Simulation of Buffer Management Policies in Networks for Grids

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

[1]Departamento de Sistemas Informáticos
Universidad de Castilla La Mancha
Campus Universitario s/n. 02071, Albacete. Spain
{agustin, blanca, carmen}@dsi.uclm.es

[2]Grid Computing and Distributed Systems Laboratory
Dept. of Computer Science and Software Engineering
The University of Melbourne
111 Barry St, Carlton VIC 3053 Australia
{anthony, raj}@csse.unimelb.edu.au

Abstract

Grid technologies are emerging as the next generation of distributed computing, allowing the aggregation of resources that are geographically distributed across different locations. The network remains an important requirement for any Grid application, as entities involved in a Grid system (such as users, services, and data) need to communicate with each other over a network. The performance of the network must therefore be considered when carrying out tasks such as scheduling, migration or monitoring of jobs. Network buffers management policies affect the network performance, as they can lead to poor latencies (if buffers become too large), but also leading to a lot of packet droppings and low utilization of links, when trying to keep a low buffer size. Therefore, network buffers management policies should be considered when simulating a real Grid system. In this paper, we introduce network buffers management policies into the GridSim simulation toolkit. Our framework allows new policies to be implemented easily, thus enabling researchers to create more realistic network models. Fields which will harness our work are scheduling, or QoS provision. We present a comprehensive description of the overall design and a use case scenario demonstrating the conditions of links varied over time.

*Corresponding author. Tel.: +34 967 59 92 00 ext. 2693; fax: +34 967 59 93 43

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 [11]. Grid systems are highly variable environments, made of a series of independent organizations that share their resources [12]. Some application domains that take advantage of the Grids are collaborative visualization [21] and medical applications [5].

The network remains an important requirement for Grid applications, as entities involved in a Grid (e.g. users, services, and data) need to communicate with each other over a network [24]. Floyd et al. demonstrated in [9] that the utilization level of the network links is heavily affected when the buffer management policy is not efficiently tuned. Thus, since the performance of the network is affected by them, network buffers policies also affect the performance of Grids.

A number of policies have been developed to manage network buffers [10, 9, 17, 3]. According to Floyd et al. [10], Random Early Detection (RED) algorithm is useful to detect incipient network congestion in packet-switched networks, which is similar to computational Grids since the Grid infrastructure is built on existing public network. However, RED algorithm might not be efficient under varying traffic conditions, as it requires constant tuning of its parameters to be able to work efficiently. This makes RED not suitable to be used when providing network Quality of Service (*QoS*), as a misconfiguration of RED parameters would seriously affect the performance received by users [9]. Therefore, Floyd et al. [9] suggest using Adaptive RED (ARED) algorithm to overcome this drawback.

Kumar et al. [17] present a buffer management framework for achieving end-to-end proportional loss differentiation in networks, which is also based on RED. Aweya et al. [3] present a technique for enhancing the effectiveness of RED by dynamically changing the threshold settings as the number of connections and system load changes. Kesselman et al. [16] introduce a novel general non-preemptive buffer management scheme, which considers the queues ordered by their size. Gazi et al. [14] propose a threshold-based dynamic buffer management policy, decay function threshold, to regulate the lengths of very active queues and avoid performance degradations.

The contribution of this paper is as follows. We design and implement buffer policies, such as FIFO, RED and ARED on GridSim [26], an open-source Grid simulation tool, since it can simulate both computational and data Grids. Moreover, several researchers have been using this simulator (such as [7, 23, 25]). More importantly, GridSim allows the flexibility and extensibility to incorporate new components into its existing infrastructure. We decided to implement these two versions of RED because RED (or policies based on it) is a widely used buffer management policy, as we showed above. Hence, our work benefits researchers for evaluating and improving their scheduling works against volatile network conditions.

This paper is organized as follows: Section 2 provides a overview on several Grid and network simulation tools. Section 3 provides a brief explanation of the buffer management policies implemented in this work. Section 4 describes the implementations on GridSim, which are supported by the results depicted in Section 5. Section 6 concludes the paper and suggests some guidelines for future work.

2 Related Work

As we mentioned previously, simulations are essential for carrying out research experiments in Grid systems. Thus, a number of simulation tools have been developed, such as GridSim [26], OptorSim [4], SimGrid [13], and MicroGrid [20]. Sulistio et al. [26] provide a detailed comparison of these simulation tools in terms of their network functionalities and features. The simulation tool we use for our work is GridSim. Moreover, [26] also provides an in-depth explanation of its network components. In this paper, we are interested in the simulation of network buffer management policies, and none of these tools provide the mentioned functionality.

Simulations are also widely used in the networking research area. Examples of such simulators are NS-2 [1], DaSSF [19], OMNET++ [27] and J-Sim [22]. Although 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 users QoS requirements for application processes. In addition, we believe simulating TCP and UDP connections are sufficient to model a real world behavior, because Grid users are mostly interested in finding out

round trip time and available bandwidth of a host. Therefore, these network simulators perform other complex functionalities which are not needed in simulating a Grid computing environment [26].

As a result, we decided to extend GridSim with a better network model rather than integrating an existing simulator tool into it. We decided it because GridSim is a very versatile tool that can be extended in an easy and efficient way. Also, integrating a network simulator would be more complicated and time-consuming than extending GridSim, and such integration would not be so interesting for the purposes of Grid researchers.

3 Buffer Management Policies

The aim of our work is the extension of the GridSim Toolkit with network buffer management policies. Thus, in this section, we provide an overview on the policies we have implemented, namely RED, Adaptative RED, and FIFO. FIFO is an straightforward policy, which just enqueues packets into buffers, and packets get dropped when buffers are full. The other two policies will be explained the next.

3.1 Random Early Detection

One of the most widely used policies is Random Early Detection (RED) [10]. RED routers detect incipient congestion by computing the average queue size. The router could notify congestion to connections either by dropping packets arriving at the router or by setting a bit in packet headers. When the average queue size exceeds a preset threshold, the router drops or marks each arriving packet with a certain probability, where the exact probability is a function of the average queue size. RED routers keep the average queue size low while allowing occasional bursts of packets in the queue [10].

During congestion, the probability that the router notifies a particular connection to reduce its window is roughly proportional to that connection's share of the bandwidth through the router. RED routers are designed to accompany a transport-layer congestion control protocol such as TCP. The RED router has no bias against bursty traffic and avoids the global synchronization of many connections decreasing their transmission window at the same time [10].

Algorithm 1 Calculation of average queue size.

1: **if** queue is non-empty **then**
2: $avg \leftarrow old\_avg + w_q(q - old\_avg)$
3: **else**
4: $avg \leftarrow (1 - w_q)^{(time - q\_time)/s} \times old\_avg$
5: **end if**

The RED congestion control mechanisms monitor the average queue size for each output queue, and, using randomization, choose connections to notify of that congestion. Transient congestion is accommodated by a temporary increase in the queue. Longer-lived congestion is reflected by an increase in the computed average queue size, and results in randomized feedback to some of the connections to decrease their windows [10].

The RED algorithm can be graphically seen in Figure 1. The parameters are listed here: avg: current average queue size; old_avg: previous average queue size; w_q: queue weight, the significance of the current queue size when calculating the average queue size; q: current queue size; q_time: the time when the queue got empty for the last time; $time$: current time; s: typical transmission time. max_{th}: maximum threshold, upper limit for the average queue size; min_{th}: minimum threshold, lower limit for the average queue size; max_p: the maximum probability for an incoming packet to get dropped; For each incoming packet, the average queue size is calculated. If it is below the minimum threshold, the packet is enqueued. If it is above the maximum threshold, the packet is dropped. Otherwise, the packet is dropped with some probability. The average queue size calculation is explained by Algorithm 1. This algorithm works as follows: if the queue is not empty, the average queue size is calculated based on old_avg, q, and w_q (line 2). Otherwise, the it is calculated based on old_avg, w_q, $time$, and q_time (line 4).

RED algorithm has been used as a basis for the development of other algorithms. Among the algorithms developed based on RED we can find Adaptive RED [8] [9].

3.2 Adaptive RED

Adaptive RED [8] [9] is based on the assumption that the resulting average queue length is quite sensitive to the level of congestion and to the RED parameter settings, and is therefore not predictable

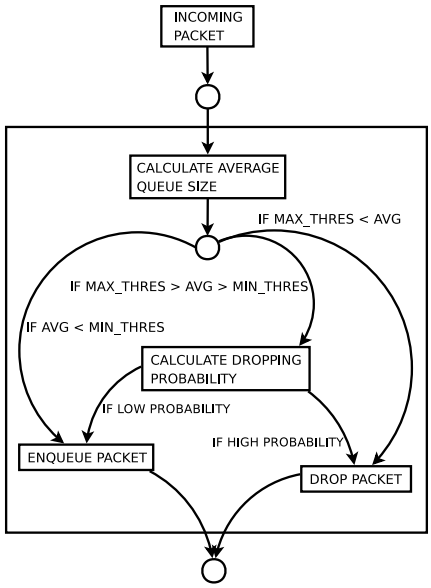


Figure 1: Random Early Detection (RED) algorithm.

Algorithm 2 The Adaptative RED algorithm.

repeat
  Every $interval$ seconds
  if ($avg > target$ and $max_p <= high\_limit$) **then**
    $max_p \leftarrow max_p + \alpha$ {increase max_p}
  else
    if ($avg < target$ and $max_p >= low\_limit$) **then**
      $max_p \leftarrow max_p \times \beta$ {decrease max_p}
    end if
  end if
until end of simulation

in advance. Authors claim that adaptive RED removes the sensitivity to parameters that affect RED's performance and can reliably achieve a specified target average queue length in a wide variety of traffic scenarios. Thus, Adaptative RED is similar to RED, but it updates the max_p parameter with a given frequency, so that the average queue length is kept at a reasonable level at all times. The update procedure is shown in Algorithm 2, and its parameters are the following: $interval$: a period of time; $high\_limit$: a top limit for the maximum dropping probability (max_p); α: increment; low_limit: : a low limit for max_p; β: decrease factor ($\beta < 1$); $target$: the desired average queue length. Also, ARED calculates w_q and

min_{th} based on the speed of the link, thus choosing their values more accurately than just choosing them by hand. The equations used for that are $w_q = 1 - exp(-1/C)$ and $min_{th} = max\left[5, \frac{delay_{target} \times C}{2}\right]$ [9]. In these equations, we can see two new parameters, namely C, which is the link capacity in packets per second, and $delay_{target}$, which is the target average queuing delay.

The Algorithm 2 works as follows: every $interval$ seconds, the average queue size is checked (it is calculated as Algorithm 1 says), and it is compared with the $target$. Also, the max_p is compared with the $high\_limit$ (line 3). If the queue size is too large, and the dropping probability is smaller than the high limit, then increase the dropping probability (line 4). Otherwise, if the queue is too small, and the dropping probability is higher than the low limit (line 6), then decrease the dropping probability (line 7).

These three algorithms (FIFO, RED, and ARED) have been implemented in GridSim, and the implementation will be explained in the next section.

4 Implementation of Buffers Management Policies in GridSim

In this section we will first explain the classes architecture developed to implement the network buffers management policies, followed by an explanation on the interactions between entities.

4.1 Architecture

We have implemented FIFO, RED and ARED as management algorithms for finite buffers in routers, and implementations have been carried out on GridSim. In order to provide GridSim with this new functionality, several classes have been developed. These classes are depicted in bold font Figure 2, and will be explained the next:

- **`FnbUser`**: This class implements the users of our Grid environment. Its functionality can be summarized as follows: (1) creation of jobs; (2) submission of jobs to resources; (3) reception of succeeded jobs.
- **`FnbSCFQScheduler`**: This class is based in the `SCFQScheduler` GridSim class with some variations to support finite buffers. This is an abstract class, and by extending it, new policies can

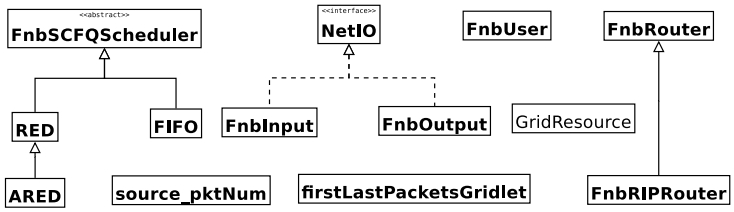


Figure 2: Classes created for the network finite buffers functionality.

be implemented. We have implemented three policies: RED, ARED and FIFO. This class has an abstract function, the `enque(Packet)` function, which is called everytime a new incoming packet arrives at the router.

- **`RED`**, **`ARED`**, and **`FIFO`**: These are three buffer management policies. They implement the `enque(Packet)` function, which calls the buffer management algorithm every time a new incoming packet arrives at the router. These classes behave as was explained in Section 3. As ARED policy is based on RED, `ARED` class extends `RED`. This way, new policies based on RED can be easily implemented, just by extending `RED`. These four classes (including `FnbSCFQScheduler`) are the most important classes of our model, as they implement the policies to manage network buffers.

  Apart from that, when a packet gets dropped, we have to inform the user involved in that transmission. We do it after a certain delay, thus emulating the expiration of a time-out. This is essential as the simulator does not deal well with entities waiting for an event that never arrives. In other words, if a packet gets dropped, the user should be informed, because the job to which that packet belongs is failed. Thus, that job will not arrive back to the user after completion. In real UDP transmissions, users just inject packets into the network, and do not care if there are lost packets. Work on implementing TCP on GridSim will be considered as future work.

- **`NetIO`**: It is an interface class, providing some functions to deal with IO ports.

- **`FnbInput`**: This class implements `NetIO`, and it is based on the `Input` GridSim class. The differences between `FnbInput` and `Input` are mainly in the `getDataFromLink()` function.

As packets may get dropped, input ports must check if all the packets belonging to a transmission have arrived properly. Hence, the transmission is successfull only if all the packets have arrived. Therefore, if any packet belonging to a job's transmission gets dropped, the job will get filtered at the input port of the receiver, not reaching the receiver itself. We do that because we are considering only UDP transmissions, so no detection of lost packets is performed. As mentioned before, implementing TCP on GridSim is part of the future work.

- **`FnbOutput`**: This class implements `NetIO`, and it is based on the `Output` GridSim class. The difference between `FnbOutput` and `Output` is the way how it receives notifications from routers when a packet is dropped. In this case, if the output port belongs to a user, the port will inform the user about that. Another strategy would be retransmitting the lost packet, and this is considered as a part of the future work. When a packet is dropped, and the output port of the user is informed about that, the port has to match the packet ID to the job it belongs to. Then, the port tells the user which job has suffered the dropping. This way a user can deem a job as failed if any of its packets got dropped. This has been done in order to avoid the fact that users keep waiting for a job that never arrives. In real world with UDP transmissions, output ports only have to send packets to the other end of transmissions.
- **`FnbRouter`** and **`FnbRIPRouter`**: based on GridSim classes `Router` and `RIPRouter`, these classes were modified to include some statistics, such as dropped packets counters.
- **`FirstLastPacketsGridlet`**: An array of objects of this class is used to keep the number of packets each job (*gridlet* using the terminology of the simulator) is made of. This is necessary for the output port to be able to match a packet ID to the gridlet it belongs to.
- **`Source_pktNum`**: This class is used in the input ports of users/resources, to make sure that all the packets of a job arrive at the user/resource. If any of the packets of a job do not arrive, that job will be considered as failed. In this case, the input port will filter that job, hence the user/resource will not receive it.
- **`GridResource`**: An original GridSim class, used to execute users' jobs.

4.2 Functionality

The process of creating an experiment in GridSim requires the following steps:

- Initialize the GridSim package by calling `GridSim.init()` and `GridSim.initNetworkType(GridSimTags.NET_BUFFER_PACKET_LEVEL)` methods. This way we decide we want to use the finite buffer functionality.
- Create one or more Grid resource and Grid user entities. Each resource must have number of processors, speed of processing and internal process scheduling policy.
- Build a network topology by connecting Grid user and resource entities.
- Finally, run the experiment by calling `GridSim.startGridSimulation()` method.

Figure 3 depicts a use case in which a user sends a job to a resource, and all the packets reach the resource. The same situation would be for the opposite direction, this is, from the resource to the user. From left to right, we can see that an user submits a job, hence his/her output port has to split that job into a number of packets. The output port also creates a `firstLastPacketGridlet` object containing the ID of the first and the last packet of that job, in this case, 0 and 2. Then, packets are transmitted through the network to the router, which calls `FnbSCFQScheduler` for each incoming packet. The `FnbSCFQScheduler` object runs the policy algorithm to determine whether each packet is dropped or not, and this is done by the child class implementing the chosen policy. In this case, no packet is dropped, hence they are forwarded to the input port of the resource. As packets arrive at the input port of the resource, the port counts them. When all the packets of the job have arrived, then the job has successfully reached the resource, hence the input port sends the job to the `GridResource` entity where it will be executed.

Figure 4 shows an scenario in which one of the packets is dropped. This happens when the `FnbSCFQScheduler` runs the policy algorithm. In this case, the SCFQ scheduler informs the output port of the user about the dropping. In turn, the output port checks which job this dropped packet belongs to, and informs the user. This way, the user knows that this job will not get executed, so his/her will not wait for the output of that job to come back. Also, the input port of the resource will discover the

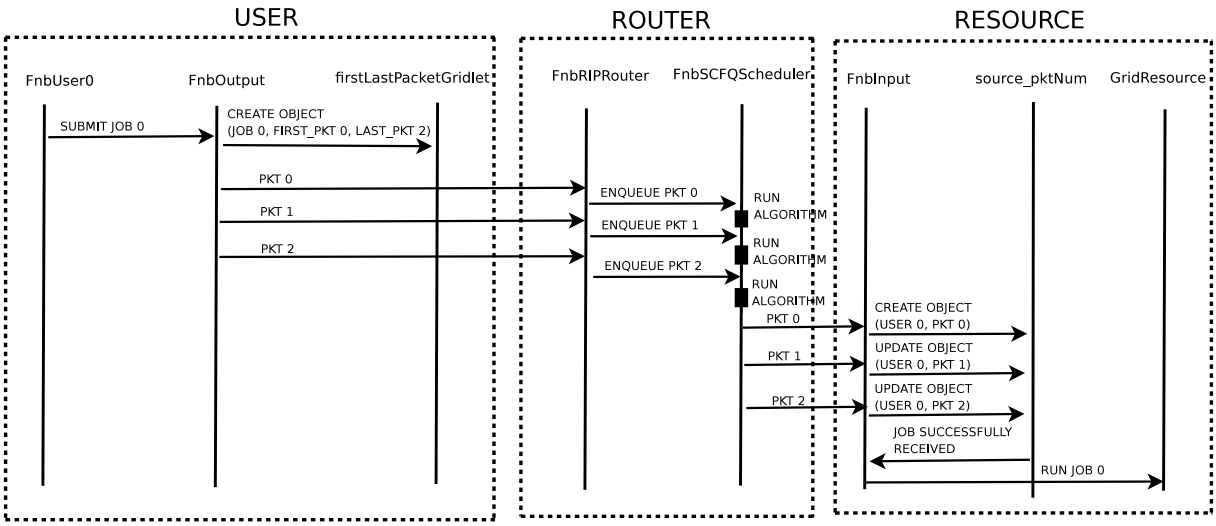


Figure 3: Sequence diagram showing a transmission with no dropped packets.

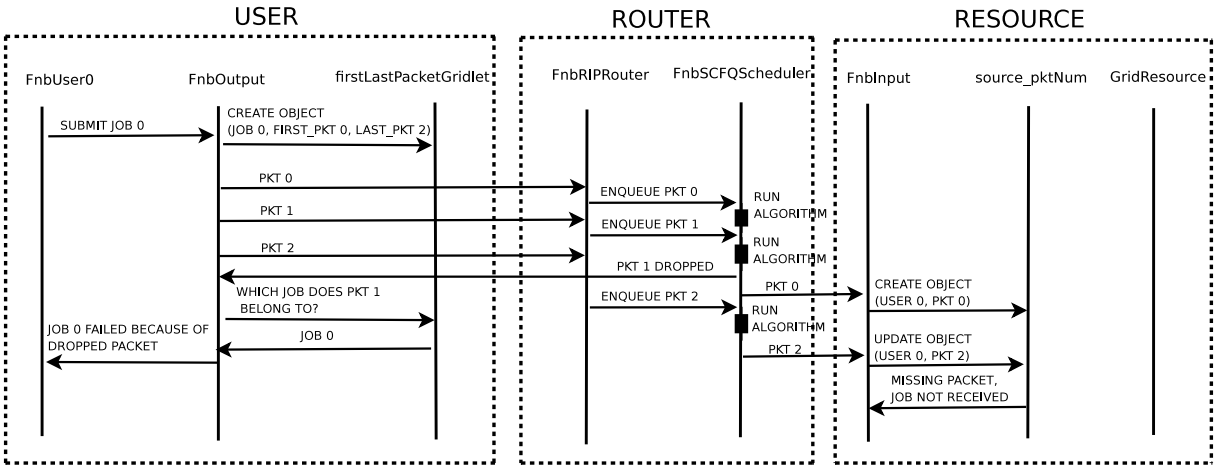


Figure 4: Sequence diagram showing a transmission with one dropped packet.

dropping when a packet does not reach it. In this case, the port will filter that job, not sending it to the resource. Next, we show the usefulness of our work with a use case scenario.

5 Use Case Scenario

The aim of this experiment is to show GridSim's ability to simulate an adequate-size Grid testbed. Therefore, we create a network scenario based on the on the EU DataGRID Testbed 1, as shown in Figure 5 [15]. For this experiment, our main concern is the network behavior in a Grid environment. Hence, we are trying to look at how different buffer management policies affect the network performance.

In this scenario we will compare the RED and ARED, using a FIFO policy as a base case. Table 1 summarizes the characteristics of simulated resources, which were obtained from a real LCG testbed [18]. The parameter regarding to a CPU rating is defined in the form of MIPS (Million Instructions Per Second) as per SPEC (Standard Performance Evaluation Corporation) benchmark. Moreover, the number of nodes for each resource have been scaled down by 10, because of memory limitation on the computer we ran the experiments on, and also for time restrictions. The complete experiments would require more than 2GB of memory, and would take several weeks of processing. Finally, each resource node has four CPUs.

For this experiment, we create 100 users and distribute them among the locations, as shown in Table 1. Each user has 10 jobs and the processing power of each job is 1400000 Million Instructions (MI), which means that each job takes about 2 seconds if it is run on the CERN resource. Also, I/O files sizes are 24 MB. All jobs have the same parameters that are taken from ATLAS online monitoring and calibration system [2]. Moreover, we set the job duration time to be small because it does not influence the performance of the network buffer management policy.

In order to create congestion on a link, we have chosen 20% of the uses to submit their jobs to the resource `CERN`. Thus, the link between this resource and `Router0` (shaded in Figure 5) will be heavily used. So, all the statistics presented here are those collected at the link between `CERN` and `Router0`.

To simplify the experiment set-up, some parameters are identical for all network elements, such as the maximum transfer unit (MTU) of links is 1,500 bytes and the latency is 10 milliseconds. Links will be scaled down by 1000, for the same reasons mentioned above.

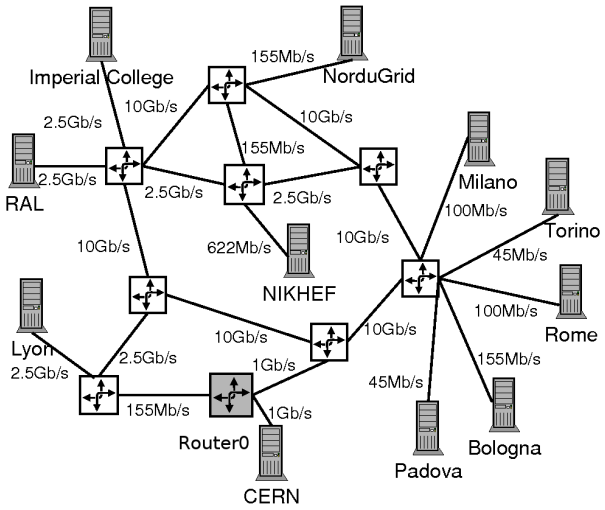


Figure 5: EU DataGRID Testbed 1.

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

Table 1: Resource specifications.

Table 2 specifies the values for the parameters of the simulations running RED algorithm. In order to calculate the thresholds, we have considered the rule $max_{th} = 3 \times min_{th}$, as suggested in [10]. The value for w_q has been chosen too small, so that we can appreciate the improvement achieved by ARED, which calculates that parameter based on the speed of the link. Table 3 specifies the values for the parameters of the simulations running Adaptative RED algorithm, and these values are those used

Parameter	Value
max_{th}	150 packets
min_{th}	50 packets
max_p	0.02
w_q	0.0001

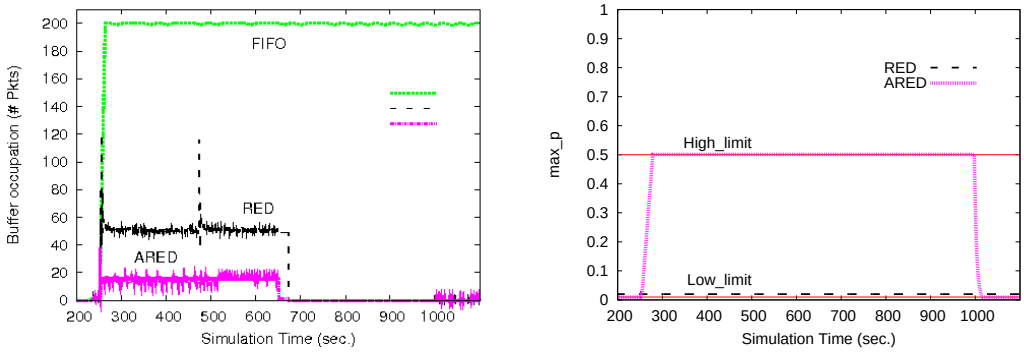
Table 2: Values of RED parameters.

Parameter	Value
$interval$	0.5 seconds
α	$min(0.001, max_p/4)$
β	0.9
$target$	$[min_{th} + 0.4 \times (max_{th} - min_{th}),$ $min_{th} + 0.6 \times (max_{th} - min_{th})]$
$delay_{target}$	0.005 seconds
low_limit	0.01
$high\_limit$	0.5

Table 3: Values of Adaptative RED parameters.

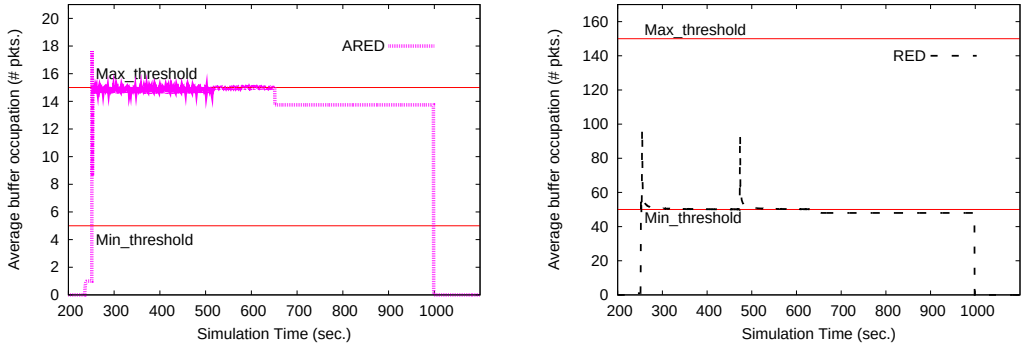
in [9]. The value for max_{th} follows the same rule as for RED. The maximum buffer size for all the simulations is 200 packets, and this is the only parameter for the base case running FIFO.

Figure 6 shows the first performance results that are collected at the beginning of our simulations. Figure 6 (a) shows variations on the buffer occupation. We can see that around time 250, buffer occupations suffer an increase, and this is because users start submitting their jobs to the resource. Thus, the buffer of that link receives a lot of incoming packets. When FIFO is running, the buffer gets saturated, as FIFO imposes no restrictions on the buffer occupations. As opposed to it, both RED and ARED can keep buffer occupations at reasonable levels, far away from saturation. In order to achieve that, ARED increases the max_p, and this can be seen in Figure 6 (b). Regarding RED, the buffer occupation has several spikes, and this is because the w_q has been chosen to a non-optimal parameter and RED cannot detect congestion efficiently. As opposed to it, ARED chooses w_q based on the link features, thus the buffer occupation remains more stable.



(a) Buffer occupation. (b) max_p.

Figure 6: Timelines showing the progress of buffer occupations and max_p.



(a) Average buffer occupation for ARED. (b) Average buffer occupation for RED.

Figure 7: Timelines showing the progress of average buffer occupations for RED and ARED.

Figure 7 shows average buffer occupation for RED and ARED. We do not present this statistic for FIFO, as we can see in Figure 6 (a) that the buffer occupation reaches the full buffer capacity (200 packets) and does not change. As we mentioned above, ARED can keep the average buffer occupation quite stable, as opposed to RED, which shows some spikes. Both of them can keep the average buffer size between the thresholds. Recall that thresholds are different for RED and ARED, since they are automatically calculated (in the case of ARED), and chosen by hand (in the case of RED). Because of this, ARED's thresholds are lower ($min_{th} = 5\ packets$ and $max_{th} = 15\ packets$) than RED's ($min_{th} = 50\ packets$ and $max_{th} = 150\ packets$). This way, the difference in average buffer occupation between both policies showed in Figure 6 (a) is explained.

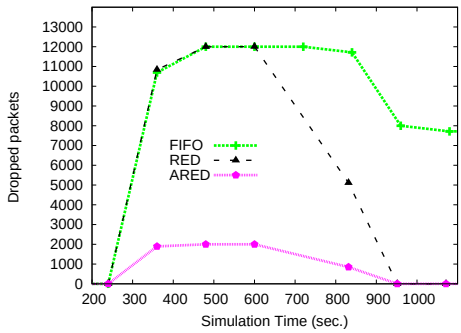
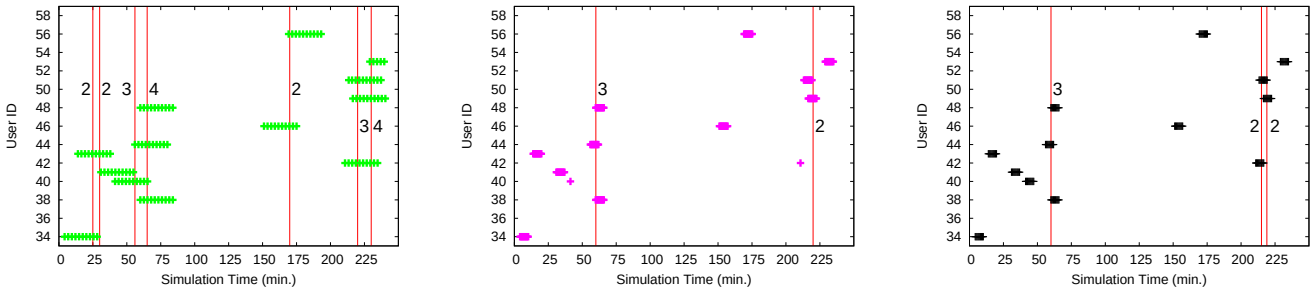


Figure 8: Statistics on dropped packets.

Figure 8 shows the dropped packets at the link between resource `CERN` and the router it is directly connected to. We can see that the policy which drops more packets at this link is FIFO, because it does nothing to control the buffer occupations. Thus, too many packets reach this link, and fill the buffer. Then, all the packets reaching this link when the buffer is full will get dropped. On the other hand, RED and ARED does perform that kind of control, thus the amount of packets that reach this link is lower. RED and ARED schedulers at each link in the topology filter packets when the average queue size becomes too high, thus the amount of packets that reach this link is lower. Recall that the current infrastructure does not provide retransmission of dropped packets, and this improvement is considered as future work.

As for the users point of view, Figure 9 shows statistics regarding the moment when users from the location `CERN` receive the first dropped packet for a job. In a real environment, this means the retransmission of the lost packet, and if it is a TCP connection, the decrease of the transmission window and the retransmission of all the packets from the lost one onwards. As we explained in Section 3, avoiding global synchronization is one of the aims of the policies based on RED. In this figure, we can see that there are few users who receive a dropped packet at the same time (the vertical lines in the figure show the moment when more than 1 user gets a dropped packet at the same time, and number on them show how many users get synchronized) when using RED or ARED policies. Figure 9 (a) shows that up



(a) FIFO. (b) ARED. (c) RED.

Figure 9: Timelines showing the moment when users from CERN receive a packet dropped.

to 4 users get a dropped packet at the same time, and Figure 9 (b) and (c) show no more than 3. Also, this happens less frequently when RED or ARED is being used, than with FIFO.

6 Conclusion and Future Work

Grid technologies are emerging as the next generation of distributed computing, allowing the aggregation of resources that are geographically distributed across different locations. Due to the large scale and distributed management of Grids, the use of simulation tools is essential to carry out research efficiently. Thus, simulation tools should cover the main features of a real Grid system, but this was not totally true for the network of Grids.

In this paper we propose an extension to one of the most widely used simulation tools to cover this gap. More precisely, we have introduced finite network buffers and network buffers management policies into GridSim. Three management policies have been implemented, namely FIFO, RED, and ARED, but more policies can be implemented using the current framework. This way, researchers will be able to create more realistic network models, thus improving their research in several key fields in Grids, such as scheduling, or QoS provision.

As for future work, we are planning to use the improved simulation tool to carry out research aimed at providing network QoS in Grids. This will be done by integrating this functionality into the Grid network broker outlined in [6]. Moreover, the functionality explained in this paper can be extended to include

retransmissions of dropped packets. Furthermore, we are thinking on implementing TCP in GridSim as another future step.

Acknowledgement

This work has been jointly supported by the Spanish MEC and European Commission FEDER funds under grants “Consolider Ingenio-2010 CSD2006-00046” and “TIN2006-15516-C04-02”; by JCCM under grants PBC-05-007-01, PBC-05-005-01 and José Castillejo. This research is also partially funded by the Australian Research Council and the Department of Education, Science and Training.

References

[1] The network simulator - ns-2. Web Page, 2007. `http://www.isi.edu/nsnam/ns/`.

[2] ATLAS online monitoring and calibration system. Web Page, 2007. `http://dissemination.interactive-grid.eu/applications/HEP`.

[3] J. Aweya, M. Ouellette, D. Y. Montuno, and A. Chapman. Enhancing TCP performance with a load-adaptive RED mechanism. *Intl. Journal of Network Management*, 11(1):31–50, 2001.

[4] W. H. Bell, D. G. Cameron, L. Capozza, A. P. Millar, K. Stockinger, and F. Zini. Simulation of dynamic grid replication strategies in OptorSim. In *Proc. of the 3rd Intl. Workshop on Grid Computing (GRID’02)*, London, UK, 2002.

[5] R. Buyya, S. Date, Y. Mizuno-Matsumoto, S. Venugopal, and D. Abramson. Neuroscience instrumentation and distributed analysis of brain activity data: a case for escience on global grids. *Concurrency and Computation: Practice and Experience*, 17(15):1783–1798, 2005.

[6] A. Caminero, C. Carrión, and B. Caminero. Designing an entity to provide network QoS in a Grid system. In *Proc. of the 1st Iberian Grid Infrastructure Conference (IberGrid)*, Santiago de Compostela, Spain, 2007.

[7] E. Elmroth and P. Gardfjäll. Design and evaluation of a decentralized system for grid-wide fairshare scheduling. In *Proc. of the 1st Intl. Conference on e-Science and Grid Computing (eScience)*, Melbourne, Australia, 2005.

[8] W. Feng, D. D. Kandlur, D. Saha, and K. G. Shin. A self-configuring RED gateway. In *Proc. of the INFOCOM Conference*, New York, USA, 1999.

[9] S. Floyd, R. Gummadi, and S. Shenker. Adaptive RED: An algorithm for increasing the robustness of RED's active queue management. Technical report, AT & T Center for Internet Research at ICSI, Aug. 2001.

[10] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *Transactions on Networking*, 1(4):397–413, 1993.

[11] I. Foster and C. Kesselman. *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 2 edition, 2003.

[12] I. T. Foster. The anatomy of the Grid: Enabling scalable virtual organizations. In *Proc. of the 1st Intl. Symposium on Cluster Computing and the Grid (CCGrid)*, Brisbane, Australia, 2001.

[13] K. Fujiwara and H. Casanova. Speed and accuracy of network simulation in the simgrid framework. In *Proc. of the 1st Intl. Workshop on Network Simulation Tools (NSTools)*, Nantes, France, 2007.

[14] B. Gazi and Z. Ghassemlooy. Dynamic buffer management using per-queue thresholds: Research articles. *Intl. Journal Communications and Systems*, 20(5):571–587, 2007.

[15] W. Hoschek, F. J. Janez, A. Samar, H. Stockinger, and K. Stockinger. Data management in an international data grid project. In *Proc. of the 1st Intl. Workshop on Grid Computing*, Bangalore, India, 2000.

[16] A. Kesselman and Y. Mansour. Harmonic buffer management policy for shared memory switches. *Theoretical Computer Science*, 324(2-3):161–182, 2004.

[17] A. Kumar, J. Kaur, and H. Vin. End-to-end proportional loss differentiation. Technical Report TR-01-33, University of Texas, USA, 2001.

[18] LCG Computing Fabric Area. Web Page, 2007. `http://lcg-computing-fabric.web.cern.ch`.

[19] J. Liu and D. M. Nicol. *DaSSF 3.1 User's Manual*. Dartmouth College, April 2001.

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

[21] F. T. Marchese and N. Brajkovska. Fostering asynchronous collaborative visualization. In *Proc. of the 11th Intl. Conference on Information Visualization*, Washington DC, USA, 2007.

[22] J. A. Miller, R. S. Nair, Z. Zhang, and H. Zhao. JSIM: A JAVA-based simulation and animation environment. In *30th Annual Simulation Symposium (ANSS'97)*, Atlanta, USA, 1997.

[23] A. Ramakrishnan, G. Singh, H. Zhao, E. Deelman, R. Sakellariou, K. Vahi, K. Blackburn, D. Meyers, and M. Samidi. Scheduling data-intensive workflows onto storage-constrained distributed resources. In *Proc. of the 7th Intl. Symposium on Cluster Computing and the Grid (CCGrid)*, Rio, Brazil, 2007.

[24] A. Roy. *End-to-End Quality of Service for High-End Applications*. PhD thesis, Dept. of Computer Science, University of Chicago, 2001.

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

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

[27] A. Varga. The omnet++ discrete event simulation system,. In *Proc. of the European Simulation Multiconference (ESM)*, Prague, Czech Republic, 2001.