A Job Self-Scheduling Policy for HPC Infrastructures

F. Guim, J. Corbalan *

Barcelona Supercomputing Center {francesc.guim,julita.corbalan}@bsc.edu

Abstract. The number of distributed high performance computing architectures has increased exponentially these last years. Thus, systems composed by several computational resources provided by different Research centers and Universities have become very popular. Job scheduling policies have been adapted to these new scenarios in which several independent resources have to be managed. New policies have been designed to take into account issues like multi-cluster environments, heterogeneous systems and the geographical distribution of the resources. Several centralized scheduling solutions have been proposed in the literature for these environments, such as centralized schedulers, centralized queues and global controllers. These approaches use a unique scheduling entity responsible for scheduling all the jobs that are submitted by the users.
In this paper we propose the usage of self-scheduling techniques for dispatching the jobs that are submitted to a set of distributed computational hosts that are managed by independent schedulers (such as MOAB or LoadLeveler). It is a non-centralized and job-guided scheduling policy whose main goal is to optimize the job wait time. Thus, the scheduling decisions are done independently for each job instead of using a global policy where all the jobs are considered. On top of this, as a part of the proposed solution, we also demonstrate how the usage of job wait time prediction techniques can substantially improve the performance obtained in the described architecture.

1 Introduction

The increasing complexity of the local systems has led to new distributed architectures. These forthcoming systems are composed of multiple computational resources with different characteristics and policies. In these new distributed scenarios, traditional scheduling techniques have evolved into more complex and sophisticated approaches where other factors, such the heterogeneity of the resources [22] or the geographical distribution [11], have been taken into account.

Distributed HPC architectures are usually composed of several centers containing many hosts. In the job scheduling strategies proposed in literature, jobs are submitted to one centralized scheduler that is responsible for scheduling all the submitted jobs to all the computational resources available in the system. Therefore, users submit jobs to this scheduler and it schedules them according to a global scheduling policy. It takes into account all the queued jobs and the resources available in the center to decide which jobs have to be submitted to, where and when.

* This paper has been supported by the Spanish Ministry of Science and Education under contract TIN2004-07739-C02-01.

Similar to the philosophy of the AppLeS project [3], in this paper we propose not to use a global scheduler or global structures for managing the jobs submitted in these scenarios. Rather, we propose using self-scheduling techniques for dispatch the jobs that users want to submit to the set of distributed hosts. In this architecture, the jobs are scheduled by their own dispatcher and there are no centralized scheduling decisions. The dispatcher is aware of the status of the different resources that are available for the job, but it is not aware of the rest of the jobs that other users have submitted to the system. Thus, the job itself decides which is the most appropriate resource for it to be executed. The target architectures of our work are distributed systems were each computational resource is managed by an independent scheduler (such as MOAB or SLURM). Different from AppLeS approach, we propose the interaction between the job dispatcher and the local schedulers. Thus, the presented work proposes the usage of two level of scheduling layers: at the top, the job is scheduled by the dispatcher (the schedule is based on the information provided by the local schedulers and their capabilities); and once the resource is selected and the job submitted, the job become scheduled by the local resource scheduler.

For this purpose, we have designed the ISIS-Dispatcher. It is a scheduling entity that is associated to one and only one job. Therefore, once the user wants to submit a job, a new dispatcher is instantiated. It is responsible for submitting the job to the computational resource that best satisfies the job requirements and that maximizes its response time. The ISIS-Dispatcher has been designed for deployment in large systems, for instance groups of HPC Research Centers or groups of universities. The core of the ISIS-Dispatcher algorithm is based on task selection policies. We also have described a new task selection policy (Less-WaitTime) based on the job wait time prediction. The main advantage of this is that it takes into account the capacity of the available resources while the others not (i.e: Less-WorkLeft, Less-Queued-Jobs etc.).

In this paper we have evaluated the different task assignment other policies proposed in the literature and the Less-WaitTime policy (LWT). The evaluation of the presented architecture shows how the self-scheduling policy can achieve good performance results (in terms of resource usage and job performance). Furthermore, we state how the usage of prediction techniques for the waiting time used in the new Less-WaitTime policy can substantially improve the overall performance. The main reason for this improvement is caused by the fact that it takes into account the status of the resource and its capacity (i.e: number of processors), while the original techniques only considered the status of the resources (i.e: number of queued jobs) and were designed for homogeneous architectures with resources having the same configurations.

The rest of the paper is organized as follows: in sections 2 and 3 we present the background of the presented work and our main contributions; in the section 4 we describe the proposed scheduling architecture, the task selection policies that have been evaluated in the dispatcher and a description of how the prediction of the wait time metric is computed in each center; next the simulation environment used is described, including the models and modules that it includes; in section 6 the experiments and their evaluation studied in this work are presented; and finally, in section 7 the conclusions are presented.

2 Motivation and related work

2.1 Backfilling policies

Authors like Feitelson, Schwiegelshohn, Rudolph, Calzarossa, Downey or Tsafrir have modeled logs collected from large scale parallel production systems. They have provided inputs for the evaluation of different system behavior. Such studies have been fundamental since they have allowed an understanding how the HPC centers users behave and how the resources of such centers are being used. Feitelson has presented several works concerning this topic, among others, he has published papers on log analysis for specific centers [15][27], general job and workload modeling [13][17][14], and, together with Tsafrir, papers on detecting workload anomalies and flurries [38]. Calzarossa has also contributed with several workload modellization surveys [4][5]. Workload models for moldable jobs have been described in works of authors like Cirne et al. in [7][8], by Sevcik in [32] or by Downey in [9].

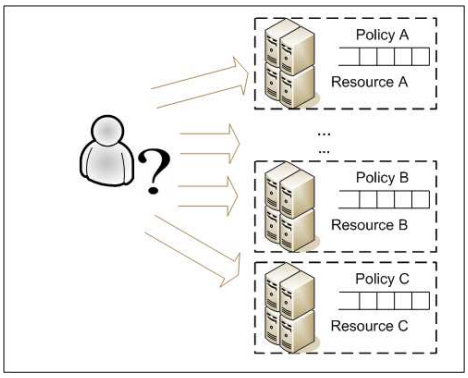


Fig. 1: Heterogeneous multi-host architectures

Concerning job scheduling policies, backfilling [34] policies have been the main goal of study in recent years. As with research in workload modeling, authors like S-H Chiang have provided the community with many quality works regarding this topic. In [18] general descriptions of the most commonly used backfilling variants and parallel scheduling policies are presented. Moreover, a deeper description of the conservative backfilling algorithm can be found in [37], where the authors present policy characterizations and how the priorities can be used when choosing the appropriate job to be scheduled. Other works are [19] and [6].

More complex approaches have been also been proposed by other researchers. For instance, in [28] the authors propose maintaining multiple job queues which separate jobs according to their estimated run time, and using a backfilling aggressive based policy. The objective is to reduce the slowdown by reducing the probability that short job is queued behind a long job. Another example is the optimization presented by

Shmueli et al. in [33] which attempts to to maximize the utilization using dynamic programming to find the best packing possible given the system status.

2.2 Global Scheduling

The previously-discussed works have analyzed how local centers behave when jobs are submitted to a specific host managed by one scheduler. In such conditions jobs are executed in the host to which they were submitted. However, in the current HPC centers, they may have many hosts managed by one centralized scheduler, or even more than one host managed by independent schedulers. In these cases, there is the possibility that a job submitted to a Host A could start earlier in Host B of the same center, or even that it could achieve more performance (i.e.: improving the response time) in another Host C. In recent years, scheduling research activities have started to focus on these new scenarios where many computational resources are involved in the architectures.

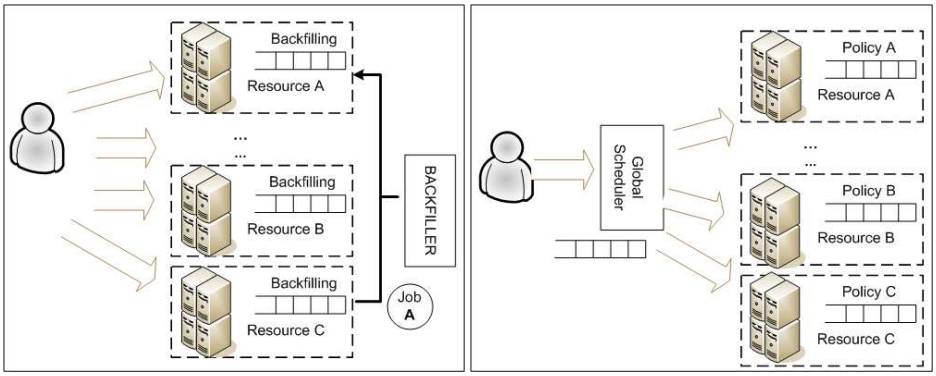
In the coming large distributed systems, like grids, more global scheduling approaches are being considered. In these scenarios users can access a large number of resources that are potentially heterogeneous, with different characteristics and different access and scheduling policies (see Figure 1). Thus, in most cases the users do not have enough information or skills to decide where to submit their jobs. Several models have been proposed in the literature to solve the challenges open in these architectures. We will discuss the most referenced models:

1. Model 1: There are K independent systems with their own scheduling policies and queuing systems, and one or more external controllers. In this scenario users submit jobs to the specific host with a given policy, and a global scheduling entity tries to optimize the overall performance of the system and the service received by the users. For example, as is exemplified in figure 2a the controller may decide to backfill jobs among the different centers [39].
2. Model 2: There is a centralized global scheduler that manages the jobs at the global level and at local level schedulers and queuing systems are also installed. In this situation the users submit jobs to the centralized scheduling system that will later submit the job to the selected scheduling system of a given center. Jobs are queued at the two different levels: first at the global scheduler queue and second at the local scheduler queue (see Figure 2b). This is the typical brokering approach.
3. Model 3: There is a centralized dispatcher that schedules and manages all the jobs but no local schedulers are installed. The local computational nodes only carry out the resource allocation since all the scheduling decisions are taken by the centralized dispatcher. The jobs are queued only at the upper level. (see Figure 3a)
4. Model 4: There is one centralized global queue where all the jobs are queued. The local computational schedulers pull jobs from the global queue when there are enough available resource for run them. In this way the scheduling decisions are done independently at the local level. In this situation (see Figure 3b) the users submit jobs to this centralized queue.

In [39], Yue proposes to apply a global backfilling within a set of independent hosts where each of them is managed by an independent scheduler (Model 1, Figure 2a).

The core idea of the presented algorithm is that the user submits the jobs to a specific system, managed by an independent scheduler. A global controller tries to find out if the job can be backfilled to another host of the center. In the case that the job can be backfilled in another host before it starts, the controller will migrate the job to the selected one. As the algorithm requires the job runtime estimation provided by the user, this optimization is only valid in very homogeneous architectures. This solution may not scale in systems with a high number of computational hosts. Furthermore, other administrative problems may arise, for instance it is not clear if the global *backfiller* presented could scan the queues of all the host involved in the system due to security reasons or VOs administration policies [21].

Sabin et al. studied in [22] the scheduling of parallel jobs in a heterogeneous multi-site environment (Model 2, Figure 2b). They propose carrying out a global scheduling within a set of different sites using a global meta-scheduler where the users submit the jobs. Two different resource selection algorithms are proposed: in the first one the jobs are processed in order of arrival to the meta-scheduler, each of them is assigned to the site with the least instantaneous load; in the second one when the job arrives it is submitted to K different sites (each site schedules according to a conservative backfilling policy), once the job is started in one site the rest of the submissions are canceled (this technique is called multiple requests, MR).



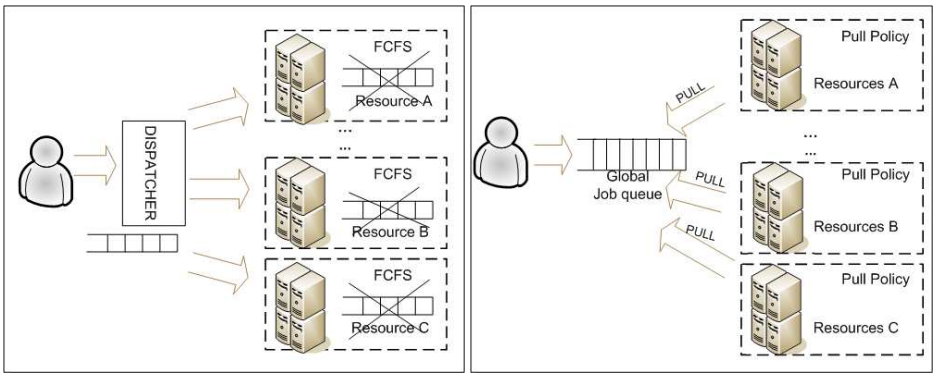
(a) Global Backfilling (b) Global Scheduler

Fig. 2: Proposed solutions (I)

In [11] they analyze the impact of geographical distribution of Grid resources on machine utilization and the average response time. A centralized Grid dispatcher that controls all the resource allocations is used (Model 3, Figure 3a). The local schedulers are only responsible for starting the jobs after the resource selection is made by the Grid Scheduler. Thus, all the jobs are queued in the dispatcher while the size of the job wait queues of the local centers is zero. In this model, a unique global reservation table is used for all the Grid and the scheduling strategy used consists of finding the

allocation that minimizes the job start time. A similar approach is the one presented by Schroeder et al. in [31], where they evaluate a set of task assignment policies using the same scenario (one central dispatcher).

In [30] Pinchak et al. describe a metaqueue system to manage the jobs with explicit workflow dependencies (Model 3 , Figure 3b). In this case, a centralized scheduling system is also presented. However the submission approach is different from the one discussed before. Here the system is composed of a user-level metaqueue that interacts with the local schedulers. In this scenario, instead of the push model, in which jobs are submitted from the metaqueue to the schedulers, placeholding is based on the pull model in which jobs are dynamically bound to the local queues on demand.



(a) Global Dispatcher (b) Global queue using the Pull mechanism

Fig. 3: Proposed solutions (II)

In the previously discussed works, using the global policies, the utilization of the available computational resources have been increased. Furthermore, the service received by the users has also been improved. However, in very large domains these approaches may not scale. Therefore, implementing a centralized scheduling algorithms in these architectures is not appropriate.

In the AppLess project [3][2] Berman et al. introduced the concept of application-centric scheduling in which everything about the system is evaluated in terms of its impact on the application. Each application developer schedules their application so as to optimize their own performance criteria without regard to the performance goals of other applications which share the system. The goal is to promote the performance of an individual application rather than to optimize the use of system resources or to optimize the performance of a stream of jobs. In this scenario the applications are developed using the AppLess framework and they are scheduled by the Apples agents. These agents do not use the functionalities provide by the resource management systems. Therefore, they rely on systems such as Globus [20], Legion [23], and others to perform that function.

2.3 Task assignment policies

The scheduling policy presented in this paper uses task assignment policies to decide where the jobs should be submitted. The subject of job or task assignment policy has been studied in several works and several solutions have been proposed by the research community. Some of the task assignment policies that have been used in the literature are:

- The *Random* policy. The jobs are uniformly distributed among all the available clusters.
- The *Less-SubmittedJobs* policy. The job *i* is assigned to the host *i mod h* (it generates a Round Robin submission).
- The *Shorts-Queue* policy. The jobs are submitted to the host with the least submitted jobs.
- The *Least-WorkLeft* policy. The jobs are submitted to the host with the least pending work. Where it is computed as $PW = \sum_{\forall jobs} RequestedTime_{job} * Processor_{job}$.
- The *Central-Queue* policy. The jobs are queued in a global queue. The hosts pull the jobs from the global queue when enough resources are available.
- The *SITA-E* policy (proposed in [26]). The jobs are assigned to the host based on their runtime length. Thus, *short* jobs would be submitted to **host 1**, *medium* jobs to **host 2** and so on. This policy uses the runtime estimation of the job. In this case the duration cutoffs are chosen so as to equalize load.
- The *SITA-U-opt* policy (proposed in [31]). It purposely unbalances load among the hosts, and the task assignment is chosen so as to minimize the mean slowdown.
- The *SITA-U-fair* policy (also proposed in [31]). Similar to the *opt*, they base the assignment to unbalance the host load. However, the goal for this policy is to maximize fairness. In this SITA variant, the objective is not only to minimize the mean slowdown, but also to balance the slowdown for large jobs equal to short jobs.

The evaluations presented in [31][26] concerning the performance of all these task assignment policies have shown that the SITA policies achieve better results. Schroeacher et al. stated that the Random policy performs acceptably for low loads, but for high loads, however, the average slowdown explodes. Furthermore, the SITA-E showed substantially better performance than Least-Work-Left and Random policies for high loads. However, the Least-Work-Left showed lower slowdown than the SITA-E. SITA-U policies showed better results than the SITE-E. SITA-U-fair improved the average slowdown by a factor of 10 and its variance by a factor of 10 to 100. Harchol presented similar work in [1], where the same situation is studied, however the presented task policy does not know the job duration. Although both SITA-U and SITA-E policies have shown promising performance results, they cannot be used for the Self-Scheduling policy described in this paper due to the fact that they assume having knowledge of the global workload.

3 Paper contribution

To summarize, all the previous scenarios have two common characteristics:

– The scheduling policies are centralized. Thus, the users submit the jobs to a global scheduler.
– They assume that the local resources are homogeneous and are scheduled according the same policy.

Our proposal will be deployed in scenarios with the following conditions:

– The users can submit the jobs directly to the computational resources. Also, they should be able to submit the jobs using the described dispatching algorithm.
– The computational resources can be heterogeneous (with different number of processors, computational power etc.). Also, they can be scheduled by any run-to-completion policy.
– The local schedulers have to provide functionalities for access to information concerning their state (such as number of queued jobs).

In this paper we study the use of job-guided scheduling techniques in scenarios with distributed and heterogeneous resources. The main contributions of this work are:

– The scheduling policy is a job-guided policy. The users submit the job using the ISIS-Dispatcher (see figure 4). Each job is scheduled by an independent Dispatcher.
– Similar to AppLess, the ISIS-Dispatcher is focused on optimizing the job metrics with the scheduling decisions, for instance the job average slowdown, the job response time, the wait time or the cost of the used resources assigned to the job.
– The application has not to be modified for use the proposed architecture. The scheduling is totally transparent to the user and the application.
– The scheduling is done according the information and functionalities that the local schedulers provide (see *job X* of the figure 4). Thus, there is an interaction between the two scheduling layers.
– We keep the local scheduling policies without important modifications. Centers do not have to adapt their scheduling policies or schedulers to ISIS-Dispatcher. They have to provide dynamic information about the system status, for example: the number of queued jobs, when a job would start if it were submitted to the center or which is the remaining computational work in the center.
– In the simulation environment used in the evaluation, we modeled the different levels of the scheduling architecture. Consequently, not only were the dispatcher scheduling policy modeled, but also the local scheduling policies used in the experiments are modeled using an independent reservation table for each of them.
– We propose and evaluate the use of the Less-Waittime task assignment policy that is based on the wait time predictions for the submitted jobs. The evaluation for this policy is compared with the task assignment policies described above, which can be applied in the job-guided scheduling ISIS-Dispatcher policy (SITA policies cannot).

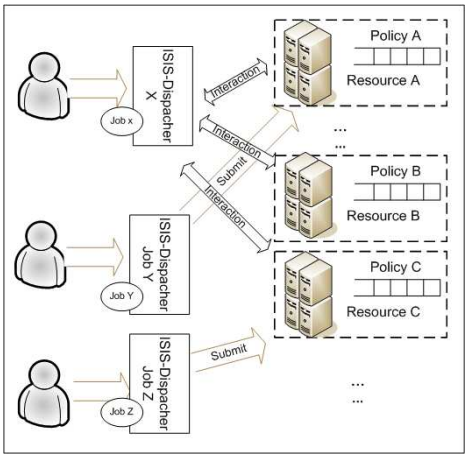


Fig. 4: ISIS-Dispacher architecture

4 The ISIS-Dispatcher

The main objective of this work is to provide a scheduling component that will be used by the user or any other component (e.g.: components that requires self-scheduling or self-orchestration) to decide where to submit their jobs. Figure 4 provides an example of a possible deployment of the proposed architecture. In this example, there are three different users that have access to several computational resources. Each computational resource has its own scheduling policy, different configurations and a particular state (queued jobs, running jobs etc.). When the user wants to submit the job to these resources an instance of the ISIS-Dispatcher is executed and associated to the job. It will decide which is the most appropriate center to submit the job to.

As we have already mentioned, the dispatcher chooses the resource based on a set of metrics provided by each of their schedulers. In this paper we evaluate two different sets of metrics: run time metrics concerning the status of the resource (like the number of queued jobs), and the estimated wait time metric. This prediction information could also be provided by an external prediction service.

In the rest of the section we describe the submission algorithm, the task assignment policies that have been evaluated in this paper, and finally, the prediction model that has been used for evaluated the *Less-WaitTime*.

4.1 The submission algorithm

When the user wants to submit a Job α to the system, he/she contacts the ISIS-Dispatcher which manages the job submision and provides the static description of the job $req_\alpha = \{\partial_1, .., \partial_n\}$. In this evaluation, the user provides the script/executable file, the number of requested processors and the requested runtime time. Once the dispatcher has accepted the submission, it carries out the following steps:

1. For each computational resource σ_i (with particular configuration, characteristics and managed by a given center) in all the resources $\{\sigma_1,..,\sigma_n\}$ available to the user:
   (a) The dispatcher checks that the static properties of σ_i match to the static description of the job $\{\partial_1,..,\partial_n\}$.For example, it would check that the computational resource has equal or more processors than that requested by the job [1].
   (b) In affirmative cases, the dispatcher contacts the predictor service and requests a prediction for the job runtime in the given resource. $RTPred_\alpha(\{\partial_1,..,\partial_n\},\sigma_i)$. In the evaluation presented in this paper, the prediction used was the user run time estimation provided in the original workloads. However, we are currently evaluating the use of datamining techniques to predict the run time in these distributed architectures.
   (c) Once the dispatcher receives the job runtime prediction for the given job in the given resource. For each metric γ_i that has to be optimized ($\{\gamma_1,..,\gamma_n\}$):
      i. It contacts the scheduler that manages the resource and requests the value of the metric: $\alpha_{\gamma_i,\sigma_i} = LocalModule.Perf(RTPred_\alpha, Reqs_\alpha)$
      ii. It adds the performance metric returned to the list of metrics ($metrics_{\{\alpha,\sigma_i\}}$) for the job in the given resource.
2. Given all the list of retrieved metrics, $metrics_\alpha = \left\{\alpha_{\{\gamma_i,\sigma_1\}},..,\alpha_{\{\gamma_m,\sigma_n\}}\right\}$, where a metric entry is composed of the metric value and the resource where the metric was requested. Using an optimization function, $\alpha_{\gamma_i,\sigma_j} = SelectBestResource(metrics_\alpha)$, the best resource is selected based on the metrics that have been collected.
3. The dispatcher will submit the job to the center that owns the resource.

The function *SelectBestMetric* used in the evaluation of this paper is a simplified version of the once presented above. In each of the evaluation experiments, in step (c) of the previous algorithm, only one metric per computational resource was used. ($\left\{\alpha_{\{\gamma_1,\sigma\}},..,\alpha_{\{\gamma_n,\sigma\}}\right\}$).

4.2 Task Assignment Policies

For this paper we evaluated four different task assignment policies

- The Less-JobWaittime policy minimizes the wait time for the job. Given a static description of a job, the local resource will provide the estimated wait time for the job based on the current resource state. We implemented a prediction mechanism for different sets of scheduling policies (EASY-Backfilling, LXWF-Backfilling, SJF-Backfilled First and FCFS) that use a reservation table that simulates the possible scheduling outcome taking into account all the running and queued jobs at each point of time (see below).
- The Less-JobsInQueue policy submits the job to the computational resource with the least number of queued jobs. (The presented Shortest-Queue in the background).
- The Less-WorkLeft policy submits the job to the computational resource with the least amount of pending work.
- The Less-SubmittedJobs policy submits the jobs to the center with the least number of submitted jobs.

[1] In architecture with thousands of resources, it is not feasible to contact all the resources. Future versions will include heuristics to decide which hosts the dispatcher has to connect to, and which not.

4.3 Job wait time prediction

The Less-JobWaittime task assignment policy submits the job to the center that returns the lowest predicted wait time. The approach taken in this evaluation was that each center has to provide such predictions. However, other architectures can also be used, for instance having several prediction/model services. In that case no interactions with the local centers would be required.

How to predict the amount of time that a given job will wait in the queue of a system has been explored by other researchers in several works [10][35][36][29]. What we propose in this paper is the use of reservation tables. The reservation is used by the local scheduling policies to schedule the jobs and decide where and when the jobs will start.

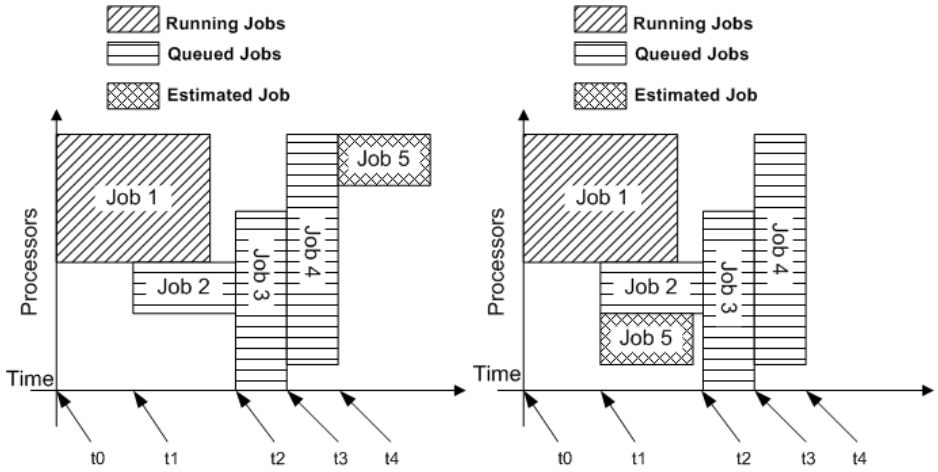
In this paper the prediction mechanism uses the reservation table to estimate the possible scheduling outcome. It contains two different types of allocations: allocations for those jobs that are running; and pre-allocations for the queued jobs. The status of the reservation table in a given point of time is only one of all the possible scheduling outcomes and the current scheduling may change depending on the dynamicity of the scheduling policy. Also, the accuracy of the job runtime estimation or prediction has an impact on the dynamicity of the scheduling outcomes, mainly due to the job runtime overestimations.

The prediction of the wait time of a job α at time T_1, that requires α_{time} and α_{cpus}, in a given resource, will be computed with: the earliest allocation that the job would receive in the resource given the current outcome if it was submitted at time T_1. Obviously, this allocation will depend on the scheduling policy used in the center, and probably will vary in different time stamps. All the scheduling events are reflected on the status of the reservation table. The prediction technique presented in this work is mainly designed for FCFS and backfilling policies. The information that is used for allocating the job in the reservation table is: the number of required processors and its estimated runtime.

Figures 5 provide two examples of how a prediction for a new job would be computed in the two scheduling policies used in this paper. In both examples the current time is t_1, there is one job running (*Job 1*), and three more queued (*Job 2*, *Job 3* and *Job 4*). If a prediction for the wait time for the job *Job 5* was required by a given instance of the ISIS-Dispatcher, the center would return $t_4 - t_1$ in the case of FCFS (Figure 5a) and would return 0 in the case of Backfilling (Figure 5b).

5 Simulation characteritzation

In this section we describe the simulation environment that was used to evaluate the presented policy and architecture: first we describe the C++ event-driven simulator that was used to simulate local and distributed High Performance Computing Architectures; and second, we characterize all the experiments that were designed to evaluate the current proposal.



(a) Prediction for FCFS (b) Prediction for Backfilling

Fig. 5: Scenario All SJF-Backfilled First

5.1 The Alvio Simulator

All the experiments were conducted using the C++ event-driven Alvio simulator [24]. The simulator models the different components that interact in local and distributed architectures. Conceptually, it is divided into three main parts: the simulator engine, the scheduling polices model (including the resource selection policies), and the computational resource model. A simulation allows us to simulate a given policy with a given architecture. Currently, three different policies have been modeled: the First-Come-First-Served, the Backfilling policies can be used, and finally, the ISIS-Dispatcher scheduling policy. For the backfilling policies the different properties of the wait queue and backfilling queue are modeled (SJF, LXWF and FCFS) and different numbers of reservations can also be specified.

The architecture model allows us to specify different kind of architectures. Currently, cluster architectures can be modeled, where the host is composed of a set of computational nodes, where each node has a set of consumable resources (currently Memory Bandwidth, Ethernet Bandwidth and Network bandwidth). Although the use of these consumable resources can be simulated in a high level fashion, for the experiments presented in this paper it has not been used.

The local scheduling policies (all excluding the ISIS-Dispatcher) use a set of job queues and a reservation to schedule the jobs. The reservation table that is linked to a given architecture has the running jobs allocated to the different processors during the time. One allocation is composed of a set of buckets that indicate that a given job α is using the processors β from $\alpha_{startTime}$ until $\alpha_{endTime}$. Depending on the policy configuration, the scheduling policy will temporarily allocate the queued jobs (for instance, to estimate the wait time for the jobs). The distributed scheduling ISIS-Dispatcher policy does not have a reservation table because it does not allocate the jobs to the proces-

sors. Furthermore, the local scheduling policies must provide a functionality that allows querying metrics concerning the current state of the local system. This functionality will be used by the dispatcher to decide where to submit the jobs.

5.2 Experiments

In this section we present the workloads used in the simulations and the scenarios that were designed to evaluate the proposal.

5.3 Workloads

The design and evaluation of our experiments were contrasted and based on the analytical studies available for each of the workloads that we used in our simulations:

- The San Diego Supercomputer Center (SDSC) Blue Horizon log (144-node IBM SP, with 8 processors per node)
- The San Diego Supercomputer Center (SDSC-SP2) SP2 log (128-node IBM SP2)
- The Cornell Theory Center (CTC) SP2 log [16] (512-node IBM SP2).

For the simulation we used traces generated with the fusion of the first four months of each trace (FUSION). The following section describes the simulation: first we simulated the four months for each trace independently; second, using the unique fusion trace, different configurations of a distributed scenario composed by the three centers were simulated. We chose these workloads because they contain jobs with different levels of parallelism and with run times that vary substantially. More information about their properties and characteristics can be found in the workload archive of Dror Feitelson [12].

5.4 Simulation Scenarios

In all the scenarios presented below, all the metrics presented in section 4 were evaluated. In the second and third scenarios we also evaluated what happens when the characteristics of the underlying systems have different configurations, in terms of scheduling policies and computational resource configurations.

The characteristics of each of the evaluated scenarios are:

1. In the first scenario (ALL-SJF), all the centers used the same policy: Shortest Job Backfilled First. The number of processors and computational resources were configured in exactly the same way as the original.
2. In the second scenario (CTC/4), the SFJ-Backfilled first was also used for all the centers. However, in this case we emulated what would happen if the CTC center performed four times slower than the two others. In this case, all the jobs that ran to this center spent four times longer than the runtime specified in the runtime of the original workload [2]. The main goal is to evaluate the impact of having resources with different computational power.

[2] This is only a first approximation. Future studies may use more detailed models

3. In the last scenario (CTC-FCFS), the SDSC and SDSC-Blue also used the SJF-Backfilled First policy. However, the CTC center used the FCFS scheduling policy. As in the first scenario, the computational resource configuration was exactly the same as the original.

The first scenario evaluates situations where all the hosts available have the same scheduling policy. Thus, each computational host is managed by the same scheduling policy and each computational unit (the processors) of all the hosts has the same power. We defined the two other scenarios to evaluate how the presented scheduling policy behaves with heterogeneous computational resources and with different scheduling policies. In the second scenario we evaluated the impact of having heterogeneous resources. In this situation the CTC processors perform four times slower than the processors of the other two centers. In the last scenario we evaluated the impact of having different scheduling policies in the local hosts.

Center	**Estimator**	**BSLD**	**SLD**	**WaitTime**
SDSC	Mean	8,66	12,9	2471
	STDev	47,7	86,07	8412
	95_{th} Percentile	17,1	18,92	18101
SDSC-Blue	Mean	6,8	7,6	1331
	STDev	29	36	5207,2
	95_{th} Percentile	28,5	29	8777
CTC	Mean	2,8	3,03	1182
	STDev	23	27,1	4307,3
	95_{th} Percentile	2,3	2,5	6223
CTC/4	Mean	19,8	20,467	9664
	STDev	57,23	58,203	20216
	95_{th} Percentile	114,3	116,3	54450
CTC FCFS	Mean	12,833	14,04	3183,3
	STDev	66,54	77,03	9585
	95_{th} Percentile	32,403	32,65	32996,4

Table 1: Performance Variables for each workload

6 Evaluation

6.1 The Original Workloads

Table 1 presents the performance metrics for the simulation of the workloads used in this paper (CTC, SDSC and SDSC-Blue) with SJF Backfilling in each center. We also include the simulations for the CTC with the other two different configurations that were used in the experiments of the distributed scenarios: the first includes the CTC,

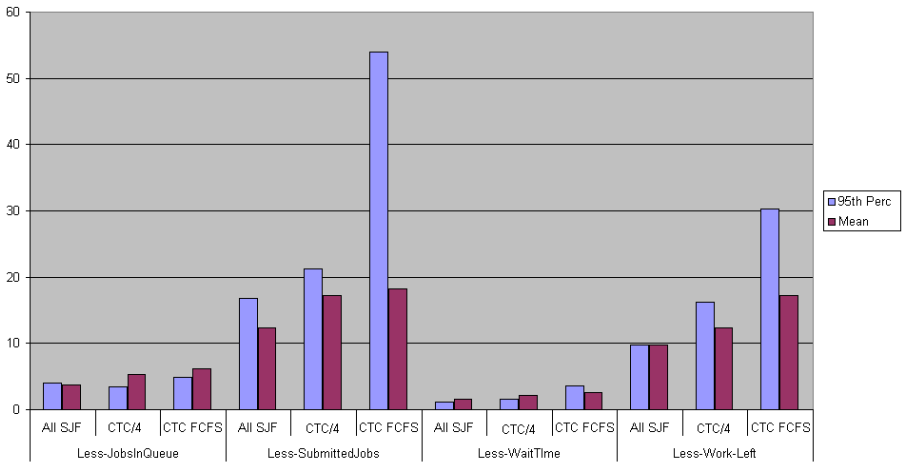


Fig. 6: Bounded Slowdown

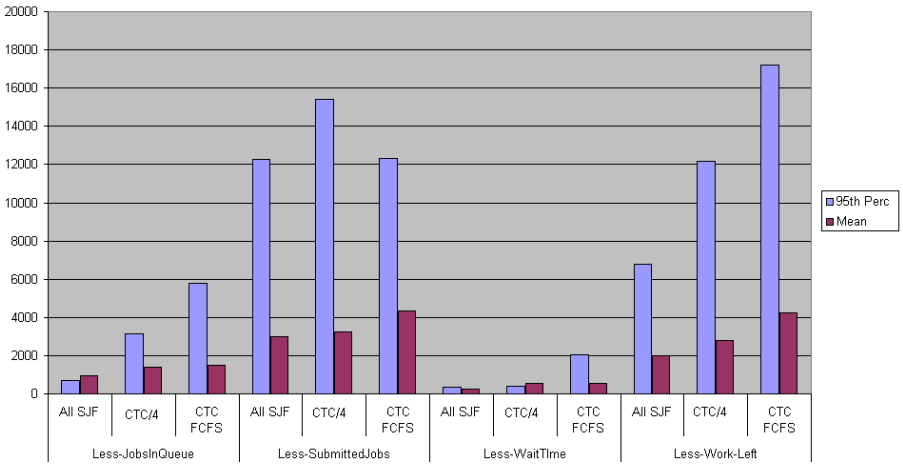


Fig. 7: Wait time

and the second also includes the CTC simulation, but using the FCFS policy. As can be observed, the workload that has the best slowdown and wait time is the CTC. The SDSC and SDSC-Blue have a similar average bounded slowdown, however, the 95_{th} percentile of the SDSC-Blue is one order of magnitude greater than the SDSC. In terms of wait time, jobs remain longer in the wait queue in the workload of the SDSC than the other two. In terms of 95_{th} percentile the jobs spend three times longer in the SDSC than in the CTC.

The performance obtained when reducing the computational power and the policy of the CTC center (Table 1) is not surprising. Using FCSC or reducing by four the computational power of the CTC significantly increases the slowdown and wait time for the CTC workload. The capacity of the resource of execute the same workload was reduced four times. Thus, the original scenario cannot cope with the same job stream. The main concern was to evaluate later this configuration in the distributed scenario.

Center	**Estimator**	**Ratio BSLD**	**Ratio SLD**	**Ratio WaitTime**
SDSC	Mean	5,44	7,5	10,98
	STDev	5,1	9,27	4,36
	95_{th} Percentile	14,1	14,92	53,2
SDSC-Blue	Mean	4,2	4,4	5,9
	STDev	3,1	3,6	2,7
	95_{th} Percentile	23,5	22	25,8
CTC	Mean	1,8	1,7	5,2
	STDev	2,5	2,9,1	2,3
	95_{th} Percentile	1,6	1,6	18

Table 2: Ratio: Original Job Perf. / ISIS Less-WaitTime Job Perf.

6.2 The first scenario: all centers with SJF-Backfilled First

Figure 6 presents the average and 95_{th} percentile for the average slowdown in the three presented scenarios and the different task assignment policies studied. In the *scenario 1* (all centers with SJF), the Less-JobQueuedJobs and Less-WaitTime policies showed the best performance. However, the first one obtains a slowdown (1,7) twice as small as the second one (3,9). The other two policies performed substantially worst. The average slowdown and the 95_{th} percentile are three or even ten times greater than in the others. For instance the average slowdown of the Less-Waitime is around two while the same slowdown for the Less-WorkLeft in the same scenario is around ten. The average wait time in this scenario (see Figure 7) presented similar behavior to the slowdown. However, the percentile shows that in the case of the Less-SubmittedJobs and Less-WorkLeft the wait time of the jobs has a high variance. This fact is also corroborated by the standard deviation that the wait time experiments in both policies (see Figure 11).

The Less-WorkLeft policy takes into account the amount of pending work and the Less-JobsInQueue not. Therefore, we expected that the first policy one would perform much better than the second one. However, the presented results showed the contrary. Analyzing both simulations we have stated that in some situations the Less-WorkLeft policy unbalances excessively the number of submitted jobs. As shown in table 3 it submits around 800 jobs more to the SDSC center than the Less-JobsInQueue. The figures 8 and 9 show that the amount of queued jobs in the SDSC is substantially bigger in the Less-WorkLeft policy in this specific interval of the simulation. This unbalance is caused by the characteristics of the stream of jobs that are submitted during this interval to the system. The initial part of this stream is composed by several jobs that requires from 256 processors until 800 processors and that have large runtime. Because of the capacity of the SDSC center (128 processors), these jobs can only be allocated to the CTC center (412 processors) and the SDSC-Blue (1152 processors). This causes that an important amount of smaller jobs (with less than 128 processors) have to be submitted to the SDSC center for accumulate the same amount of pending work that are assigned to the other two centers. Thus, as we state in [25], this stream of jobs composed by jobs that requires all the processors of the host and jobs that requires small number of processors causes an important fragmentation in the scheduling of the SDSC. These situations occurs several times in the simulation and they decrease substantially the performance achieved by the Less-WorkLeft policy.

What the results suggest is that the Less-SubmittedJobs policy has the worst performance of all the assignment policies, since the choice of where the job is submitted does not depend on the static properties of the job (estimated runtime and processors). Regarding the other two policies, the Less-JobsInQueue policy performs substantially better than the Less-WorkLeft.

Table 2 provides the ratio for the job performance variables in the original scenario (where jobs where submitted to the independent centers) against the performance for the jobs in the ALL-SJF scenario using the Less-WaitTime policy. The results show that the jobs of all the centers obtained substantially better service in the new scenario. For instance, the average bounded slowdown in SDSC is 5.44 times greater than the average bounded slowdown for the jobs in the *original workloads*. On the other hand, the resource usage achieved by the Less-WaitTime policy has been improved. As the table 4 shows, the average of used processors per hour in the centers has been improved. Although the SDSC-Blue has experimented a soft drop in its processors usage, the CTC has experimented a notoriously increment in its processors usage. Also, as can be been seen in the number of running jobs per hour the packing of jobs has been improved.

6.3 Second and third scenarios: CTC/4 and CTC with FCFSC

The other two scenarios analyzed in the paper show that the ISIS-Dispacher scheduling policy is able to react in heterogeneous environments where the computational capabilities of the different centers can vary (in the *scenario 2* with a resource with less computational power and in the *scenario 3* with a resource with different scheduling policy). Compared to the *scenario 1* the performance shown in both scenarios experienced only a small drop. Thus, the system was able to schedule the jobs to the different resources adapting to the different capabilities of each of the available centers. This fact can be

Resource	*Less-JobsInQueue*	*Less-WorkLeft*
CTC	10788	10912
SDSC	1953	2560
SDSC-Blue	9550	8819

Table 3: Number of submitted jobs per host

Center	**Variable**	**Original Workload**	**Less-WaitTime Scenario**
SDSC	Running Jobs	5,1	6,02
	Number of used CPUs	52,3	58,3
SDSC-Blue	Running Jobs	4,5	4
	Number of used CPUs	492,8	435,2
CTC	Running Jobs	148,5	282,7
	Number of used CPUs	18,2	23,4

Table 4: Average of processors used and running jobs per hour

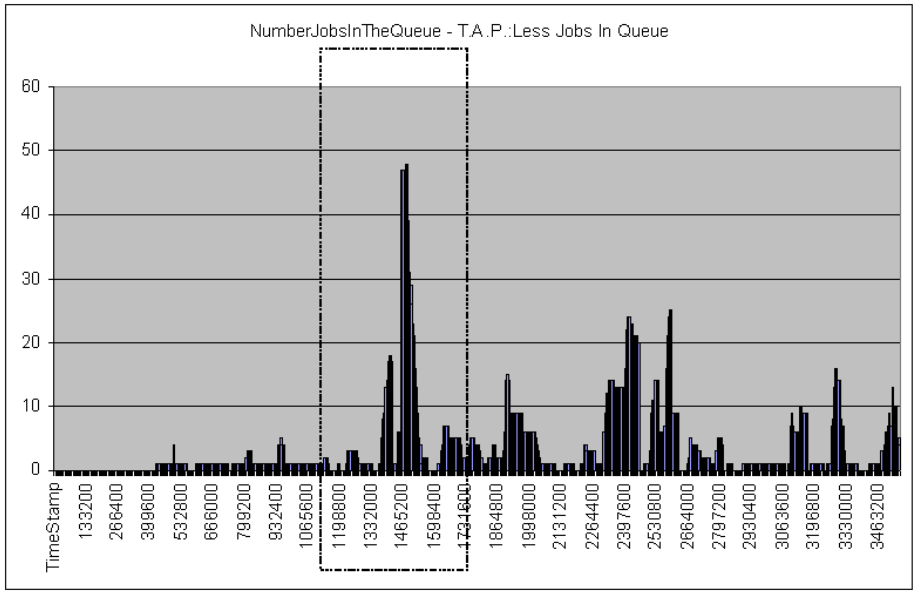


Fig. 8: Number of queued jobs in the SDSC

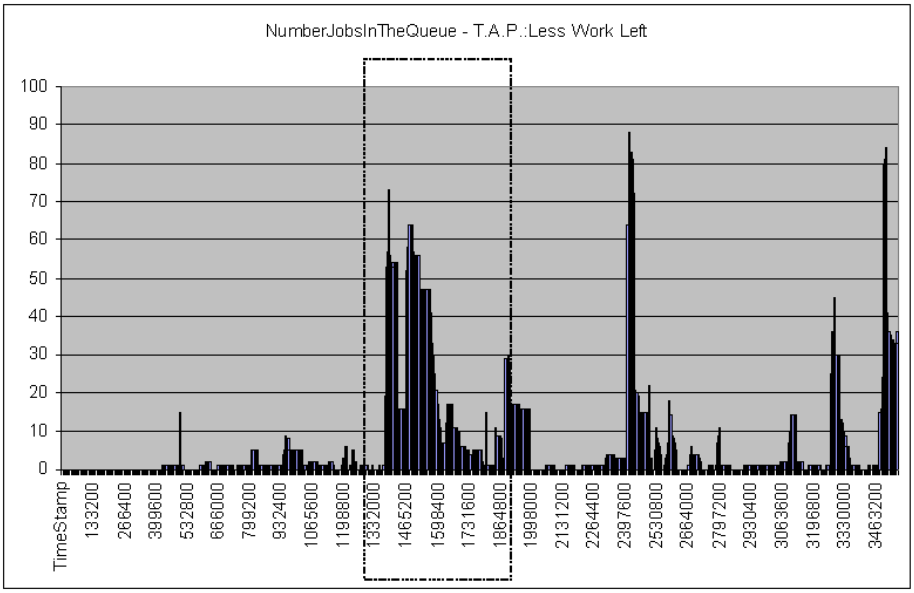


Fig. 9: Number of queued jobs in the SDSC

observed in figure 12 where the normalized amount of workload done by each center is described for *scenarios 1 and 2*. Clearly, in the situation where the CTC center used a scheduling policy with lower performance, the amount of workload was automatically balanced from this center to the SDSC-Blue and to the SDSC center (similar properties were found in the *scenario 3*). Regarding the performance achieved by each of the task assignment policies used in the experiments, the results show similar behaviors to those we observed in the first scenario.

Clearly, independently of the configuration used, using the Less-WaitTime assignment policy in the ISIS-Dispatcher scheduling policy obtained the best performance results in all the scenarios that were evaluated. It has demonstrated that it is better able to adapt to the difference configuration of the local centers, and to provide a similar service to all the jobs.

7 Conclusions and future Work

In this paper we have presented the use of a job-guided scheduling technique designed to be deployed in distributed architectures composed of a set of computational resources with different architecture configurations and different scheduling policies. Similar to the AppLeS project, the main goal of the technique presented here is to provide the user with a scheduling entity that will decide the most appropriate computational resource to submit his/her jobs to. We support the interaction between the job dispatcher and the local schedulers. Thus, the presented work proposes the usage of two level of

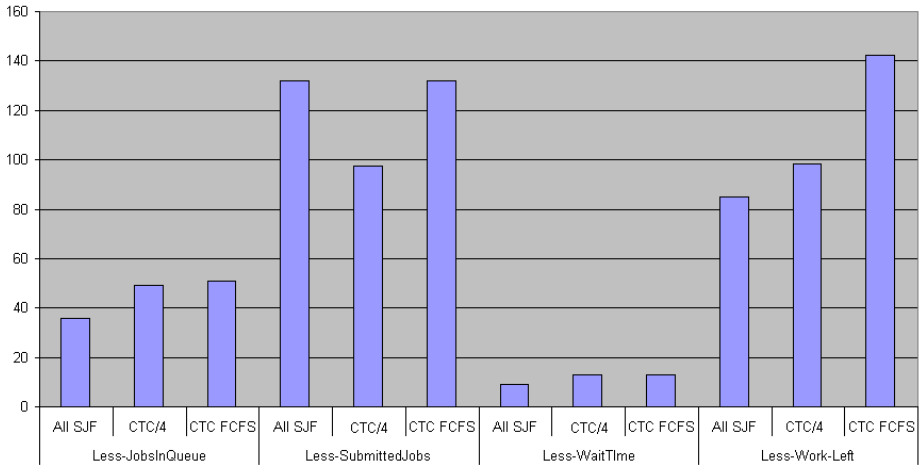


Fig. 10: Bounded Slowdown Standard Deviation

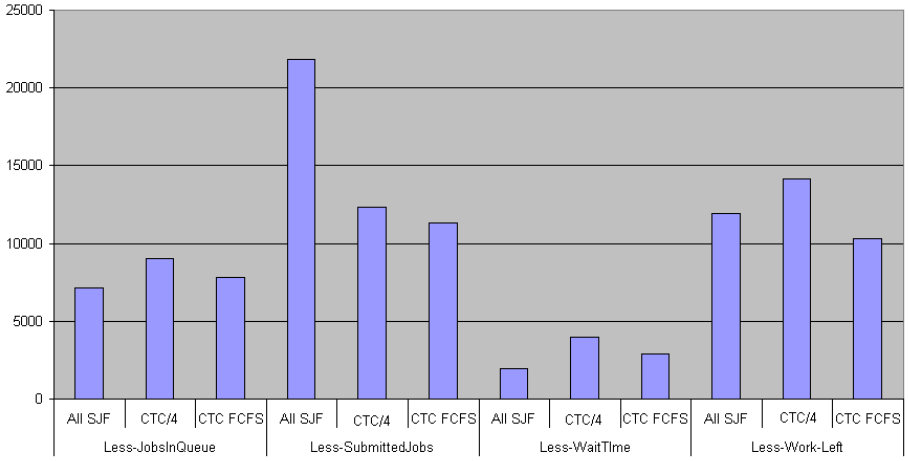
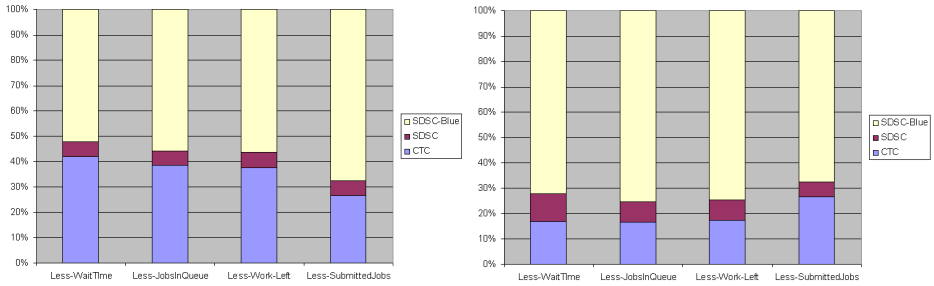


Fig. 11: Wait time Standard Deviation



(a) All With SJF (b) CTC with FCFS

Fig. 12: Work distribution in the ISIS-Scenarios

scheduling layers: at the top, the job is scheduled by the dispatcher (the schedule is based on the information provided by the local schedulers and their capabilities); and once the resource is selected and the job submitted, the job become scheduled by the local resource scheduler.

The scheduling policy presented here uses a set of task assignment policies to decide where the jobs are finally submitted. This paper has also evaluated how the most representative task assignment scheduling policies presented in the literature perform in the policy presented here (including the Less-WorkLeft, Less-Less-SubmittedJobs and Less-JobsInQueue policies). Furthermore, a task selection policy using the wait time prediction and focused on reducing the wait time for jobs has been proposed.

We have evaluated the proposal in three different scenarios using the workloads CTC, SDSC and SDSC-Blue from the Workload Log Archive. The first scenario was composed of a set of centers with the same scheduling policies and different computational resources (different numbers of processors); the second was composed of a set of centers with different scheduling policies (two with SJF-Backfilled First and one with FCFS) and different computational resources ; and finally, the last one was composed of centers with the same policies and different computational resources with different computational power (one of the centers performed four times slower than the other two). Although the scheduling proposal presented in this paper is non-centralized and the dispatcher does not store any information regarding the progress of the global scheduling, it has been shown that using the appropriate task assignment policy (in this analysis the Less-Waittime policy showed the most promising results) it is able to achieve good global performance, adapting to the underlying center resource characteristics and to the local scheduling policies. Furthermore, not only the job wait time has been improved, the resource usage and the job packing have been also improved.

In future work we plan to use prediction techniques to estimate job runtime rather than user estimates. We are currently working on prototypes where the run time of jobs is estimated using C45 algorithms and discretization techniques (and other datamining techniques). In such scenarios, users will only have to provide the number of requested processors and the static description of the job. We will have to evaluate the impact of prediction and user runtime estimation errors on such architectures.

We will extend the current submission algorithm including other negotiation mechanisms between the local centers and the dispatcher, for instance using Service Level Agreement negotiations or advanced reservations. Furthermore, the ISIS-Dispatcher will be alive during the complete job cycle of life monitoring the job evolution. It will be able to decide to migrate the job to other resources or to carry out other scheduling decisions to achieve better performance. In the current version the dispatching algorithm contacts to all the schedulers that matches the job requirements to gathering the scheduling information. Future version of this algorithm will include user and job heuristics for reduce the amount of schedulers to be queried. Thus, the number of communications will be reduced.

References

1. N. Bansal and M. Harchol-Balter. *Analysis of SRPT scheduling: investigating unfairness*. 2001.
2. F. Berman and R. Wolski. Scheduling from the perspective of the application. pages 100–111, 1996.
3. F. Berman and R. Wolski. The apples project: A status report. 1997.
4. M. Calzarossa, G. Haring, G. Kotsis, A. Merlo, and D. Tessera. A hierarchical approach to workload characterization for parallel systems. *Performance Computing and Networking, Lect. Notes Comput. Sci. vol.*, page pp. 102109, 1995.
5. M. Calzarossa, L.Massari, , and D. Tessera. Workload characterization issues and methodologies. *In Performance Evaluation: Origins and Directions, Lect. Notes Comput. Sci.*, page pp. 459482, 2000.
6. S.-H. Chiang, A. C. Arpaci-Dusseau, and M. K. Vernon. The impact of more accurate requested runtimes on production job scheduling performance. *8th International Workshop on Job Scheduling Strategies for Parallel Processing*, Vol. 2537:103 – 127, 2002.
7. W. Cirne and F. Berman. A comprehensive model of the supercomputer workload. *4th Ann. Workshop Workload Characterization*, 2001.
8. W. Cirne and F. Berman. A model for moldable supercomputer jobs. *15th Intl. Parallel and Distributed Processing Symp.*, 2001.
9. A. B. Downey. A parallel workload model and its implications for processor allocation. *6th Intl. Symp. High Performance Distributed Comput.*, Aug 1997.
10. A. B. Downey. Using queue time predictions for processor allocation. *3rd Workshop on Job Scheduling Strategies for Parallel Processing*, Lecture Notes In Computer Science; Vol. 1291:35 – 57, 1997.
11. C. Ernemann, V. Hamscher, , and R. Yahyapour. Benefits of global grid computing for job scheduling. *5th IEEE/ACM International Workshop on Grid Computing*, 2004.
12. D. D. G. Feitelson. Parallel workload archive, 2006.
13. D. G. Feitelson. Packing schemes for gang scheduling'. *Job Scheduling Strategies for Parallel Processing*, Lect. Notes Comput. Sci. 1162:pp. 89–110, 1996.
14. D. G. Feitelson. Workload modeling for performance evaluation. *In Performance Evaluation of Complex Systems: Techniques and Tools, Lect. Notes Comput. Sci. vol. 2459*, pages pp. 114–141, 2002.
15. D. G. Feitelson and B. Nitzberg. Job characteristics of a production parallel scientific workload on the nasa ames ipsc/860. *In Job Scheduling Strategies for Parallel Processing, Lect. Notes Comput. Sci.*, vol. 949:pp. 337–360, 1995.
16. D. G. Feitelson and L. Rudolph. Workload evolution on the cornell theory center ibm sp2. *Job Scheduling Strategies for Parallel Processing*, 1162:pp. 27–40, 1996.

17. D. G. Feitelson and L. Rudolph. Metrics and benchmarking for parallel job scheduling. *In Job Scheduling Strategies for Parallel Processing, Lect. Notes Comput. Sci.*, vol. 1459:pp. 1–24, 1998.
18. D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn. Parallel job scheduling - a status report. *Job Scheduling Strategies for Parallel Processing: 10th International Workshop, JSSPP 2004*, 3277 / 2005:9, June 2004.
19. D. G. Feitelson and A. Weil. Utilization and predictability in scheduling the ibm sp2 with backfilling. *Proceedings of the 12th. International Parallel Processing Symposium*, pages 542–546, 1998.
20. I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *J Intl - International Journal of Supercomputer Applications.*, 1997.
21. I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the Grid: Enabling scalable virtual organizations. *Lecture Notes in Computer Science*, 2150, 2001.
22. S. Gerald, K. Rajkumar, R. Arun, and S. Ponnuswamy. Scheduling of parallel jobs in a heterogeneous multi-site environment. *JSSPP*, 2003.
23. A. S. Grimshaw, W. A. Wulf, J. C. French, A. C. Weaver, and P. F. Reynolds Jr. Legion: The next logical step toward a nationwide virtual computer. (CS-94-21), 8, 1994.
24. F. Guim, J. Corbalan, and J. Labarta. The internals of the alvio-simulator: Simulator of hpc infraestructures (upc-dac-rr-cap-2007-2). Technical report, Architecture Computer Deparment - Technical University of Catalunya, 2005.
25. F. Guim, J. Corbalan, and J. Labarta. Modeling the impact of resource sharing in backfilling policies using the alvio simulator. *Submitted to 15th Annual Meeting of the IEEE / ACM International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, 2007.
26. M. Harchol-Balter, M. E. Crovella, and C. D. Murta. On choosing a task assignment policy for a distributed server system. *Journal of Parallel and Distributed Computing*, 59(2):204–228, 1999.
27. V. L. K. Windisch, R. Moore, D. Feitelson, , and B. Nitzberg. A comparison of workload traces from two production parallel machines. *In 6th Symp. Frontiers Massively Parallel Comput.*, pages pp.319–326, 1996.
28. B. G. Lawson and E. Smirni. *Multiple-Queue Backfilling Scheduling with Priorities and Reservations for Parallel Systems*. Springer Verlag, 2002. Lect. Notes Comput. Sci. vol. 2537.
29. H. Li, J. Chen, Y. Tao, D. Groep, , and L. Wolters. Improving a local learning technique for queue wait time predictions. *Cluster and Grid computing*, 2006.
30. C. Pinchak, P. Lu, and M. Goldenberg. Practical heterogeneous placeholder scheduling in overlay metacomputers: Early experiences. *Job Scheduling Strategies for Parallel Processing*, pages 205–228, 2002. Lect. Notes Comput. Sci. vol. 2537.
31. B. Schroeder and M. Harchol-Balter. Evaluation of task assignment policies for supercomputing servers: The case for load unbalancing and fairness. *Cluster Computing 2004*, 2004.
32. K. C. Sevcik. Application scheduling and processor allocation in multiprogrammed parallel processing systems. *Performance Evaluation*, pages pp. 107–140, 1994.
33. E. Shmueli and D. G. Feitelson. *Backfilling with Lookahead to Optimize the Performance of Parallel Job Scheduling*. Springer Verlag, 2003. Lect. Notes Comput. Sci. vol. 2862.
34. J. Skovira, W. Chan, H. Zhou, and D. A. Lifka. The easy - loadleveler api project. *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, Lecture Notes In Computer Science; Vol. 1162 archive:41 – 47, 1996.
35. W. Smith, V. E. Taylor, and I. T. Foster. Using run-time predictions to estimate queue wait times and improve scheduler performance. *Proceedings of the Job Scheduling Strategies for Parallel Processing*, Lecture Notes In Computer Science; Vol. 1659:202 – 219, 1999.

36. W. Smith and P. Wong. Resource selection using execution and queue wait time. predictions. page 7.
37. D. Talby and D. Feitelson. Supporting priorities and improving utilization of the ibm sp scheduler using slack-based backfilling. *Parallel Processing Symposium*, pages pp. 513–517, 1999.
38. D. Tsafrir and D. G. Feitelson. Instability in parallel job scheduling simulation: the role of workload flurries. *In 20th Intl. Parallel and Distributed Processing Symp*, 2006.
39. J. Yue. Global backfilling scheduling in multiclusters. *Asian Applied Computing Conference, AACC 2004*, pages pp. 232–239, 2004.