Performance-driven Coordinated Grid Scheduling

Ivan Rodero, Francesc Guim and Julita Corbalan

Computer Architecture Department, Technical University of Catalonia (UPC)
Jordi Girona 1-3, Modul D6, 08034 Barcelona, Spain
{irodero, fguim, juli}@ac.upc.edu

Abstract

Grid computing has emerged as a way to share geographically and organizationally distributed resources that may belong to different institutions or administrative domains. In this context, the scheduling and resource management is usually performed by a grid resource broker. The scheduling task consists of distributing the jobs among the different centers resources and the need to coordinate the grid with the underlying scheduling levels which have already been identified. However, there is still a lack of policies for this approach. In this paper we describe and evaluate our coordinated grid scheduling strategy. We take as a reference the FCFS job scheduling policy and the matchmaking approach for the resource selection. We also present a new job scheduling policy based on backfilling (JR-backfilling) that aims to improve the workloads execution performance, avoiding starvation and the SLOW-coordinated resource selection policy that considers the average bounded slowdown of the resources as the main parameter to perform the resource selection. From our evaluation, based on trace-driven simulations of real grid systems, we state that our proposed coordinated strategy can substantially improve the workloads execution performance as well as the resource utilization.

I. Introduction

In recent years a lot of effort has been devoted to the research of grid scheduling and resource management. These scenarios are usually composed of different computing resources owned by one or more different centers or organizations. Thus, the users and high level schedulers can access different resources which are potentially heterogeneous: from a single PC to a High Performance Computing (HPC) system with a complex scheduling system, through standard middleware services. In this context, the scheduling task consists of distributing the jobs among the different resources. Since the scheduling is performed on the top of the local schedulers, this component is usually called meta-scheduler. At this layer, the scheduling algorithms may be more sophisticated than in local HPC systems because they must take into account new requirements as well as looking for the best trade-off between satisfying the user requirements and global system performance. Furthermore, a meta-scheduler usually has no ownership or control over the local resources and the local systems may have their own local policies that can conflict with the grid scheduling strategy. In particular, there are conflicting performance goals between the users and the resource owners. While the users are focused on optimizing the performance of a single application for a specified cost goal, the resource owners are targeted on obtaining the best system throughput or minimizing the response time.

A grid system may contain a set of distributed resources that may belong to different virtual organizations (VOs). It may also contain different kinds of resources (supercomputers, cluster, PCs, etc.) with their specific security and management mechanisms. Usually a grid resource broker is in charge of the management of jobs and resources (job submission and monitoring, resource discovery and monitoring, resource selection, etc.). Grid resource brokers collect resource information from grid information services such as Globus MDS. To access the resources, grid resource brokers can use different middleware software. Moreover, the resources may be managed by a Local Resource Management System (LRMS) with their own local policies that must be respected. Figure 1 depicts a general grid scheduling scenario. To solve these problems in [24] we presented the eNANOS architecture that considers all the scheduling layers that can be involved in a job execution: from the CPU scheduling to a grid system. This aims at providing a good low level support to improve the scheduling at higher levels.

In literature there are several grid scheduling strategies such as those based on economic issues [1], adaptive scheduling [15], multi-criteria [9], co-allocation [20], or application-centric scheduling [3]. However, although some works have discussed the need for coordinating the scheduling layers and the required attributes for coordinating the different scheduling entities [26], there is still a lack of policies for this approach.

In this paper we present and evaluate our grid scheduling strategy based on the coordination between the grid and cluster scheduling layers to improve the job execution by having good acknowledgement of the job behavior in both layers. In particular, we target enhancing the applications execution performance in terms of execution time, slowdown and response time, and the resource utilization. While the most common scheduling strategies in grid systems select the resources to dispatch a job depending on the job requirements and resources characteristics (for example free CPU slots), we propose to the use of dynamic performance information to select the most appropriate resources and to delegate the responsibility for the low level scheduling and allocation to the underlying scheduling systems.

We studied the grid scheduling and resource selection strategies through simulation tools and workloads based on real grid traces. In particular, we have extended the Alvio simulation framework for grid systems. Our evaluations compare the most common grid scheduling approach that selects the job and resources to be executed using matchmaking algorithms with our

This research has been supported by the Spanish Ministry of Science and Technology under contract TIN2007-60625.

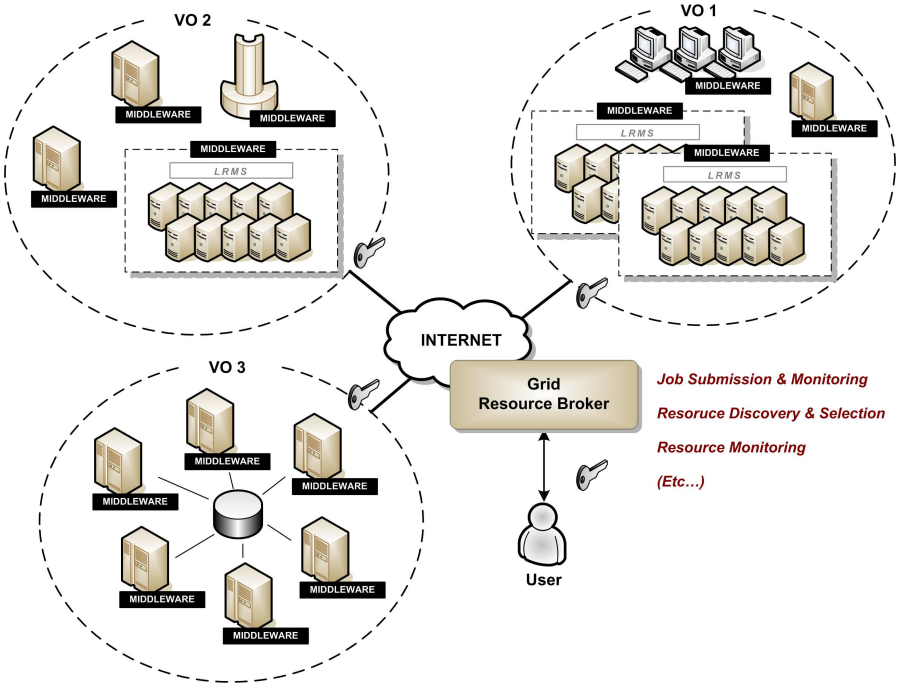


Fig. 1: General grid scenario

approach that performs the selection by considering information that comes from the underlying scheduling levels. We take as a reference the First Come First Serve (FCFS) job scheduling policy with a matchmaking-based resource selection policy. We present an aggressive variant of backfilling (*JR-backfilling*) that aims at improving the applications execution performance thereby avoiding starvation. Finally, we also present the *SLOW-coordinated* policy that filters the resources matching the most important requirements at grid level, and uses performance information from the local schedulers (the average bounded slowdown of the finished jobs) to select the resources to submit a job. The comparison of these approaches provides clear results and supports the argument that coordinating the grid and cluster scheduling layers can improve the workloads execution as well as the resource utilization.

The rest of the paper is organized as follows. Section II presents the related work. Section III presents our scheduling strategy and describes our proposed policies. Section IV presents the evaluation methodology. Section V presents the results obtained and, finally, section VI discusses the conclusions and some future projects.

II. Related Work

The scheduling problem consists of, given a set of jobs with their characteristics and requirements, a set of resources, and the system status, deciding which jobs to start executing and in which resources. In the literature there are several job scheduling policies, such as the FCFS approach [27] and its variants, for example those proposed by Feitelson et al. [6] and by Schwiegelshohn et al. [28], being the most popular policies that do not use estimated application information. Other policies that use estimated applications information (for example the execution time) have been widely studied such as Smallest Job First (SJF) [19], Largest Job First (LJF) [33], Smallest Cumulative Demand First (SCDF) [17] [29] or Backfilling [18] [30] [21] which is the most used in HPC systems. Backfilling applies a FCFS order, but it allows the advancing of small jobs in the queue if they do not delay the execution of previously queued jobs.

In the field of grid computing several resource selection policies have also been proposed, the matchmaking approach [32] being the most popular one. Other specific techniques have also been developed, such as those based on economic issues proposed by Buyya et al. [1], those based on adaptive scheduling proposed by Llorente et al. [15], those based on multi-criteria [9], those based on co-allocation proposed by Epema et al. [2] [4] or those based on advanced reservations [7] [31].

Other proposals have been developed. For example, Guim et al. [13] presented how using data mining prediction techniques (applied to historical information) can substantially improve the performance of the global distributed infrastructure by the *"grid backfilling"* meta-scheduling policy. In addition, similar to a global backfilling approach, Sabin et al. [25] presented the scheduling of parallel jobs in a heterogeneous multi-site environment. They propose carrying out a global scheduling within a set of different sites using a global meta-scheduler with two different resource selection algorithms.

With reference to the coordination of the grid scheduling systems with the underlying layers some works have discussed the necessity of coordinating the scheduling layers. For example, Schwiegelshohn and Yahyapour [26] study the required attributes to coordinate the different scheduling entities. The Open Grid Forum (OGF) that has emerged as a standardization organization, has addressed this problem within its Grid Scheduling Architecture Research Group (GSA-RG) [22]. This group has proposed a grid scheduling architecture based on the study of several use cases and has studied the required scheduling attributes. However, there is still a lack of policies for on this approach, without considering estimated application information.

III. The Scheduling Strategy

In this section we present three different scheduling approaches. The first one is a grid scheduling configuration that is based on the First Come First Serve (FCFS) policy for job scheduling, and based on the matchmaking approach for resource selection. This configuration is one of the most used in grid scheduling and, moreover, this is the configuration that we have exploited most within the eNANOS grid resource broker. Therefore, we take this configuration as a reference to compare the new grid job scheduling (*JR-backfilling*) and resource selection (*SLOW-coordinated*) policies that we also describe in this section.

A. The reference configuration

As mentioned previously, we have implemented our grid resource management approach within the eNANOS grid resource broker. This performs the main job scheduling based on the FCFS policy and the resource selection based on the matchmaking approach. Since our brokering system allows the user to indicate estimated job information (such as the estimated execution time), we also implemented other policies using this information. However, we want to research new grid scheduling policies which consider neither estimations nor predictions concerning job execution because users may be unaware of such parameters. Moreover, if users provide wrong job execution estimations they can improve the planning of their applications but may penalize significantly the performance of the systems scheduling.

Therefore, we will take the FCFS policy as a reference scheduling configuration to compare with our grid scheduling approach. Since the FCFS policy executes the jobs in the same order of arrival in the system, it becomes inefficient with the matchmaking approach when there are not enough resources to allocate a given job. In this case, the rest of queued jobs (that potentially may be allocated immediately) remain blocked until there are free resources to allocate it. This is why different variants and improvements of the FCFS policy have been proposed such as backfilling. We will try to solve this problem of the FCFS policy with the *JR-backfilling* policy which is an aggressive variant of backfilling.

Concerning resource selection, our strategy is based on the Condor matchmaking approach [32] that has been extensively used in many projects, including Condor-G [8] which is the grid version of Condor. As was discussed in [32] *matchmaking* tries to optimize both user interests (planning) and systems performance (scheduling). Matchmaking requires some steps: jobs and resources advertise their characteristics and requirements in classified advertisements (ClassAds), scans the known ClassAds and creates pair that satisfy each other's constraints and selects the best resources to execute the job.

The ClassAd language is a flexible and expressive framework for matching resources requests (i.e., jobs) with resource offers (i.e., machines). It allows users to define custom attributes for both resources and job through a set of uniquely named expressions. Each named expression is called an attribute and each attribute has a name and a value. ClassAds are schema-free and do have not any predefined allocation policy. On the other side, the resources publish information to matchmaker. Figure 2 shows the matchmaking architecture.

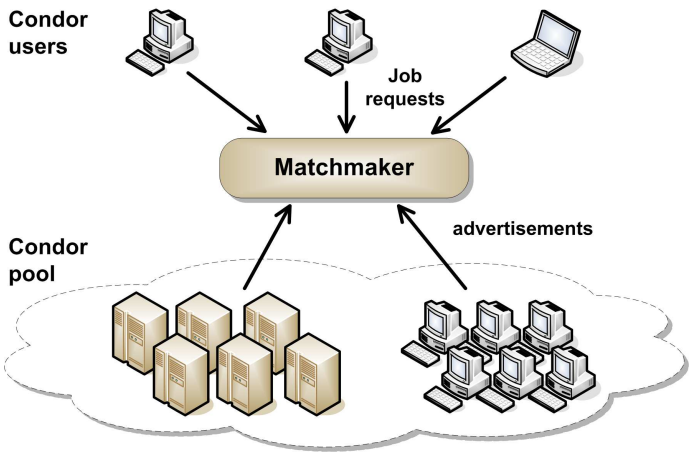


Fig. 2: Matchmaking architecture

The matchmaker assigns significance to the *Requirements* and *Rank* attributes. *Requirements* indicates a constraint and *Rank* measures the desirability of a match. Thus, the matchmaking algorithm matches the requirements of job requests with available resources and, then, obtains a rank value that is used to choose from among compatible matches. Listing 1 shows an example of a Condor job ClassAd.

```
[
  MyType = "Job"
  TargetType = "Machine"
  Requirements =
    ((other.Arch == "INTEL" && other.OpSys == "LINUX") &&
    other.Disk > my.DiskUsage &&
    other.LoadAvg < 50)
  Rank = (Memory * 100) + (MFlops * 25)
  Cmd = "/home/irodero/appls/appl_exe"
  DiskUsage = 6000
]
```

Listing 1: Condor job ClassAd example

Our resource selection approach uses a set of hard user criterion (resource requirements) and soft ones (recommendations) with a predefined set of priorities to compute the rank values. It then selects the resource with the higher rank value to submit the selected job as matchmaking does. The set of priorities for the different attributes are defined from the feedback of previous experiments using different input parameters and system configurations. In the ClassAd example shown in listing 1 the memory is multiplied by a factor of 10,000 to compute the rank value. The priorities that we use in eNANOS have the same function as this factor. More details of the eNANOS broker job description and resource requirements language can be found in [23]. Listing 2 shows the same requirements of listing 1 but expressed in our requirement description language.

```
<?xml version="1.0" encoding="UTF-8"?>
<CRITERIA>
  <Memory-Processor>
    <Attribute Name="RAMAvailable"  Operator=">=" Value="100"   Importance="SOFT" Priority="100"/>
    <Attribute Name="LoadLast15Min" Operator="<=" Value="50"    Importance="HARD" Priority="1"/>
    <Attribute Name="Vendor"        Operator="==" Value="INTEL" Importance="HARD" Priority="1"/>
  </Memory-Processor>
  <FileSystem-OperatingSystem>
    <Attribute Name="DiskAvailable" Operator=">=" Value="6000"  Importance="HARD" Priority="1"/>
    <Attribute Name="OS Name"       Operator="==" Value="LINUX" Importance="HARD" Priority="1"/>
  </FileSystem-OperatingSystem>
  <Others>
    <Attribute Name="Middleware"    Operator="==" Value="GT3"   Importance="HARD" Priority="1"/>
    <Attribute Name="MFlops"        Operator=">=" Value="1"     Importance="SOFT" Priority="25"/>
  </Others>
</CRITERIA>
```

Listing 2: eNANOS hard and soft resource requirements example

We will take the regular matchmaking that we perform in eNANOS as a reference resource selection policy to compare with the resource selection strategy that we propose in this paper. In particular, we propose to use dynamic performance information from the underlying scheduling levels (*SLOW-coordinated* policy) to improve the efficiency of the resource selection in terms of workload execution time and resource utilization.

B. The JR-backfilling policy

The ***JR-backfilling*** policy is a job scheduling policy based on backfilling. As commented previously, backfilling follows a FCFS policy but allows advancing jobs in the queue if they do not delay the execution of previously queued jobs. Therefore, backfilling usually advances short or small jobs. Moreover, the decision to advance jobs is performed from estimated job information provided by the user or by predictions. As a result, the main problem of backfilling is that it requires the estimated job execution time to advance jobs because when a backfilled job delays the execution of previously queued jobs, it must be killed. Since we are not considering any estimated job information we can not apply a regular backfilling policy.

The idea of *JR-backfilling* is to advance only small jobs, but the problem is how to define "small" jobs. We do not consider estimated job information but we have the resource requirements of each job that must be provided by the user side. Our

strategy consists on using the resource requirements of the jobs to determine which are "small" jobs in order to advance their execution. In particular, we compute **J**ob **R**ank values based on these requirements. The *JR-backfilling* policy is evaluated following three main steps:

1) Filters jobs that match their resource requirements with the available resources.
2) Computes the rank value of each from its resource requirements. If job resource requirements were not matched in the previous step, its rank value is -1.
3) Selects the job with the lowest positive rank value.

Figure 3 shows a simple scenario of the *JR-backfilling* policy evaluation. It shows two resources with their current status and a job queue with the resource requirements of the queued jobs. It also shows the priorities of the different resource requirements attributes. We can appreciate that *job 1* does not match its resource requirements with any resource. Consequently, it is not considered in the following steps. The rest of the jobs match their resource requirements and, after computing the rank values for each job, the algorithm selects *job 20* because it has the lowest rank value (requires fewer CPUs than *job 3*). In this scenario, *job 20*, which is the last in the queue, will be executed first.

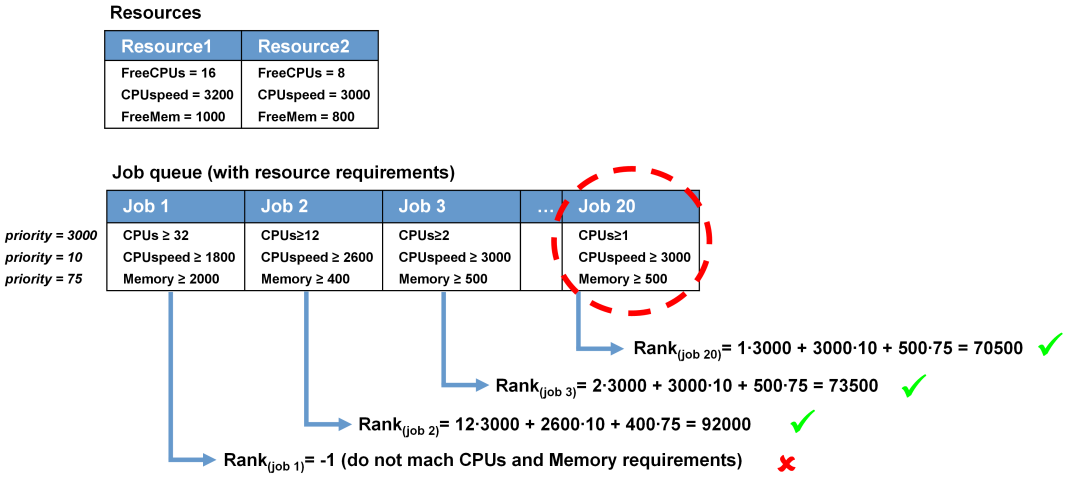


Fig. 3: JR-backfilling policy simple evaluation example

The main objectives of the *JR-backfilling* policy are to minimize the workload execution time, job waiting time, job response time, average bounded slowdown and to maximize the resource utilization. However, the main problem of advancing jobs without considering estimated job information is that it can lead to starvation. For example, in figure 3 the scheduling policy selects *job 20* to be executed because it requires fewer CPUs than *job 3*. However, *job 20* execution may take a long time and it also may delay the execution of other applications that were queued before. In this scenario, the job waiting time of several applications would increase and, consequently, their response time and the average bounded slowdown would increase as well. In this situation, the obtained performance may be even worse than with a regular FCFS policy.

Therefore, starvation can penalize the performance significantly because it increases job waiting time and, as a result, job response time and average slowdown. Since our objectives are to minimize these metrics, the JR-backfilling policy also aims to avoid starvation. To solve this problem we apply a **reduction factor** to each job rank value obtained from the matchmaking process. The reduction factor is higher for the last queued jobs to promote the jobs at the head of the queue. Therefore, when a job can not be scheduled and another one is advanced, if the reduction factor is applied, it will not penalize significantly the performance of the previously queued jobs. The reduction factor is obtained from previous experiments. We chose the reduction factor that provided the best performance results on set of workloads and system configurations. Therefore, another important objective of the *JR-backfilling* policy is to find the best tradeoff between advancing jobs and not penalizing queued jobs.

Figure 4 shows the same scenario as figure 3 but applies a reduction factor to the rank value of jobs. The reduction factor penalizes 10% multiplied by the position of each job in the queue. After applying this reduction factor *job 20* rank value becomes very high because it is at the end of the queue and *job 3* now has the lowest rank value. Consequently, *job 3* is selected.

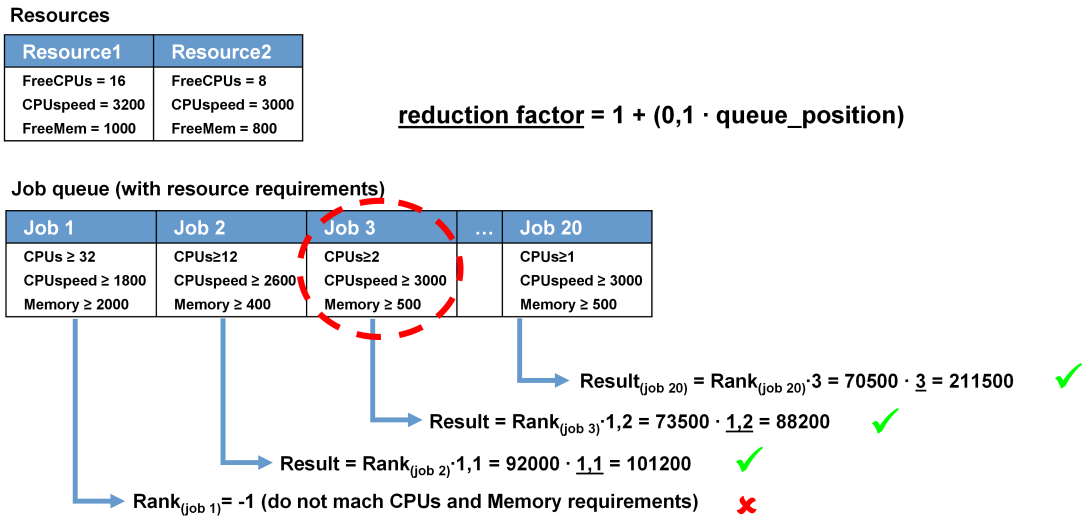


Fig. 4: JR-backfilling policy simple evaluation example with reduction factor

The pseudo-code of the JR-backfilling policy is shown in listing 3 but it does not include loops and memory usage optimizations for simplicity. The input parameters are a queue of jobs and a set of resource requirements for each of the queued jobs. It returns a job from the queue following the *JR-backfilling* scheduling policy, or "NULL" in the case that any job can be dispatched. **ComputeJobRank** matches the given resource requirements to the available resources and computes the rank value depending on these resource requirements, the resource information, and the attributes priorities. If any resource match the resource requirements it returns -1. **ReductionFactor** is the fixed reduction factor that was described previously.

```
FUNCTION: JR-backfilling
IN:
  JOBS = {job_1,...,job_n}
  ∀ job ∈ JOBS, REQS(job) = {req_1,...,req_m}
  RESOURCES = {res_1,...,res_n}

OUT:
  JOB

BEGIN:

  selected = -1
  minRank = -1
  if ( ComputeJobRank(REQS(job_1), RESOURCES) != -1 ){
    RETURN (job_1)
  }
  else{
    for ( i = 2; i ≤ n; i++ ){
      tmpRank = ComputeJobRank(REQS(job_i), RESOURCES)
      tmpRank = tmpRank * i * ReductionFactor
      if ( tmpRank≥0 && (tmpRank<minRank || minRank==-1) ){
        minRank = tmpRank
        selected = i
      }
    }
    if ( selected != -1 ){
      RETURN (job_selected)
    }
  }
RETURN (NULL)
```

Listing 3: JR-backfilling policy pseudo-code

C. The SLOW-coordinated policy

The ***SLOW-coordinated*** policy is a resource selection policy. Although it uses the matchmaking approach in part, it differs significantly from the regular matchmaking because the novelty of this policy is that it takes dynamic performance information from the underlying scheduling layer as the main parameter to select the appropriate resources to submit a job. In fact, the main objective of the *SLOW-coordinated* policy is to maximize the resource utilization. As a dynamic performance metric we use the *average bounded slowdown of finished jobs*. The bounded slowdown (BSLD) of a job is defined as:

$$BSLD_{job} = max\left(1,\ \frac{runtime_{job} + waittime_{job}}{max(runtime_{job},\ threshold)}\right)$$

Although a threshold of 10 seconds is used in many works in the context of parallel job scheduling to limit the influence of very short jobs on the average bounded slowdown, we define a threshold of 60 seconds, because in grid scenarios jobs may take longer. Therefore, we define the average bounded slowdown of a resource finished jobs as:

$$AVG\ BSLD_{resource} = \frac{\sum_{i=1}^{finished\_jobs} BSLD(job_i)}{\#finished\_jobs}$$

Listing 4 shows an example of these performance metrics computations for a resource with three finished jobs. Although the waiting time of job_1 is higher than the waiting time of job_2, since the proportion of waiting time respect the runtime is higher for job_2, the bounded slowdown of job_1 is better than the bounded slowdown of job_2. Job_3 obtains a very high regular slowdown (SLD) because the execution time is very short compared to the other jobs. However, the bounded slowdown is significantly shorter because of the threshold of 60 seconds.

runtime(job_1) = 7000 seconds, waittime(job_1) = 500 seconds
runtime(job_2) = 4000 seconds, waittime(job_2) = 400 seconds
runtime(job_3) = 30 seconds, waittime(job_3) = 500 seconds

$BSLD_{job_1} = \frac{7000+500}{7000}$ = **1.07**

$BSLD_{job_2} = \frac{4000+400}{4000}$ = **1.10**

$SLD_{job_3} = \frac{30+500}{30}$ = 17.67 $BSLD_{job_3} = \frac{30+500}{60}$ = **8.83**

$AVG\ BSLD_{resource} = \frac{1.07+1.10+8.83}{3}$ = **3.66**

Listing 4: Resource performance metrics computation example

The best slowdown value that a job can obtain is 1 when the job waiting time is 0. Therefore, the average bounded slowdown of a resource must be minimized to improve its performance.

The *SLOW-coordinated* policy is evaluated following three main steps:

1) Filters the resources that match the main resource requirements like matchmaking. We consider the main resource requirements to be requirements of static attributes, such as the total number of CPUs or the total amount of RAM memory. Dynamic attributes, such as the CPU load or the amount of free memory, are not considered in the matching algorithm.
2) Collects the *average bounded slowdown of finished jobs* for each of the previously selected resources.
3) Selects the resource with the lowest average bounded slowdown because the resources with lower slowdown perform better.

Figure 5 shows a simple scenario of the *SLOW-coordinated* policy evaluation. It shows that the user resource requirements include having at least 4 CPUs to execute its job. We can appreciate that two of the resources (the first and the fourth one) do not match the resource requirements because they have fewer than 4 CPUs. After collecting the average bounded slowdown of the finished job of the resources that match the resource requirements ($\geq$4 CPUs), the grid resource broker selects the resource with the lowest average bounded slowdown (the third resource with BSLD=4). Although the fourth resource has a smaller average bounded slowdown than the selected resource it is not considered because it does not match the resource requirements.

Therefore, the resource selection with SLOW-coordinated is based on the coordination with the local scheduling systems. It is important to emphasize that it will submit a job to the resource with lower average bounded slowdown even though other resources may have higher regular matchmaking rank values (obtained from other resource requirements or recommendations).

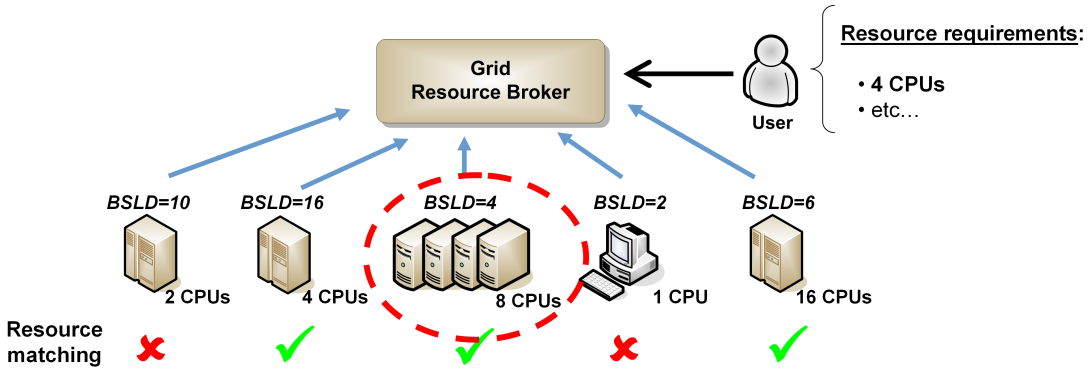


Fig. 5: SLOW-coordinated policy simple evaluation example

The main effect of this strategy on the system performance should be to balance the performance among the resources to obtain similar average slowdown values in the different resources. The summarized pseudo-code of this policy is described in listing 5. The input parameters are a job with its requirements and a set of resources. It returns the resource to dispatch the given job following the *SLOW-coordinated* resource selection policy. If any resource matches the resource requirements it returns NULL. **MatchRequirements** matches the static attributes of the given resource requirements to the available resources. If the requirements are matched it returns "true". Otherwise it returns "false". **AVG_BSLD** returns the average bounded slowdown of the finished jobs of the given resource.

```
FUNCTION: SLOW-coordinated
IN:
  JOB
  REQS(JOB) = {req_1,...,req_m}
  RESOURCES = {res_1,...,res_n}

OUT:
  RESOURCE

BEGIN:

  selectedResource = NULL
  minBSLD = -1
  for ( i = 1; i ≤ n; i++ ){
    if ( MatchRequirements(REQS(JOB), res_i) == true ){
      if ( AVG_BSLD(res_i) < minBSLD || minBSLD == -1 ){
        minBSLD = AVG_BSLD(res_i)
        selectedResource = res_i
      }
    }
  }
RETURN (selectedResource)
```

Listing 5: SLOW-coordinated policy pseudo-code

IV. Evaluation Methodology

We have used simulation mechanisms for our policy evaluations. These simulations allow us to research policies for large and complex configurations with numerous jobs and high demand of resources and to easily include modifications and refinements in the policies.

A. The Alvio simulation framework

The Alvio Simulator [11] is a C++ event driven simulator that has been designed and developed to evaluate scheduling policies in HPC architectures. It supports evaluation of schedulers in a large range of facilities from local centers to interoperable grid environments simultaneously. It allows research on job scheduling strategies in very different scenarios that may be composed of different VOs. It has been designed in order to provide an easy mechanism to extend its functionalities. Thus, extending this

simulator with our models (i.e., adding new scheduling strategies or new resource models) required only a reasonable amount of effort in terms of development and design.

1) Local systems model: The simulator models different components which interact in local and distributed architectures. Conceptually, it is divided into three main parts: the *Simulator Engine*, the *Scheduling Polices* and the *Computational Resource Model* (CRM). A simulation of a local scheduling scenario allows us to simulate a given policy with a given architecture. Currently, the following local policies have been modeled:

- Local Resource Selection Policies (RSP): First Fit, First Continuous Fit, and Less Consume policies.
- Local Job Scheduling Polices (LJSP): the First Come First Serve policy, the backfilling policy, and finally, the Resource Usage Aware backfilling (RUA-Backfilling [12]). For backfilling policies, the different properties of the wait queue and backfilling queue are modeled (Sort Job First, LXWF and First Come First Serve) and different numbers of reservations can also be specified.

Like other simulators, given a workload and an architecture definition, Alvio is able to simulate how the jobs would be scheduled using a specific job scheduling policy (such as First Come First Serve, or the backfilling policies). The main contribution of this simulator at this level is that it not only allows the modeling of the jobflow in the system, but also the simulation of different resource allocation policies. To do this, it uses a reservation table that models how each job is allocated to each node of the architecture. As a result, the researcher is able to validate how different combinations of scheduling policies and resource selection policies impact on the performance of the system.

The other new capability of this simulator is the ability to model the local resource usage on the jobs that are running in the system. For each job, the researcher can specify the different fields that are specified in the Feitelson Standard Workload Format (SWF) [5]. However, at the local level, in addition to the SWF fields, for each job, the user can specify the memory, ethernet and network bandwidths. Consequently, depending on the configuration of the simulation, the impact of considering the penalty introduced in the job runtime due to resource sharing can be evaluated, as it implements a job runtime model and resource model that try to estimate the penalty introduced in the job runtime when sharing resources.

2) Multi-site systems model: The simulator allows the local scenario to be extended by having several instances of this scenario. As depicted in figure 6, different machines (e.g., clusters, single box servers, etc.) can be specified (including their architecture definition, local job scheduling policy, and local resource selection policy) and the different meta-scheduling policies that can be specified to schedule the jobs. When the simulation starts, all the different layers of the model are instantiated, from the local reservation tables (which model how the jobs are mapped to the processors) to the brokering component that manages the jobs submitted to the system.

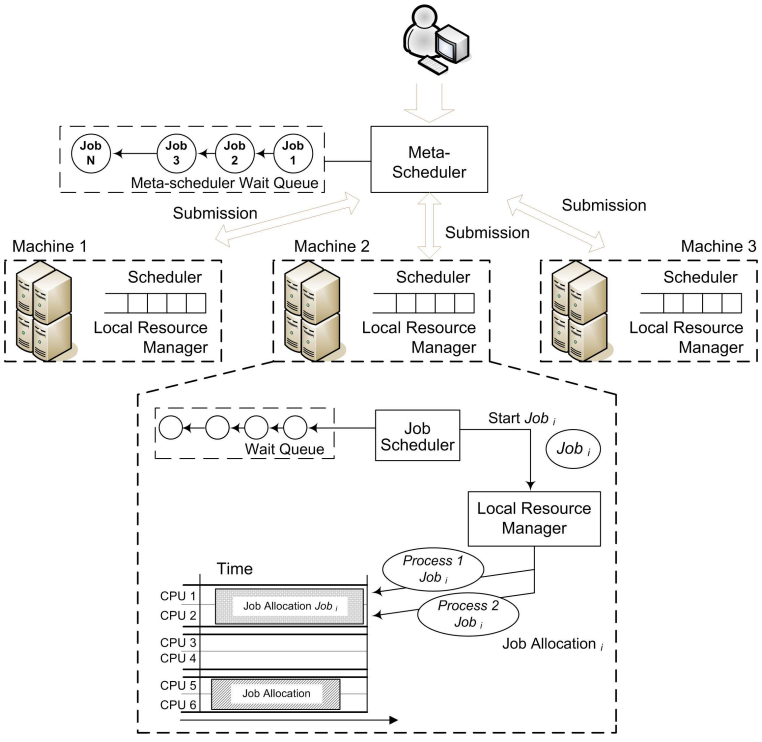


Fig. 6: Brokering system

In grid scenarios, a grid resource broker usually requires the specification of the job requirements from the user. In many cases, the meta-scheduling policies use these requirements to carry out the matchmaking with the local resources. To allow this, we have extended the Standard Workload Format (SWF) to specify the job requirements in the workload to be simulated. Each requirement is composed of an identifier (e.g., *Vendor*), an operator (e.g., *EQUAL*) and a value (e.g., *Intel*). In the current version of the simulator, the following requirements can be specified for each grid job: memory in MB (e.g., *1024 MB*), processor vendor (e.g., *Intel*, *AMD*), processor clock speed in MHZ (e.g., *1200 MHZ*), operating system (e.g., *Linux*, *AIX*), number of processors (e.g., *4 processors*) and disk size in MB (e.g., *1000 MB*). Concerning the meta-scheduling policies, two different kinds of policies can be used:

- Multi-site scheduling policies: the non-centralized ISIS-Dispatcher policy [10]. In this scenario, jobs are scheduled by their own dispatcher and there are no centralized scheduling decisions.
- Brokering scheduling policies: First Come First Serve, RealTime, Earliest Deadline First, or Job Rank (JR)-backfilling policies for the job scheduling, and resource selection based on the matchmaking approach.

B. The workloads

In our evaluation, we have used traces of the DAS-2, Grid5000, and Sharcnet systems from the Grid Workloads Archive [16][14]. We have selected two weeks of job submissions for each workload trace. Thus, we have avoided the initial warm up period of the systems (around the first 50,000 jobs) to skip the unrepresentative data. The selected trace fragments are sized with 11,318 jobs for DAS-2, 12,719 jobs for Grid5000, and 13,283 for Sharcnet.

We have analyzed the traces in order to select representative fragments. We have manually reduced the inter arrivals times of the jobs. Increasing the stream of jobs allows us to increase the pressure on the system incrementing the load. Moreover, we have adjusted the execution times and we have limited the memory and CPU demand (up to 512 CPUs) to scale the experiments according to a reduced scenario which simplifies the analysis of the results.

Figure 7 shows the distribution of the workloads job arrivals. The DAS-2 distribution is approximately linear and the Sharcnet one is by chunks. In the latter one, we have pairs of intervals where many jobs arrive and where there are no job arrivals. In the figure, we can also find two different Grid5000 workloads. The one as GRID5000_OUTLIER is an unrepresentative portion of the trace because the majority of jobs arrive at the first 30% of the workload duration and the rest of time there are only a few job arrivals. In fact, the simulation with the outlier trace gave us irregular and incoherent results compared to the other workloads. However, the other Grid5000 workload follows a more uniform distribution and it will be used in our experiments. More details regarding these traces can be found in [16].

In our experiments, we have defined three different domain types, one for each workload system: *"DOM_small"* for DAS-2, *"DOM_medium"* for Grid5000, and *"DOM_big"* for Sharcnet. The resources of each domain are based on real testbeds in terms of number of clusters, CPU architecture, and OS. Moreover, they are scaled in terms of memory and disk demand. For the DAS-2 system, we have modeled 6 resources with a total of 400 CPUs, 12 resources with a total of 985 CPUs for the Grid5000 system, and 12 resources with a total of 3,712 CPUs for the Sharcnet system. However, to simplify the experiments, we have chosen a subset of the CPU available architectures (*Intel*, *AMD*, and *PowerPC*) and Operating Systems (*Linux*, *AIX*, and *Solaris*). While DAS-2 is a homogeneous multi-cluster (only has *Intel* and *Linux*), the other two systems are quite heterogeneous in term of number of CPUs, architectures and OS (for example, Grid5000 has 50% *Intel*, 40% *AMD* and 10% *PowerPC*, and 60% *Linux*, 30% *AIX* and 10% *Solaris*).

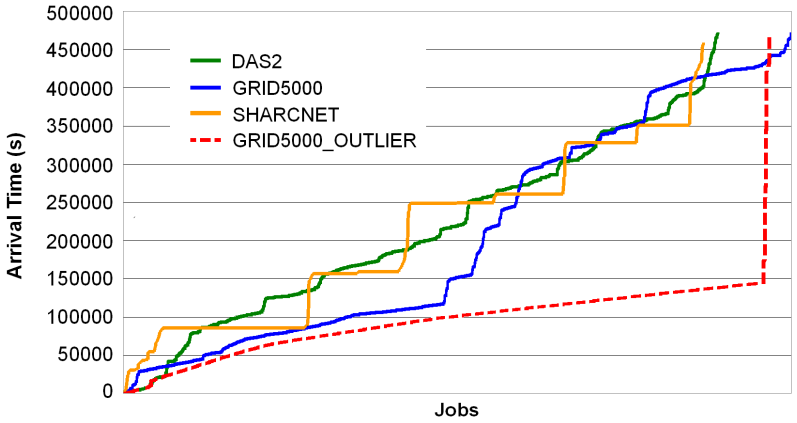


Fig. 7: Workloads job arrivals

The original traces do not include resource requirements for each job. To fix this, we have generated a requirements trace per workload to be used by the simulator as explained in the previous sub-section. To generate these requirements, we have used a perl script that defines the jobs requirements based on the original trace, the resources characteristics, and a combination of input parameters. In particular, we have used the CPU and memory demand, and for the disk utilization we have used a combination of the job duration with the CPU and memory usage, with a randomized factor. For the remaining attributes we have used percentages for each CPU architecture and OS type, applying a random distribution by bursts (sized from 3 to 6).

C. Metrics

We use the following metrics for evaluating our strategies:

- Total workload execution time
- Average job waiting time
- Average bounded slowdown (BSLD). We define BSLD for a given job:

$$BSLD_{job} = max\left(1, \frac{runtime_{job} + waittime_{job}}{max(runtime_{job},\ threshold)}\right), with\ a\ threshold\ of\ 60\ seconds$$

- Average CPUs and nodes utilization

The units for the first two metrics are seconds and hours, for average CPUs and nodes utilization are percentages, and the slowdown has no units. In our experiments, we try to minimize all the metrics except the average CPU and nodes utilization that should be maximized.

V. Results

In this section we present the results obtained using three different broker system configurations:

- ***FCFS***: this is the reference configuration that we use to compare with our proposed policies to evaluate their performance. It uses the *FCFS* job scheduling policy and the regular matchmaking approach to perform the resource selection.
- ***RANK***: it uses the *JR-backfilling* job scheduling policy and the previously described matchmaking-based approach for the resource selection. Thereby, it will help us to evaluate the influence of the job scheduling policy on the performance metrics and resource utilization.
- ***SLOW***: it uses the *JR-backfilling* job scheduling policy and the *SLOW-coordinated* resource selection policy. Thus, it will help us to evaluate the influence of the resource selection policy as well as the combination of our proposed approaches.

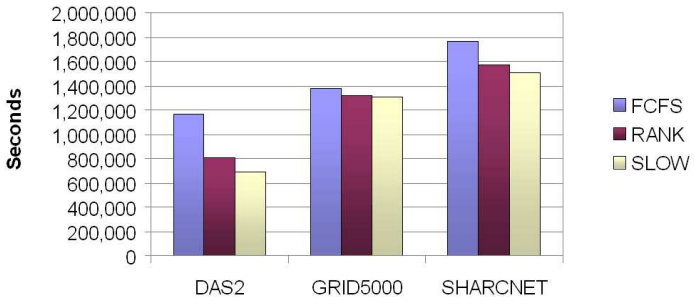
Figure 8 shows the workloads and jobs performance results with the three different policies. In particular, it shows the workloads execution time, and the 95^{th} percentile of the average job waiting time and average bounded slowdown. The 95^{th} percentile is a mathematical calculation widely used to judge scheduling performance metrics. A percentile is, in a cumulative frequency distribution, one of the 99 values of a variable that divide its distribution into 100 parts of equal frequency. In practice, only certain of the percentiles are used such as median (or 50^{th} percentile), the lower and the upper quartiles (respectively, the 25^{th} and 75^{th} percentiles), or the 95^{th} percentile that cuts off the upper 5% of a frequency distribution. The reason this statistic is so useful in measuring performance metrics, is that it gives an accurate picture of these metrics results and helps filter out outlier values.

In figure 8a we can appreciate that the execution times of the different workloads follow a similar pattern with the three policies. In general, with the *FCFS* configuration the workloads execution times are the longest ones, with the *RANK* configuration the execution times are shorter (around 10% on average) and with the *SLOW* configuration the execution times are the shortest (6.6% on average with respect to *RANK* and around 20% on average with respect to *FCFS*). In the case of the DAS-2 workload, the execution time difference between the different policies is very significant (up to 40% between *FCFS* and *SLOW* policies). With the Sharcnet workload the execution times with the different configurations follow a similar pattern. With the Grid5000 workload the workload execution time difference is not very significant.

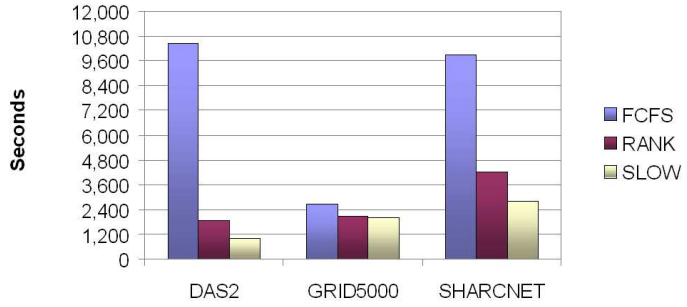
From our experiments we have also stated that, although the execution time of workloads are limited by the job arrival times, inefficient scheduling (such as what the regular *FCFS* policy does) penalizes significantly the workloads execution time because the jobs remain queued for a longer time.

In the presented results we have not considered the matchmaking algorithms processing time. Since our proposed policies only consider the static attributes of the resource requirements, the processing time for selecting a job and matching the appropriate resources may be lower. However, the number of resource requirements attributes is not very numerous and, therefore, the processing time of the different algorithms should not be significantly different.

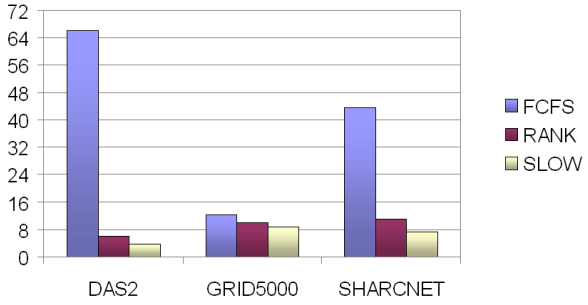
Figure 8b shows the average job waiting times. The waiting times and the workload execution times follow a similar pattern with the different policies. However, the waiting times present a higher difference between the *FCFS* and *RANK* policies, especially with the DAS-2 workload (5.5 times shorter with *RANK* with respect to *FCFS*) and with the Sharcnet workload.



(a) Workload total execution time



(b) Average job waiting time



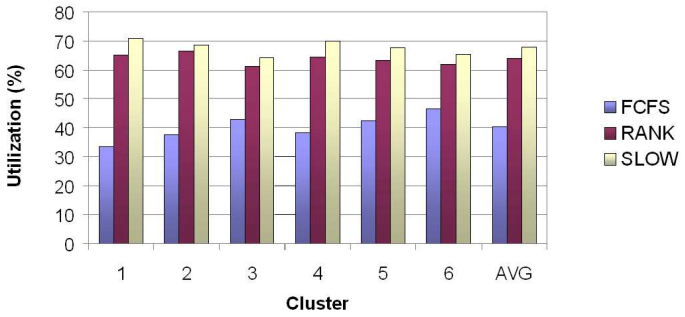
(c) Average bounded slowdown

Fig. 8: Workloads and job performance results

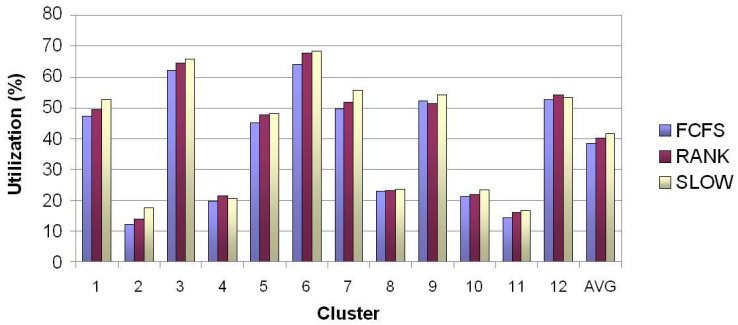
Figure 8c shows the average job bounded slowdown. It is worth noting that, since the average job waiting times (and consequently the job response time) and bounded slowdown are linked, they follow very similar patterns. Moreover, we can appreciate that the bounded slowdown difference between the *FCFS* and the other policies is higher (around 8%) than the other metrics difference. From the performance metrics results we can appreciate a clear behavior: *RANK* has better results than *FCFS*, and *SLOW* has better results than *RANK*.

In figure 9 we focus on the resource utilization. It shows the CPU utilization of each cluster of the systems and the average values with the different configurations. The different configurations follow a similar pattern with the three different systems: most clusters have better results with the *RANK* configuration than with the *FCFS* policy, and with the *SLOW* configuration they have better results than with the *RANK* configuration. The average of the clusters resource utilization also follows this pattern. However, in some clusters the results are very similar with the three different configurations (such as clusters 4 and 12 of the Grid5000 system). Moreover, in some clusters the *RANK* and *SLOW* configurations have similar results while the *FCFS* configuration the utilization is lower (such as cluster 5 of Grid5000 system or clusters 2, 5 and 12 of Sharcnet system).

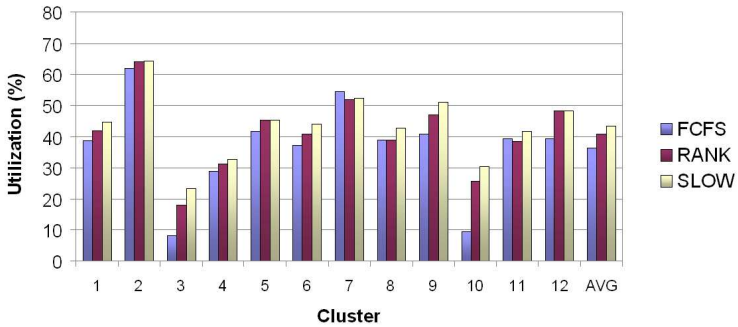
Figure 9a shows that for the DAS-2 system the difference between the *FCFS* policy and the other policies is very significant (up to 40%). This makes sense because the jobs remain blocked in the job queue for more time intervals and, therefore, the job



(a) DAS-2 utilization (%)



(b) Grid5000 utilization (%)



(c) Sharcnet utilization (%)

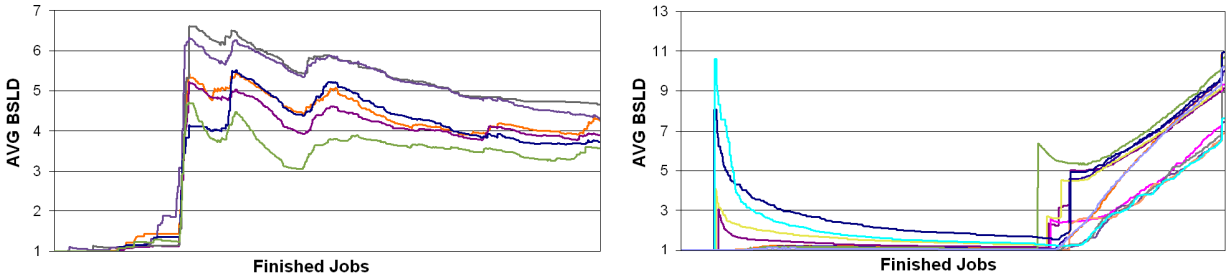
Fig. 9: Resources utilization results

waiting times may become longer. Moreover, the resources may be unused during these time intervals. In figures 9b and 9c the resource utilization is very different depending on the cluster because the resources of these two system are not homogeneous like the DAS-2 resources. Actually, the average clusters resource utilization is similar for Grid5000 and Sharcnet systems (near 40%, which is the average utilization of the DAS-2 system with the *FCFS* configuration). However, the resource utilization difference between the three policies is smaller with the Grid5000 system (around 7% between *FCFS* and *SLOW*) than with the Sharcnet system (around 16% between *FCFS* and *SLOW*).

Therefore, the results show that *JR-backfilling* policy can improve the average resource utilization significantly with respect to the reference *FCFS* policy, especially when the resources are more homogeneous. When the resources are quite heterogeneous (such as the resources of the Grid5000 system) jobs that have similar resource requirements are executed in almost the same clusters because there are only a few resources available that match the main resource requirements. They also show that the *SLOW-coordinated* policy can also improve the resource utilization with respect to the regular matchmaking.

Figure 10 shows the evolution of the average bounded slowdown of the finished jobs of the different clusters in two different workloads using the *SLOW* configuration. Each data series of the figure shows the bounded slowdown of a cluster. They clearly converge to a similar value, which is very close to the total workload average bounded slowdown shown in figure 8c. This indicates that the coordinated strategy can perform a good balancing of performance among the different clusters. Moreover, it is shown that the different clusters follow similar patterns.

Since in the DAS-2 system the resources are homogeneous, probably it is easier to balance the performance. However, in the Sharcnet case that is shown in figure 10b the performance balancing is a bit worse and there are some peaks during the workload execution. These peaks can be explained by the fact that the Sharcnet workload distribution is in bursts having job submission peaks. Moreover, the performance balancing is worse probably because the Sharcnet resources are more heterogeneous.



(a) DAS-2 (b) Sharcnet

Fig. 10: Evolution of the clusters average bounded slowdown

VI. Conclusions and Future Work

In this paper we have described and evaluated our grid scheduling strategy that is based on the coordination between the grid and cluster scheduling layers by having good acknowledgement of the job behavior in both layers. In particular, we have presented a new job scheduling policy based on backfilling (the *JR-backfilling* policy), and a new resource selection policy (the *SLOW-coordinated* policy) that considers the average bounded slowdown of the resources rather than a rank from the resource requirements matching, to improve the resource selection. The main objectives of these polices are to minimize the workload execution time, job waiting time, job response time, average bounded slowdown and to maximize the resource utilization.

We have evaluated the proposed policies with simulation tools and traces from real grid systems. We have taken as a reference one of the most used configurations in grid scheduling system that uses the FCFS job scheduling policy and the matchmaking approach for the resource selection. The results obtained with our evaluation clearly support the argument that coordinating the grid and cluster scheduling layers can improve the workloads execution performance as well as the resource utilization.

The obtained results show that the *JR-backfilling* policy performs better than the regular *FCFS* policy and the *SLOW-coordinated* policy performs better than the regular matchmaking approach in terms of workloads execution time, job waiting time, average job bounded slowdown, and resource utilization. Moreover, we have stated that the best results have been obtained with the combination of the *JR-backfilling* and the *SLOW-coordinated* policies. They reduce the workload execution time up to 20% and the average job bounded slowdown around 30% (on average). We can therefore conclude that delegating the responsibility for performing the resource allocation to the local scheduling systems improves the workloads execution performance. We also conclude that using the resource requirements to advance jobs is a good choice when at the grid scheduling layers there is no neither estimated nor predicted job information available.

We have also shown that the performance metrics of the DAS-2 and Sharcnet systems follow similar patterns with the different configurations. With the Grid5000 system the performance improvement is lower because its resources are more heterogeneous. Therefore, we can conclude that when the resources are more heterogeneous, the workloads execution performance (the execution time, the waiting time and, consequently, the average job bounded slowdown) is significantly worse because there are fewer resources available to allocate jobs with a certain set of resource requirements.

We have also stated that our scheduling strategy improves the average resource utilization significantly with respect to the reference *FCFS* configuration, especially when the resources are more homogeneous. In particular, with the DAS-2 system the resource utilization is around 40% lower with the *FCFS* configuration because the jobs remain blocked in the queue for long time intervals. However, with the other two systems the difference between the clusters utilization and the different configurations is less significant (around 10% on average). This means that the load performance is better balanced in systems that have more homogeneous resources. When the resources are quite heterogeneous (such as the resources of the Grid5000 system) jobs that have similar resource requirements are executed in almost the same clusters because there are only a few resources that match the resource requirements.

Finally, through the study of the evolution of the average bounded slowdown of the finished jobs, we have stated that our scheduling strategy balances the performance among the clusters. We also have seen that it is more difficult to balance the performance among the clusters when the resources are more heterogeneous. Therefore, we have stated that the *SLOW-coordinated* policy improves the resource utilization with respect to the regular matchmaking approach as well as balancing the performance among the different clusters.

There are different lines of work that we plan to address in the near future. On one hand, we are interested in studying the *JR-backfilling* policy deeply. One of the issues that we are studying is the impact of the factor that multiplies the job rank values, also considering a dynamic one. On the other hand, we plan to implement a new resource selection policy combining the matchmaking approach with the SLOW-coordinated policy. We propose using performance metric as a new attribute in the matchmaking process as well as applying a dynamic factor to the rank values depending on the workload and system behavior. Moreover, since the performance and resource utilization improvement is not very significant when the resources are more heterogeneous, we plan to evaluate more complex scenarios. In fact, the main area of our current work is the grid interoperability where numerous heterogeneous resources can be found. We believe this scenario offers plenty of research opportunities for coordinated scheduling strategies.

REFERENCES

[1] D. Abramson, R. Buyya, J. Giddy, A Computational Economy for Grid Computing and its Implementation in the Nimrod-G Resource Broker, Future Generation Computer Systems 18 (2002) 1061–1074.

[2] S. Banen, A. Bucur, D. Epema, A Measurement-based Simulation Study of Processor Co-allocation in Multicluster Systems, in: Job Scheduling Strategies for Parallel Processing (JSSPP), 2002, pp. 105–128, LNCS 2862.

[3] F. Berman, R. Wolski, Scheduling from the Perspective of the Application, in: IEEE International Symposium on High-Performance Distributed Computing (HPDC), 1996, pp. 100–111.

[4] A. Bucur, D. Epema, Trace-Based Simulations of Processor Co-Allocation Policies in Multiclusters, in: IEEE International Symposium on High Performance Distributed Computing (HPDC), 2003, pp. 70–79.

[5] S. Chapin, W. Cirne, D. Feitelson, J. Jones, S. Leutenegger, U. Schwiegelshohn, W. Smith, D. Talby, Benchmarks and Standards for the Evaluation of Parallel Job Schedulers, in: Job Scheduling Strategies for Parallel Processing (JSSPP), 1999, pp. 66–89, LNCS 1659.

[6] D.G. Feitelson and L. Rudolph, Parallel Job Scheduling: Issues and Approaches, in: Job Scheduling Strategies for Parallel Processing (JSSPP), 1995, pp. 1–18, LNCS 949.

[7] I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, A. Roy, A Distributed Resource Management Architecture that Supports Advance Reservations and Co-allocation, in: International Workshop on Quality of Service, 1999.

[8] J. Frey, T. Tannenbaum, M. Livny, I. Foster, S. Tuecke, Condor-G: A Computation Management Agent for Multi-Institutional Grids, in: IEEE International Symposium on High-Performance Distributed Computing (HPDC), San Francisco, CA, USA, 2001.

[9] Grid Resource Management System (GRMS) Web Site. `http://www.gridlab.org/grms`

[10] F. Guim, J. Corbalan, A Job Self-Scheduling Policy for HPC Infrastructures, in: Job Scheduling Strategies for Parallel Processing (JSSPP), 2008, pp. 51–75, LNCS 4942.

[11] F. Guim, J. Corbalan, J. Labarta, Modeling the Impact of Resource Sharing in Backfilling Policies Using the Alvio Simulator, in: MASCOTS, Istambul, 2007.

[12] F. Guim, I. Rodero, J. Corbalan, The resource Usage Aware Backfilling, in: Job Scheduling Strategies for Parallel Processing (JSSPP), 2009, p. 22.

[13] F. Guim, I. Rodero, J. Corbalan, A. Goyeneche, The Grid Backfilling: a Multi-Site Scheduling Architecture with Data Mining Prediction Techniques, Grid Middleware and Services: Challenges and Solutions (2008) 137–152.

[14] Grid Workloads Archive Web Site. `http://gwa.ewi.tudelft.nl/`

[15] E. Huedo, R. Montero, I. Llorente, A Framework for Adaptive Execution in Grids, Software: Practice and Experience 34 (2004) 631–651.

[16] A. Iosup, H. Li, M. Jan, S. Anoep, C. Dumitrescu, L. Wolters, D. Epema, The Grid Workloads Archive, Future Generation Computer Systems 24 (2008) 672–686.

[17] S. Leutenegger, M. Vernon, The Performance of Multiprogrammed Multiprocessor Scheduling Algorithms, ACM SIGMETRICS Performance Evaluation Review 18 (1990) 226–236.

[18] D. Lifka, The ANL/IBM SP Scheduling System, in: Job Scheduling Strategies for Parallel Processing (JSSPP), 1995, pp. 295–303, LNCS 949.

[19] S. Majumdar, D. Eager, R. Bunt, Scheduling in Multiprogrammed Parallel Systems, ACM SIGMETRICS Performance Evaluation Review 16 (1988) 104–113.

[20] H. Mohamed, D. Epema, KOALA: a Co-allocating Grid Scheduler, Concurrency and Computation: Practice & Experience 20 (2008) 1851–1876.

[21] A. Mu'alem, D. Feitelson, Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling, IEEE Transactions on Parallel and Distributed Systems 12 (2001) 529–543.

[22] OGF Grid Scheduling Architecture Research Group (GSA-RG) Web Site. `https://forge.gridforum.org/sf/projects/gsa-rg`

[23] I. Rodero, J. Corbalan, R. Badia, J. Labarta, eNANOS Grid Resource Broker, in: European Grid Conference 2005, Amsterdam, 2005, pp. 111–121, LNCS 3470.

[24] I. Rodero, F. Guim, J. Corbalan, J. Labarta, eNANOS: Coordinated Scheduling in Grid Environments, in: International Conference on Parallel Computing (ParCo), Malaga, Spain, 2005, pp. 81–88.

[25] G. Sabin, R. Kettimuthu, A. Rajan, P. Sadayappan, Scheduling of Parallel Jobs in a Heterogeneous Multi-site Environment, in: Job Scheduling Strategies for Parallel Processing (JSSPP), 2003, pp. 87–104, LNCS 2862.

[26] U. Schwiegelshohn, R.Yahyapour, Attributes for Communication Between Grid Scheduling Instances, Grid Resource Management - State of the Art and Future Trends (2003) 41–52.

[27] U. Schwiegelshohn, R. Yahyapour, Analysis of First-Come-First-Serve Parallel Job Scheduling, in: 9th annual ACM-SIAM symposium on Discrete algorithms, vol. 38, 1998, pp. 629–638.

[28] U. Schwiegelshohn, R. Yahyapour, Improving First-Come-First-Serve Job Scheduling by Gang Scheduling, in: Job Scheduling Strategies for Parallel Processing (JSSPP), 1998, pp. 180–198, LNCS 1459.

[29] K. Sevcik, Application Scheduling and Processor Allocation in Multiprogrammed Parallel Processing Systems, Performance Evaluation 19 (1994) 107–140.

[30] J. Skovira, W. Chan, H. Zhon, The EASY - Loadleveler API Project, in: Job Scheduling Strategies for Parallel Processing (JSSPP), 1996, pp. 41–47, LNCS 1162.

[31] Q. Snell, M. Clement, D. Jackson, C. Gregory, The Performance Impact of Advance Reservation Meta-Scheduling, in: Job Scheduling Strategies for Parallel Processing (JSSPP), 2000, pp. 137–153, LNCS 1911.

[32] D. Thain, T. Tannenbaum, M. Livny, Distributed computing in practice: the Condor experience., Concurrency - Practice and Experience 17 (2-4) (2005) 323–356.

[33] Y. Zhu, M. Ahuja, On Job Scheduling on a Hypercube, IEEE Transactions on Parallel and Distributed Systems 4 (1993) 62–69.