University of Castilla-La Mancha

A publication of the

Computing Systems Department

Grid Metascheduling Using Network Information: A Proof-of-Concept Implementation

by

Luis Tomás, Agustín Caminero, Blanca Caminero, Carmen Carrión

Technical Report **#DIAB–08–04–2** April, 2008

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"; jointly by JCCM and Fondo Social Europeo under grant "FSE 2007-2013"; and by JCCM under grants "PBC-05-007-01", "PBC-05-005-01".
A shorter version of this work has been submitted to the 9th ACM/IFIP/USENIX International Middleware Conference.

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

Grid Metascheduling Using Network Information: A Proof-of-Concept Implementation

Luis Tomás Agustín Caminero Blanca Caminero Carmen Carrión
Instituto de Investigación en Informática de Albacete (I^3A)
Universidad de Castilla-La Mancha.
02071 - Albacete, SPAIN
{luistb,agustin,blanca,carmen}@dsi.uclm.es

Abstract

The aggregation of geographically distributed resources in the context of a particular application has been made possible thanks to the deployment of Grid technologies. As Grids are extremely distributed systems, requirements on the communication network should also be taken into account when performing usual tasks such as scheduling, migrating or monitoring of jobs. Note that users, services, and data need to communicate with each other over networks, thus the network should be used in an efficient and fault-tolerant way. However, most of the existing efforts to provide QoS in Grids do not take into account network issues, and focus instead on processor workload and disk access. The authors have previously proposed a framework for providing network-aware grid metascheduling. The usefulness of this approach was shown by means of simulation. In this work, a proof-of-concept implementation of a metascheduler, developed as an extension to the GridWay metascheduler, is presented and evaluated. This extension harnesses network information to perform job scheduling tasks. Results show that the response time perceived by Grid users is reduced when data on network performance are used when scheduling jobs.

1 Introduction

Grid systems are highly variable environments, made of a series of independent organizations that share their resources. During the last decade, Grid computing has emerged as an enabling technology for many data and/or computing intensive applications. Through the use of Grid technologies it is possible to aggregate dispersed heterogeneous resources for solving various kinds of large-scale parallel applications in science, engineering and commerce [1]. A well-known example of such applications is the Grid-based worldwide data-processing infrastructure deployed for the LHC experiments at CERN [2].

Grid users should perceive some Quality of Service (QoS) on their application performance, in terms of response delay or number of jobs finished per time unit [3]. Though, this is quite difficult to achieve due to the large scale of interconnected networks that are inherent part of such a extensively distributed environment [4].

Achieving an *end-to-end* QoS is often difficult, as without resource reservation any guarantees on QoS are often hard to satisfy. However, for applications that need a timely response (i.e., collaborative visualization [5]), the Grid must provide users with some kind of assurance about the use of resources – a non-trivial subject when viewed in the context of network QoS [4]. In a VO, entities communicate with each other using an interconnection network – resulting in the network playing an essential role in Grid systems [4].

A *meta-scheduler* is an entity entrusted with the care of the users' interests in a Grid, such as GridWay [6] or Condor/G [7]. The user will provide the metascheduler with a job template, and from that moment on, the metascheduler will take care of any interaction with the Grid in order to get the job executed (such as selection of resources, data transfers, authentication issues, job monitoring and migrations).

Among the many issues that must handled with when designing metaschedulers, the authors are focusing their research on the selection of the computing resources where user's jobs will be executed. As the corresponding section will show, current metaschedulers take their scheduling decisions focusing just on the computing power (and utilization) of the available resources. So, a metascheduler might decide that the most suitable resource to run a user's job is the most powerful one. However, if there is some overloaded link in the path between the resource and data sources, other less powerful computing resource with a less loaded network connection might be more suitable to run that job. In this case, the less powerful computing resource would be a better choice.

This scenario is even more harmful for those Grid applications that need access to huge amounts of input data from remote locations, provide analysis data for remote visualization, or exchange data between distant machines [1]. Then, it is clear that the communication requirements of these applications are subject to the network constraints, which have a direct impact on the final performance of applications. Hence, the network status must be considered when the resource selection process is performed [4].

The enhancement of application performance through the use of network-related information when performing job scheduling decisions is the topic of interest of this work. Authors have already developed a reference framework with this regard [8]. Among others, it includes an entity known as *Grid Network Broker, GNB*, integrated within the metascheduler, that provides job scheduling decisions based on information on network condition. In this paper, a proof-of-concept implementation of such entity, made by extending the GridWay metascheduler [6], is presented. Also, experiments that show the goodness of this approach are included.

The paper is organized as follows. In Section 2 a brief overview of the general metascheduling framework is presented. Then, the extensions implemented on the GridWay metascheduler are outlined in Section 3. Section 4 presents the experiments carried out to illustrate the scheduling approach. Section 5 presents other proposals aimed at the provision of network quality of service in Grids. Finally, Section 6 draws conclusions and suggests guidelines for future work.

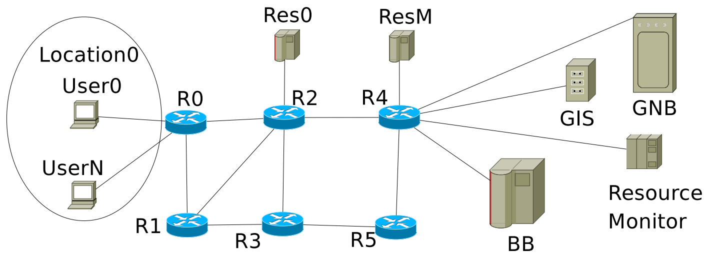


Figure 1: Topology.

2 GNB: A framework for metascheduling in Grids

Communication-intensive and real-time applications require some type of QoS brokering services in order to provide guaranteed reservation of network capacity along with computational resources. The authors have proposed in [8] a flexible Grid resource management architecture, which is aimed at guaranteeing the QoS requirements specified by the applications. One of the key points of this architecture is an entity called *Grid Network Broker* or **GNB**. This entity will check the status of the network and will create both network reservations and service allocation in order to provide network QoS to Grid applications.

Our scenario is depicted in Figure 1 and has the following entities [9]: **Users**, each one with a number of jobs; **computing resources**, e.g. clusters of computers; **routers**; **GNB** (*Grid Network Broker*), a network-aware scheduler; **GIS** (*Grid Information Service*), such as [10], which keeps a list of available resources; **resource monitor** (for example, Ganglia [11]), which provides detailed information on the status of the resources; **BB** (*Bandwidth Broker*) such as [12], which is in charge of the administrative domain, and has direct access to routers. BB can be used to support reservation of network links, and can keep track of the interconnection topology between two end points within a network. The interaction between components within the architecture is as follows:

- Users ask the GNB for a resource to run their jobs. Users provide features of jobs.
- The GNB collects the requests and after a given time interval, it performs two operations for each request. First, it performs scheduling of that request job to a computing resource, and second, asks the BB to perform connection admission control (CAC).
- The GNB makes use of the GIS in order to get the list of available resources, and then it gets their current load from the resource monitor.
- The GNB makes use of the BB in order to carry out operations requiring the network. The BB should implement some functions to check if a transmission can be routed through a specific link. By making the BB have this functionality, we assure the independence and autonomy of each administrative domain.

The architecture of the GNB is shown in Figure 2. It includes the following modules:

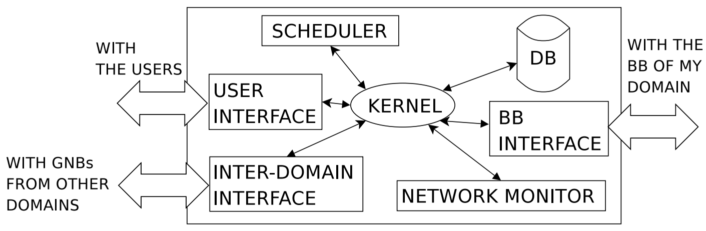


Figure 2: Architecture of the GNB.

User interface. This module allows users to submit requests to the GNB. More precisely, a user can ask the GNB to allocate a computing resource to run his/her jobs. As a consequence, the 'Scheduler' module will order the available and suitable resources from the best to the worst, and choose the best of them. Then, GNB will optionally ask the BB to perform a connection admission control test.

Scheduler. This module selects the resource where every job will be run, attending to different criteria. In order to do this, the 'Scheduler' needs to retrieve a list of the available resources from the GIS module.

Several scheduling algorithms can be implemented within this module. The authors have proposed several of them in [8] [13] [14]. These algorithms always take into account network information when taking scheduling decisions. Moreover, in [9] the authors proposed a scheduler which incorporates autonomic computing techniques.

BB interface. This module deals with the BB, so that it performs tasks such as connection admission control. When performing CAC, the BB assesses whether a new connection can be admitted through the path between the user and the resource. This way, links between a user and a given resource (previously selected by the 'Scheduler' module) can be checked. The BB also communicates with edge routers as well as with the core routers within the domain. The routers have the ability of handling the application traffic, thus they have to be configured by the BB (e.g., issuing proper commands related to DiffServ [15] configuration), in order to apply the adequate forwarding priority to every type of traffic. By making the BB have these functionalities, we assure the independence and autonomy of each administrative domain.

Interdomain interface. A GNB communicates with GNBs from other neighbor administrative domains in order to find a suitable resource for running jobs that do not fit in the local domain. This issue has been explored in [16], where some techniques from peer-to-peer networks have been applied.

Database. All the information the GNB manages and uses must be stored in a database. Examples are information on resource configuration (CPU type and speed, amount of RAM and disk, etc.).

Network monitor. The purpose of this module is to periodically run tests on the network so that information on current load and delay conditions can be retrieved and stored in the 'Database module'. These tests are usually based on sending probes through different paths in the network (as in the IPerf tool [17]), thus adding traffic overhead. The more frequently tests are run, the more accurate estimations on network performance are, but the more overhead is caused. Thus, this is an extremely sensitive issue because a trade-off must be achieved between overhead and accuracy on network status.

Previous research on the GNB architecture has been carried by means of simulation. The authors have developed a simulated model of the GNB architecture with the GridSim simulation tool [18]. A working implementation of a metascheduler which integrates network information when taking scheduling decisions has been developed now. It has been devised by extending the features of the GridWay metascheduler. GridWay [6] has been chosen among others mainly due to the availability of its source code, and to the fact that it is a Globus project [19]. Also, its capabilities to add new criteria to perform the filtering and sorting of the candidate resources have been exploited. Work done on GridWay is explained in the next section.

3 Extensions to the GridWay metascheduler

The GridWay metascheduler [6] enables large-scale, reliable and efficient sharing of computing resources (clusters, computing farms, servers, supercomputers...), managed by different Local Resource Management (LRM) systems, (such as PBS, SGE, LSF, or Condor) within a single organization (enterprise Grid) or scattered across several administrative domains (partner or supply-chain Grid). GridWay is a Globus project, adhering to Globus philosophy and guidelines for collaborative development. With GridWay, a Grid infrastructure can be exploited and managed in the same way as a local computing cluster.

Users willing to submit jobs to the Grid infrastructure managed by GridWay need to generate a *job template*. This template is a text file (with extension ".jt") which includes the information needed for job execution, such as the names of input, output and executable files, as well as some control parameters related to scheduling, performance, fault tolerance or resource selection, among others. More details can be found in [20].

The work presented here is focused on how GridWay selects resources for job execution. The parameters related to this issue are `REQUIREMENTS` and `RANK`. They together allow users to specify the criteria used to select the most appropriate resource to run their jobs, in a two-step process (see Algorithm 1). The `REQUIREMENTS` tag is processed first. Within this tag, the user can specify the *minimal requirements* needed to run the job. Thus, it acts as a filter on all the resources known to GridWay. As a result, only the resources that fulfill the `REQUIREMENTS` condition are considered in the next step. Then, with the `RANK` tag, the characteristics taken into account when ordering resources are specified. This means that all the resources that fullfill the `REQUIREMENTS` specifications are ordered following the criteria specified with the `RANK` tag (i.e., the set of resources is ordered with regard to their amount of free RAM). For both tags, several characteristics as type of CPU, operating system, CPU speed, amount of free memory, etc. can be specified. Many of these values are gathered through the Globus GIS module, while others (specifically the dynamic ones, such as amount

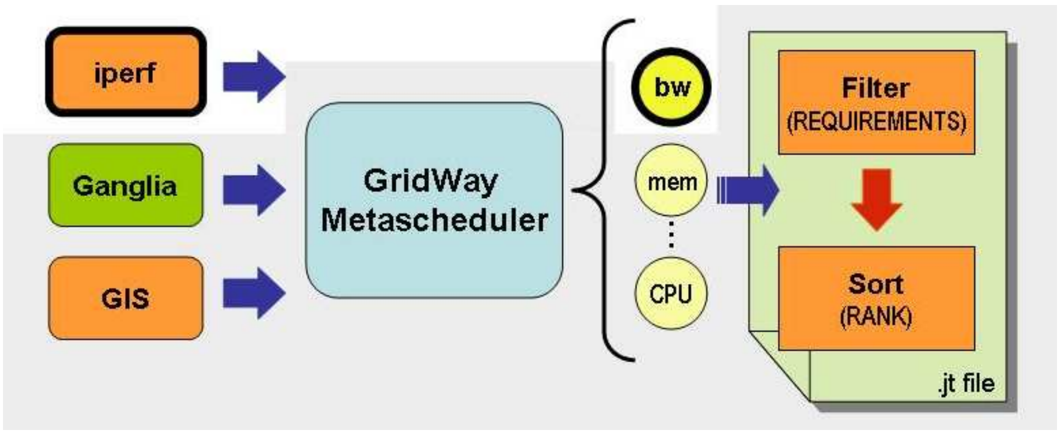


Figure 3: Conceptual view of the extensions introduced to GridWay.

of free RAM) are monitored through Ganglia.

Algorithm 1 GridWay resource selection algorithm

```
1: Let $S$ = set of resources known to GridWay
2: $S^R$ = {s $\in S$ / $s$ fulfils REQUIREMENTS condition }
3: return $s' \in S^R$ / $\forall s \in S^R$, RANK($s'$) > RANK ($s$)
```

Among the parameters supported by GridWay, there is nothing related to the network conditions between the candidate nodes. Therefore, the modifications introduced are aimed at including into GridWay the possibility to use information about the status of the network connections when performing the scheduling operations. First, it has been necessary to integrate the computation of the state of the interconnections to every resource belonging to the Grid. Second, one additional tag has been added so that the aforesaid information can be used in the job scheduling process. The parts which are not shaded in Figure 3 illustrate the extensions introduced in GridWay.

The state of the network connections has been measured as the available bandwidth between the node that submits jobs and every other node in the Grid infrastructure. The *iperf* tool [17] is used to this end. It makes an estimation of the available bandwidth by sending a series of packets, with a certain size, through the specified port and for a certain time (variable, 10 sec. by default). It should be noted that the larger this time is, the greater accuracy will be obtained when computing the available bandwidth towards an specific host. However, on the other hand, it is not desirable that the above-mentioned time was too long, because it could overload the network. Thus, a trade-off must be reached. Two additional tags (`NET_PORT` and `NET_TIME`) have been added into the GridWay configuration file (`GW_LOCATION/etc/gwd.conf`), to be able to specify the port and the time used by the *iperf* test, respectively.

The bandwidth estimation carried out by *iperf* is then integrated into the GridWay scheduling process by defining a new attribute named `BANDWIDTH`. The information generated by *iperf* is stored in this new attribute. The `BANDWIDTH` variable can then be used both in the `REQUIREMENTS` as well as in the `RANK` expressions, so that candidate resources can be filtered out and/or sorted taking into account the available bandwidth from the scheduler node to each one of them.

Thus, if the variable `BANDWIDTH` is included in the `RANK` tag, the list of hosts is sorted out from higher to lower available bandwidth. In this way, the scheduler will take this factor into account. Hence, it selects the first host of the list, i.e, the host whose connection with the scheduler node (the node running GridWay) has better bandwidth. If the variable `BANDWIDTH` is included is in `REQUIREMENTS` tag, it refers to the limit. In other words, it means that the available bandwidth in the path from the GridWay node to the resource needs to be greater than, less than, or equal than a certain value. The obtained list of hosts is filtered by that pattern, and in this way, the scheduler will only submit jobs to those nodes that fulfill the condition.

It is worthwhile to note that several parameters can be combined within the `RANK` expression. Different weights can be assigned to each parameter, so that it is possible to modify the relative importance of each of them. For example, in this way the user can select a larger weight to the `BANDWIDTH` parameter, in case the job involves heavy network use.

4 Experiments and results

Several benchmarks have been executed in order to measure the performance improvement when bandwidth information is used to do the scheduling tasks. The results are presented in this section.

First, features of the testbed are described, then details on the workload used are given, and finally, the results of our experiments are depicted and analyzed. Tests are divided into two groups. First, different type of tests with changing network traffic conditions are executed. Second, focus is placed only in one test (*VP test of NGB*), and a number of *GridWay* configuration parameters are varied. Then, new resources are added to the Grid, in order to make a more heterogeneous environment.

4.1 Testbed description

The functional validation of the network-aware scheduling, described in the previous sections, has been carried out in a controlled Grid environment. Figure 4 shows the network topology used as testbed.

The testbed is composed by four machines that are in the same subnet, *AulasunX.uclm.es* with $X \in [1, 4]$. Moreover, these hosts have the same characteristics, both hardware and software. There are also two more machines, *GridWayI3A.uclm.es* and *Globus1.uclm.es*, with different hardware and software characteristics. These machines belong to the same administrative domain as *Aulasun* (which is *UCLM*) but they are located in another subnet. The *GridWayI3A.uclm.es* machine carries out the scheduler tasks. Table 1 outlines the main characteristics of the computing resources.

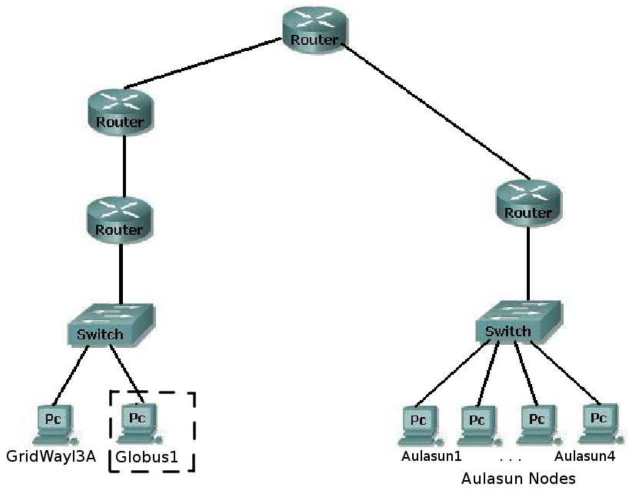


Figure 4: Network Diagram

Machine	Hardware		Software
	CPU	RAM	
GridWayI3A.uclm.es	2 CPU Intel Pentium 4, 3 GHz	2 GB	Globus (v. 4.0.5)
Aulasun1.uclm.es	2 CPU AMD Opteron 244, 1.8 GHz	1 GB	Globus (v. 4.0.3)
Aulasun2.uclm.es	2 CPU AMD Opteron 244, 1.8 GHz	1 GB	Globus (v. 4.0.3)
Aulasun3.uclm.es	2 CPU AMD Opteron 244, 1.8 GHz	1 GB	Globus (v. 4.0.3)
Aulasun4.uclm.es	2 CPU AMD Opteron 244, 1.8 GHz	1 GB	Globus (v. 4.0.4)
Globus1.uclm.es	1 CPU Intel Pentium 4 3 GHz	1 GB	Globus (v. 4.0.4)

Table 1: Characteristics of the computational resources

4.2 Workload used

To validate and evaluate performance results of the implemented grid network-aware meta-scheduler we have run different *NAS Grid Benchmarks (NGB)* [21]. These jobs have different workflow dependencies. Some jobs are computational intensive while others are network demanding. Therefore, the NGBs allow us to explore a big spectrum of running conditions. Figure 5 shows the workflows of the tests executed in our Grid system.

- **SPs3**: This test is made of several "SP.jt" jobs requiring different computational resources to be executed. This job is one of the subjobs of the *"ED"* test of the NGB benchmarks. Job dependencies of this test are despited in Figure 5 (a).

- **VPModif**: This test is similar to the "VP" test of the NGB benchmarks but, some dependencies between jobs have been deleted. The goal is getting a greater parallelism. Figure 5 (b) shows the diagram of this test. The circular nodes are *"BT"* jobs, square nodes are *"MG"* jobs and trapezium nodes are *"FT"* jobs. All of them (*"BT"*,*"MG"* and *"FT"*) are defined in the *NGB* benchmarks.

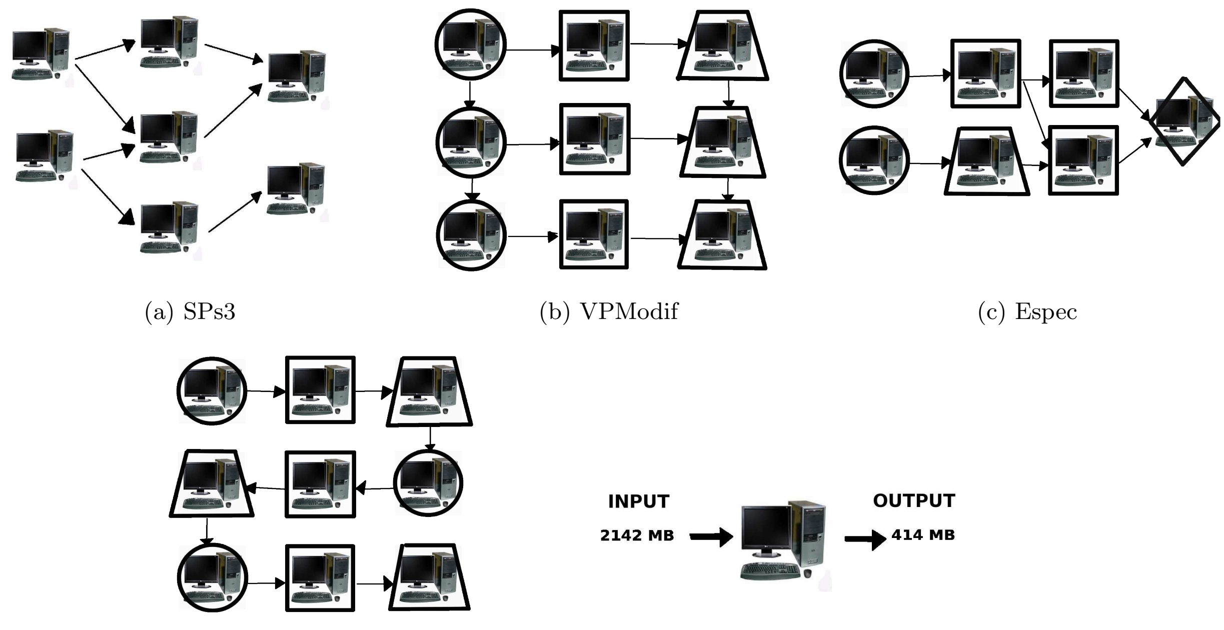


Figure 5: Workflows of the GNB tests.

- **Espec**: The diagram for this test is shown in Figure 5 (c). Please note that the transfers carried out in these jobs are different, being the execution more asymmetric than it may seem when its diagram is seen.

  Circular nodes are jobs of type *BT iteration 0*. Trapezium nodes are other type of *BT* job but with *iteration 3*. Squared nodes are job of type *BT iteration 3* but with an increased input file, doing this job heavier in network traffic. At the end, rhombus node is a job of type *BT iteration 2*.

- **HC**: Figure 5 (d) shows the diagram of this test. In this case, there are not heavy network transfers ("only" 32 MB files).

  Circular nodes are *"BT"* jobs, squared nodes are *"SP"* jobs and trapezium nodes are *"LU"* jobs. All of them are defined in the *NGB* benchmarks.

- **Heavy Traffic (HT)**: This job is based on several big size files sent for executing one job, which is of *BT* type of the *NGB* benchmarks. The number of inputs files that it accepts has been increased. The diagram of this test can be seen in Figure 5 (e).

In order to carry out the scheduling of jobs to computing resources, GridWay has a number of scheduling policies, named *RESOURCE SCHEDULING POLICIES*. They allow administrators to influence the usage of resources made by the users. These policies are:

- Fixed Priority (RP) Policy: each resource is assigned a fixed priority (range [1,99]) based on either the information management that discovered it, or its FQDN.

- Rank (RA) Policy: it prioritizes resources based on their RANK (as defined in the job template).

- Failure Rate (FR) Policy: resources with persistent failures are banned.

- Usage (UG) Policy: with this policy, resources are prioritized based on the estimated execution time of a job (on each resource). The estimation is derived from the sum of two contributions:
  $T = (1 - w) * T\_history + w * T\_last$, where T is the estimated execution time for a job, $T\_history$ is the average execution time for jobs in that resource, T_last is the execution time of the last job executed in that resource, and w is the weight that the last execution time has when calculating the estimated execution time for the current job.

The overall priority of a resource is computed as a weighted sum of the contribution of each policy:
$ResourcePriority = Wrp * RP + Wfr * FR + Wug * UG + Wra * RA$, where each Wx is the weight that policy x has when calculating the overall priority of resources.

As we want to illustrate the importance of the network when performing the scheduling of jobs to computing resources, all the above policies have been disabled, except Rank. We do this so that scheduling is performed based on the bandwidth information only. Next section we are going to present the experiments conducted with this regard.

4.3 Performance evaluation

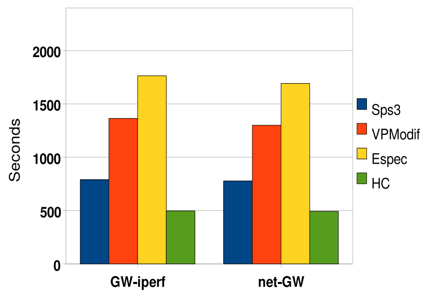
The performance evaluation presented in this paper is divided into several parts. First, we are going to evaluate the influence of background network and CPU load. Then, we are going to evaluate the addition of new resources, so that a more heterogeneous Grid environment is created. After that, we are going to study the influence of the parameters of the monitoring tools. Finally, the cost of having our proposal working is going to be analyzed.

4.3.1 Influence of background network and CPU load.

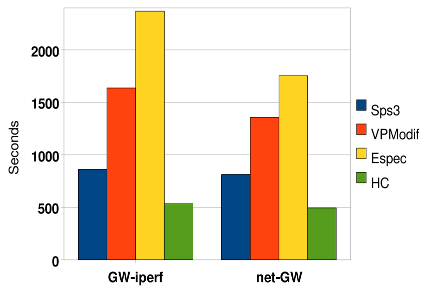
To validate the changes developed in the GridWay scheduling algorithm described in the previous section, we have run the NGB tests in the proof Grid platform. We have limited the available computational resources to the four Aulasun hosts (topology is depicted in Figure 4, and host *globus1* is not enabled at the moment). The reason for this choice is to check the behavior of the network-aware scheduling by using CPU nodes with equal computational performance. Host *GridWayI3a* will carry out the scheduling task but will not execute any job.

The aim of this first evaluation is to show the usefulness of our proposal under varying network and CPU conditions. First, we are going to evaluate the influence of background network load. To do so, an ideal situation has been considered in which the network is unloaded. Then, we have emulated a more real working conditions by delivering background network traffic. The background network traffic is created by one file transfer from one node to another, by means of the *globus-url-copy* command, and only one transfer is carried out in the whole topology. A more in-depth study on the influence of background network traffic is presented in a subsequent section.

Figure 6 shows the results for the different tests (*SPs3*, *VPModif*, *Espec* and *HC*) under both scenarios, that is, with and without background network traffic. The label **GW** refers to the performance of using the original scheduling algorithm provided by the GridWay tool. So, bandwidth information is not used for scheduling. The label **GW-iperf** refers to the original scheduling algorithm by GridWay which has been extended to calculate the bandwidth information, although this information is not considered for the scheduling. On the other hand, **net-GW** refers to the results of the new implemented scheduling algorithm, which considers bandwidth information for the scheduling.



(a) Without background traffic



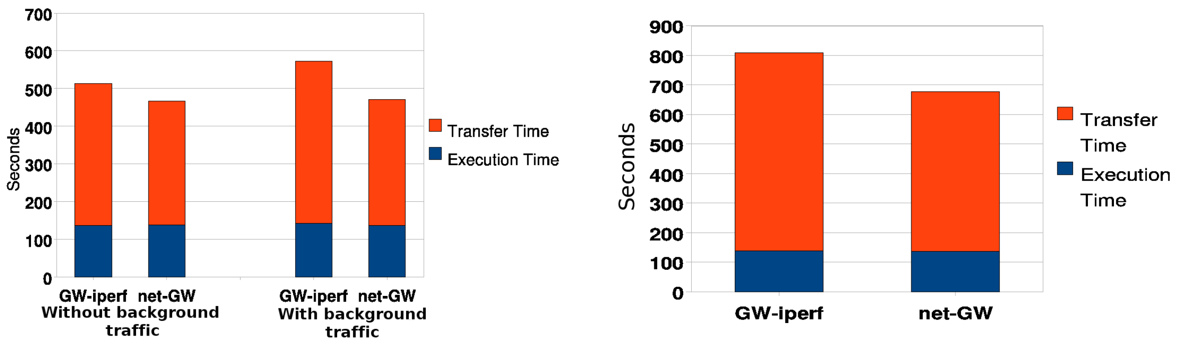
(b) With background traffic

Figure 6: Completion time of different type of tests

In Figure 6 (a) shows the scenario without background traffic, and we can see that there is little difference in terms of completion time between running the same test with and without using bandwidth information in the schedulers. Using bandwidth information (when net-GW is running), small improvements are obtained in spite of the fact that both the computational resources have the same features and they are located in the same subnet. Thus, those differences can be harnessed, and due to the transfer of big IO files, an improvement on the test completion time can be obtained.

Figure 6 (b) shows the second scenario, when the network background traffic is generated (because of file transfers between two nodes). A noticeable completion time increase is seen when tests are executed without using bandwidth information (when GW-iperf is running). This is because a node which is transmitting or receiving can be chosen to execute a job. Hence, its network link may be heavily used, and the transmissions on this link will take longer than on another node's link that is not being used at the moment. On the contrary, if bandwidth information is used (when net-GW is running), a node whose network link is not being used will be selected because its bandwidth is higher than others.

However, those results have been obtained for the aforesaid executions with the indicated nodes under this set of traffic conditions. The background traffic is generated by transfers made with *globus-url-copy* command which simulate client executions using the Grid. If the traffic is increased, other differences would be seen. That is, if the traffic is increased to one node, greater differences will be produced depending on whether bandwidth information is used or not. On the contrary, if traffic is increased drastically, for example to all *Aulasun* nodes, and only these nodes belong to the Grid, the transfer speed will decrease



(a) Sequential executions (b) Simultaneous executions

Figure 7: Heavy Traffic test

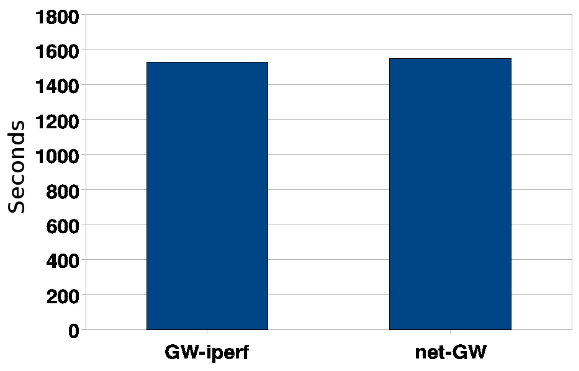
drastically due to there is not any node with bandwidth features equal to the first execution (which had no background network traffic). Finally, if traffic to each node is different, improved times will be obtained, intermediate between the last two options.

Also, Figure 6 (b) shows that as long as background traffic is increased and GW-iperf is running (bandwidth information is not used for the scheduling), the time to perform the tests grows up more than in the absence of background. This is because there is a greater number of transfers and more dependences among the jobs composing the test. On the other hand, the figure shows that when there is more traffic and net-GW is used (this is, bandwidth information is used for scheduling), the time to perform the tests almost does not grow up. In fact, it is slightly better than if background traffic is not present and GW-iperf is running (see Figure 6 (a)). In case a greater number of nodes is provided, a greater difference among the executions may be obtained if new nodes present better bandwidth features.

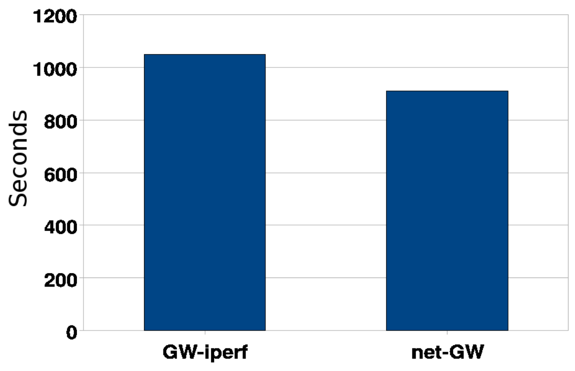
The completion times for TestHC are smaller (several tens of seconds, instead of several hundreds) because those jobs have a shorter total execution time and their files are not very big. For this type of test, it should have been defined the importance of CPU speed in the job template, weighted with the bandwidth, or setting a minimum required. This is because these jobs are heavier at computation than at data transfers, at least for this Grid and under the studied traffic conditions. If link speed were lower or there were more traffic, transfers would be heavier and computation softer.

Now we are going to present another set of tests, which are different than the previous ones as only one job is executed each time. That job has a big input file, doing the job heavy in data transfer and light in computation. This way, the improvements of using bandwidth information are highlighted. This is shown in Figure 7. As the previous tests, these ones are aimed at showing the usefulness of using bandwidth information under varying traffic conditions.

Figure 7 shows the completion times for jobs, and transfer and execution times have been distinguished. As before, experiments have been conducted with and without background network traffic, running GW-iperf and net-GW as scheduling algorithms. Figure 7 (a) shows results of tests that have been executed one after another. We can see that the presence of network traffic has hardly influenced the completion time when net-GW is running.



(a) Execution of TestVPModif



(b) Execution of TestSPs3

Figure 8: Execution of tests with background CPU load

Provided there were a greater number of transfers or links were slower, this difference among net-GW and GW-iperf executions would be more highlighted. This is because transfers last more and selecting a node with faster link improves transfer time significantly. Also, all the Grid resources are similar, since execution time is the same for all the experiments.

Figure 7 (b) presents results of tests that have been executed simultaneously, this is, some jobs are scheduled by GW-iperf and some others by net-GW. The file transfers of one of those tests are the background network traffic for the other. The time increase needed to perform the transfers can be seen. This is because the tests are submitted from the same node (*GridWayI3A.uclm.es*) to the same subnet (*Aulasun*), that implies slow transfers between them. But net-GW chooses a resource with an unloaded link whilst GW-iperf does not consider bandwidth information, thus creating time difference between them.

Using bandwidth highly improves the time necessary to perform transfers, even though the job scheduled by net-GW finishes before the job scheduled by GW-iperf and release the network link. In that moment, the GW-iperf job can increase its transfer speed.

Once we have studied the influence of background network traffic, we move on to study the influence of background CPU load. This evaluation has been performed by means of tests submitted to nodes that are executing an unknown background CPU load. The tests executions are the next:

- For test *VPModif* with CPU background load, the results shown at Figure 8 (a) are obtained. Emphasizing that the background CPU load leaves only one free processor to *Aulasun* nodes but it does not use bandwidth. Because of this, there is not a big difference between our proposal net-GW which uses bandwidth information, and GW.

  This figure depicts that there are not any improvement when using net-GW. Due to the test parallelism and the shortage of free nodes, jobs must wait for the nodes to be released in order to use them. In this way, the link bandwith is underused because there are not nodes to choose. VPModif test requires 3 nodes at a time so, since nodes are busy, jobs will have to wait. Also, a node with less bandwidth can be chosen. Thus, the use of net-GW does not make any difference.

- For test *SPs3* the results shown in Figure 8 (b) are obtained. In this case, there is an improvement when using net-GW, because this test is not as parallel as the previous one and it does not have to wait for free nodes. This is because executions are shorter and resources are released before. Thus, nodes with better bandwidth are chosen, which leads to better completion time.

4.3.2 Resource addition.

Once the evaluation of the influence of both network and CPU background load has been presented, we proceed with the second evaluation. For this evaluation, two new hosts are added to the Grid, so that a more heterogeneous Grid environment is created. We do that in order to analyze the advantages of using bandwidth information in schedulers tasks, as well as testing other parameters that can influence the goodness of schedulers.

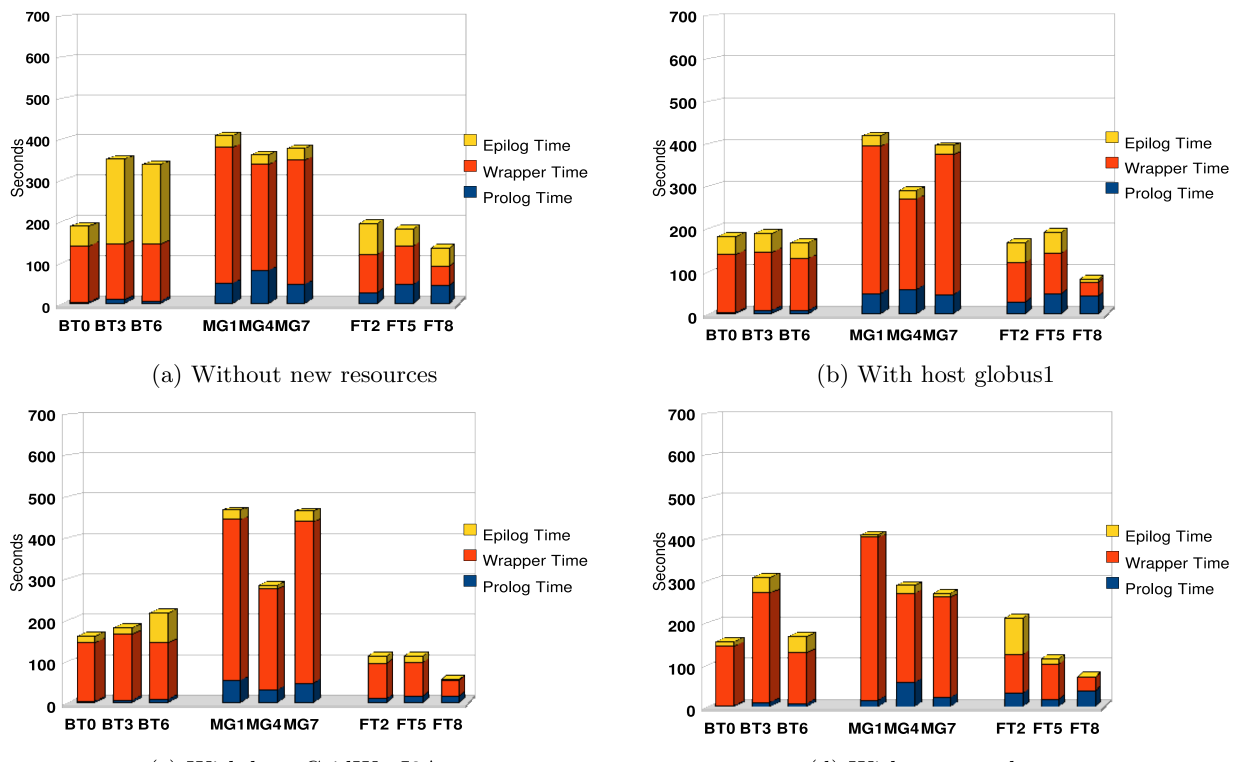
The two new machines included are the next, and draw the topology presented in Figure 4:

- **GridWayI3A.uclm.es**: Although this machine was already in the Grid (because it was entrusted with the jobs scheduler), it was not allowed to run jobs. Now the possibility of executing jobs in this host has been added.
- **Globus1.uclm.es**: This new machine is in the same subnetwork as the previous one and because of this, it has a better bandwidth than *Aulasun* machines. Thanks to this, it can highlight the advantages of using bandwidth information at scheduling, getting greater improvements when there is background traffic between *Aulasun* nodes.

In this evaluation phase only one type of test will be executed, and different characteristics have been changed. This has been done in order to get information about how each of those characteristics affect the time taken by the job scheduling and the total execution time of that test. This set of configurations has been studied by executing test “VP” of NGB benchmarks [21], because it is a type of test that have big file transfer and it is heavily enough at computation to get interesting conclusions of its results. The characteristics that have been modified among different executions are the next:

- New resources have been added, in order to see their influence at scheduler tasks.
- Host information update time has been changed, paying special attention to *iperf* command time, that is, the accuracy with which the tool calculates bandwidth.
- Traffic has been generated, which alters the execution environment.

Firstly, we have tested the addition of new hosts and their influence at total execution time of tests. Since there are more resources (which are heterogeneous in terms of hardware, software and link bandwidth), more remarkable results have been obtained when net-GW is used. Figure 9 shows the execution results using net-GW without background traffic. X axes are labeled with the different types of job that are executed in test “VP” (*BT, MG and FT*)



(a) Without new resources

(b) With host globus1

(c) With host GridWayI3A

(d) With two new hosts

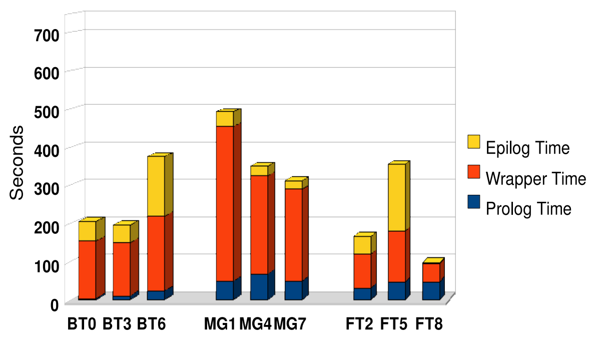
Figure 9: VP executions with different number of resources without background network traffic, using net-GW

with the iteration number with which they are executed (see [21]). As the figure shows, there are noticeable changes in the completion time between jobs. Also, there are variations for a given job depending on the node which executes it.

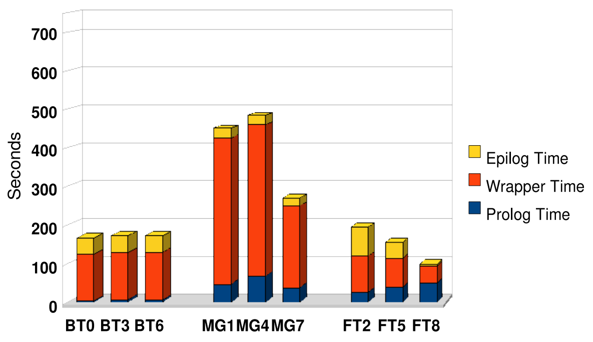
Improvements of total completion time are obtained if either host *globus1* or *GridWayI3A* are added (see Figure 9 (b) and (c)), as more resources are available to execute jobs, and these nodes have better bandwidth than *Aulasun* nodes. We must pay attention to the *prolog* and *epilog* times as it is then when file transfers are performed. *Wrapper* time is the execution time of jobs in the resources chosen.

There are not noticeable improvements at *wrapper* time, but there is a great improvement at prolog and epilog times (in which these new nodes are involved). This is because the new nodes (which are located in the same subnetwork as the scheduler node) have been chosen to run jobs. In this case, less jobs are executed in *Aulasun* nodes (whose network links are worse since they are in a different subnetwork). For the test whose results are depicted in Figure 9 (b), 5 jobs have been executed in *globus1* node. For the test whose results are depicted in Figure 9 (c), 6 jobs have been executed in *GridWayI3A* node. For the test whose results are depicted in Figure 9 (d), 5 jobs have been executed in *GridWayI3A* node and 3 in *globus1* node.

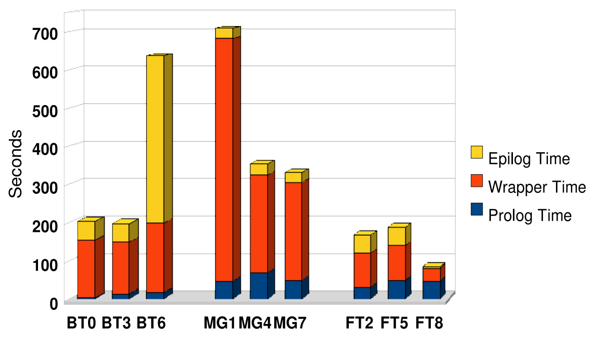
Also differences between adding *globus1* or *GridWayI3A* are shown. This is because



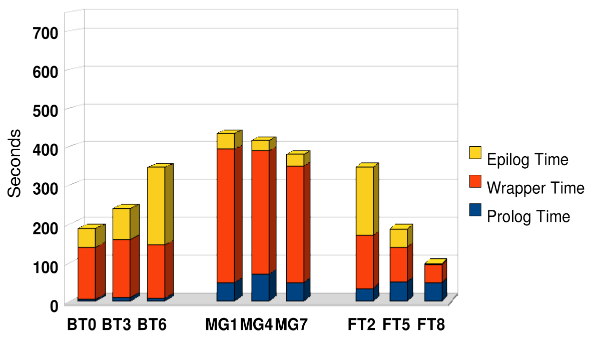
(a) Without new resources



(b) With host globus1



(c) With host GridWayI3A



(d) With two new hosts

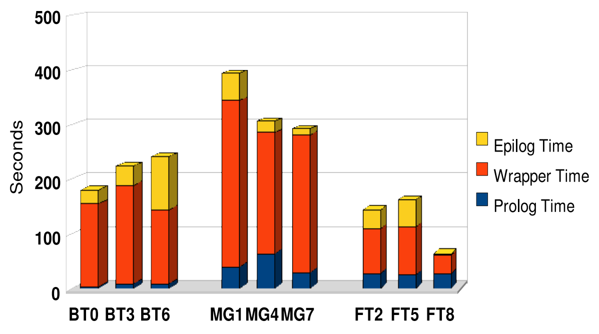
Figure 10: VP executions with different number of resources without background traffic, using GW-iperf

the second node, since it is the local node (that runs the scheduler), present a better transfer rate and for this reason the total completion time is improved.

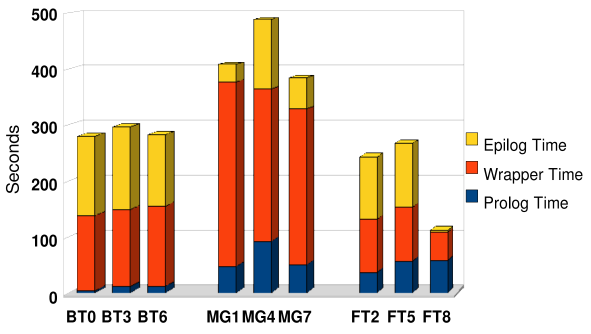
Finally it is observed that, when the two nodes are added, completion times are improved even more due to now there are two resources with better bandwidth features instead of only one (see Figure 9 (d)). Thus, because the jobs test are executed in parallel, allow greater improvements. The reason for this is that in the previous cases we had only 1 new resource, thus some jobs had to be sent to *Aulasun* nodes. With both new resources, no job has to be sent to *Aulasun* since one of this two new nodes is available.

If GW-iperf is used, the results presented in Figure 10 are obtained. There are light differences among them due to when GW-iperf is used (other policies are disabled), the scheduler chooses the node to execute each job in the same order as nodes were discovered. This is, when a job arrives, if the first node has idle processors, the job will be dispatched to it. This is true although the new hosts are available, because *Aulasun* hosts were discovered first. Thus, they will be chosen even if they have worse bandwidth features.

Figure 10 (b), showing the addition of *globus1*, presents better results than with the original topology, although worse than for the same case when net-GW is used. The reason for this is that *globus1* is the first host of the list that the scheduler has, and while *globus1* has idle CPUs, it will be selected to run jobs. This selection is the correct because *globus1*



(a) net-GW



(b) GW-iperf

Figure 11: Executions of VP test with background traffic

has the greater bandwith for this test since it is in the same subnetwork as the scheduler. On the other hand, if in the middle of the experiment, the bandwidth towards this node decreases (more transfers to this node are been carried out), and for this reason the node is not the best any more, the completion time will suffer an increase if GW-iperf is used.

Figure 11 presents results for the complete topology (including *globus1* and *GridWayI3A*) and with background network traffic, for both GW-iperf and net-GW. As before, the background network traffic is created by one file transfer from one node to another, by means of the *globus-url-copy* command, and only one transfer is carried out in the whole topology.

As we can see, transfers are slower for both *prolog* and *epilog* (which include the transfer of input/output files). It is during transfer time when improvements are obtained, since *wrapper* (or execution) time is very similar for all the executions for each job. When net-GW is used, prolog and epilog times are highly improved, since nodes with better bandwidth are chosen (*globus1* and *GridWayI3A*), and *Aulasun* nodes are only used when those two ones were busy.

Figure 12 shows results with different background traffic conditions in the *Aulasun* subnetwork, and using the complete topology. These conditions are explained the next.

- *No traffic*: No background traffic is delivered.
- *Traffic to one node*: One file is transfered from one node to another, by means of the *globus-url-copy* command. Only one transfer is carried out in the whole topology.
- *Heavy traffic to one node*: Similar to the previous one, but with two transfers between the same pair of nodes.
- *Different traffic to 2 nodes*: A combination of the last two. One node sends one file to another node, and the same source node sends two files to another destination node. There are 3 file transfers at a moment in the whole topology.
- *Traffic to all the nodes*: One node sends files to all the other nodes of *Aulasun*.

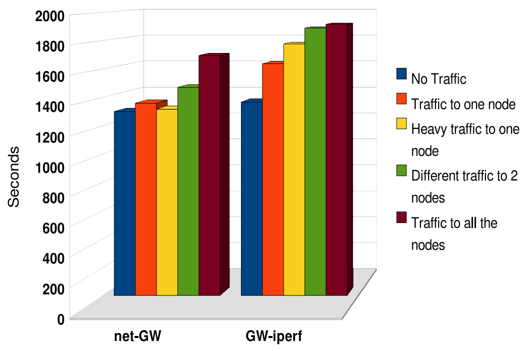


Figure 12: Completion times of VP test with variable background traffic

As we can see, as we increase the background traffic, executions of GW-iperf suffer an important increase in the completion time, whilst net-GW only suffer an important increase for the highest background traffic configuration. If there is traffic to only *one node*, the completion time using net-GW remained stable (only an insignificant increase is shown). When there is background traffic to *more than one node*, net-GW jobs suffer a clearer increase in the completion time, which is specially clear when there is background traffic to all the nodes. However, these increases are lower than for the GW-iperf executions. Besides, provided GW-iperf is used, the completion times suffer a remarkable increase for all the background traffic configurations studied.

4.3.3 Adjusting the network monitoring tool.

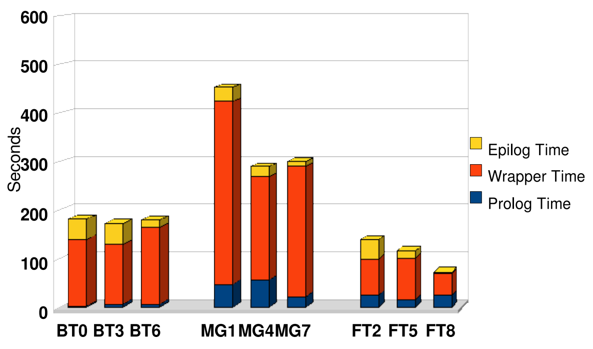
The next performance evaluation to be presented is the influence of the bandwidth update time. This is performed with the full Grid topology including all the computing resources, and results are presented in Figure 13.

In this point, including *GridWayI3A* as a working node has a remarkable significance, since the greater the update time of hosts information is, the lesser its load will be. For this reason, *GridWayI3A* will have more availability to run jobs.

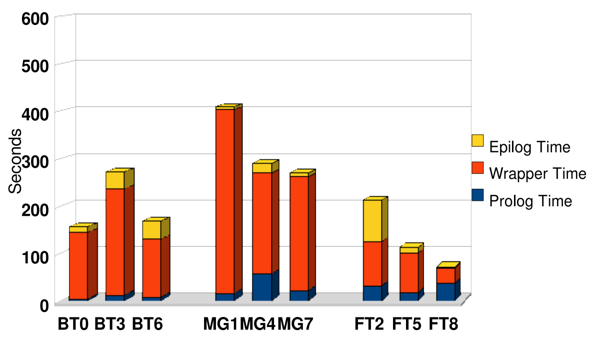
Figure 13 presents results of executions of VP tests, with net-GW, when the monitoring interval is being varied. This interval includes the monitoring of the bandwidth and the computing resources. It shows that if interval time is too short, this is worse for the scheduler. This is because the whole network (specially *GridWayI3A* node) are busy serving the requests of the monitoring tools, thus they will not be able to proceed with the jobs executions. Recall that we have two monitoring tools running, namely iperf (monitors the network bandwidth) and Ganglia (monitors the other parameters, such as free memory or CPU availability).

When the interval time is reduced the transfers time is increased. For example, when time interval is 100 seconds, prolog times add about 208 seconds and epilog times about 215, and when this interval is reduced to 15 second transfer times grow up to 321 seconds for prologs and 312 for epilogs.

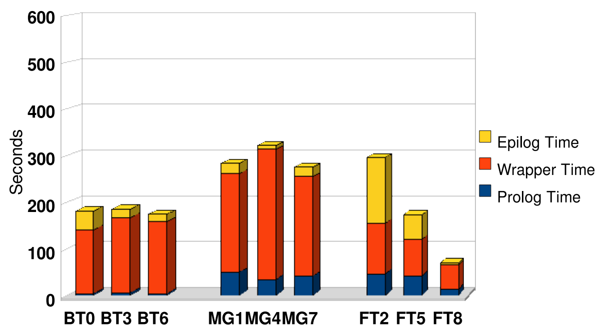
Also, if time between two updates is too long, execution time is increased due to the



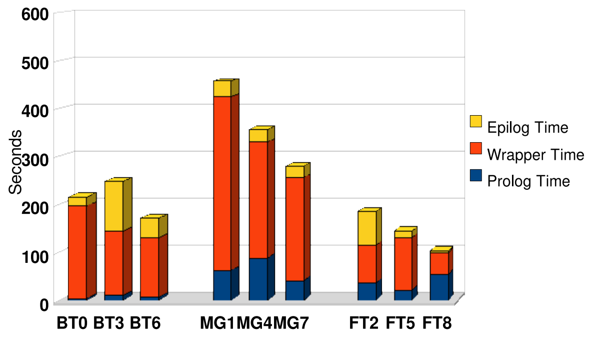
(a) Every 100 seconds



(b) Every 60 seconds



(c) Every 30 seconds



(d) Every 15 seconds

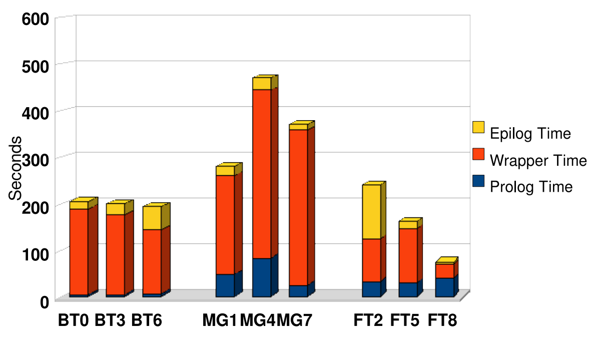
Figure 13: Executions of VP test changing monitoring interval time of resources

information about bandwidth is not up-to-date enough. Thus, it is possible that the current host status does not correspond with the stored, making bad scheduling decisions. For this reason, finding a trade-off to set this time is necessary. This time should not be too big because the information about hosts would not be up-to-date. Nevertheless, it should not be too small, so that nodes do not get busy answering monitoring requests. For our Grid environment, the time that gets the best results is nearly 1 minute. This is because we have a small environment, not too variable and also the scheduler node is a worked node.

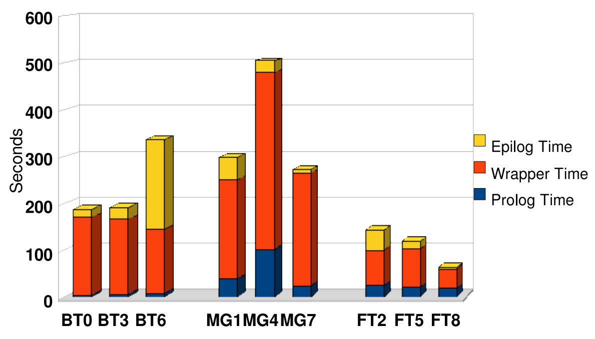
Figure 14 shows the completion times for VP executions when varying the monitoring interval and the *iperf* sending time. Recall that iperf sending time is the time interval during which iperf sends packets in order to calculate the bandwidth of links. Up to now, the iperf sending time has been set to its default value (1 sec.).

We can see that when the sending time of *iperf* tool is increased, prolog and epilog times are incremented too. Also, wrapper times are increased because *GridWayI3A* has to perform the monitoring calculations, which require CPU. Thus, jobs executed in *GridWayI3A* suffer wrapper time increase. Jobs allocated to other nodes will also suffer some increase as these nodes have to deal with *GridWayI3A* for prolog and epilog tasks.

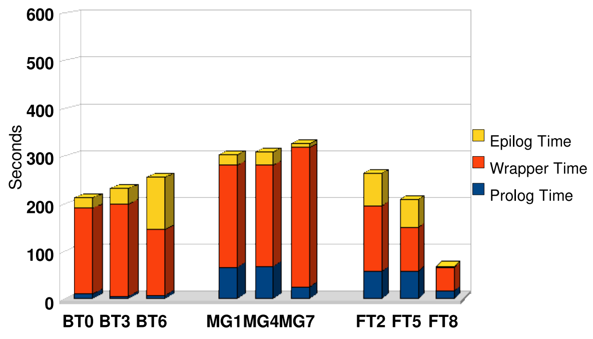
Figure 14 (d) shows that when this monitoring load is extreme, and the *GridWayI3a* spends more time updating hosts information than scheduling jobs, all file transfers are delayed, and transfer times add near 735 seconds for prologs and to 712 seconds for epilogs.



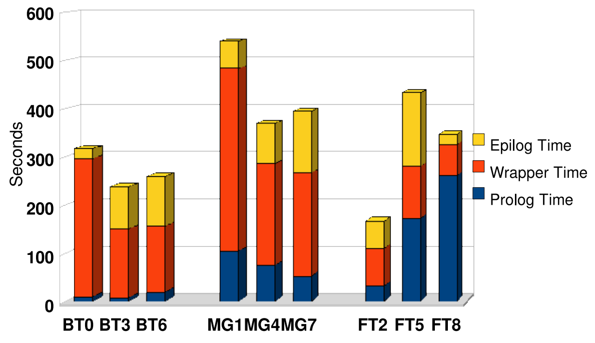
(a) T.Iperf=2s;T.Monitoring=60s



(b) T.Iperf=5s;T.Monitoring=100s



(c) T.Iperf=5s;T.Monitoring=60s



(d) T.Iperf=5s;T.Monitoring=15s

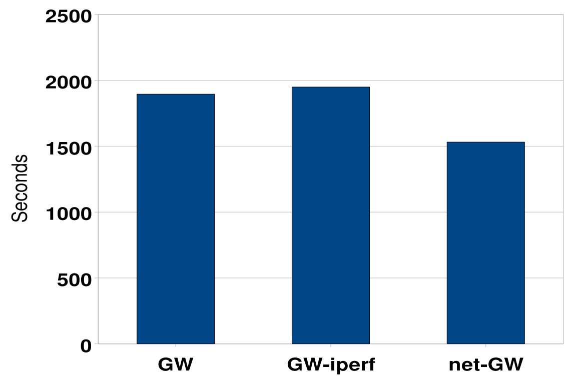
Figure 14: Executions of VP test changing iperf command time and monitoring time of resources

After studying different values for the iperf sending time, we deem the default value (1 sec.) as the best one, combined with a monitoring interval of 60 sec.

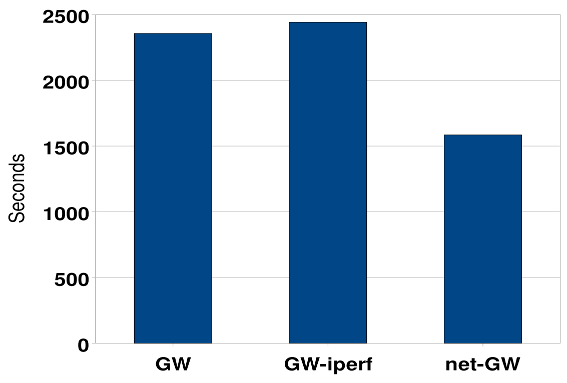
4.3.4 A cost estimation of the proposed scheduling.

Finally, the results of executing VP test using GW, GW-iperf and net-GW are compared, in the presence and absence of background network traffic. The background traffic used in this experiment is *traffic to one node*, as explained before. Recall that GW is the original GridWay scheduling algorithm, GW-iperf is the same algorithm but with overhead caused by calculating bandwidth information, and net-GW is our new proposal which considers the bandwidth information for the scheduling.

Figure 15 shows these comparisons. Figure 15 (a) presents results without background traffic, and we can see that the calculation of the bandwidth information creates a negligible overhead (see columns GW and GW-iperf). When this information is used to perform the scheduling (net-GW column) there is a remarkable gain. In the presence of background traffic (Figure 15 (b)), these differences are emphasized. We can also see that when net-GW is running, there is not noticeable differences between having or not background traffic. This is because a resource with better bandwidth is chosen to execute the jobs, with and without background traffic.



(a) Without background traffic



(b) With background traffic

Figure 15: Overhead caused by iperf

5 Related work

The topic of interest of our work is the provision of network QoS in a Grid system. This field has been explored by a number of research projects, namely GARA [4], NRSE [22], G-QoSM [3], GNRB [23], and VIOLA [24]. They are briefly reviewed below.

General-purpose Architecture for Reservation and Allocation (GARA) [4] is a framework whose aim is providing programmers and users with an uniform way to access different types of resources (CPU, network, storage). These uniform mechanisms have a modular structure which allows an easy development of high-level services. On the other hand, GARA has limitations when performing reservations across difference domains, as it must exist in all the traversed domains, which leads to scalability problems.

The *Network Resource Scheduling Entity* (NRSE) [22] suggests that signalling and per-flow state overhead can cause end-to-end QoS reservation schemes to scale poorly to a large number of users and multi-domain operations – observed when using IntServ and RSVP, as also with GARA [22]. This problem has been tackled in NRSE by storing the flow/application state only the end sites of the communication. Although NRSE has demonstrated its effectiveness in providing DiffServ QoS, it is not clear how a Grid application developer would make use of this capability – especially as the application programming interface is not clearly defined [3].

Grid Quality of Service Management (G-QoSM) [3] is a framework to support QoS management in computational Grids in the context of the Open Grid Service Architecture (OGSA). G-QoSM is a generic modular system that, conceptually, supports various types of resource QoS, such as computation, network and disk storage. This framework aims to provide three main functions: 1) support for resource and service discovery based on QoS properties; 2) provision for QoS guarantees at application, middleware and network levels, and the establishment of Service Level Agreements (SLAs) to enforce QoS parameters; and 3) support for QoS management of allocated resources, on three QoS levels: ‘guaranteed’, ‘controlled load’ and ‘best effort’. G-QoSM also supports adaptation strategies to share resource capacity between these three user categories.

The *Grid Network-aware Resource Broker* (GNRB) [23] is an entity that merges the functionalities of a Grid Resource Broker and a Network Resource Manager. Because of this, new mapping/scheduling strategies can be implemented, so that both computational and network conditions are taken into account. The GNRB, using network status information, can reserve network resources to satisfy the QoS requirements of applications. The architecture is centralized, with one GNRB per administrative domain – potentially leading to the GNRB becoming a bottleneck within the domain. Also, GNRB is a framework, and does not enforce any particular algorithms to perform scheduling of jobs to resources.

The aim of the research being carried out by the authors is to provide network QoS by means of an efficient scheduling. Many of the above efforts do not take network capability into account when scheduling tasks. Among the reviewed proposals, those which provide scheduling of users' jobs to computing resources are GARA, G-QoSM and GNB, and the schedulers used are DSRT [25] and PBS [26] in GARA, whilst G-QoSM uses DSRT. Thus, the proposed GNB framework is the only proposal which considers the network. These schedulers (DSRT and PBS) only pay attention to the load of the computing resource, thus a powerful unloaded computing resource with an overloaded network could be chosen to run jobs, which decreases the performance received by users, especially when the job requires a high network I/O.

Finally, VIOLA [24] provides a metascheduling framework that provides co-allocation support for both computational and network resources. It is able to negotiate with the local scheduling systems to find and to reserve a common time slot to execute various components of an application. The metascheduling service in VIOLA has been implemented via the UNICORE Grid middleware for job submission, monitoring, and control. A key feature in VIOLA is the network reservation capability, that allows the network to be treated as a resource within a metascheduling application. In this context, VIOLA is somewhat similar to the author's approach – in that it also considers the network as a key part in the job allocation process. However, the key difference is the focus in VIOLA on co-allocation and reservation – which is not always possible if the network is under ownership of a different administrator.

It should also be mentioned a previous effort in including network-related information into the GridWay metascheduler, which was reported in [27]. In that case, the internals of the `RANK` computing mechanism were altered in order to include bandwidth and latency information, obtained through the *iperf* tool and the Network Weather Service (NWS) [28]. The work presented here follows a similar idea, but the way it has been deployed into GridWay allows to include network related restrictions when filtering out resources too (i.e., through the use of the `BANDWIDTH` parameter within the `REQUIREMENTS` expression). Moreover, with the approach presented here, the user can combine the network-related information with any other GridWay parameter, both in the `REQUIREMENTS` and in the `RANK` sections, to better match the needs of the jobs to the features of the available resources.

6 Conclusions and future work

A large number of projects in science and engineering would not have been feasible without the capabilities for resource aggregation that Grid computing offers. One of the issues that still need research is the impact of the network characteristics when taking decisions on where to schedule jobs for execution. The authors have previously proposed a theoretical framework for metascheduling, where network conditions were taken into account by the metascheduling entity.

In this work, a working proof-of-concept implementation of a Grid network-aware metascheduler has been presented and evaluated. Tests have been carried out using the NAS Grid Benchmarks (NGB) as workload. Results regarding the influence of background network and CPU loads, heterogeneity of resources, monitoring parameters, and cost are presented. They clearly show that taking into account network conditions (such as available bandwidth) when scheduling jobs to resources greatly improves completion times, and the overhead created is totally harnessed to reduce the completion time of applications. Thus, Grid users receive back the outcome of their jobs with a lower response time. This is quite useful specially for interactive applications such as collaborative visualization [5], and those integrated within the Interactive European Grid Project [29].

As future work, it is planned to extend the current implementation with a number of features, such as considering network features altogether with CPU features to perform a more complex and efficient scheduling.

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”; jointly by JCCM and Fondo Social Europeo under grant “FSE 2007-2013”; and by JCCM under grants “PBC-05-007-01”, “PBC-05-005-01”.

References

[1] Foster, I., Kesselman, C.: The Grid 2: Blueprint for a New Computing Infrastructure. 2 edn. Morgan Kaufmann (2003)

[2] CERN: LHC Computing. Web page at `http://www.interactions.org/LHC/computing/index.html` (2008)

[3] Al-Ali, R., et al.: Network QoS Provision for Distributed Grid Applications. Intl. Journal of Simulations Systems, Science and Technology, Special Issue on Grid Performance and Dependability **5**(5) (December 2004)

[4] Roy, A.: End-to-End Quality of Service for High-End Applications. PhD thesis, Dept. of Computer Science, University of Chicago (2001)

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

[6] E. Huedo, R.S.M., Llorente, I.M.: A modular meta-scheduling architecture for interfacing with pre-ws and ws grid resource management services. Future Generation Computing Systems **23**(2) (2007) 252–261

[7] Condor-G Project. Web page at `http://www.cs.wisc.edu/condor/condorg/` (2008)

[8] Caminero, A., Carrión, C., Caminero, B.: 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)

[9] Caminero, A., Rana, O., Caminero, B., Carrión, C.: An Autonomic Network-Aware Scheduling Architecture for Grid Computing. In: Proc. of the 5th Intl. Workshop on Middleware for Grid Computing (MGC), Newport Beach, USA (2007)

[10] Fitzgerald, S., Foster, I., Kesselman, C., von Laszewski, G., Smith, W., Tuecke, S.: A directory service for configuring high-performance distributed computations. In: Proc. 6th Symposium on High Performance Distributed Computing (HPDC), Portland, USA (1997)

[11] Massie, M.L., Chun, B.N., Culler, D.E.: The Ganglia distributed monitoring system: design, implementation, and experience. Parallel Computing **30**(5-6) (2004) 817–840

[12] Sohail, S., Pham, K.B., Nguyen, R., Jha, S.: Bandwidth Broker Implementation: Circa-Complete and Integrable. Technical report, School of Computer Science and Engineering, The University of New South Wales (2003)

[13] Caminero, A., Caminero, B., Carrión, C.: Network-aware Grid Scheduling. In: Proc. of the Intl. Conference on Grid computing, high-performAnce and Distributed Applications (GADA), Vilamoura, Portugal (2007)

[14] Caminero, A., Carrión, C., Caminero, B.: On the efficient use of network dynamic information to perform grid scheduling. In: Proc. of the 2nd Iberian Grid Infrastructure Conference (IberGrid), Porto, Portugal (2008)

[15] Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z., Weiss, W.: Internet RFC 2475 : An architecture for differentiated services. (1998)

[16] Caminero, A., Rana, O., Caminero, B., Carrión, C.: Providing network QoS support in Grid systems by means of peer-to-peer techniques. In: Proc. of the 2008 Intl. Conference on Grid Computing and Applications (GCA'08), Las Vegas, USA (2008)

[17] NLANR/DAST: Iperf - The TCP/UDP Bandwidth Measurement Tool. Web page at `http://dast.nlanr.net/Projects/Iperf/` (2008)

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

[19] The Globus Alliance. Web page at `http://www.globus.org` (2008)

[20] GridWay Team: "GridWay 5.2 Documentation: User Guide". Distributed Systems Architecture Group, Universidad Complutense de Madrid. (2007)

[21] Frumkin, M., Van der Wijngaart, R.: Nas grid benchmarks: a tool for grid space exploration. In: Proceedings of 10th IEEE International Symposium on High Performance Distributed Computing. (2001)

[22] Bhatti, S., Sørensen, S., Clark, P., Crowcroft, J.: Network QoS for Grid Systems. The Intl. Journal of High Performance Computing Applications **17**(3) (2003)

[23] Adami, D., et al.: Design and implementation of a grid network-aware resource broker. In: Proc. of the Intl. Conference on Parallel and Distributed Computing and Networks, Innsbruck, Austria (2006)

[24] Waldrich, O., Wieder, P., Ziegler, W.: A Meta-scheduling Service for Co-allocating Arbitrary Types of Resources. In: Proc. of the 6th Intl. Conference on Parallel Processing and Applied Mathematics (PPAM), Poznan, Poland (2005)

[25] Chu, H.H., Nahrstedt, K.: CPU service classes for multimedia applications. In: Proc. of Intl. Conference on Multimedia Computing and Systems (ICMCS), Florence, Italy (1999)

[26] Mateescu, G.: Extending the Portable Batch System with preemptive job scheduling. In: SC2000: High Performance Networking and Computing, Dallas, USA (2000)

[27] Fuentes, A., Huedo, E., Moreno, R., Martin-Llorente, I.: A grid scheduling algorithm considering dynamic interconnecting network. In: The 3rd Cracow Grid WorkShop. (2003)

[28] Wolski, R., Spring, N.T., Hayes, J.: The network weather service: A distributed resource performance forecasting service for metacomputing. Future Generation Computer Systems **15**(5–6) (1999) 757–768

[29] Interactive European Grid Project. Web page at `www.interactive-grid.eu` (2008)