THE GRID BACKFILLING: A MULTI-SITE SCHEDULING ARCHITECTURE WITH DATA MINING PREDICTION TECHNIQUES*

Francesc Guim and Ivan Rodero and Julita Corbalan
Barcelona SuperComputing Center
Jordi Girona 29, 08034 Barcelona, Spain
{francesc.guim,irodero,julita.corbalan}@bsc.es

A. Goyeneche
Centre for Parallel Computing, Cavendish School of Computer Science
University of Westminster
115 New Cavendish Street, London, W1W 6UW
goyenea@wmin.ac.uk

Abstract In large Grids, like the National Grid Service (NGS), or large distributed architecture different scheduling entities are involved. Despite a global scheduling approach would archive higher performance and could increment the utilization of global system in these scenarios usually independent schedulers carry out its own scheduling decisions.

In this paper we present how a coordinated scheduling among all the different centers using data mining prediction techniques can substantially improve the performance of the global distributed infrastructure, and can provide a uniform access to the user to all the heterogeneous Grid resources. We present the Grid Backfilling meta-scheduling policy that optimizes the global utilization of the system resources and increases substantially the response time for the jobs. We also present how data mining techniques applied to historical information can provide very suitable inputs for carrying out the Grid Backfilling meta-scheduling decisions.

Keywords: Grid Computing, Meta-Scheduling, Backfilling, Prediction

*This research work is carried out under the FP6 Network of Excellence CoreGRID funded by the European Commission (Contract IST-2002-004265)

1. Introduction

The number of computational resources has increased exponentially these last decades. Scheduling policies have been adapted to these new scenarios where several independent resources have to be managed. Thus, the local policies, such as the FCFS, Gang Scheduling or Backfilling policies have also evolved to more sophisticate approaches for considering issues like multicluster environments, heterogeneous systems or the geographical distribution of the resources. Several global scheduling solutions have been proposed in the literature for these environments, such as centralized schedulers, centralized queues or global controllers.

The backfilling policies have been demonstrated to be the most effective policies in the local high performance computing centers. Some research works have extended the traditional backfilling policies for a distributed environments (see section 2). However one the major problem that they have is that the runtime of the scheduled applications is supposed to be known, or at least a closer estimation at submission time and should by provided by the user. However user in most of the cases will not have enough information (f.e: the user does not know in which cluster its grid application will run) or enough skills (f.e: the user just wants to submit its fluid dynamic application without knowing how many minutes it will take in a given allocation of nodes and cpus) for specify how long it will his/her job to run.

In this paper we present the Grid Backfilling scheduling policy. It extends the algorithm for the backfilling traditional policy presented in [16]for distributed architectures using a prediction service. The main goal of the presented policy is optimizing the overall performance of the system backfilling the jobs to the different available computational resources when possible requiring the minimum information from the user. In this paper we present the usage of data mining techniques for implement a prediction service that is used by the Grid Backfilling policy for estimate the job runtime. All the presented algorithms and techniques have been evaluated using a set of worklogs collected from the UK National Grid Service.

The rest of the paper is organized as follows: firstly, in the background section, we provide a discussion for the more relevant proposals concerning scheduling policies for HPC centers, including the natural evolution for the scheduling policies from local centers to more global approaches (such as: multi-site or grid architectures); secondly, we describe the Grid Backfilling meta-scheduling scheduling policy internals and the data mining techniques used for predict the run time for the jobs; and finally, we present the policy evaluation and the conclusions.

2. Background

In the area of job scheduling strategies for parallel processing the Gang Scheduling [6]and the backfilling policies have been the main goal of study these last years. Authors like Feitelson, S-H Chiang or Tsfrir have provided to the community many quality works regarding this topic. In [16]Skovira et. al presented the first paper about the EASY algorithm and its performance in the LoadLeveler system. General descriptions about the most used backfilling variants and parallel scheduling policies can be found in the report that Dror. G. Feitelson et al. provides in [7].

In the forthcoming scheduling architectures, like Grids or very heterogeneous computational resources, prediction techniques are having a crucial relevance due to user in most of the cases does not have enough information or enough skills for specify how long the job to run. Tsafrir et al. have presented several works analyzing the impact of the usage of prediction techniques rather user estimates in the backfilling policies [19]. They also formalized how the algorithm have to be extended for allow the deployment of this policies in real HPC centers.

In the current HPC infrastructures, centers may have more than one host managed by independent schedulers. In theses cases, can occurs that a job submitted to a *Host A* could start earlier in *Host B* of the same center. This global optimization has been proposed in [21]by Yue. The author proposes to apply a global backfilling within a set of independent hosts where each them is managed by an independent scheduler. The core idea of the presented algorithm is that user submits the jobs to an specific system, with an independent scheduler, and a global controller tries to find out if the job could be backfilled in another host of the center. In the case that a job can be backfilled in another host the controller will migrate the job to the chosen one. The idea is interesting due to they improve the global throughput of the center and decrease the response time of the applications. However, the algorithm requires the job runtime estimation provided by the user and not always he/she is able to provide it. This work it is only valid in very homogeneous architectures. If a job is finally executed to a different host from where the user submitted it, both configurations must be exactly the same. Otherwise, the user runtime estimation loses its validity due to it may would differ for the new host.

Similar this global backfilling approach, Sabin et al. [8]have presented the scheduling of parallel jobs in a heterogeneous multi-site environment. They propose carry out a global scheduling within a set different sites using a global meta-scheduler where users submit the jobs. Two different resource selection algorithms are proposed: the jobs are processed in order of arrival to the meta-scheduler, end each of them is assigned to the site with less instantaneous load; when the job arrives it is submitted to K different sites (each site schedules

using a conservative or aggressive backfilling), once the job is started in one site the rest of submissions are canceled (technique is called multiple requests, MR). This multi-site approach still does not take into account that when the local schedulers are scheduling using the backfilling optimization the run times for the jobs may differ between two different sites.

More centralized approaches have been proposed in the literature. For instance in [5]they analyze the impact of geographical distribution of Grid resources on the machine utilization and the average response time. A centralized Grid dispatcher that controls all resource allocations is used. The local schedulers are only responsible for starting the jobs after their allocation by the Grid scheduler. Thus all the jobs are being queued in the dispatcher while the size of job wait queues of the local centers is zero. A similar approach is the once presented by Schroeder et al. in [15], where they evaluate a set of task assignment policies using the same scenario (one central dispatcher).

In [14]Pinchak et al. describe a metaqueue system to manage jobs with explicit workflow dependencies. Here the placeholder scheduling creates a user-level metaqueue that interacts with the local schedulers and queues of the overlay meta computer. In this case, instead of push model, in which jobs are submitted from the meta queue to the schedulers, placeholder is based on the pull model in which jobs are dynamically bound to the local queues on demand.

In this paper we present the Grid Backfilling scheduling policy. It extends the ideas provided by Yue, in its Global Backfilling, and uses the ideas proposed by Tsfrir about the usage of prediction in backfilling scheduling policies. The global scheduler has a reservation table with all the computational resources available on the Grid, and tries to allocate the job to the earliest available allocation. The processes of finding out the allocation is based on a prediction of the job runtime. This prediction is done using a technique built on top of C45 classifying trees, predicts the time that a given job would take to complete on a given computational node given its statical characterization.

2.1 Data mining and prediction techniques

Data mining can provide to the scheduler estimations that can guide to the meta-scheduler to carry out a more intelligent scheduling decisions. In the presented work we have derived this information correlating the past executions of similar jobs in similar resources or with similar future load using decision trees for the prediction algorithm. In the literature several prediction methodologies have been proposed. However, as will be introduced in the later paragraphs, almost all of them have been basically focused in predicting the job performance variables (such as runtime) for local environments or sites. In [10]we have proposed a set of prediction techniques that provide to the users hints about

where to submit the jobs given a Grid architecture. For example, we estimate to the user how much a job will wait in a given broker before get executed with a specific job requirements (the number of processors, the input files etc.).

Works like those presented by Peter A. Dinda in [3], propose the usage of linear mathematical models for predicting the runtime for the applications submitted by the users. Dinda proposes the usage of the time series (AR, MA ARMA and ARIME) and a windowed mean for carry out host load estimations, and using such estimation for predict the job runtime. Recently, Yuanyuan presented in [22]new models, also based in time series, for predict the runtime for Grid applications. Other works [2]have proposed the usage of a state-transition model to characterize the resource usage of each program in its past executions.

In [4]Allen B. Downey characterizes the applications describing the speedup of the application on a family of curves that are parameterized by a job's average parallelism and its variance.

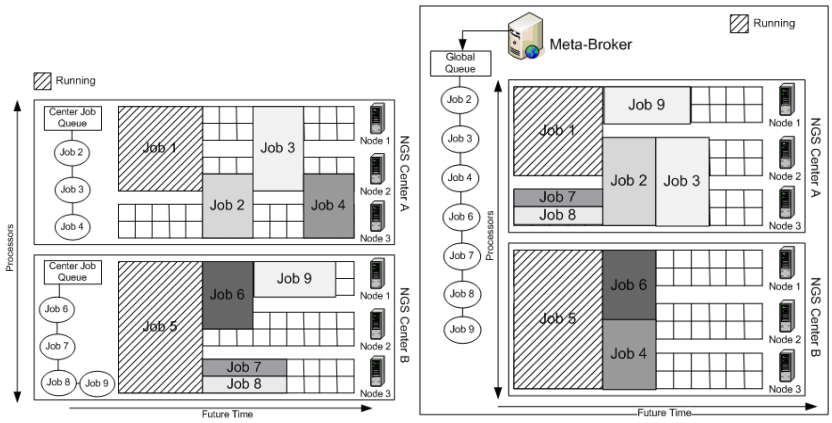
The other statistical approach that is also commonly used is the simulation. An example is the Dimemas simulator developed in the Barcelona Supercomputing Center [13]. The simulator reconstructs the execution trace file by estimating the time to execute each computation burst and communication burst.

Data mining techniques has became very popular during this lasts years. They are being used in a very wide range of areas, including the job performance prediction. Warren Smith et al. in [17]presented a first approximation to these techniques. Their work is mostly based on the work that previously Gibbons [9]presented before, which consisted in a static clustering of the workloads and a later usage of the mean and median inside this clusters.

3. The Grid Backfilling policy

As has been introduced in the previous section, in multi-site or Grid architectures, the local schedulers can optimize the performance (f.e: response time) for the jobs that are currently queued to the local system or site. However, in such environments, although they are doing all the best for achieve the highest performance in the local system or site, the schedule of the whole system might be improved substantially with a global scheduling. This situation is illustrated in the figure 1a. It presents two different centers with two different schedulers. Each scheduler is carrying out a local schedule using a SJF-Backfilling allocation policy. Although the presented reservation tables are closed to the optimal in each system, there are several holes that could be filled by allocating jobs of other centers. This second situation is illustrated in the figure 1b. Jobs are scheduled using a global backfilling scheduling policy to the different centers using the meta-scheduler architecture. In this picture can be observed how the holes have been used for backfilling jobs from other centers, and how the

global performance for the system (f.e: the throughput) and for the jobs (f.i: the wait time) have been improved.



(a) Traditional backfilling approach (b) Grid Backfilling approach

Figure 1: Traditional VS Grid Backfilling

We define the Grid Backfilling policy based on four different events: Job arrival, Job starts, Job completion and Job deadline missed. The last event occurs when a miss-prediction in the runtime of a job occurs. The meta-scheduler is warned from the local scheduler for update its reservation table according the new estimated runtime. In the introduction section, we have already introduced the work that Tsafrir et al. have presented concerning the usage of predictors in the SJF-Backfilling policy [19]. The approach that we have followed in the design of the predictor internals, how the scheduler manages the deadline misses and how the predictor and the scheduler interacts is based on the formalizations provided in the formers work.

The different elements that are evolved in the architecture are:

- The Job α that is submitted to the system is characterized by:
  - The static description of the job $req_\alpha = \{\partial_{\{1,\alpha\}},..,\partial_{\{n,\alpha\}}\}$ where each propierty is a pair value $\partial_{\{i,\alpha\}} = \{JobPropierty, value\}$.
- A set of computational resources $\{\sigma_1,..,\sigma_n\}$ available on the system. Like the job definition, each resource is descrived by:
  - A set of capabilities $cap_{\sigma_i} = \{\partial_{\{1,\sigma_i\}},..,\partial_{\{n,\sigma_i\}}\}$, where each capability is composed by a pair key value $\partial_{\{i,\sigma_i\}} = \{ResCapability, value\}$ (f.i: $\partial_{\{i,\sigma_i\}} = \{AvailableProcessors, 256\}$).

  - The computational resource is composed by a set of nodes $\{\partial_1,..,\partial_n\}$, where each node is composed by a set of processors $\left\{\rho_{\{1,\partial\}},..,\rho_{\{n,\partial\}}\right\}$

- The Prediction Service γ.
- The Meta-Scheduler β with its own global reservation table.

In the following subsections we present how the meta-scheduler behaves in each of the four enumerated events.

3.1 Job Arrival

When a job α with requirements req_α [1] is submitted to the meta-scheduler, the following scheduling algorithm is carried out:

- It pushes the job α to the global wait queue.
- It computes all the possible allocations to the current free nodes $outcomes_\alpha$ where the job would be able to run. Note that:
  - The outcome is composed by set of nodes $\{\partial_1,..,\partial_n\}$ of the same computational resource σ_i.
  - The resource σ_i satisfies the requirements of the job $\forall\partial \in cap_{\sigma_i} sat(\partial, req_\alpha)$
- For each of the $outcomes_\alpha$ where the job would be able to run:
  - It queries to the prediction service the estimation r_α of the runtime for the job α in the computational resource σ of the outcome.
  - It stores the estimation and the allocation into a local hash table Ω.
- If Ω is not empty, the meta-scheduler will choose the allocation following the backfilling algorithm $allocation_\alpha = \{\partial_1,..,\partial_n\}$ in Ω that maximizes the response time for the job. If the job can be backfilled or started (it would be the first of the wait queue), the job is deleted from the wait queue and started to run.
- If $allocation_\alpha$ is not null. It will start the job. (see 3.2)

The priority used in the global queue is the LXWF presented by S.-H. Chiang in [1]. We have chosen this backfilling derivate due to as has been proved

[1] For carry out this allocation only static requirements are need from the user: the number of processors, the executable start and the input/output files. The dynamic requirements, such as the runtime, will be estimated by the prediction service

in several works [19][1]it achieves good performance and the jobs do not suffer starvation. However, the Shortest-Job-Backfilled-First policy proposed by Tsfrir in [19]is also a valid candidate for the job selection. The evaluation results shown that both achieved similar performance.

3.2 Job starts

When a job α is chosen to start to the allocation $allocation_\alpha = \{\partial_1, .., \partial_n\}$:

- The meta-scheduler contacts to the scheduler that manages the resource specified in the allocation, provides the job credentials with it requirements and requires to start the job.
- When the job starts to run, it updates the information of the global reservation table.
- It contact to the prediction service and provide all the details the job submission, including the global id assigned to the job. This global id will be used for match the job information provided by the broker once the job is finished with its historical data base.

In this study we have considered that the local scheduler accepts the meta-scheduler resource selection about where the job has to run (nodes $\{\partial_1, .., \partial_n\}$). Thus, the local scheduler only make the resource allocation and has no say in the matter of how the scheduling is done. However, in future extensions of this policy would include interactions between all the scheduling layers of the architecture. For example, the meta-scheduler could contact to the local schedulers and start a negotiation for the acceptance of the proposed allocation

3.3 Job completion

Once the job has been executed, the scheduler contacts to the meta-scheduler providing the feedback for the job execution. This feedback includes information about variables for the job execution, for instance: run-time, disk used, memory used, final status etc. When the meta-scheduler receives a job completion notification:

- It will provide this information to the Prediction Service for allow its future usage by the prediction techniques.
- Following the algorithm presented in 3.1 it will try to allocate the head job of the global queue. If there are enough computational resources the job will start (3.2).
- Following also the same algorithm it will try to backfill (3.2) the jobs queued in the global queue. The backfilling variance used in the more

aggressive once, using only one reservation. That means that jobs will be backfilling only if the start time for the first job of the queue is not delayed.

3.4 Job deadline missed

In those cases that the prediction service made a wrong prediction underestimating the runtime for the job, the local scheduler will notify to the metascheduler that a deadline missed has been reached. In normal backfilling policies, this job would be killed due to it would interfere with the execution of the following job. However, using prediction we can not take this approach. In the model proposed by Tsafrir the estimate runtime for the job is extended an t_α time. This amount of time is computed by the deadline miss managers. We have tested two times of deadline miss managers:

- Gradual Deadline miss manager: extends the job prediction runtime gradually.
- Exponential Deadline miss manager: extend the job runtime prediction in following an exponential distribution.

4. The runtime predictions

We have already emphasized that data mining techniques [11]are especially interesting because they can be applied to a very different kind of data independently of its nature. On the contrary, some statistical algorithms require the normality of the input variables for assure the correctness of the resulting conclusions. Moreover, the most interesting of their characteristics is that some of them they have an autonomic learning mechanism, and they are able to derive or discover new knowledge without the necessary interaction of a third part (user, expert or other software component).

There are several techniques can be used for the performance prediction, for example Bayessian Networks, C.45 trees, ID3 trees, K-Means or X-Means. In the presented work we have used C45 trees and discretization techniques for predicting the job runtimes. In this section we present how we have constructed and validated the model that has been used later in the grid backfilling evaluation.

4.1 The prediction model

The prediction model is built on top of the C45 decision trees algorithm. Its goal is to predict the runtime for a submitted application using the static information provided by the user. The model has been constructed and validated using the Weka [12]software and following the next steps:

- The log file used for generate the model has been preprocessed.
- The continuous variables have been converted to nominal variables.
- We have carried out an study for the selection of the response and input variables for the tree model.
- We have constructed of the decision tree validated model.

4.1.1 Log preprocessing. The log contains variables concerning the job performance (like percentage of processor used or virtual memory used), variables concerning the job identification (Grid node, job id, job owner and queue), variables concerning the data that has been used for the job execution (output, input and error paths and the working directory), and finally, variables concerning the dates of the events for the job (start time, end time, queuing time). For lack of space we do not provide a detailed analysis about the characteristics of the NGS workload used in the simulations and workload creation. However we already carried a deeper study of these traces, a characterization of them can be found in [10].

Tsafrir presented a very interesting work about detecting and deleting workload anomalies in [20]. The authors highlighted a set of phenomena, like workload flurries, that should be taken into account when analyzing workloads and proposed a set of techniques for identifying and filtering such anomalies. Before constructing the model this anomalies have been localized and deleted in the log.

4.1.2 Continuous variables discretization. We have discretized all the continuous variables into nominal values. This process has been iterated several times until find the appropriate bins according to the performance obtained in the evaluation of the resulting trees. Two different types of configurations have been tested:

- First, varying the number of been in which the continuous variables are discretized. Initially we tested several number of bins from 3 till 10. However, the final number of bins has been chose using the *findNumBins* option for the *unsupervised attribute disctretize* filter of the Weka. The interesting application of this methodology is that can be carried out in the prediction service without any external supervision.
- Secondly, varying the criteria of discretization. The options of *desiredWeightOfInstancesPerInterval*, *makeBinary* and *useEqualFrequency* for the *unsupervised attribute disctretize* filter.

The discretization for the continuous variable run time is shown in the table 1. The discretization for the rest of continous variables is not shown due to they were rejected in the construction of the C45 tree.

Interval Predicted	**Numerical prediction**
(-inf-12390]	12390
(12390-24780]	24780
(24780-37170]	37170
(37170-49560]	49560
(49560-inf)	49560

Table 1: Run time predictions

TP-Rate	*FP-Rate*	*Precision*	*Recall*	*F-Measure*	*Class*
0.966	0.301	0.92	0.966	0.942	(-inf-12390]
0.359	0.06	0.4	0.359	0.379	(12390-24780]
0.388	0.058	0.324	0.338	0.331	(24780-37170]
0.113	0.001	0.77	0.113	0.194	(37170-49560]
0.388	0.002	0.659	0.335	0.444	(49560-inf)

Table 2: Acurracy by class

4.1.3 Selection for the input variables,. The set of variables that have been chosen for built the C45 decision tree have been: executable name, number of requested processors, user id, group id and the site where the job would be submitted and the response variable runtime.

Other variables, such as the input files, working directory or output files, have been ruled out for two main reasons: the sizes of the pruned trees were big and the non availability of the variables at the submission time. Using the variables concerning the data used in the job execution resulted in a trees with a thousands of nodes with a low recall and precisions.

4.1.4 Construction and tree validation. The resulting tree used for predicting the job runtime has 130 nodes and 120 leaves. The variables that provide more information (those that are in the upper nodes) are mainly the user credential and the number of processors used in the job execution. The resulting model has been tested using the cross validation technique with ten folds. The resulting model has classified correctly 86% of the instances. The performance of the constructed model is presented in the detailed accuracy by class presented in the table 1, and the confusion matrix 3. As can be observed in the results obtained in the cross validation analysis the model has shown good behaviour. Moreover, the main errors in the instance classification are only wrong classification between the classes *a, b and c* and we exepcted that such kind of errors should have high impact on the schedule performance.

a	*b*	*c*	*d*	*e*	*Classified as*
10354	186	160	7	15	a = (-inf-12390]
482	493	392	0	5	b = (12390-24780]
235	451	352	1	3	c = (24780-37170]
95	93	175	47	7	d = (37170-49560]
91	9	9	6	58	e = (49560-inf)

Table 3: Confusion matrix

4.2 The prediction mechanism

The presented C45 tree is providing a prediction for the interval of time that a given job will need to be completed. However, the algorithm presented in the proceeding section requires a numerical value for carrying the scheduling. The prediction service, once the submitted job has been classified in one of the 5 presented classes, uses the hash 2 for return the numerical value for the estimation. The upper bound of the estimated class is used due we want to avoid overestimating the runtime for the job.

5. The evaluation

In the experiments we have simulated four different scenarios: in the first three scenarios we have simulated independently the workload of each center using the SJF-Backfilling variant, thus we have evaluated how the different workload would behave with such policy; the last scenario has consisted on simulating the Grid Backfilling policy and the workload generated with the fusion of the four different workloads. The original log traces were collected form the NGS during five different month. In the NGS the users accessed to the different computational resources using the Globus infrastructure and they had to decide in resource their jobs had to be executed. Thereby there were no global scheduling.

The evaluation of this policy has been tested using the simulation methodology. The simulation has used a model that characterizes the computational resources for NGS architecture (see its characteristics in [10]) including the centers of Oxford, Manchester, LR and Leeds, and has simulated the Grid Backfilling policy presented in the previous sections. We have used the Alvio-simulator that is a C++ event-driven simulator similar to the EASY simulator implemented by Tsafrir et al. in the paper [18], but modeling the architectures (the resources and its capabilities) for the centers.

The statistical analysis for the error in the runtime prediction using the presented prediction scheme (4) in the simulation has shown an average error of 160% and the median error is -1.7%. Our experience in the prediction errors evaluation has shown this prediction performance values can be used to under-

Center	Estimator	BSLD	SLD	WaitTime	Backfilled Jobs/Day
Manchester	Mean	1,1	1,9	247	0,3
	STDev	1,23	1,4	841	0,12
	95_{th} Percentile	1,4	1,8	123	1
Leeds	Mean	2,4	2,5	4266,2	0,37
	STDev	3,6	3,8	3150	1,9
	95_{th} Percentile	4,3	4,21	19856	2,4
LR	Mean	2,8	3,03	1182	2,3
	STDev	23	27,1	4307,3	1,2
	95_{th} Percentile	2,3	2322	6223	3
Oxford	Mean	4,04	5,9	6390	1,3
	STDev	29	89,2	19420	4
	95_{th} Percentile	9,1	10,1	54750	8
GridBackfilling	Mean	1,12	1,17	153,32	3,5
	STDev	0,5	0,45	1200,25	5,1
	95_{th} Percentile	1,4	1,9	2200.25	14

Table 4: Performance Variables for each workload and the Grid Backfilling

stand how well the predictor behaves. However, the real benefit of the usage of a given prediction technique relies on the performance achieved on a specific scheduling policy. As has been stated in the simulation results the errors that prediction service had during the simulation were acceptable and demostrated that the presented architecture can be deployed in real systems.

Table 4 presents the average, standard deviation and $95_{th}Percentile$ for the variables Slowdown, Bounded Slowdown, Wait time and Backfilled Jobs per day for each center independently and for the global architecture with the Grid Backfilling policy. It is clear that a global scheduling carried out in top of all the centers improves qualitatively the service provided and reduces substantially the response time for the submitted jobs. The average wait time of all the centers has been reduced qualitatively. For instance the Manchester average wait is almost two times bigger than the average wait time experimented in the Grid Backfilling. Furthermore, the average wait time of the Oxford center is around forty times bigger than the Grid Backfilling once. The variable *Backfilled Jobs/Day* shows how the global backfilling approach give more chances to the jobs to start earlier rather than using independent scheduling per centers. The percentile 95_{th} shows how the ratio of backfilled jobs is at minimum two times bigger than the once achieved in the independent configurations.

6. Conclusions and future work

In this paper we have presented the usage of backfilling scheduling techniques in distributed environments using a global reservation table and prediction techniques. We have shown how the analyzed policy provides a uniform

access to the whole computational resources available in the environment and how it achieves good performance optimizing the usage of all the available resources. The policy has been evaluated using logs collected from the National Grid Service that contained the jobs that user submitted during four months to each of the four centers: Oxford, Leeds, Manchester and LR. The average wait and average wait time for all the jobs submitted to the Grid has been reduced one order of magnitude respect to the average wait time that the job of the same workloads had in the original architecture.

The usage of prediction techniques in scheduling policies has been proposed in several works for usage of such prediction rather user estimates in the scheduling decisions. In this paper we have also presented how data mining techniques are very suitable for predicting the run time for Grid environments, and how they can be used in the Grid Backfilling with high success. We have presented a prediction methodology based on classification trees C45 that having as an input static information about the job (executable name and number of requested processors) predicts the run time for this job in a given site or center. The predictions have shown only a median of error of -1.7% respect the original runtime, and a mean of 160%. The real benefit of the usage of a given prediction technique has been proved on the performance achieved scheduling policy evaluation. The usage of such techniques is specially important due to abstarct the user to the underlying complexities of the system, and allows to the scheduler to decide where to submit jobs having an estimation of how long this job would run in each of the possible allocations.

Future extensions of this policy will include interactions between all the scheduling layers of the architecture. In this extension, the meta-scheduler would reach an agreement with the local schedulers in terms of where the job can be finally allocated. On the other hand, the current allocation selection algorithm is focused on optimizing the response time for the submitted applications and does not take into account other of its requirements. We would like to take in to account economic criterias, soft and hard requirements or resource matching criteria in the allocation selection.

References

[1] 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.

[2] M. V. Devarakonda and R. K. Iyer. Predictability of process resource usage : A measurement based study on unix. IEEE Tans. Sotfw. Eng., pp. 15791586, 1989

[3] P. Dinda. Online prediction of the running time of tasks. Cluster Computing SIGMETRICS/Performance, pages 225236, 2002.

[4] A. B. Downey. Using queue time predictions for processor allocation. 3rd JSSPP, Lecture Notes In Computer Science; Vol. 1291:35 57, 1997.

[5] C. Ernemann, V. Hamscher, , and R. Yahyapour. Benefits of global grid computing for job scheduling. 5th IEEE/ACM International Workshop on Grid Computing, 2004.

[6] D. G. Feitelson and M. A. Jette. Improved utilization and responsiveness with gang scheduling. pages 238261, 1997.

[7] 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.

[8] S. Gerald, K. Rajkumar, R. Arun, and S. Ponnuswamy. Scheduling of parallel jobs in a heterogeneous multi-site environment. JSSPP, 2003.

[9] R. Gibbons. A historical application profiler for use by parallel schedulers. Job Scheduling Strategies for Parallel Processing 1997, 1997.

[10] A. Goyenechea, F. Guim, I. Rodero, G. Terstyansky, and J. Corbalan. Extracting performance hints for grid users using data mining techniques: a case study in the ngs. Mediterranean Journal: Special issue on data mining, 2006.

[11] J. Han and M. Kamber. Book: Data mining: Concepts and techniques. Book, 2001.

[12] G. Holmes, A. Donkin, and I. Witten. Weka: A machine learning workbench. In Proc 2nd Australia and New Zealand Conf. on Intelligent Information Systems, 1994

[13] T. C. Jess Labarta, Sergi Girona. Analyzing scheduling policies using dimemas. 3rd Workshop on environment and tools for parallel scientific computation, 1997.

[14] C. Pinchak, P. Lu, and M. Goldenberg. Practical heterogeneous placeholder scheduling in overlay metacomputers: Early experiences. Job Scheduling Strategies for Parallel Processing, pages 205228, 2002. Lect. Notes Comput. Sci. vol. 2537.

[15] B. Schroeder and M. Harchol-Balter. Evaluation of task assignment policies for supercomputing servers: The case for load unbalancing and fairness. Cluster Computing 2004.

[16] 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.

[17] 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.

[18] D. Tsafrir, Y. Etsion, , and D. G. Feitelson. Modeling user runtime estimates. In the 11th JSSPP ,Lecture Notes in Computer Science, Vol.3834:pp. 135, 2006.

[19] D. Tsafrir, Y. Etsion, and D. G. Feitelson. Backfilling using system-generated predictions rather than user runtime estimates. In the IEEE TPDS, 2006.

[20] 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.

[21] J. Yue. Global backfilling scheduling in multiclusters. Asian Applied Computing Conference, AACC 2004, pages pp. 232239, 2004.

[22] Y. Zhang, W. Sun, , and Y. Inoguchi. Cpu load predictions on the computational grid. Cluster and Grid computing, 2006.