4

Autonomic Network-Aware Metascheduling for Grids: A Comprehensive Evaluation

Agustín C. Caminero[1], Omer Rana[2],
Blanca Caminero[3] and Carmen Carrión[4]
[1]*The National University of Distance Education*
[2]*Cardiff School of Computer Science*
[3,4]*The University of Castilla La Mancha*
[1,3,4]*Spain*
[2]*UK*

1. Introduction

Grid technologies allow the aggregation of dispersed heterogeneous resources for supporting large-scale parallel applications in science, engineering and commerce (Foster & Kesselman (2003)). Current Grid systems are highly variable environments, made of a series of independent organizations sharing their resources, creating what is known as *Virtual Organizations* (VOs) (Foster & Kesselman (2003)). This variability makes Quality of Service (*QoS*) highly desirable, though often very difficult to achieve in practice. One reason for this limitation is the lack of control over the network that connects various components of a Grid system. 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 (Marchese & Brajkovska (2007))), 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 (Roy (2001)). In a VO, entities communicate with each other using an interconnection network – resulting in the network playing an essential role in Grid systems (Roy (2001)).

In (Caminero et al. (2009a)), authors proposed an autonomic network-aware Grid scheduling architecture as a possible solution, which can make scheduling decisions based on the current status of the system (with particular focus on network capability). This architecture focuses on the interchange of I/O files between users and computing resources, and the impact these interchanges have on the performance received by users. This work presented a performance evaluation in which the proposal was compared with one existing non network-aware meta-scheduler. Besides, authors have also presented an evaluation in which the proposal is compared with an existing network-aware meta-scheduler (Caminero et al. (2009b)).

The main contribution of this work is a comprehensive evaluation in which our proposal is compared with existing commercial meta-schedulers and heuristics found in literature, both network-aware and non-network-aware. These proposals are the GridWay meta-scheduler (Huedo et al. (2007)), the Gridbus Broker (Venugopal et al. (2008)), Min-min (Freund

et al. (1998)), Max-min (Freund et al. (1998)), and XSufferage (Casanova et al. (2000)). Thus, the benefits of taking the network into account when performing meta-scheduling tasks is evaluated, along with the need to react autonomically to changes in the system.
The structure of the paper is as follows: Section 2 reviews the two existing meta-schedulers against which our proposal is evaluated along with other heuristics found in literature. Section 3 shows a sample scenario in which an autonomic scheduler can be used and harnessed. Section 4 shows the solution we propose – consisting of a network-aware autonomic meta-scheduling architecture capable of adapting its behavior depending on the status of the system. In this section the main functionalities of the architecture are presented, including models for predicting latencies in a network and in computing resources. Section 5 presents a performance evaluation of our approach. Section 6 draws conclusions and provides suggestions for future work.

2. Meta-scheduling in Grids

Since the purpose of this work is the provision of QoS in Grids by means of an efficient meta-scheduling of jobs to computing resources, existing proposals aimed at this point are reviewed in this Section. Imamagic et al. (2005) provide a survey of Grid meta-scheduling strategies, some of them are reviewed below. Also, an in-depth study of meta-schedulers in the context of TeraGrid is presented in (Metascheduling Requirements Analysis Team (2006)), where key capabilities of meta-schedulers are identified, and meta-schedulers are classified based on them.
In this section two widely used meta-schedulers are reviewed, along with other proposals found in literature. These will be used to compare our proposal. The meta-schedulers reviewed are GridWay and Gridbus, and the heuristics are Min-min, Max-min, and XSufferage.

2.1 GridWay meta-scheduler

The *GridWay* meta-scheduler (Huedo et al. (2007)) 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 (Mateescu (2000)), SGE (Gentzsch (2001)), or Condor (Litzkow et al. (1988))) within a single organization (enterprise Grid) or scattered across several administrative domains (partner or supply-chain Grid). GridWay is a Globus project (Globus Projects (2010)), adhering to Globus philosophy and guidelines for collaborative development. Besides, GridWay comes with the Globus releases from version GT4.0.5 onwards. GridWay has been used for a variety of research works, among others (Vázquez et al. (2010)); (Bobroff et al. (2008)); (Tomás et al. (2010)). Thus, it can be concluded that GridWay is a widely used tool, and that is the reason why it has been chosen to compare our proposal with.
GridWay works as follows. Users willing to submit jobs to the Grid infrastructure managed by GridWay need to generate a *job template*. This template 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 meta-scheduling, performance, fault tolerance or resource selection, among others.
The way how GridWay performs the meta-scheduling is explained the following. There are two parameters related to this issue within the job template, namely `REQUIREMENTS` and

`RANK`. They together allow users to specify the criteria used to select the most appropriate computing resource to run their jobs, in a two-step process.
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 fulfill 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. Network status is not one of them at present. Many of these values are gathered through the Globus GIS module (Czajkowski et al. (2001)), while others (specifically the dynamic ones, such as amount of free RAM) are monitored through Ganglia (Massie et al. (2004)).

2.2 The gridbus broker

The *Gridbus Broker* (Venugopal et al. (2008)) is a network-aware meta-scheduler that mediates access to distributed resources by (a) discovering suitable data sources for a given analysis scenario, (b) discovering suitable computing resources, (c) optimally mapping analysis jobs to computing resources, (d) deploying and monitoring job execution on selected resources, (e) accessing data from local or remote data source during job execution and (f) collecting and presenting results. The broker supports a declarative and dynamic parametric programming model for creating Grid applications (Venugopal et al. (2006)).
The Gridbus Broker has been designed to operate with different Grid middleware frameworks and toolkits such as Globus 2.4 (Foster & Kesselman (1997)), that primarily runs on Unix-class machines, and Alchemi (Luther et al. (2005)), which is a .NET based Grid computing platform for Microsoft Windows-enabled computers. Hence, it is possible to create a cross-platform Grid implementation using the Gridbus Broker (Venugopal et al. (2006)).
The meta-scheduling of jobs is performed as follows. The broker tries to choose the best resource from the users' point of view, this is, the resource that can have jobs executed the fastest. Thus, Gridbus works by trying to minimize the amount of data transfer involved for executing a job by dispatching jobs to compute servers which are close to the source of data. The meta-scheduler uses the *average job completion ratio* (the ratio of the number of jobs completed to the number of jobs allocated) to evaluate the performance of the computing resources. The job completion ratio of a resource is calculated as follows:

$$r_S = \frac{J_C}{J_Q} \tag{1}$$

where r_s is the job completion ratio for a particular resource, J_C is the number of jobs that were completed on that particular resource in the previous polling interval, and J_Q is the number of jobs that were queued on that resource in the previous allocation. Then, the meta-scheduler calculates the average job completion ratio, R_S, at the N^{th} polling interval as:

$$R_S = R_S^{'} * (1 - 1/N) + r_S/N \tag{2}$$

where $R_S^{'}$ is the average job completion ratio for the $N - 1^{th}$ polling interval. The averaging of the ratio provides a measure of the resource performance from the beginning of the

Algorithm 1 Min-min scheduling algorithm.

1: Let T = bag of independent tasks
2: Let t_i = a task
3: Let R = set of computing resources
4: Let r_j = a computing resource
5: Let C_i^j = expected completion time of task t_i in the computing resource r_j
6: **repeat**
7: **for all** t_i in T **do**
8: **for all** r_j in R **do**
9: calculate C_i^j
10: **end for**
11: **end for**
12: find the task t_i with the *minimum* expected completion time
13: find the resource r_j that obtains the *minimum* expected completion time for task t_i
14: map task t_i to resource r_j
15: remove t_i from T
16: **until** T is empty

meta-scheduling process and can be considered as an approximate indicator of the future performance of that resource.

Each resource is assigned a *job limit*, the maximum number of jobs that can be allocated among those jobs waiting for execution, which considers its average job completion ratio and the resource share available for Grid users. The meta-scheduler then iterates through the list of unassigned jobs one at a time. For each job, it first selects the data host that contains the file required for the job and then, selects a compute resource that has the highest available bandwidth to that data host. If this allocation plus previously allocated jobs and current running jobs on the resource exceeds the job limit for that resource, then the meta-scheduler looks for the next available nearest compute resource.

Thus, it is necessary to keep the bandwidth between each data host and each compute resource, which makes this approach unfeasible for a real-sized Grid, because of the huge amount of bandwidth data to keep. An example of such a real-sized Grid is LHC Computing Grid (LCG (LHC Computing Grid) Project (2010)), which has around 200 sites and tens of thousands of CPU (for a map showing real time information, see (GridPP, Real Time Monitor (2010))).

2.3 Heuristics for meta-scheduling

Three heuristics are reviewed in this section, namely Min-min, Max-min, and XSufferage. *Min-min* (Freund et al. (1998)) is presented in Algorithm 1, which works as follows. First, an estimation on the completion time of each task in each computing resource is calculated (line 9). Then, the task with the minimum completion time is chosen (line 12), and mapped to the computing resource providing that completion time (line 14). After that, the task is removed from the bag of tasks (line 15), and estimated completion times are updated for all the remaining tasks (line 9).

Max-min (Freund et al. (1998)) is similar to *Min-min*, the only difference is that instead of getting the minimum expected completion time (line 13), the *maximum* expected completion time is chosen. Max-min is likely to do better than the Min-min heuristic in cases where

Algorithm 2 XSufferage scheduling algorithm.

1: Let T = bag of independent tasks
2: Let t_i = a task
3: Let R = set of computing resources
4: Let r_j = a computing resource
5: Let C_i^j = expected completion time of task t_i in the computing resource r_j
6: Let m_i = minimum completion time for task t_i
7: Let m_i^2 = second minimum completion time for task t_i
8: Let S_i = sufferage value of task t_i
9: **repeat**
10: **for all** t_i in T **do**
11: **for all** r_j in R **do**
12: calculate C_i^j
13: **end for**
14: **end for**
15: **for all** t_i in T **do**
16: calculate m_i and m_i^2
17: find the computing resource r_j that gives m_i
18: $S_i = m_i^2 - m_i$
19: **end for**
20: find the task t_i with the *maximum* S_i
21: map task t_i to resource r_j
22: remove t_i from T
23: **until** T is empty

there are many more shorter tasks than long tasks. For example, if there is only one long task, Max-min will execute many short tasks concurrently with the long task. The resulting makespan might just be determined by the execution time of the long task in theses cases. Min-min, however, first finishes the shorter tasks and then executes the long task, increasing the makespan.

XSufferage (Casanova et al. (2000)) calculates the priority of a task based on its sufferage value. The sufferage of a task is calculated as the difference between the least and the second least expected completion time for that task. Algorithm 2 presents this approach. First of all, and in the same way as the algorithms presented before, the expected completion time of each task in each computing resource is calculated (line 12). Then, the sufferage value for each task is calculated as explained above (line 18). The next step is finding the task with the highest sufferage value, this is, the task that would suffer the most if it were not mapped to the resource with the lowest estimated completion time. Then, this task is mapped to the resource providing the lowest estimated completion time for it (line 21), and it is removed from the bag of tasks (line 22).

3. Sample scenario

A sample scenario in which an autonomic scheduler could be used to improve the QoS perceived by users is depicted in Figure 1. In this scenario, a user has a number of jobs, m, and requires computing resource(s) to run his jobs. The user will have some Quality of Service

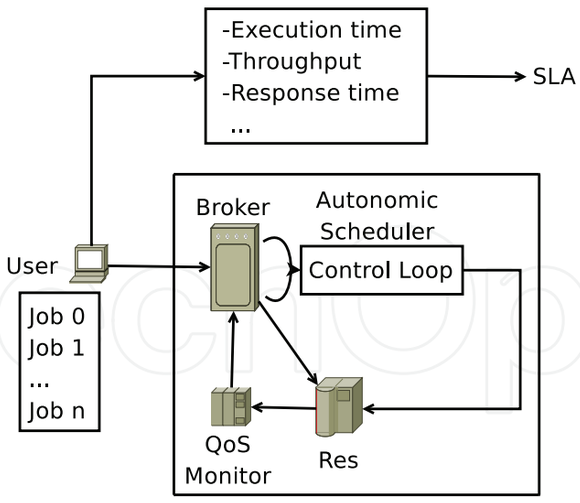


Fig. 1. Scenario.

(*QoS*) requirements, such as execution time, throughput, response time, etc. Each such QoS requirement could be represented as $(a_1, a_2, ..., a_n)$, where each a_i is a constraint associated with an attribute (e.g. $a_1 : throughput > 5$). Hence, a user's job m_i could be expressed as a set of constraints $\{a_j\}$ over one or more resources – expressed by the set $R = \{r_1, ..., r_k\}$. Let us consider that the QoS requirement of the user is that all his jobs must be finished within one hour. This deadline includes the transmission of jobs and input data to the resources, and the delivery of results. In order to have his jobs executed within the deadline, the user will contact a broker, and will tell it his QoS requirements. The role of the broker is to discover resource properties that fulfill the QoS constraints identified in the job. The constraint matching is undertaken by the broker – by contacting a set of known registry services. Once suitable resources have been discovered, the next step involves establishing Service Level Agreements (*SLAs*) to provide some degree of assurance that the requested QoS requirements are likely to be met by the resources.

When the user has already submitted his jobs to the computing resource, the broker will monitor the resource, to check whether the QoS requirements of the user are being met or not. The broker must decide when corrective action should be taken if particular QoS constraints are not being met. For instance, in our example, the user requirement is a one hour deadline; hence, the broker may consider that after 30 minutes half of the jobs should be finished (assuming that all the jobs require only one CPU). If this is not true, and less jobs have been finished, then the broker may decide to allocate more resources to run those jobs. This way, the broker modifies the initial set of resources allocated to run this user's jobs so that the deadline can be met. A key requirement in this scenario is for the broker to develop a *predictive* model identifying the *likely* completion times of the jobs that have been submitted by the user, and identifying whether the QoS constraints are *likely* to be violated.

Consider the scenario where the deadline for a job cannot be met. In this case, the broker can attempt to migrate this job to another resource, but depending on the type of job (for instance, if this job cannot be checkpointed) we have lost the time up to this moment (if after 30 minutes we discover that the job will not meet the deadline and decide to move it, we have only 30 minutes left to have the job executed in the new resource within the 1 hour deadline). Therefore, when choosing resources, it is necessary to take into account the

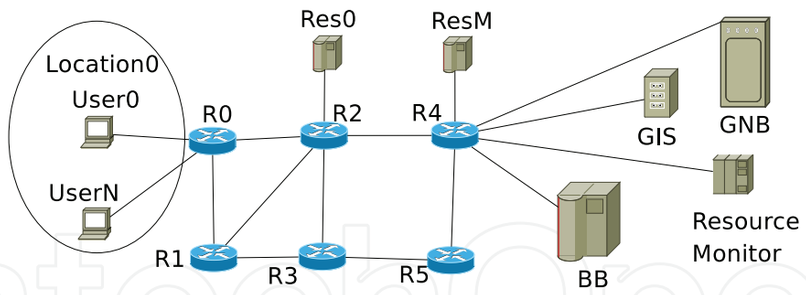


Fig. 2. Topology.

characteristics of each resource and each job during the resource allocation process. Therefore, we propose the following alternative resource selection strategy. When the user submits his resource request to the broker, including his QoS requirements, the broker will perform a resource selection taking into account the features of the resources available at this moment. Also, features of the jobs and their QoS requirements should also be taken into account. When the jobs have been allocated to a resource, the broker performs the monitoring. If the broker infers that the requirement will not be met (for instance, after 30 minutes it may deduce that the 60 minute deadline will not be met) it will proceed with a new resource allocation. Again, this resource allocation must be done taking into account the features of resources, jobs and QoS requirements. This scenario is focused on *high throughput computing*, where there are no dependencies among jobs.

4. Network-aware scheduling

Our initial approach, whose structure was presented in (Caminero et al. (2007)), provides an efficient selection of resources in a single administrative domain, taking into account the features of the jobs and resources, including the status of the network. However, only data associated with Grid jobs was considered – which is unlikely to be the case in practice. We improve upon this by considering a more realistic way of checking the status of the network. Also, we extend our previous proposal by means of considering autonomic computing (Caminero et al. (2009a;b)). This means that our model will use feedback from resources and network elements in order to improve system performance. Our scheduler will therefore adapt its behavior according to the status of the system, paying special attention to the status of the network.
Our scenario is depicted in Figure 2 and has the following entities (Caminero et al. (2009a;b)):

- **users**, each one has a number of jobs to run;
- **computing resources**, e.g. may consist of a single machine or clusters of machines;
- **routers**;
- **GNB** (*Grid Network Broker*), an autonomic network-aware scheduler;
- **GIS** (*Grid Information Service*), such as (Fitzgerald et al. (1997)), which keeps a list of available resources;
- **resource monitor** (for example, Ganglia (Massie et al. (2004))), which provides detailed information on the status of the resources;

- **BB** (*Bandwidth Broker*) such as (Sohail et al. (2003)), 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 jobs and deadlines.
- The GNB performs two operations for each job. First, it performs scheduling of that job to a computing resource, and second, performs connection admission control (CAC). The GNB uses link latencies to carry out the scheduling, and effective bandwidth of the network to perform the CAC. The information on the network status is collected by means of SNMP queries that the GNB sends to the computing resources, as Section 4.3 explains. This way, the GNB gets real information on the current status of the network. Once the GNB has decided the computing resource where a job will be executed, it carries on with the next scheduling request.
- 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. We assume the independence and autonomy of each administrative domain.
- Once the GNB has chosen a computing resource to run a job, it submits the job to that resource. On the completion of a job, the GNB will get the output sent back from the resource, and will forward it to the user. Also, the GNB will update information about the accuracy of its decisions, considering CPU and transmission delays.

The monitoring of the network and resources is carried out with a given frequency, called *monitoring interval*. In order to perform scheduling more efficiently, each monitoring interval is divided into a number of *subintervals*, each with an associated effective bandwidth. When the monitoring takes place, all the subintervals are updated with the effective bandwidth obtained from monitoring. As the GNB performs scheduling of jobs to computing resources, the effective bandwidth of each subinterval is updated independently, so that it reflects the jobs that are being submitted to resources. This way, GNB tries to infer the effect of the jobs being transmitted over the network. This process is depicted in Figure 3. In this figure, subintervals are numbered from 0 to 9, from left to right. We can see that when the monitoring took place, the effective bandwidth of all the subintervals was set to 100 Mbps. Then, *job* 0 (requiring 35 Mbps) is scheduled to be submitted to a resource in subinterval 1. Thus, the effective bandwidth of that subinterval is updated to 65 Mbps. Similarly, jobs 1, 2 and 3 are scheduled and submitted to a resource in different subintervals. As a result of that, subintervals 2, 3, 4, and 5 have their effective bandwidths updated, and set to 0 Mbps. The number of subintervals a job occupies is calculated based on the bandwidth required by the job, by subtracting the bandwidth of the job from the effective bandwidth of the subinterval where it is being submitted. If the new effective bandwidth of the subinterval is < 0.0, then it is set to 0.0 and the next subinterval is updated with the remaining of the bandwidth of the job. Thus, we update consecutive subintervals until the whole job is submitted to the resource.

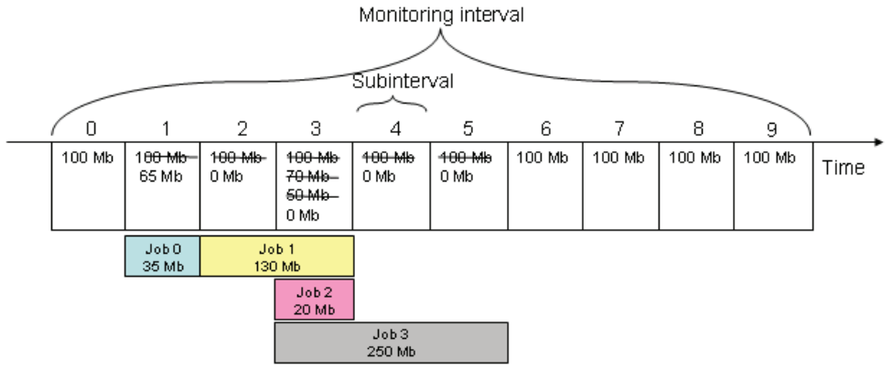


Fig. 3. Monitoring intervals and subintervals.

Algorithm 3 Scheduler algorithm.

1: Let u = a user (job owner)
2: Let R = set of computing resources
3: **for all** r_i in R **do**
4: $latency_{job}(u, r_i) = latency_{cpu(r_i)} * TOLERANCE^{r_i}_{cpu} + latency_{network(u,r_i)} * TOLERANCE^{r_i}_{net}$
5: increment(i)
6: **end for**
7: Rmin = Ordered set {min($latency_{job}(u, r_i)$)}, $\forall(i)$

4.1 Scheduler

The GNB performs scheduling for each job request on arrival. Algorithm 3 explains how the GNB performs the scheduling. It needs to predict the latency for a job in each computing resource (line 4). For this, it is necessary to consider the network latency between the GNB and the resource, as well as the CPU latency of the resource. The former is described in Section 4.3 and is based on the effective bandwidth of the network, whilst the estimation of the CPU latency is explained in Section 4.4. The resources are then sorted based on the expected latency of the job, from the resource with the smallest latency to the biggest one (line 7). The result is an sorted list of computing resources, from the best to the worst one.

The terms $TOLERANCE^{r_i}_x$, $x = \{net, cpu\}$, represent the accuracy of the previous predictions carried out by the GNB for the resource r_i. For $TOLERANCE^{r_i}_{net}$, for instance, we consider the last measurement for network latency, collected from the last job that came back to the GNB after being executed at resource r_i, and the network latency estimation for that job. Equations 3 and 4 show the actual formulas used, where MB represents the size of the job in mega-bytes, and MI represents the length of the job in millions of instructions.

$$TOLERANCE^{r'_i}_{net} = \frac{t^{real}_{net} - t^{estimated}_{net}}{MB} \tag{3}$$

$$TOLERANCE^{r'_i}_{cpu} = \frac{t^{real}_{cpu} - t^{estimated}_{cpu}}{MI} \tag{4}$$

This is, however, the current $TOLERANCE$ (the accuracy of the predictions for the last job that was executed). Therefore, we must use a predictive model to determine the future values for

Algorithm 4 CAC algorithm.

```
1: Let j be a job
2: Let j_bw be the bandwidth required for the job j
3: repeat
4:    for all r_i ∈ Rmin do
5:       if (resEffectiveBandwidth(r_i, j)! = OK) then
6:          discard(r_i)
7:       else
8:          choose(r_i)
9:       end if
10:   end for
11: until (choose(r_i)) or (i == sizeOf(Rmin))
```

these terms. A predictive model can be similar to the one used by Transport Control Protocol (TCP) for computing the retransmissions time-outs (Stevens (1994)), for instance. Hence, we can consider:

$$diff = TOLERANCE_x^{r_i'} - TOLERANCE_x^{r_i} \tag{5}$$

$$TOLERANCE_x^{r_i} = TOLERANCE_x^{r_i} + diff * \delta \tag{6}$$

where δ reflects the importance of the last sample in the calculation of the next $TOLERANCE$. GNB keeps a $TOLERANCE$ for each computing resource, for network and for CPU. Using one $TOLERANCE$ for pair $< user, computing\ resource >$ has also been considered, and discarded, since GNB would not "learn" from scheduling decisions already made for other users. By modifying $TOLERANCEs$, the GNB reacts to changes in the status of the system. Thus, the autonomic features of GNB lie on the $TOLERANCEs$.

4.2 Connection admission control (CAC)

Once the scheduler has sorted the available resources, the connection admission control (CAC) algorithm is activated. The overall algorithm is shown in Algorithm 4. It is necessary to estimate the effective bandwidth between two end points in the network – these being the GNB and the computing resource from which an estimate is desired. Thus, for each resource, we check the effective bandwidth of the path between the GNB and the resource (line 5). If the test returns OK, the resource is accepted for its execution (line 8). Otherwise, we check the next resource (line 6).

The effective bandwidth test is carried our according to Algorithm 5. First, we calculate the monitoring subinterval when scheduling is performed. Recall that GNB monitors computing resources every *monitoring interval*. This interval is divided into a number of subintervals, and each subinterval has an associated effective bandwidth. When the GNB performs the monitoring, the effective bandwidth of all subintervals is updated with the value obtained. As GNB schedules jobs to computing resources, the effective bandwidth of subintervals is updated separately.

If the effective bandwidth of the subinterval in which the job is to be scheduled is higher or equal to the bandwidth required by the job, then the effective bandwidth of the subinterval is updated (the job's bandwidth is subtracted from it) (line 12). If the effective bandwidth of the subinterval is lower than the bandwidth required by the job, then this subinterval's effective

Algorithm 5 Resources' Effective Bandwidth Test.

Let r be a computing resource
Let j be a job
Let j_{bw} be the bandwidth required for the job j
Let s_i be the subinterval of the moment when the job is to be scheduled
Let m be the monitoring interval
Let n be the number of subintervals m is divided in
while $(j_{bw} > 0)\&(i < n)$ **do**
  if $(effectiveBandwidth(s_i, r) >= j_{bw})$ **then**
    $effectiveBandwidth(s_i, r) = effectiveBandwidth(s_i, r) - j_{bw}$
    $j_{bw} = 0$
  else
    $effectiveBandwidth(s_i, r) = 0$
    $j_{bw} = j_{bw} - effectiveBandwidth(s_i, r)$
    $i = i + 1$
  end if
end while
if $j_{bw} = 0$ **then**
  return OK
else
  return NO_OK
end if

bandwidth is set to 0.0, and the next subintervals are also updated. This is undertaken because the job cannot be transmitted in one subinterval. We consider all the subintervals till the end of this monitoring interval, and in this moment the effective bandwidth is updated with the new data collected from the system.

4.3 Calculating the performance of the network.

A useful way to estimate the latency of a link would be sending ping messages with real data payloads (the size of the I/O files of the jobs, for instance), but this approach is not acceptable, because these ping messages would congest the network. An alternative approach would be for the GNB to send a query to all the computing resources it knows. This query asks for the number of transmitted bytes, for each interface that the query goes through (the *OutOctets* parameter of SNMP (McCloghrie & Rose (1991))). Using two consecutive measurements ($m1$ and m_2, m_1 shows X bytes, and m_2 shows Y bytes), considering the time of measurement (m_1 collected at time t_1 seconds and m_2 at t_2 seconds), and the capacity of the link C, we can calculate the effective bandwidth of a link l as follows:

$$eff\_bw(l) = C - \frac{Y - X}{t_2 - t_1} \tag{7}$$

Then, the effective bandwidth of a network path is calculated as the effective bandwidth of the bottleneck link. This procedure can be graphically seen in Figure 4. In this figure, `GNB` wants to know the effective bandwidth of the resource `Res`, so it sends a query to the resource. This query goes through routers `R0` and `R1`. Each router fills the query message with the number of *OutOctets* forwarded through that interface. For the first query (depicted as *monitoring round 1*), the interface `GNB-R0` has forwarded 134 *bytes* up to now, the interface `R0-R1` 1234 *bytes*,

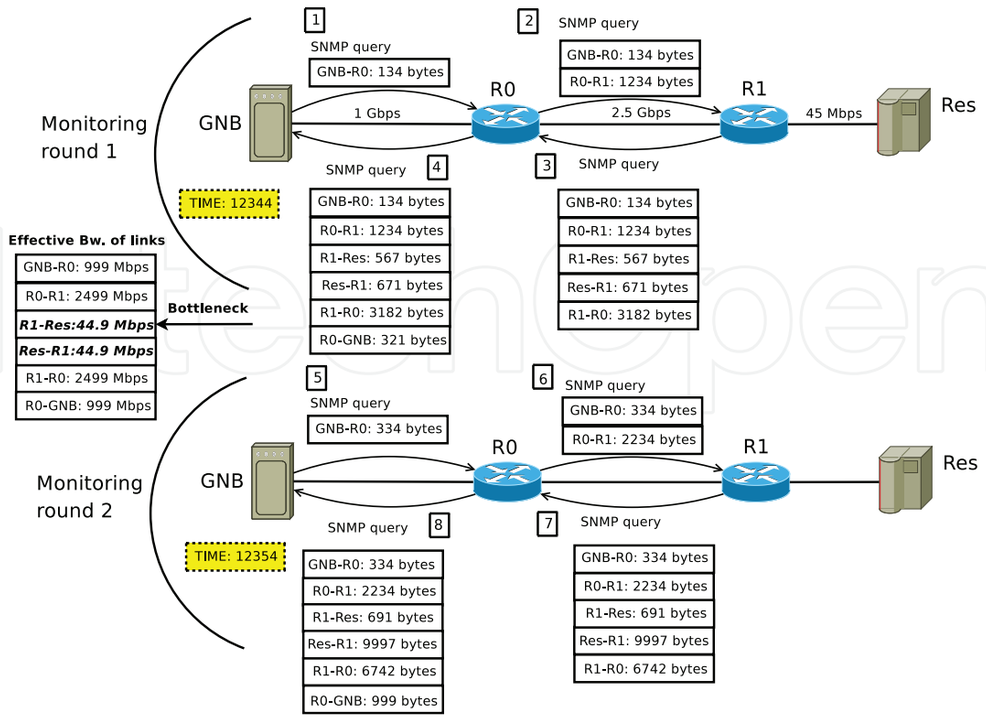


Fig. 4. Calculation of the effective bandwidth of a network path.

and so on. Once we have two measurements (which we have after *monitoring round 2*), GNB can calculate the effective bandwidth for each link in the path to the resource. Then, the effective bandwidth of the path to the resource is computed as the effective bandwidth of the bottleneck.

Since the GNB is a part of the infrastructure of an administrative domain, it is allowed to get such information using SNMP. Another possibility is that BB, which is in charge of the administrative domain and has direct access to routers, gets such information and presents it to the GNB. This way, independence and autonomy of the administrative domain is kept.

Once we have the effective bandwidth of a network path, we can calculate the latency of a job over a network path just by dividing the job's I/O file size in MB (mega bytes) by the effective bandwidth. These data (I/O file sizes) are known since I/O files are stored in the user's computer.

4.4 Calculating CPU latency

The CPU latency of a job is estimated as follows:

$$CPU\_lat \propto \left(\frac{jobs\_already\_submitted + 1}{cpu\_speed} \right) * current\_load \tag{8}$$

meaning that the CPU latency of a job in a computing resource is proportional to the jobs already assigned to the resource, the CPU speed and current load of the resource. $Jobs\_already\_submitted + 1$ is the number of jobs already submitted to that resource by the architecture in the current monitoring round (assuming all the jobs require only one CPU) plus the current job which is being checked, and cpu_speed and $current\_load$ are metrics obtained from a resource monitor. $Jobs\_already\_submitted$ refers to job submissions in the current

Location	Res. Name	# Nodes	CPU Rating	Policy	# Users
Loc_0	Res_0	41	49,000	Space-shared	35
Loc_1	Res_1	17	20,000	Space-shared	10
Loc_2	Res_2	2	3,000	Time-shared	5
Loc_3	Res_3	5	6,000	Space-shared	10
Loc_4	Res_4	67	80,000	Space-shared	35
Loc_5	Res_5	59	70,000	Space-shared	70
Loc_6	Res_6	12	14,000	Space-shared	40

Table 1. Resource specifications.

monitoring interval – the monitoring information available at the GNB does not contain them. Thus, these jobs must be considered for the scheduling of jobs in the next interval.
Current_load is an average load, for example $load\_fifteen$, $load\_five$ or $load\_one$ metrics measured by Ganglia. It is normalized to lie in the range $[0,1]$, meaning 0 totally idle, 1 totally busy. Also, a prediction on the next *current_load* can be calculated in a similar way to $TOLERANCE$s and effective bandwidth.

5. Experiments and results

The main contribution of this work is an comprohensive evaluation in which our proposal is compared with existing commercial meta-schedulers and heuristics found in literature, both network-aware and non-network-aware. This evaluation is presented in the current section.
A simulated model using GridSim Toolkit (Sulistio et al. (2008)) has been developed. This has been decided because simulations allow the creation of repeatable and controlled experiments. The experiments carried out have been divided in two groups. First, the autonomic network-aware meta-scheduling strategy (ANM) has been compared with existing Grid meta-scheduling strategies, which are neither network-aware nor autonomic. One of them is similar to the way how meta-scheduling is performed in the GridWay meta-scheduler – which, given a job submission, selects a computing resource at any given time based on the number of currently available CPUs. This has been chosen because it is a commercial meta-scheduler, which is released along with the Globus Toolkit from version GT4.0.5 onwards. The way how GridWay performs the meta-scheduling is presented in Section 2.1, and GridWay is labeled as `GW` in figures. The other heuristics are Max-Min, Min-min and XSufferage algorithms, explained in Section 2.3.
Second, ANM has been compared with the Gridbus strategy (presented in Section 2.2), which is network-aware. Thus, the way how ANM checks the status of the network and computing resources is compared with Gridbus.
For ANM, several details must be mentioned. As both the meta-scheduling algorithm and the CAC use the effective bandwidth of the network, a combined version of these has been utilized in the implementation. Hence, when calculating the estimated latency of a job in a computing resource, it is considered the effective bandwidth of the path from the broker to the resource. If the path does not have enough bandwidth, the estimated completion time will be infinity – thus the resource will not be chosen to run the job. Results focus on network and CPU latencies, makespan, and load balancing over resources, and illustrate that ANM outperforms the other meta-scheduling strategies.

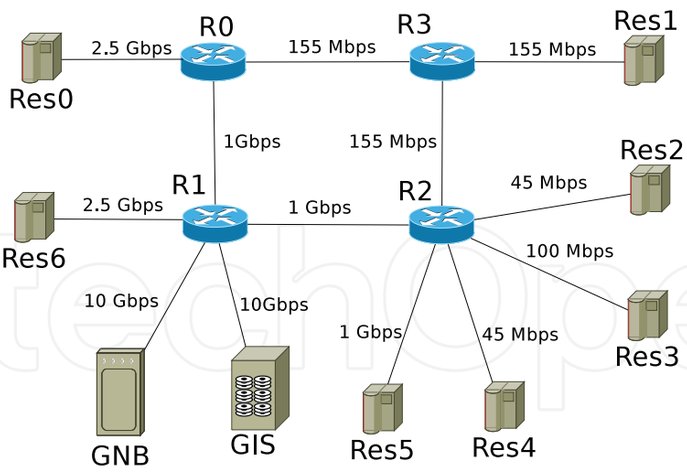


Fig. 5. Topology simulated.

Location Name	# Users
Loc_0	16
Loc_1	10
Loc_2	2
Loc_3	5
Loc_4	16
Loc_5	35
Loc_6	16

Table 2. Users' locations.

Figure 5 shows the topology used in experiments. In order to reduce the amount of memory and the length of our simulations, the bandwidth of the links appearing in the figure was scaled down by 0.1 %.
Table 1 summarizes the characteristics of the simulated resources, which were obtained from a real LCG testbed (LCG Computing Fabric Area (2010)). The parameters regarding to a CPU rating are 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 are scaled down by 10, for the same reasons mentioned before. Finally, each resource node has four CPUs.
Each computing resource also has local (non-Grid) computing load. Variable load follows the default local load provided by GridSim. This load is based on traces obtained from common observation of relative local usage behavior of resource per hour in a day (Buyya & Murshed (2002)). The local load is not the same for all the resources. Resources `Res_0`, `Res_5` and `Res_6` have a full local load that covers around 95 % of the computing power of the resources. That is, only around 5 % of the computing power of each CPU at those resources is available for Grid users. For the other resources, the local load is nearly 0 %. This has been decided in order to simulate a real Grid scenario, in which resources may have local load, that may differ between resources.
For these experiments, 100 users were created and distributed among the locations, as shown in Table 2. Each user executes three *bags-of-tasks* (Cirne et al. (2003)), with four jobs each one.

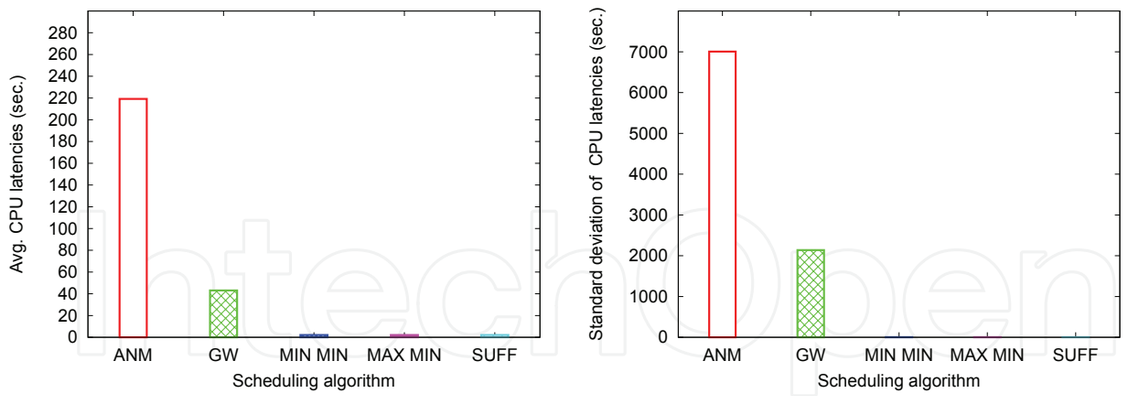
Job type	Power	Network
0	low (1,200,000 MI)	low (24 MB)
1	low (1,200,000 MI)	high (48 MB)
2	high (2,400,000 MI)	low (24 MB)
3	high (2,400,000 MI)	high (48 MB)

Table 3. Job types.

Users want to submit all the jobs in a bag at the same time, and the time difference between bags submission is 80,000 sec (approximately 1 day).
Each job in each bag has different characteristics. The characteristics are *heavy* and *low* processing requirements. Jobs with *heavy* processing requirements have 2,400,000 MI, which means that each job takes about 4 seconds if it is run on the `Loc_5` resource. On the other hand, jobs with *low* processing requirements have 1,400,000 MI, which means that each job takes about 2 seconds if it is run on the `Loc_5` resource. For the size of IO files, there are also *heavy* and *low* requirements. Jobs with *heavy* I/O requirements have I/O files whose sizes are 48 MB, and for jobs with *low* I/O requirements, I/O file sizes are 24 MB. Thus, there are four different job types, which are depicted in Table 3. Since these jobs do not require more than a few seconds of processing in the `Loc_5` resource, they are not CPU intensive, but I/O intensive. Features of jobs have been decided keeping in mind those from the ATLAS online monitoring and calibration system (ATLAS online monitoring and calibration system (2010)).
The GNB performs the monitoring of the network and computing resources with a 600 seconds (10 minutes) interval. This way, every 600 seconds the GNB gets information on the real status of the system, giving the load on the computing resources and the effective bandwidth of network paths. The GNB performs the meta-scheduling of jobs to computing resources as jobs arrive. At first, it was considered that the GNB would perform the meta-scheduling after a particular interval, i.e. jobs would be queued at the GNB, and from time to time, it would schedule the jobs received. However, this approach was considered to be inaccurate as it would synchronize jobs – i.e. jobs would be submitted to the corresponding computing resources at the same time, thus overloading the network. Another way of avoiding the synchronization would be submitting jobs to resources with a given delay, so that jobs scheduled at the same meta-scheduling round would not be submitted to the computing resource at the same time.
For the Max-Min, Min-min, XSufferage and Gridbus algorithms, the meta-scheduling interval is the same as the monitoring interval (600 seconds). Thus, at each monitoring interval they also perform the meta-scheduling of those jobs received during last 600 seconds.

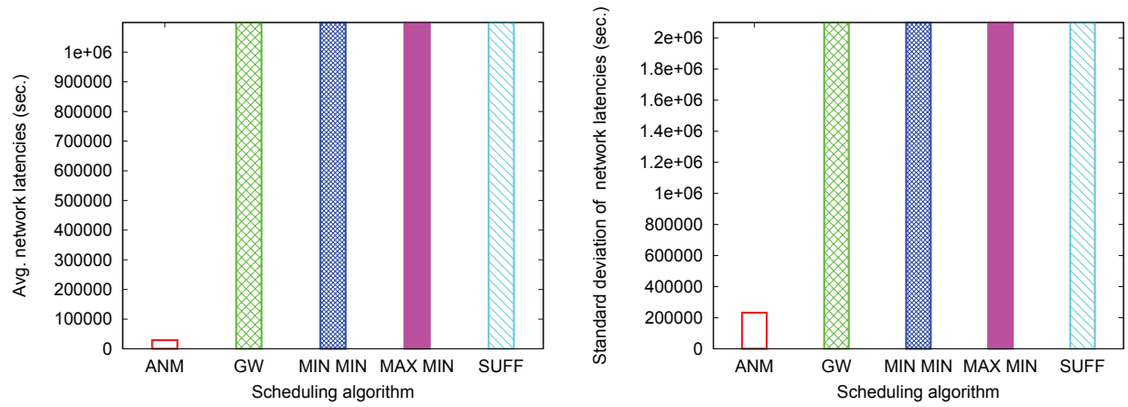
5.1 Network-awareness vs. Network-awareless

Figure 6 shows the average and standard deviation for the CPU latencies for all the jobs, for all the meta-scheduling policies studied. It can be seen that Min-min, Max-min and XSufferage outperform the other strategies, including ANM, in terms of average CPU latency (see Figure 6 (a)). This is because they only consider CPU when deciding to which computing resource a job should be submitted, and ANM performs the worst of all of them, followed by GridWay. With regard to the standard deviation (see Figure 6 (b)), ANM strategy shows bigger results than the other strategies. As before, Min-min, Max-min and XSufferage present the smallest standard deviation. This means that for ANM, CPU latencies are less stable – meaning that there is more difference between values. On the contrary, the CPU latencies of



(a) Average (b) Standard deviation

Fig. 6. Statistics on CPU latencies



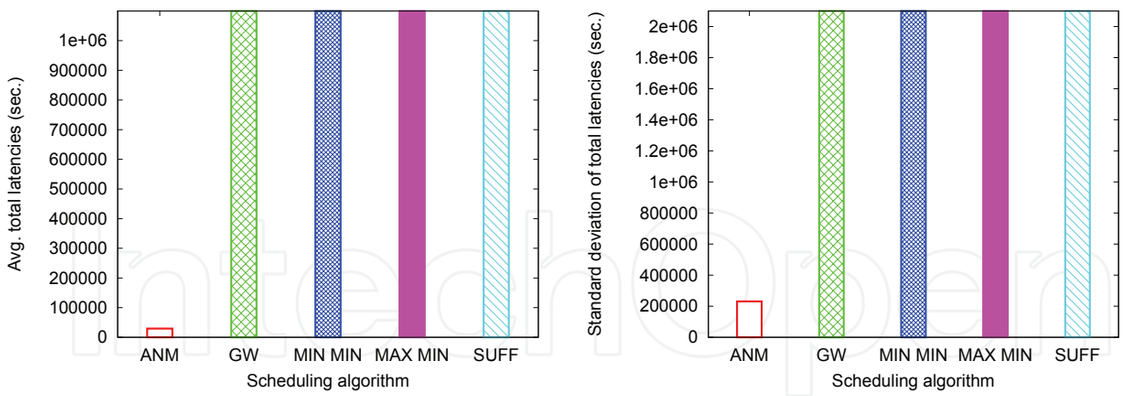
(a) Average (b) Standard deviation

Fig. 7. Statistics on network latencies.

jobs for Max-min (also for Min-min and XSufferage) are very similar, being less difference between them.

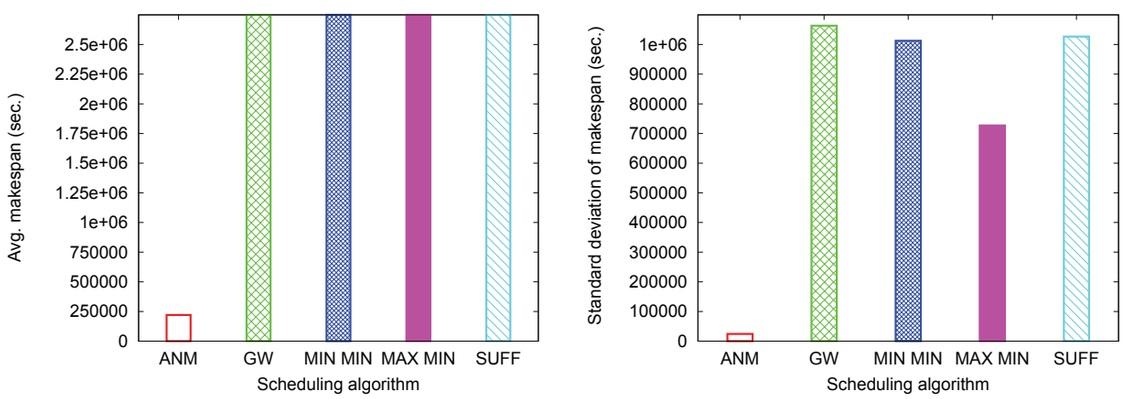
Figure 7 presents the average and standard deviation of network latencies. Here it can be seen that ANM clearly outperforms the others (since it presents lower average and standard deviation than the other strategies), because jobs are I/O intensive and the performance of the network is significant – which is not considered by the other strategies. So, not only average network latencies are smaller for ANM than for the other strategies studied, but also network latencies are more stable, since the standard deviation is also smaller. Total latencies (Figure 8) show similar tendencies.

Figure 9 presents the average and standard deviation of the makespan of users, i.e. the time users take to get all their jobs executed. As before, since the network is a key requirement for jobs, ANM outperforms the others. Regarding the average makespan (see Figure 9 (a)), GridWay, Min-min, Max-min, and XSufferage present results which are more than 10 times worse than ANM. Since ANM can have jobs executed with a lower latency (as explained above), jobs from different bags scarcely overlap each other in the network (i.e. when jobs from one bag are submitted, jobs from the previous bags have finished). On the contrary, when the others strategies are running, jobs overlap other jobs from the previous bags, thus



(a) Average (b) Standard deviation

Fig. 8. Statistics on total latencies (CPU + Network).



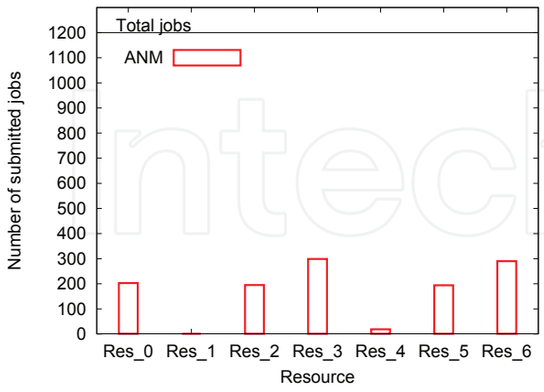
(a) Average (b) Standard deviation

Fig. 9. Statistics on users' makespan.

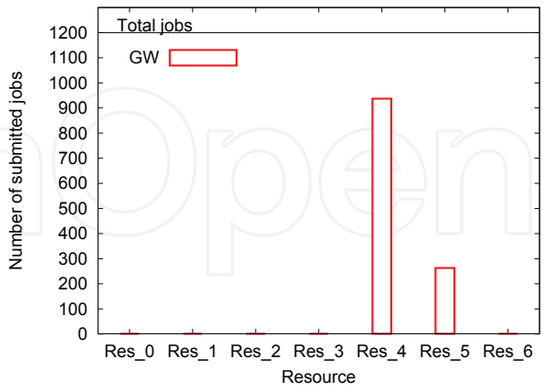
interfering each other and making the overall makespan of users higher. Also, ANM presents the smallest standard deviation, with a noticeable difference with the other strategies (see Figure 9 (b)). This means that ANM can perform more efficient meta-scheduling, as there are less variability among its results.

Figure 10 illustrates the number of jobs that have been submitted to each computing resource. Since GridWay only considers the number of idle CPUs when performing the meta-scheduling of jobs to computing resources, jobs are scheduled to the resource having more CPUs. Thus, resources `Res_0`, `Res_1`, `Res_2`, `Res_3`, and `Res_6` scarcely receive any job for execution. Only the computing resources which have more idle CPUs (namely, `Res_4` and `Res_5`) receive a noticeable amount of jobs.

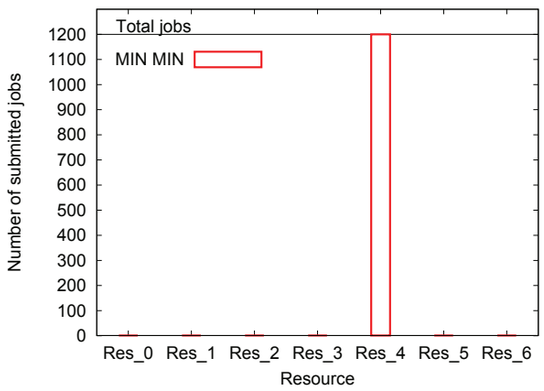
When Min-min, Max-min and XSufferage strategies are running, since they do not take the network into account when performing the meta-scheduling, they always choose the most powerful computing resource to run jobs (namely, `Res_4`). This leads to a bad overall performance, since this computing resource does not have a good network connection.



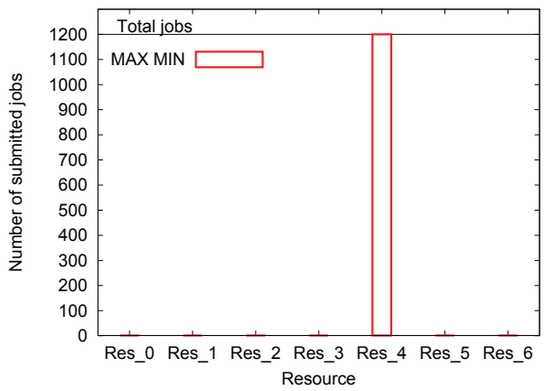
(a) ANM



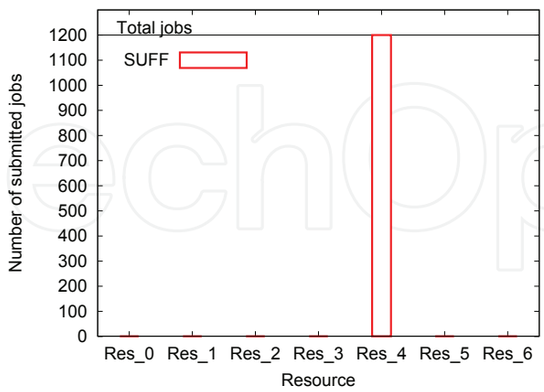
(b) GridWay



(c) Min-min

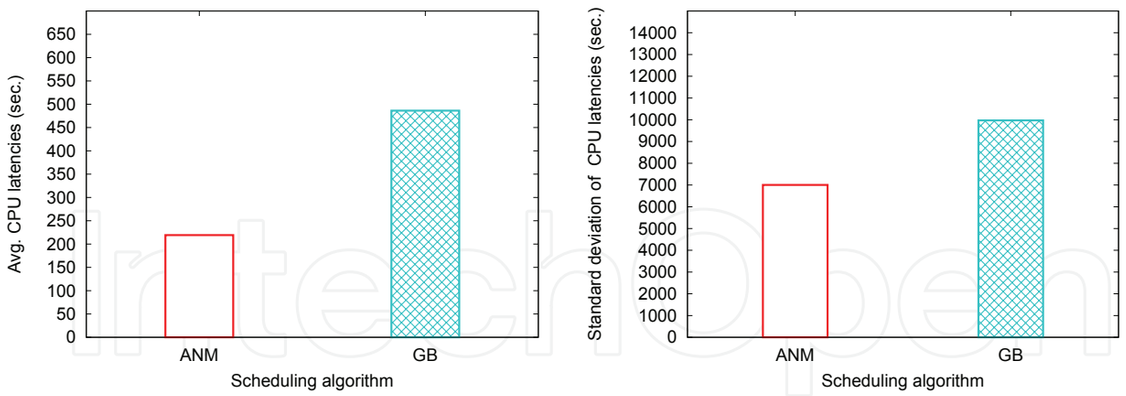


(d) Max-min



(e) XSufferage

Fig. 10. Computing resource selected by the meta-scheduling algorithm.



(a) Average (b) Standard deviation

Fig. 11. Statistics on CPU latencies.

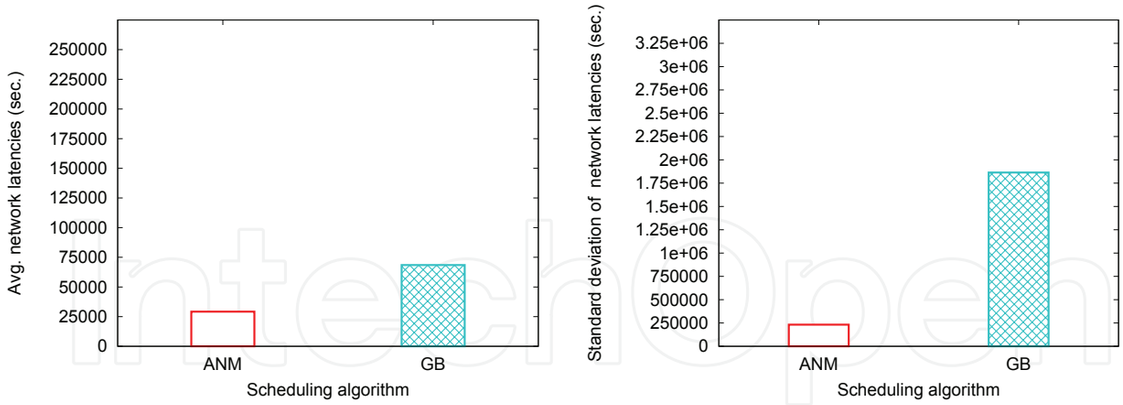
But when ANM is used, network connections are considered to perform the meta-scheduling. Thus, jobs are submitted to several resources, and all the resources receive a noticeable number of jobs. This way, network connections do not become overloaded, which heavily influences the performance received by jobs. Since jobs used in experiments are not CPU-intensive (they are I/O-intensive) this results in a better choice of resources. This leads to the better overall execution performance as previous figures shown. Besides, load is better balanced over the resources.

5.2 ANM vs. Gridbus

Now, an evaluation comparing ANM with another network-aware proposal for meta-scheduling in Grids is presented. The network-aware proposal chosen is Gridbus (Venugopal et al. (2008)), which was explained in Section 2.2. Results using Gridbus are labeled as `GB` in figures. The parameters of experiments presented are the same as the previous evaluation. Thus, results using ANM are the same.

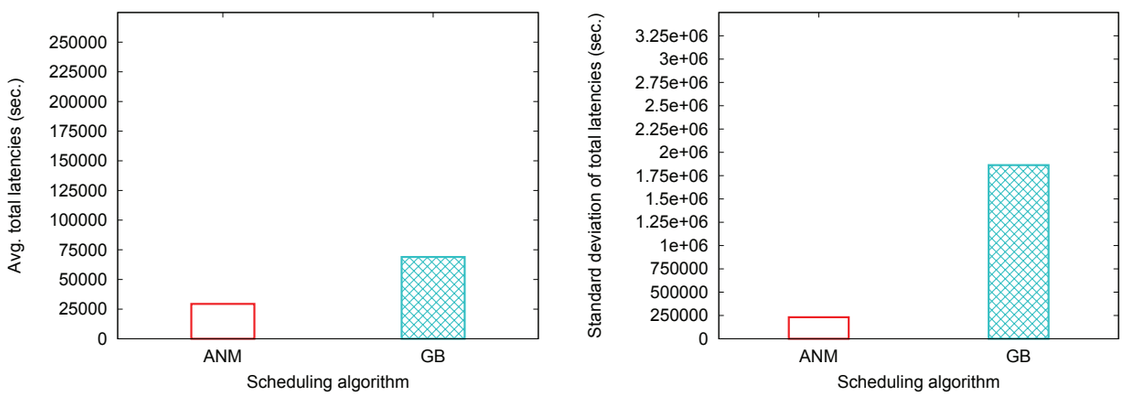
As before, the first results to be presented are regarding average and standard deviation of latencies of jobs. CPU, network and total latencies are presented. Figure 11 presents the average and the standard deviation of CPU latencies of jobs, for ANM and Gridbus. It can be seen that ANM outperforms Gridbus both in average and standard deviation. Thus, ANM makes better decisions with regard to CPU, since average latencies are lower, and CPU latencies are more stable (lower standard deviation).

Figure 12 presents average and standard deviation of network latencies. In this case, ANM also outperforms Gridbus since both average and standard deviation are lower. Similar tendencies also observed in the total latencies (shown in Figure 13) and makespan (shown in Figure 14). The reason for this better behavior of ANM compared with Gridbus lies on the computing resource chosen to execute each job, and this can be seen in Figure 15. Both ANM and Gridbus choose the computing resource to execute a job considering the network bandwidth, thus the resources that run more jobs are those whose bandwidth is the highest, namely `Res_6`, `Res_0`, `Res_5`, and `Res_3`. But the key difference between ANM and Gridbus is related to `Res_4`. This resource is the most powerful, since it has the most powerful CPUs, and it has more CPUs than the others. But the effective bandwidth between the GNB and it is the worst of all the computing resources.



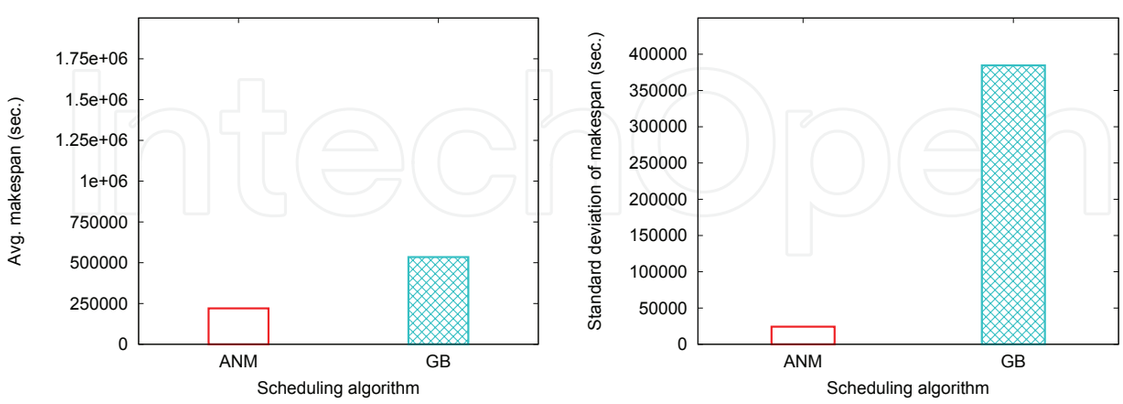
(a) Average (b) Standard deviation

Fig. 12. Statistics on network latencies.



(a) Average (b) Standard deviation

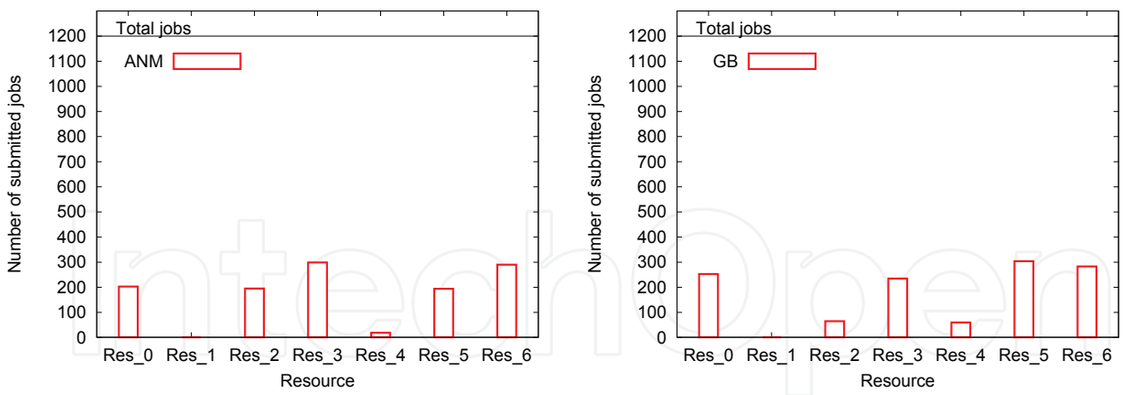
Fig. 13. Statistics on total latencies (CPU + Network).



(a) Average (b) Standard deviation

Fig. 14. Statistics on users' makespan.

Recall that for resources `Res_0`, `Res_5`, and `Res_6`, only 5 % of the computing power is available for Grid users, since they have local load covering 95 % of their computing power.



(a) ANM (b) Gridbus

Fig. 15. Computing resource selected by the meta-scheduling algorithm.

On the other hand, `Res_1`, `Res_2`, `Res_3`, and `Res_4` have almost no local load, so 100 % of computing power is available for Grid users. As explained before, when performing the meta-scheduling of a job, Gridbus starts checking the resource with the highest available bandwidth. If the job limit for it has not been reached, the resource is chosen to execute the job. Thus, the job limit for Gridbus is key. It is calculated by considering the resource share available for Grid users. Thus, since some computing resources have a heavy local load, this greatly influences the job limit for those resources. But, since jobs are not CPU intensive, the CPU load is not an important restriction for a computing resource. Because of this, loaded resources may reach their job limit, and jobs may be submitted to computing resources with lower available bandwidth.

As opposed to it, ANM can predict the performance of jobs at each resource more accurately. Thus, less jobs are submitted to resource `Res_4` (which is the most powerful, with no local load, but also has the worst network connection). Thus, jobs are submitted to other resources such as `Res_2` and `Res_3`, which provide a better performance to jobs.

6. Conclusions

The network, as the communication media for Grid applications, is a critical resource to be considered by the Grid management architecture. In this chapter, algorithms aimed at performing connection admission control (CAC) and meta-scheduling of jobs to computing resources within one single administrative domain (*intra-domain* meta-scheduling) have been presented.

This paper presents a comprehensive performance evaluation of an *autonomic network-aware meta-scheduler* (*ANM*), that combines concepts from Grid meta-scheduling with autonomic computing, in order to provide users with a more adaptive job management system. The architecture involves consideration of the status of the network when reacting to changes in the system – taking into account the workload on computing resources and the network links when making a meta-scheduling decision. Thus, the architecture provides meta-scheduling of jobs to computing resources and connection admission control, so that the network does not become overloaded.

Performance evaluation shows that ANM can schedule heterogeneous jobs onto computing resources more efficiently than existing literature approaches (namely, Min-min, Max-min, and XSufferage) and conventional meta-scheduling algorithms (that used by GridWay). Besides, it has been compared with a network-aware meta-scheduler, Gridbus. Results show that ANM can make better meta-scheduling decisions, resulting in a better performance from the users' point of view when compared with other meta-schedulers.

Regarding future work, authors are considering the implications of the technique presented in this paper with regard to different contexts, such being Cloud computing. In this case, there should be a tolerance for the time needed to deploy a virtual machine, along with the aforementioned tolerances for the network and CPU time of tasks.

Besides, a combination of Grid and Cloud resources (as in (Kim et al. (2009))) could be used. In this case, depending on the tolerances of each resource, jobs should be mapped to Grid or Cloud resources in order to minimize their completion times.

7. References

ATLAS online monitoring and calibration system (2010). Web page at `http://dissemination.interactive-grid.eu/applications/HEP`.

Bobroff, N., Fong, L., Kalayci, S., Liu, Y., Martinez, J. C., Rodero, I., Sadjadi, S. M. & Villegas, D. (2008). Enabling interoperability among meta-schedulers, *Proc. of the 8th Intl. Symposium on Cluster Computing and the Grid (CCGRID)*, Lyon, France.

Buyya, R. & Murshed, M. (2002). Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing, *Concurrency and Computation: Practice and Experience* 14: 1175–1220.

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

Caminero, A., Rana, O., Caminero, B. & Carrión, C. (2009a). Performance evaluation of an autonomic network-aware metascheduler for Grids, *Concurrency and Computation: Practice and Experience* 21(13): 1692–1708.

Caminero, A., Rana, O. F., Caminero, B. & Carrión, C. (2009b). A performance evaluation of network-aware grid meta-schedulers, *Intl. Conference on Parallel Processing Workshops (ICPPW)*, Vienna, Austria.

Casanova, H., Legrand, A., Zagorodnov, D. & Berman, F. (2000). Heuristics for scheduling parameter sweep applications in Grid environments, *Proc. of the 9th Heterogeneous Computing Workshop (HCW)*, Cancun, Mexico.

Cirne, W., Brasileiro, F., SauvÃl', J., Andrade, N., Paranhos, D., Santos-Neto, E., Medeiros, R. & Silva, F. (2003). Grid computing for bag-of-tasks applications, *Proc. of the 3rd Conference on E-Commerce, E-Business and E-Government*,, São Paulo, Brazil.

Czajkowski, K., Kesselman, C., Fitzgerald, S. & Foster, I. T. (2001). Grid information services for distributed resource sharing, *Proc. of 10th Intl. Symposium on High Performance Distributed Computing (HPDC)*, San Francisco, USA.

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

Foster, I. & Kesselman, C. (1997). Globus: A metacomputing infrastructure toolkit, *International Journal of Supercomputer Applications and High Performance Computing* 11(2): 115–128.

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

Freund, R. F., Gherrity, M., Ambrosius, S. L., Campbell, M., Halderman, M., Hensgen, D. A., Keith, E. G., Kidd, T., Kussow, M., Lima, J. D., Mirabile, F., Moore, L., Rust, B. & Siegel, H. J. (1998). Scheduling resources in multi-user, heterogeneous, computing environments with SmartNet, *Proc. of the 7th Heterogeneous Computing Workshop (HCW)*, Orlando, USA.

Gentzsch, W. (2001). Sun Grid Engine: Towards creating a compute power Grid, *Proc. of the First Intl. Symposium on Cluster Computing and the Grid (CCGrid)*, Brisbane, Australia.

Globus Projects (2010). Web page at `http://dev.globus.org/wiki/Guidelines`.

GridPP, Real Time Monitor (2010). Web page at `http://gridportal.hep.ph.ic.ac.uk/rtm/`.

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

Imamagic, E., Radic, B. & Dobrenic, D. (2005). CRO-GRID: Grid execution management system, *Proc. of 27th Intl. Conference on Information Technology Interfaces (ITI)*, Cavtat, Croatia.

Kim, H., el Khamra, Y., Jha, S. & Parashar, M. (2009). An autonomic approach to integrated HPC Grid and Cloud usage, *Proc. of the 5th Intl. Conference on e-Science (e-Science)*, Oxford, UK.

LCG Computing Fabric Area (2010). Web page at `http://lcg-computing-fabric.web.cern.ch`.

LCG (LHC Computing Grid) Project (2010). Web page at `http://lcg.web.cern.ch/LCG`.

Litzkow, M. J., Livny, M. & Mutka, M. W. (1988). Condor - A Hunter of idle workstations, *Proc. of the 8th Intl. Conference on Distributed Computer Systems (ICDCS)*, San Jose, USA.

Luther, A., Buyya, R., Ranjan, R. & Venugopal, S. (2005). Alchemi: A .NET-based enterprise Grid computing system, *Proc. of The 2005 Intl. Conference on Internet Computing (ICOMP)*, Las Vegas, USA.

Marchese, F. T. & Brajkovska, N. (2007). Fostering asynchronous collaborative visualization, *Proc. of the 11th Intl. Conference on Information Visualization*, Washington DC, USA.

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

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

McCloghrie, K. & Rose, M. T. (1991). Management information base for network management of TCP/IP-based internets:MIB-II, RFC 1213.

Metascheduling Requirements Analysis Team (2006). Final report, available at `http://www.teragridforum.org/mediawiki/images/b/b4/Metasched-RatReport.pdf`.

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

Sohail, S., Pham, K. B., Nguyen, R. & Jha, S. (2003). Bandwidth broker implementation: Circa-complete and integrable, *Technical report*, School of Computer Science and Engineering, The University of New South Wales.

Stevens, W. R. (1994). *TCP/IP Illustrated: The Protocols*, Addison-Wesley.

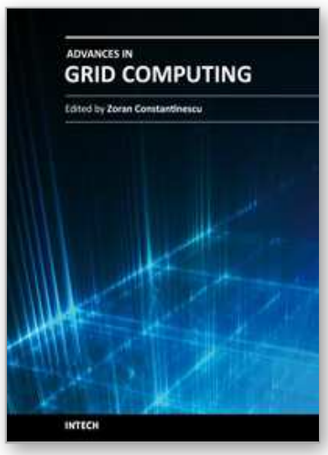
Sulistio, A., Cibej, U., Venugopal, S., Robic, B. & Buyya, R. (2008). A toolkit for modelling and simulating Data Grids: An extension to GridSim, *Concurrency and Computation: Practice and Experience* 20(13): 1591–1609.

Tomás, L., Caminero, A., Caminero, B. & Carrión, C. (2010). Using network information to perform meta-scheduling in advance in grids, *Proc. of the Euro-Par Conference.*, Ischia, Italy.

Vázquez, C., Huedo, E., Montero, R. S. & Llorente, I. M. (2010). Federation of TeraGrid, EGEE and OSG infrastructures through a metascheduler, *Future Generation Computer Systems* 26(7): 979–985.

Venugopal, S., Buyya, R. & Winton, L. J. (2006). A grid service broker for scheduling e-science applications on global data Grids, *Concurrency and Computation: Practice and Experience* 18(6): 685–699.

Venugopal, S., Nadiminti, K., Gibbins, H. & Buyya, R. (2008). Designing a resource broker for heterogeneous Grids, *Software: Practice and Experience* 38(8): 793–825.



Advances in Grid Computing

Edited by Dr. Zoran Constantinescu

ISBN 978-953-307-301-9
Hard cover, 272 pages
Publisher InTech
Published online 28, February, 2011
Published in print edition February, 2011

This book approaches the grid computing with a perspective on the latest achievements in the field, providing an insight into the current research trends and advances, and presenting a large range of innovative research papers. The topics covered in this book include resource and data management, grid architectures and development, and grid-enabled applications. New ideas employing heuristic methods from swarm intelligence or genetic algorithm and quantum encryption are considered in order to explain two main aspects of grid computing: resource management and data management. The book addresses also some aspects of grid computing that regard architecture and development, and includes a diverse range of applications for grid computing, including possible human grid computing system, simulation of the fusion reaction, ubiquitous healthcare service provisioning and complex water systems.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen