RAM	4	1050
R4	Australia National Univ. (ANU), Canberra	IBM eServer with dual Intel Xeon 2.6 Ghz, 2 GB RAM	4	1050

- background traffic: submits to all resources and other users, with inter-arrival time using a Poisson distribution approach with mean of 5 minutes. Total number of packets for each interval is uniformly distributed in [1...10]. The size of each packet is 1,500 bytes.

To investigate the advantage of having network QoS, one user from each site is chosen with a higher ToS weight. High priority users have a weight of 2 while normal users are assigned a weight of 1. Therefore high-priority users will be treated better if a `SCFQScheduler` is used.

C. Analysis

The result in table II shows the advantage of having network QoS in a shared network environment. As mentioned previously, only 4 out of 20 users are given a higher priority for sending their jobs. On average, they manage to finish all of their execution jobs faster by more than 3%.

Table III shows the average amount of time spent by one packet in a router's queue, in this case the leaf router at Melbourne. This router is chosen because it links two grid resources, hence more traffic than other leaf routers. We compare two 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 in the SCFQ router are dequeued faster than normal priority packets, thus providing better QoS to high priority users.

TABLE II

NETWORK QOS USING SCFQ PACKET SCHEDULER

Priority	With background traffic (in simulation minutes)
High	22.82
Normal	23.57

TABLE III

AN AVERAGE PACKET LIFETIME AT THE MELBOURNE LEAF ROUTER

Priority	With FIFO scheduler (in simulation seconds)	With SCFQ scheduler (in simulation seconds)
High	3.60×10^{-6}	1.20×10^{-6}
Normal	2.38×10^{-6}	2.38×10^{-6}

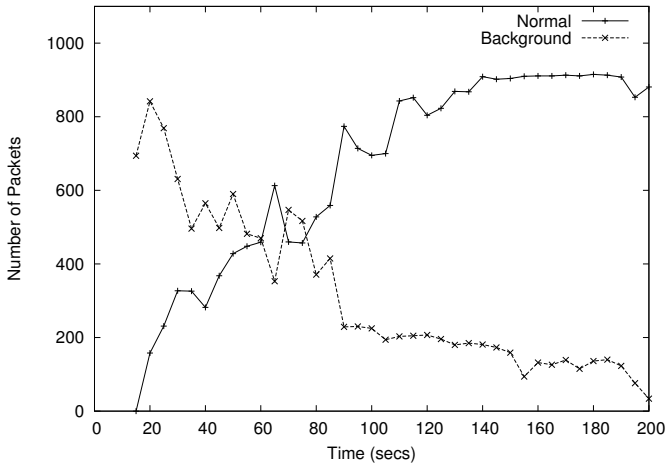

Fig. 6. Number of Packets passing through the Melbourne Leaf Router

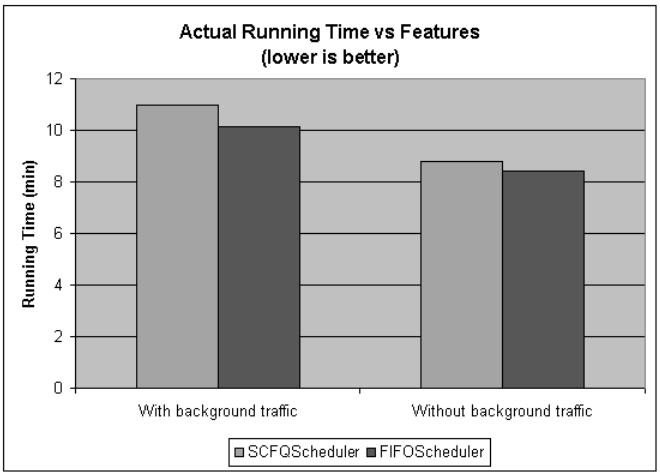

Fig. 7. Comparison between actual running time vs more features

Figure 6 shows the number of packets passing by for the Melbourne leaf router. The effect of background traffic is clearly shown in the beginning of the simulation. The background traffic decreases over time because when a user entity collects all the jobs, it will exit the simulation, hence not sending any other network packets.

It can be seen from figure 7 that using the SCFQ scheduler increases the run time of the experiment. This is because of the extra overheads associated with SCFQ as mentioned earlier. The FIFO scheduler, on the other hand, simply enqueues at the tail and dequeues at the head. With no background traffic on, the difference between `FIFOScheduler` and `SCFQScheduler` is quite small around 4%. The gap, however, is bigger around 7% with the addition of background traffic. The experiments were conducted on a Pentium 4 machine running at 2.4Ghz and equipped with 512MB RAM.

V. 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.

There are some tools available, apart from GridSim, for application scheduling simulation in Grid computing environments, such as Bricks [22], MicroGrid [23] [24], SimGrid [25] [26], and OptorSim [27]. 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 IV. Note that for Routing Table Entry column, an automatic entry means filling in 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 [22] 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 [28]. As a result, we are not able to compare it with our work in more details. Therefore, it is not included in Table IV.

MicroGrid [23] [24] 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 [29].

SimGrid [25] [26] 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) [30]. 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 III-B.

OptorSim [27] 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.

TABLE IV

LISTING OF NETWORK FUNCTIONALITIES AND FEATURES FOR EACH GRID SIMULATOR

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

From the above discussion and Table IV, 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.

VI. 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 ver3.1` 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. In addition to this, we are also planning an ability to design the network topology using scripts similar to ns-2.

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] C. Simatos, "Making simjava count," MSc. Project report, The University of Edinburgh, September 12 2002.

[6] M. Priestley, *Practical Object-Oriented Design with UML*. McGraw-Hill, 2000.

[7] G. Malkin, "RFC 2453: RIP version 2," November 1998. [Online]. Available: http://www.apps.ietf.org/rfc/rfc2453.html

[8] J. Moy, "RFC 2328: OSPF version 2," April 1998. [Online]. Available: http://www.ietf.org/rfc/rfc2328.txt

[9] J. Postel, "Internet control message protocol: Darpa internet program protocol specification," September 1981. [Online]. Available: http://www.ietf.org/rfc/rfc0792.txt

[10] 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.

[11] 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.

[12] 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

[13] C. Simatos, *The SimJava Tutorial*, The University of Edinburgh, 2002. [Online]. Available: http://www.icsa.inf.ed.ac.uk/research/groups/hase/simjava/

[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] 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.
[23] 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.
[24] X. Liu and A. Chien, "Realistic large-scale online network simulation," in *Proc. of IEEE Supercomputing 2004*, Pittsburgh, USA, November 6–12 2004.
[25] 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.
[26] 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.
[27] 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 International Journal of High Performance Computing Applications*, vol. 7, no. 4, pp. 403–416, 2003.
[28] "Bricks: A performance evaluation system for grid computing scheduling algorithms." [Online]. Available: http://www.is.ocha.ac.jp/ takefusa/bricks/index.shtml
[29] 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.
[30] 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.