Available online at www.sciencedirect.com
ScienceDirect
Future Generation Computer Systems 23 (2007) 606–615
FGCS FUTURE GENERATION COMPUTER SYSTEMS
www.elsevier.com/locate/fgcs

On incorporating differentiated levels of network service into GridSim

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

[a] *Grid Computing and Distributed Systems Laboratory, Department 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, Singapore*

Received 28 February 2006; received in revised form 4 October 2006; accepted 15 October 2006
Available online 20 November 2006

Abstract

Grid computing technologies are increasingly being used to aggregate computing resources that are geographically distributed. 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 the effects of using this shared infrastructure during simulations. In this paper, we discuss how new additions to the GridSim simulation toolkit can be used to explore network effects in Grid computing. We also investigate techniques to incorporate differentiated levels of service, background traffic and the collection of 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, which aggregates dispersed heterogeneous resources for solving various kinds of large-scale parallel applications in science, engineering and commerce [10]. In order to evaluate the performance of a Grid environment, we need to conduct *repeatable* and *controlled* experiments, which are difficult due to the 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 requires the handling of 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 [5] 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 the heterogeneous computational resources of various configurations. The GridSim toolkit has been applied successfully to simulate a Nimrod-G-like [6] Grid resource broker and to evaluate the performance of deadline and budget constrained cost- and time-optimization scheduling algorithms.

Communication networks serve as fundamental components 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 during the experiment. Resources and Grid users had all-to-all connections with specifiable bandwidths. 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 by enhancing the ability to simulate realistic network models by: (1) allowing users to create a network topology, (2) packetizing data into smaller chunks for sending over a network, (3) generating background traffic, and (4) incorporating different levels of service for sending packets.

The rest of this paper is organized as follows: Section 2 provides some relevant background on GridSim. Section 3

* Corresponding author. Tel.: +61 3 8344 1360; fax: +61 3 9348 1184.
E-mail address: anthony@csse.unimelb.edu.au (A. Sulistio).

0167-739X/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2006.10.006

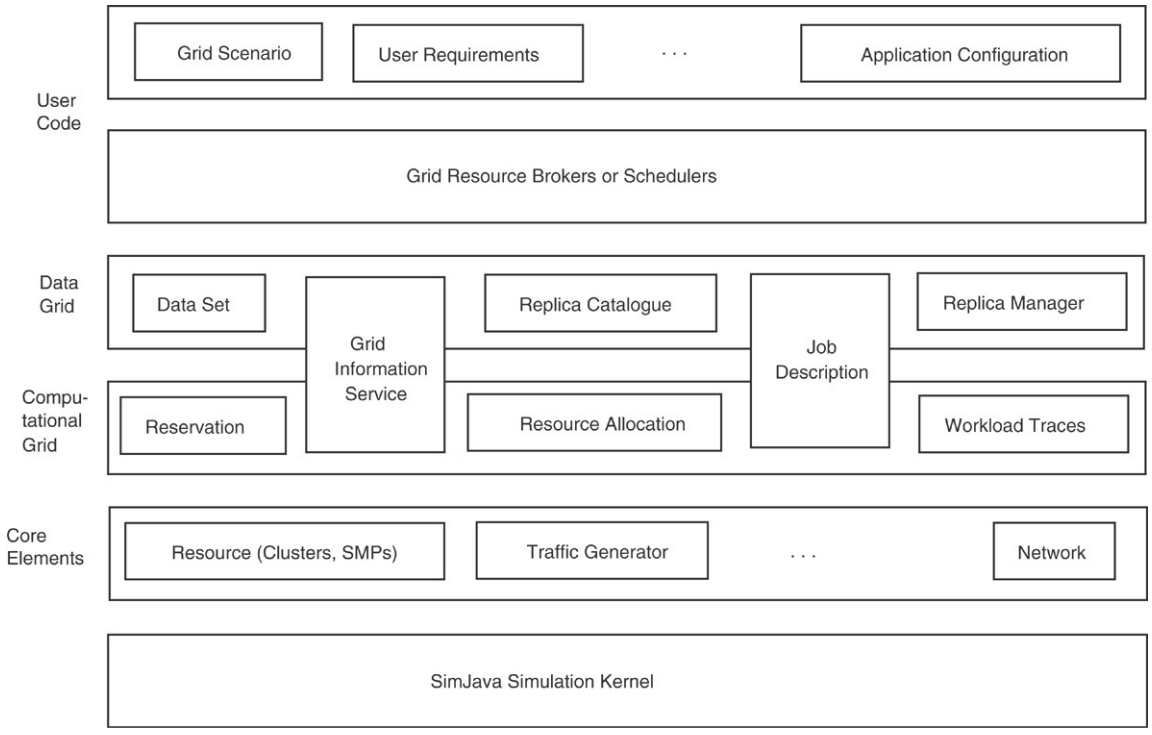


Fig. 1. GridSim architecture.

presents the design and implementation of the network additions to GridSim, 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 significant work done in the past to incorporate more functionality and extensibility into `GridSim ver3.0`, such as extending the GridSim infrastructure to support advance reservation as discussed in [26]. This allows resources to have their own schedulers and policies in reservation-based systems. However, no work has been done on improving the existing network model. Therefore, in the newer GridSim release, a new package is incorporated to enhance the capabilities of the existing network model. This package contains core network components, such as links and routers. Details of these components will be discussed in Section 3. Also, our use of the term GridSim denotes the latest version of the software throughout.

2.1. Overall GridSim architecture

GridSim is based on SimJava [23], 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 layered GridSim architecture captures the model of the Grid computing environment. The overall GridSim architecture is shown in Fig. 1.

The first layer at the bottom of Fig. 1 is managed by SimJava for handling the interaction of events among GridSim components. The second layer represents the infrastructure components of GridSim, such as network and resource hardware. The third and fourth layers are concerned with the 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 any new requirements of running Data Grids. The fifth and sixth layers allow users to extend GridSim as needed.

2.2. Features

Some of GridSim's features are outlined below:

- It allows the modeling of different resource characteristics and types;
- It enables the simulation of workload traces taken from real supercomputers;
- It supports a reservation-based mechanism for resource allocation;
- It allocates incoming jobs based on space- or time-shared mode;
- It has the ability to schedule compute- and/or data-intensive jobs as discussed in [27];
- It provides clear and well-defined interfaces for implementing different resource allocation algorithms; and
- It allows the modeling of several regional GIS components.

With these features, it gives researchers the functionality and the flexibility of simulating Grids for various topics, such as evaluating a fairshare scheduling in a decentralized architecture [9], or analyzing security solutions in Grids [20].

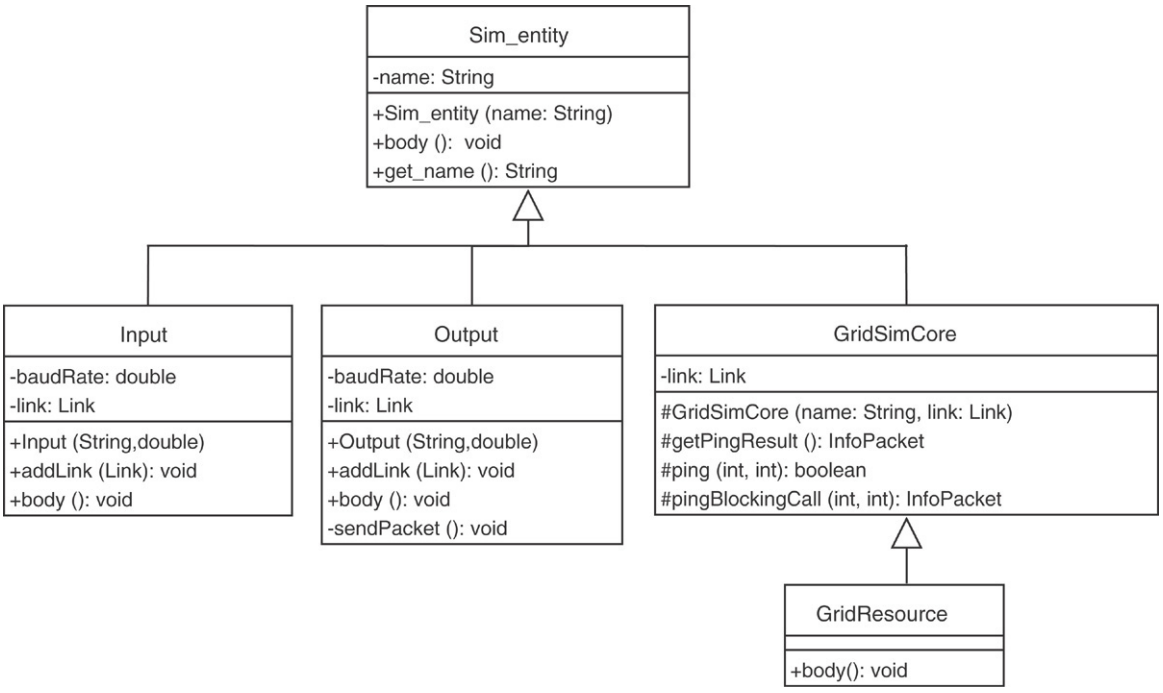


Fig. 2. A class diagram showing the relationship between GridSim and SimJava entities.

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 via the class `Sim_port`: one for sending events to other entities, and the other for receiving incoming events. In GridSim, this is represented 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 the `Sim_entity` and GridSim classes is shown in Fig. 2. In a class diagram, attributes and methods are prefixed with characters +, # and − indicating access modifiers as public, protected, and private respectively. Note that the class `GridSimCore` does not have the `body()` method because it is not necessary (since its subclass will override the method). Moreover, only attributes and methods relevant to this work are shown here, and will be discussed later.

3. Design and implementation of the 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` classes, 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 for the modeling of a communication delay. In addition, this existing design provides a clean interface between the network entities and others. Therefore, most of the changes incorporated were in the classes `Input` and `Output`, resulting in 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 Fig. 3. A detailed explanation of this figure will be given 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 amongst these network components, in Unified Modeling Language (UML) notations [21], are depicted in Fig. 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 Fig. 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

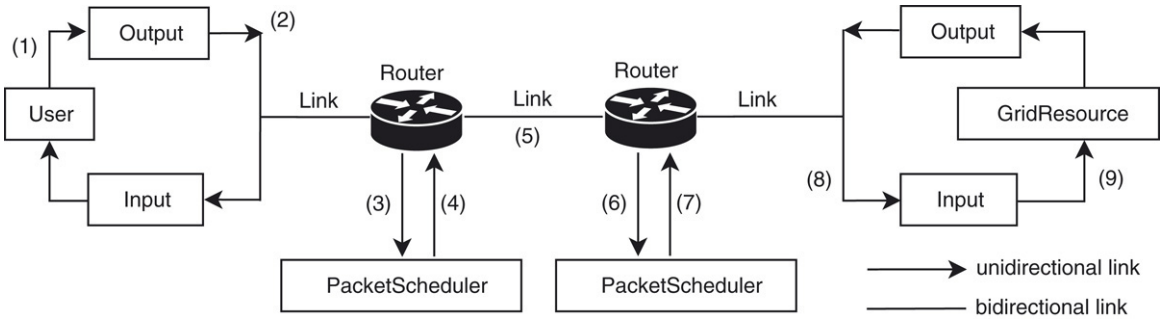


Fig. 3. Interaction amongst GridSim network components.

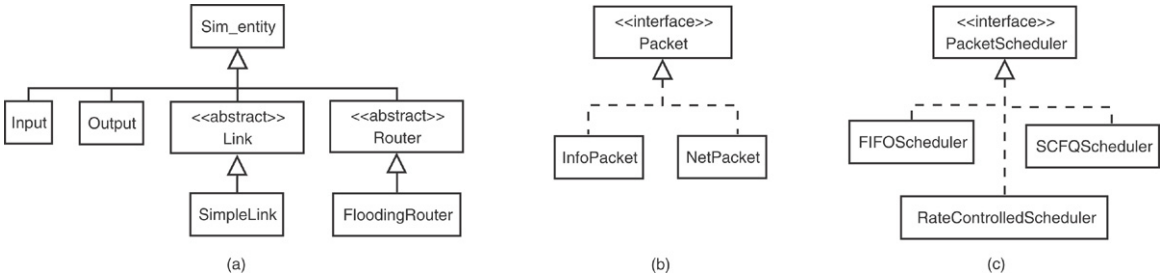


Fig. 4. Generalization and realization relationships in UML for GridSim network classes.

mentioned. The *Output* entity is responsible for splitting the data into MTU-sized packets, whereas the *Input* entity is responsible for collating the different packets in a stream, and for sending 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 Fig. 4(a). Therefore, this design allows a subclass of `Router` to determine the forwarding table at the start of the simulation, and the ability to implement it using any routing algorithms.

Routing can be done using static tables or dynamic methods, such as the Routing Information Protocol (RIP) [18] and Open Shortest Path First (OSPF) [19]. The implementation of a router in class `FloodingRouter` uses a flooding algorithm to set up its forwarding tables automatically. Since routers and other GridSim entities cannot be created and added after the simulation has started, the flooding algorithm is a sufficient method to set up a router's forwarding tables.

3.1.4. Packet

A network packet in GridSim is represented as an interface class `Packet` as shown in Fig. 4(b). Currently, there are two classes that belong to this category; these are `NetPacket` and `InfoPacket`. A `NetPacket` class is used to encapsulate data passing through the network, whereas class `InfoPacket` is devoted to gathering network information during runtime, which is equivalent to the Internet Control Message Protocol (ICMP) [22] 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 Fig. 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) [8], called Self Clocked Fair Queuing (SCFQ) [13] policy. The `RateControlledScheduler` is an implementation of a rate-jitter controlling regulator [32].

3.2. Support for network quality of service and runtime information

Jobs on Grids may have different requirements with respect to bandwidth and latency. Systems like those dealing with fire or earthquake detection require low latency and reliable delivery of packets. 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) levels while using their networks [3]. In order to support this functionality, every packet in GridSim contains a Type of Service (ToS) attribute with a default weight value of zero. 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 will 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-workconserving policies have some benefits, like lower buffer space requirements and smoothing 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 entity can use either a blocking or non-blocking method call from the class `GridSimCore`, as shown in Fig. 2. A blocking call needs to use only a `pingBlockingCall()` method, where it waits for a result to come back and prevents any other activities from progressing. In contrast, a non-blocking call uses a combination of `ping()` and `getPingResult()` methods. Hence, rather than stopping to wait for the result, the entity can continue with other activities. Both the `pingBlockingCall()` and the `getPingResult()` methods return an object of class `InfoPacket`.

3.3. Simulating with background traffic

In commercial or even academic networks, users expect to experience delays due to network traffic that does not belong to them. In order to capture this real world scenario within a simulation, GridSim supports the modeling of background traffic. This can be done by creating an instance of class `TrafficGenerator`, and storing it as an attribute inside the class `Output`. The class `TrafficGenerator` generates the inter-arrival time, packet size, and number of packets for each interval according to various distributions that are supported by SimJava [23]. 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 amongst 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 their neighboring routers in turn (except the source). Depending on the complexity of a network topology and the 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 Fig. 3, can start sending jobs to a *GridResource* entity. Each GridSim entity has I/O entities attached to it that act as buffers. 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, enqueued in a buffer, and sent to the link that connects the entity to the neighboring 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 which packet scheduler 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, in the same way as the *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). A 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 the User Datagram Protocol (UDP). There is no support for acknowledging each packet or for 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 the behavior of the network components and the packet scheduling algorithms implemented in GridSim. Hence, we are trying to look at:

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

In order to conduct this experiment, we use a network topology based on the EU DataGrid TestBed I [28]. The topology from this production Grid is chosen because we also want to show GridSim's ability to simulate an adequate-size Grid testbed. Note that the network topology can also be a generic one and not specific to Grid testbeds (Fig. 5).

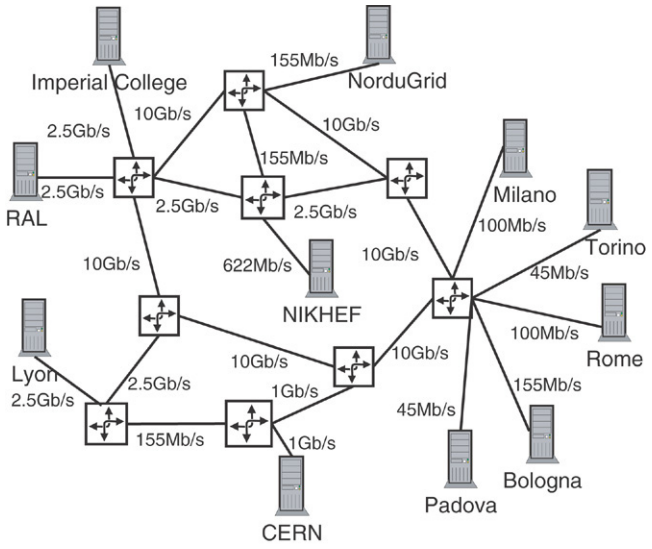


Fig. 5. Network topology.

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

4.2. Experiment setup

Table 1 summarizes all the relevant resource information. Each node of a resource is assumed to be a 2 GHz AMD Opteron processor. We took the data about the resources and scaled them down. Only the number of nodes and their CPU ratings were scaled down by 10; the network parameters are not affected. The scaling was done primarily to reduce the runtime of the experiment and its memory consumption. The complete simulation would have required more than 2 GB of memory. However, the network parameters were kept the same as in the original information.

In GridSim, total processing capability of a resource's CPU is modelled in the form of its Million Instructions Per Second (MIPS) rating as per SPEC (Standard Performance Evaluation Corporation) CPU (INT) 2000 [25] benchmarks. A space shared policy or First Come First Served (FCFS) algorithm is used to compute incoming jobs for all resources. In addition, all links share the same characteristics: 1 MTU size of 1500 bytes and a latency of 10 ms.

There are four users located on each of the resources (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 42,000k Million Instructions (MI) $\pm$ 30%, which is around 10 min 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 who submit all their jobs approximately every 5 min.

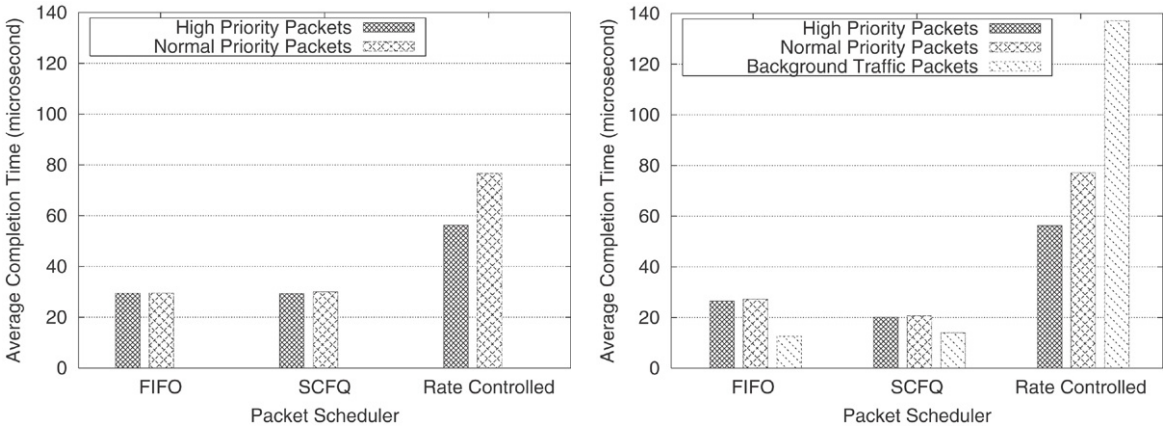
To incorporate background traffic functionality into this experiment, selected users are chosen as the sources to generate these background packets. A Poisson distribution is used, with an inter-arrival time of 1 min. In addition, the total number of packets for each interval is uniformly distributed in [500...1000], where the size of each packet is 1500 bytes.

To investigate the advantage of having differentiated network QoS levels, 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% and 35% of the network bandwidths at each link respectively, with background traffic receiving 10% of the bandwidth.

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 scheduling policy implementations, time- and space-shared are provided. However, a user can easily implement a different scheduling policy as described in [26] because of well-defined interfaces between the 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 the user's needs and requirements. This can be done by extending the `GridSimCore` class and writing the necessary code inside a `body()` method.
4. Build a network topology by connecting the Grid user and resource entities. At the moment, the connecting of these entities needs to be done manually, by first creating network objects such as `Router` and `Link`. Then, these entities are connected to a `Router` object using an `attachHost()` method. An `attachRouter()` method can be used to link one or more routers.



(a) Without background traffic.

(b) With background traffic.

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

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 Fig. 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, and hence it routes a substantial amount of incoming and outgoing traffic.

As mentioned previously, we compare two types 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 in the FIFO experiment, thus providing a better QoS to high priority users.

For the FIFO experiment shown in Fig. 6, all packets are routed based on the arrival time. Hence, there is no prioritization for these packets. On the other hand, for the SCFQ experiment as shown in Fig. 6, high priority packets are dequeued faster than normal packets by more than 2%. An interesting observation in the SCFQ experiment of Fig. 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 those times at which other packets were not there.

For the Rate Controlled experiment displayed in Fig. 6, high priority packets are dequeued faster than normal packets by approximately 36%. The main reason is because, as mentioned earlier, each class of user is assigned to a certain 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, as illustrated in Fig. 6(a) and (b) respectively.

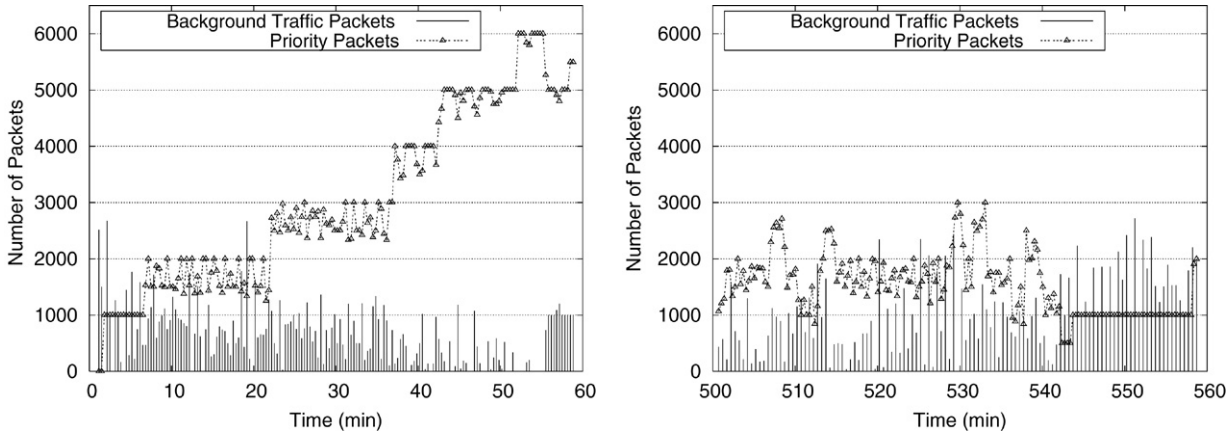
As expected, high priority packets spent less time using the SCFQ scheduler rather than the FIFO one. However, it is also interesting to note that the Rate Controlled scheduler dequeued these packets the slowest of all, as shown in Fig. 6. This is because, as stated earlier, the Rate Controlled scheduler 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 the 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 increases in memory and processing requirements, hence higher costs. When prioritization rather than guarantees are required, an SCFQ should be used. An SCFQ scheduler is also a simpler algorithm to implement than Rate Controlled schedulers.

The effect of the background traffic in the experiment is shown in Fig. 7. This figure shows the number of packets passing through the CERN router for a specific period of time. On average, the background packets accounted for 36% of total packets passed by the CERN router, as shown in Fig. 8.

5. Related work

Simulation is widely used in the networking research area. Examples of such simulators include NS-2 [29], DaSSF [17], OMNET++ [30] and J-Sim [14]. Though their support for network protocols is extensive, they are not targeted at studying Grid computing. This is because simulating Grids



(a) In the beginning of the experiment.

(b) In the middle of the experiment.

Fig. 7. Number of packets passing through the CERN router during the rate controlled experiment.

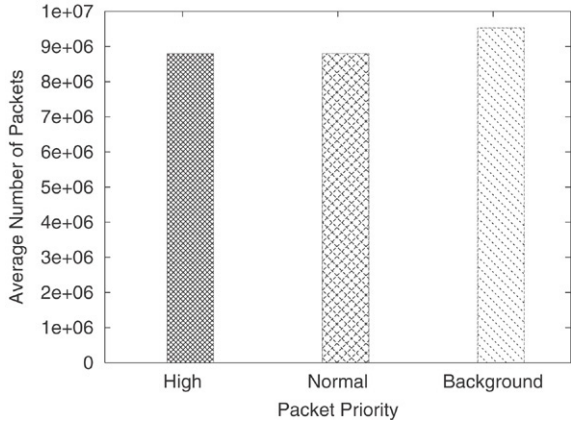


Fig. 8. Average number of packets passing through the CERN router for all experiments.

requires modeling the effects of job scheduling algorithms on Grid resources, and investigating users' QoS requirements for application processes. In addition, we believe that simulating TCP and UDP connections are sufficient to model real world behavior, because Grid users are mostly interested in finding out RTT and available bandwidth of a host, based on the differentiated services offered by various packet scheduling algorithms. Therefore, we extend GridSim to incorporate these sets of features as an alternative to building Grid services on top of a network simulator. However, we are also considering the feasibility of integrating GridSim with a network simulator such as J-Sim (written in Java), for more advanced network functionalities.

There are some tools available, apart from GridSim, for application scheduling simulation in Grid computing environments, such as Bricks [1], MicroGrid [24,16], SimGrid [7,15], and OptorSim [2]. All of these simulators also have an underlying network infrastructure, with the ability to simulate realistic experiments by using background traffic. The differences among these Grid simulators, except for Bricks, in terms of network functionalities and other existing features are highlighted in Table 2. Note that for the 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 [1] is able to specify a network's 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 was being written, this package was not available to download from its website [4]. As a result, we were not able to compare it with our work in further detail. Therefore, it is not included in Table 2.

MicroGrid [24,16] allows complex network modeling, such as transport and routing protocols and large-scale experiments, since it is based on DaSSF [17]. Hence, in terms of network capabilities, MicroGrid is the most complete of all Grid simulators. However, it is actually an emulator, meaning that the actual application code is executed on the virtual Grid modeled after Globus [11].

SimGrid [7,15] has a good network infrastructure that supports the Transmission Control Protocol (TCP) transport protocol for a reliable service. It also models background traffic by reading from a trace file generated by the Network Weather Service (NWS) [31]. NWS is used to monitor the currently 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 both modeled as resources performing specific tasks. Therefore, it does not support data packetization. In addition, requests for network status functionalities during runtime in SimGrid are limited to the 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 [2] has a very simple network infrastructure model compared to other simulation tools, since it does not support routing and transport protocol or data packetization. The background traffic functionality can only be modeled

Table 2
Listing of network functionalities and other existing 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
Data replication	Supported	Not supported	Not supported	Supported
Disk I/O overheads	Supported	Supported	Not supported	Not supported
Complex file filtering/data query	Supported	Not supported	Not supported	Not supported
Scheduling user jobs	Supported	Supported	Supported	Not supported
CPU reservation of a resource	Supported	Not supported	Not supported	Not supported
Workload trace-based simulation	Supported	Not supported	Not supported	Not supported

by using a Landau distribution. In addition, simulating with background traffic requires a configuration file that describes a network's topology in a matrix format.

From the above discussion and Table 2, GridSim has successfully incorporated QoS into a network for scheduling packets, which is not supported by other Grid simulators. In addition, GridSim provides a good set of network functionalities, some of which are not supported in the other Grid simulators. The combination of these functionalities with its previously existing features enables GridSim to model an integrated Grid platform of computing, networks, data, storage and resource allocation algorithms.

6. Conclusion and further work

The network serves as a fundamental component in Grid computing, since resources and users are connected over a network topology with shared bandwidth. Previously, GridSim did not have the ability to specify a network topology or the functionality to connect resources through network links during experiments. In this work, modifications to 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 a network's topology and its Grid resources can have on their jobs. This paper explores various types of network elements in GridSim, like routers, links, and packet schedulers; and how they can be extended to add more functionalities. Moreover, GridSim has exciting new features such as the ability to generate background traffic during an experiment, to request network information during runtime, and to provide differentiated service levels 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 levels of service, can help high priority users achieve better QoS than normal users. However, providing differentiated qualities of 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 incorporate different protocols, such as TCP, for dealing with lost packets and GridFTP (as part of the Globus Toolkit [12]) for transferring huge amounts of data. Finally, we are considering the addition of other type of network building blocks, like switches and domain gateways, or integration with a network simulator, such as J-Sim, for simulating advanced network functionalities.

Software availability

The latest version of the GridSim toolkit with source code and examples, can be downloaded from the following website: http://www.gridbus.org/gridsim/.

Acknowledgements

We thank Uros Cibej for his work on implementing the functionality of creating a network topology from a file. We also thank CS Yeo, Hussein Gibbins and anonymous reviewers for their comments.

References

[1] K. Aida, A. Takefusa, H. Nakada, S. Matsuoka, S. Sekiguchi, U. Nagashima, Performance evaluation model for scheduling in a global computing system, The International Journal of High Performance Computing Applications 14 (3) (2000) 268–279.

[2] W.H. Bell, D.G. Cameron, L. Capozza, A.P. Millar, K. Stockinger, F. Zini, Optorsim—a Grid simulator for studying dynamic data replication strategies, The International Journal of High Performance Computing Applications 7 (4) (2003) 403–416.

[3] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, W. Weiss, RFC 2475: An architecture for differentiated service, Available: http://www.ietf.org/rfc/rfc2475.txt, December 1998 (online).

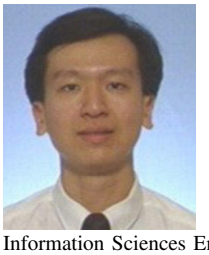
[4] Bricks: A performance evaluation system for Grid computing scheduling algorithms. Available: http://www.is.ocha.ac.jp/~takefusa/bricks/index.s.html (online).

[5] R. Buyya, 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 14 (2002) 13–15.
[6] R. Buyya, D. Abramson, 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.
[7] 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.
[8] A.J. Demers, S. Keshav, 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.
[9] E. Elmroth, P. Gardfjall, Design and evaluation of a decentralized system for Grid-wide fairshare scheduling, in: Proc. of the 1st IEEE International Conference on e-Science and Grid Computing, Melbourne, Australia, December 5–8, 2005.
[10] I. Foster, C. Kesselman (Eds.), The Grid: Blueprint for a Future Computing Infrastructure, Morgan Kaufmann Publishers, 1999.
[11] I. Foster, C. Kesselman, Globus: A metacomputing infrastructure toolkit, The International Journal of Supercomputer Applications and High Performance Computing 11 (2) (1997) 115–128.
[12] I. Foster, Globus toolkit version 4: Software for service-oriented systems, in: IFIP International Conference on Network and Parallel Computing, in: LNCS, vol. 3779, Springer-Verlag, 2005, pp. 2–13.
[13] 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.
[14] J-Sim. Available: http://www.j-sim.org/ (online).
[15] A. Legrand, L. Marchal, 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.
[16] X. Liu, A. Chien, Realistic large-scale online network simulation, in: Proc. of IEEE Supercomputing 2004, Pittsburgh, USA, November 6–12, 2004.
[17] J. Liu, D.M. Nicol, DaSSF 3.1 User's Manual, Dartmouth College, 2001, April.
[18] G. Malkin, RFC 2453: RIP version 2. Available: http://www.apps.ietf.org/rfc/rfc2453.html, November 1998 (online).
[19] J. Moy, RFC 2328: OSPF version 2, Available: http://www.ietf.org/rfc/rfc2328.txt, April 1998 (online).
[20] S. Naqvi, M Riguidel, Grid security services simulator (G3S)—A simulation tool for the design and analysis of grid security solutions, in: Proc. of the 1st IEEE International Conference on e-Science and Grid Computing, Melbourne, Australia, December 5–8, 2005.
[21] M. Priestley, Practical Object-Oriented Design with UML, McGraw-Hill, 2000.
[22] J. Postel, Internet control message protocol: Darpa internet program protocol specification, Available: http://www.ietf.org/rfc/rfc0792.txt, September 1981 (online).
[23] C. Simatos, Making simjava count, M.Sc. Project report, The University of Edinburgh, September 12, 2002.
[24] H.J. Song, X. Liu, D. Jakobsen, R. Bhagwan, X. Zhang, K. Taura, A. Chien, The microGrid: A scientific tool for modeling computational Grids, in Proc. of IEEE Supercomputing 2000, Dallas, USA, November 4–10, 2000.
[25] Spec—standard performance evaluation corporation. Available: http://www.spec.org/ (online).
[26] A. Sulistio, 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.
[27] A. Sulistio, U. Cibej, R. Buyya, 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, November 8, 2005.
[28] The European DataGrid project homepage. http://eu-dataGrid.web.cern.ch/eu-dataGrid, 2005.
[29] The network simulator—ns-2. Available: http://www.isi.edu/nsnam/ns/ (online).
[30] A. Varga, The omnet++ discrete event simulation system, in Proc. of the European Simulation Multiconference, ESM'01, Prague, Czech Republic, June 6–9, 2001.
[31] R. Wolski, N. Spring, J. Hayes, The network weather service: A distributed resource performance forecasting service for metacomputing, The Journal of Future Generation Computing Systems 15 (5–6) (1999) 757–768.
[32] H. Zhang, D. Ferrari, Rate-controlled static-priority queueing, in: INFOCOM, 1993, pp. 227–236.
[33] H. Zhang, S. Keshav, Comparison of rate-based service disciplines, in: SIGCOMM, 1991, pp. 113–121.

Anthony Sulistio is a Ph.D. student at the Department of Computer Science and Software Engineering (CSSE), University of Melbourne, Australia. He received his B.E. and M.S.S.E. degree from University of Melbourne in 2001 and 2002 respectively. His main research interests are in grid computing, grid simulation, parallel programming and software engineering.

Gokul Poduval is a Post-Graduate student at the Department of Electrical and Computer Engineering (ECE) of the National University of Singapore (NUS). His research interests are in coordinated quality of service (QoS) management in computational and network systems. He obtained his Bachelor's degree in 2003 at the same university after receiving scholarship from Singapore Airlines and Neptune Orient Lines.

Rajkumar Buyya is a Senior Lecturer and the Director of the Grid Computing and Distributed Systems Laboratory within the Department of CSSE, University of Melbourne. He has authored/co-authored over 130 papers and technical documents that include three books—Microprocessor x86 Programming, Mastering C++, and Design of PARAS Microkernel. He received B.E, M.E, and Ph.D. degrees from Mysore, Bangalore, and Monash Universities respectively. He was awarded Dharma Ratnakara Memorial Trust Gold Medal for academic excellence in Mysore University. He is currently serving as the Chair of the IEEE Technical Committee on Scalable Computing (TCSC) and Associate Editor of the Journal of Future Generation Computing Systems (FGCS), Elsevier Press, Holland.



Chen-Khong Tham is an Associate Professor at the Department of Electrical and Computer Engineering (DECE) of the National University of Singapore (NUS). His research interests are in coordinated quality of service (QoS) management in wired and wireless computer networks and distributed systems, and distributed algorithms for resource allocation, machine learning and decision making. He obtained his Ph.D. and M.A. degrees in Electrical and Information Sciences Engineering from the University of Cambridge, United Kingdom, and held a 2004/05 Edward Clarence Dyason Universitas21 Fellowship at the University of Melbourne, Australia.