

Modeling the impact of resource sharing in Backfilling Policies using the Alvio Simulator

F. Guim ¹, J. Corbalan ² and J. Labarta ³

Barcelona Supercomputing Center

Jordi Girona 13, Barcelona, Spain

¹francesc.guim@bsc.es

²julita.corbalan@bsc.es ³jesus.labarta@bsc.es

Abstract—Job scheduling policies for HPC centers have been extensively studied during these last years, specially backfilling based policies. Almost all of these studies have been done using simulation tools. These tools evaluate the performance of scheduling policies using the workloads and the resource definition as an input.

To the best of our knowledge, all the existent simulators use the runtime (either requested or real) provided in the workload as a basis of their simulations. However, the runtime of a job, even executed with a fixed number of processors, depends on runtime issues such as the specific resource selection policy used for allocate the jobs or the resource jobs requirements.

This paper is the first part of a more complex research project that analyzes the impact in the system performance of considering the resource sharing of running jobs. With this purpose we have included in our job scheduler simulator (the Alvio simulator) a performance model that estimates the penalty introduced in the application runtime when sharing the memory bandwidth. Experiments have been conducted with two resource selection policies and we present both the impact from the point of view of global performance metrics, such as average slowdown, and per job impact such as percentage of penalized runtime.

I. INTRODUCTION

Several works focused on analyzing job scheduling policies have been presented during the last decades to evaluate the performance of these policies with specific workloads in HPC centers. A special effort has been devoted on evaluating backfilling-based policies because they have demonstrated to reach the best performance results (i.e: [1] or [2]). Almost all of these studies have been done using simulation tools. To the best of our knowledge, all the existent simulators use the runtime (either requested or real) provided in the workload as a basis of their simulations. However, the runtime of a job depends on runtime issues such as the specific resource selection policy used or the resource jobs requirements.

The aim of this paper is to present a first attempt to evaluate the impact of considering the penalty introduced in the job runtime due to resource sharing (such as the memory bandwidth) in system performance metrics, such as the average bounded slowdown or the average wait time, in the backfilling policies. To do that, we have developed a job scheduler simulator (Alvio simulator) that, additionally to traditional features, implements a job runtime model that tries to estimate the penalty introduced in the job runtime when sharing resources. In this paper we have just considered in

the model the penalty introduced when sharing the memory bandwidth of a computational node.

We have simulated the impact of memory bandwidth usage in the scheduling policy using two workloads trace files (CTC and SDSC) obtained from the Dror Feitelson Parallel Workload Archive. Due to the original traces did not contained the jobs memory bandwidth usage, we have created it synthetically. For each workload three scenarios have been defined: one with 80% of jobs having a high memory bandwidth usage (HIGH), one with 50% of jobs having a *normal* memory bandwidth usage (MED), and one with 10% of jobs having a high memory bandwidth usage (LOW). These scenarios have been evaluated with the SJF-Backfilling policy, with two resource selection policies (First fit and First Contiguos fit), and with and without considering the penalty of the resource usage sharing.

Results show a clear impact of system performance metrics such as the average bounded slowdown or the average wait time. Furthermore, other interesting collateral effects such as a significant increment in the number of killed jobs have appeared. Moreover, as it is described later, the impact in these performance metrics is not only quantitative. We have stated that depending on the model used in the evaluation (considering the resource usage or not), the researcher could achieve different conclusions.

The rest of the paper is organized as follows: in section II we present the related work for some of the most representatives works of this topic; in section III we provide the formalizations for model that is used in the simulations; later, in section IV the experiments that we have evaluated are presented; in section VI the experiments evaluations are presented; and finally in section VII the collusions and future work are presented.

II. RELATED WORK

Authors like Feitelson, Uwe Schwiegelshohn, Rudolph, Calzarossa, Downey or Tsafrir have modeled logs collected from large scale parallel production systems. They have provided inputs for the evaluation of the different systems behavior. Feitelson also presented several works concerning this topic, among others, he has published about logs analysis for specific centers [3], general job and workload modelization [4], and together with Tsafrir, about detecting workload anomalies and flurries for filter them from the workloads

[5]. Calzarossa has also contributed with several workload modelization surveys [6]. Models of moldable jobs have been described in works of authors like Cirne et al. in [7], by Sevcic in [8] or by Downey in [9]. Lublin proposed a very detailed model for the rigid jobs including their arrival pattern and runtimes in [10].

In this area, a recent work, presented by Tsafirir et al. has been very valuable for evaluating the behavior for the backfilling scheduling policies and for those policies that need the user runtime as an input. In [1] they have modeled how the users estimate the jobs that are submitting to the centers.

The other outstanding topic in this area are the scheduling policies for the HPC Centers. The gang scheduling and the backfilling [11] policies have been the main goal of study these recent years. Similar to the research in the workload modeling, authors or S-H Chiang or Feitelson have provided to the community many quality works regarding to this topic. In [12] general descriptions about the most used backfilling variants and parallel scheduling policies are presented. Moreover, deeper description of the conservative backfilling algorithm can be found in [2], where the authors present its characterization and how the priorities that can be used when choosing the appropriate job to be scheduled.

To the best of our knowledge, none of existent simulation tools that evaluate the performance of the workloads in the backfilling policies have evaluated the resource sharing in the simulations. Furthermore, none of the previously enumerated research works have modeled the resource job usage in their evaluations.

III. THE ALVIO SIMULATOR

All the experiments have been conducted using the C++ event-driven Alvio simulator [13]. It simulates the execution of a workload composed by a set of HPC jobs under the control of a resource manager (i.e: LoadLeveler or Maui). It implements a run-to-completion job scheduling policy, such as FCFS or Backfilling policies, and a resource selection policy (how the job processes are mapped to the processors), such as continuous allocation, in a defined architecture (a cluster of SMPs in this paper). Furthermore, the simulator allows to simulate distributed architectures. Thus it can be configured like a unique HPC center, or like a distributed environment composed of several independent centers.

Conceptually, it's divided in three main parts: the simulator engine, the scheduling policies (including the resource selection policies), and the computational resource model. Each simulation receives as parameter the workload, the job scheduling policy and the system architecture definition. The architecture model allows to specify any configuration based on cluster architectures, where the host is composed by a set of computational nodes, where each node has a set of consumable resources (processors, Memory Bandwidth, Ethernet Bandwidth and Network bandwidth).

The job scheduling policy uses as input a set of job queues and a Reservation Table (RT). The RT represents the status of the system at a given moment and it is linked to the

architecture. It has the running jobs allocated to the different processors during the time. One allocation is composed by a set of buckets that indicate that a given job α is using the processors β from $T_{startTime}$ until $T_{endTime}$. Depending to the policy configuration, the scheduling policy will allocate temporarily the queued jobs (for instance for estimate the wait time for the jobs).

In this section we provide a formalization for the different elements managed by the Alvio simulator and that are part of the evaluated model. First we formalize the entities that are evolved in the scheduling algorithm: the job and the available resources (defined in the RT). Second, we formalize the main characteristics of a scheduling policy and the different elements that it uses for computing the schedule.

A. The Scheduling entities

The main entities that are involved in the scheduling architecture are:

- The Job α that is submitted to the system is characterized by ¹:
 - The static description of the job $chars_\alpha = \{c_{\{1,\alpha\}}, \dots, c_{\{n,\alpha\}}\}$ where each property is a pair value $c_{\{i,\alpha\}} = \{JobProperty, value\}$.
(f.i: $c_{\{i,\alpha\}} = \{Application, CPMD\}$).
 - The requirements for the job $req_\alpha = \{r_{\{1,\alpha\}}, \dots, r_{\{n,\alpha\}}\}$ where each property is a pair value $r_{\{i,\alpha\}} = \{JobRequirement, value\}$.
(f.i: $r_{\{i,\alpha\}} = \{MemoryRequired, 16MB\}$).
- A set of computational resources $\{\sigma_1, \dots, \sigma_n\}$ available on the system. Like the job definition, each resource is described by:
 - A set of capabilities $cap_{\sigma_i} = \{\partial_{\{1,\sigma_i\}}, \dots, \partial_{\{n,\sigma_i\}}\}$, where each capability is composed by a pair key value $\partial_{\{i,\sigma_i\}} = \{ResCapability, value\}$
(f.i: $\partial_{\{i,\sigma_i\}} = \{AvailableNodes, 256\}$).
 - The computational resource is composed by a set of nodes $\{n_1, \dots, n_n\}$. Each node:
 - * is composed by a set of processors $\{p_{\{1,\partial\}}, \dots, p_{\{n,\partial\}}\}$
 - * has a set of resources with a certain capacity:
 $res_{\sigma_i, n_j} = \{r_{\{1,\sigma_i, n_j\}}, \dots, r_{\{k,\sigma_i, n_j\}}\}$
(f.i: $\partial_{\{i,\sigma_i\}} = \{MemoryBandwidth, 6000MB/S\}$).

B. The Job Scheduling Policy

The scheduling policy is characterized by:

- The job selection algorithm. That given a set of jobs $\{\alpha_i, \dots, \alpha_{i+n}\}$ and a Reservation Table decides which job to start.
- The job resource allocation policy. That given a set of free processors $\{p_{\{1\}}, \dots, p_{\{n\}}\}$ and a given job α_i with a set of requirements req_α decides to which processors the job will be allocated.

¹We use the SWF plus an extension to include the resource usage in terms of memory bandwidth

- The running queue that contains the set of jobs $\{\alpha_{r1}, \dots, \alpha_{rm}\}$ that are currently running in the computational resource.
- The wait queue that contains the set of jobs that are currently waiting for be executed $\{\alpha_{w1}, \dots, \alpha_{wk}\}$.
- The Reservation Table that contains the mapping for all the jobs that are running in the processors that are currently used. More formally:
 $Allocations = \{\forall P \in \{p_{\{1\}}, \dots, p_{\{n\}}\} \Rightarrow \{a_{\alpha_{ri}, P}\}\}$
 where each cpu allocation is defined by:
 $allocation\{\alpha_i, P\} = t_0, t_1$ and indicates that the job α is allocated to the processors P from the time t_0 until t_1 . The allocations of the same processors must satisfy that they are not overlapped during the time:
 $\forall a \in \{a_{\alpha_i, P}, \dots, a_{\alpha_{i+k}, P}\} \Rightarrow \{a_{\alpha_i, P}, \dots, a_{\alpha_{i+k}, P}\} \cap a = O$
 The scheduling policy decides where to allocate the jobs based on the mapping that is shown in the reservation table.

As we have already introduced, the Reservation Table stores the information concerning how the running jobs are currently mapped into the architecture. Figure 1 provides an example of a possible snapshot of the reservation table at the point of time t_1 . Currently, there are three jobs running in three different job allocations:

$$a_1 = \{ \{ p_{\{1, node_1\}}, p_{\{2, node_1\}}, p_{\{3, node_1\}} \}, t_0, t_2 \}$$

$$a_2 = \{ \{ p_{\{4, node_1\}}, p_{\{5, node_1\}}, p_{\{1, node_2\}}, p_{\{2, node_2\}} \}, t_1, t_3 \}$$

$$a_3 = \{ \{ p_{\{5, node_2\}} \}, t_1, t_4 \}$$

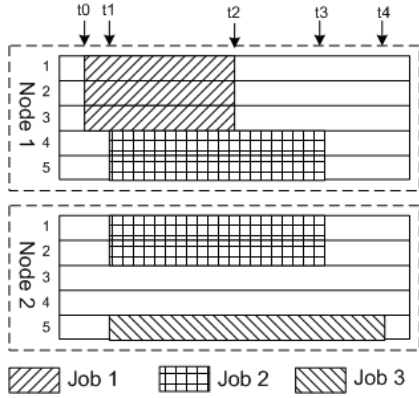


Fig. 1: Reservation Table Snapshot

C. Modeling the conflicts

The model that we have presented in the previous section has some properties that allows to simulate the behavior of a computational center with more details. Different resource selection policies can be modeled. Thanks to the Reservation Table, the scheduler knows at each moment which processors are used and which are free. Thus, while simplest models can not, our model allows to select specific processors for a given job (different resource selection policies). For example: a first continuous policy, that allocates consecutive processors to the jobs can be simulated.

Using the req_α for all the allocated jobs, the resource usage for the different resources available on the system has been modeled. Thus, using the Reservation Table, we are able to compute, at any point of time, the amount of resources that are being used in each node. Thereby, if in a interval of time $[t_0, t_1]$ the resource ∂_j of a node n_i used by the allocated processes is higher than capacity of the resource a penalty to of time $t_{penalty}$ proportionally to the resource consumption and the time $t_1 - t_0$ is added to each process. The main goal of this extended model is to include the resource usage model in the reservation table.

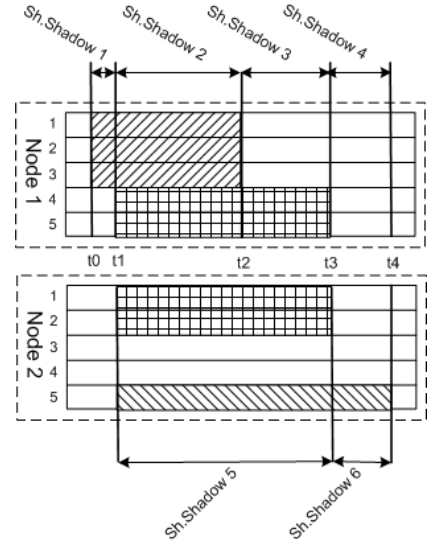


Fig. 2: Example of Shared Shadows definition

In this extended model, when a job α is allocated from $[t_{\alpha,0}, t_{\alpha,1}]$ to the reservation table to the processors p_i, \dots, p_{i+k} that belong to the nodes n_j, \dots, n_{j+k} , we check if any of the resources $\partial_{\{i, \sigma_i\}}$ that belong to each node is overloaded during any part of the interval $[t_{\alpha,0}, t_{\alpha,1}]$. In the affirmative case a runtime penalty will be added to the jobs that belong to the overloaded subintervals.

Before introducing the model used for calculating the penalties, we formalize some concepts used later:

1) *The Shared Shadows*: A *Shared Shadow* is an interval of time $[t_x, t_y]$ associated to the node n_i where all the processors of the node satisfies that: or no process is allocated to the processor or a process is fully included in the given interval in the processor. More formally, in the shadow interval time:

$$\forall p \in procs(n_i) \Rightarrow \{ \exists job_\alpha | runs([t_x, t_y], p, job_\alpha) \}$$

$$\vee \{ \neg \exists job_\alpha | runs([t_x, t_y], p, job_\alpha) \}$$

Where the function *runs* returns true if the job is running on the processor during the hole interval of time provided.

Figure 2 shows the Shared Shadows that would be defined in the example introduced in the figure 1. In this example six

different Shared Shadows would be defined, four in the first node and two in the second node. For instance: the Shadow 5 is defined in the interval $[t_1, t_3]$ in the *node 2* and contains the *job 2* in the processors 1 and 2, the *job 3* in the processor 5 and the processors 3 and 4 remain idle.

2) *The penalty function*: This function is used to compute the penalty that is associated to all the jobs that belongs to a given shared shadow due to resources overload. The input parameters for the function are:

- The interval associated to the shadow $[t_x, t_y]$.
- The jobs associated to the Shared Shadow $\{\alpha_0, \dots, \alpha_1\}$
- The node n_i associated to the shadow with its resources capacity.

The function used in this model is defined as:

$$\forall \sigma_k \in res_{\sigma_{k=1..j}, n_i} \left\{ usage_{\sigma_k} = \sum_{\alpha} req_{\{\alpha, \sigma\}} \right\} \quad (1)$$

$$Penalty = \sum_{\sigma_{k=1..j}} \frac{capacity_{\sigma_k}}{\min(usage_{\sigma_k}, capacity_{\sigma_k})} - 1 \quad (2)$$

$$PenalizedTime = 1 + (t_y - t_x) * Penalty \quad (3)$$

First for each resource in the node the resource usage for all the jobs is computed. Second, the penalty for each resource consumption is computed. This is a linear function that depends on the overload of the used resource. Thus if the amount of required resource is lower than the capacity the penalty is zero, otherwise the penalty added is proportionally to the fraction of demand and availability. Finally, the penalized time is computed by multiplying the length of the Shared Shadow and the penalty. This penalized time is the amount of time that will be added the node penalized time to all the jobs that belong to the shadow.

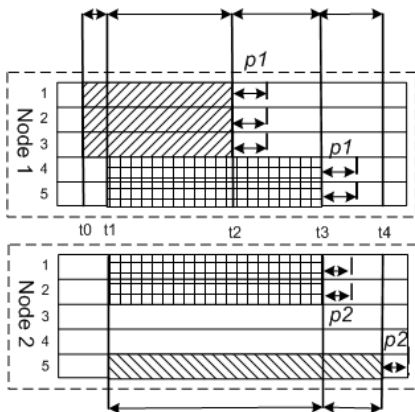


Fig. 3: Example of Shared Shadow Penalties

D. Computing the Reservation Table Penalties

In the previous two subsections III-C1 and III-C2 we have introduced what are the Shared Shadows and how the penalties for this shadows are computed to the all the jobs that are allocated to a given set of nodes. In this subsection we

introduce how the job penalized time is computed. Figure 3 shows the penalty associated to each shadow.

The algorithm is applied to the reservation table with all the allocated jobs and with no penalties assigned. It has to be applied each time that a job is allocated in the reservation table and it is divided in three main steps:

- 1) The shared shadows for all the nodes and the penalized times associated to each of them are computed. In the example, only the two Shadows 2 and 5 have associated penalties. That means that during the intervals of both shadows the resource demand for the allocated jobs was higher than the available capacity for the resources.
- 2) The penalties of each job associated to each node are computed adding the penalties associated to all the shadows where the job runtime is included. For example, as can be observed in the figure 4 , the penalty for the *job 2* in the *node 1* will be $p1$. Thus, if the job *alpha* is allocated to the the nodes n_i, \dots, n_{i+k} , it will have n different amount of penalized times ($\{ptime_1, \dots, ptime_n\}$).
- 3) The penalty time that will be finally assigned to the job will be the maximum of the penalties that job would have in the different nodes. Figure 4 shows how the penalty is assigned to each job. The penalty for the *Job 2* is $p1$ due to the penalty that the job suffers for its allocation to the node one ($p1$) is bigger than the penalty that the job will suffer for be allocated to the node two ($p2$). The finalization time for all the jobs is updated in the Reservation Table according to the computed penalties.

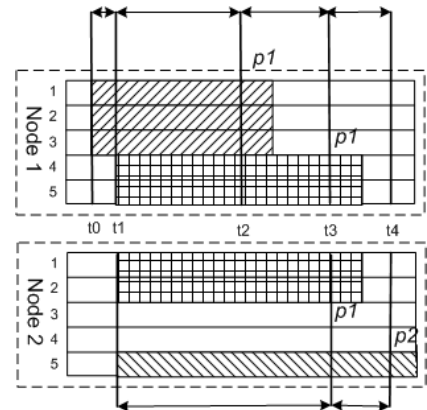


Fig. 4: Example of Final Job Penalties

IV. EXPERIMENTS

In this paper we evaluate the effect of considering the memory bandwidth usage when simulating the SJF-Backfilling policy under several workloads and resource selection policies. Two different workloads from the Feitelson workload archive [14] have been used. For each of them we have generated three different scenarios: with high (HIGH), medium (MED), and low (LOW) percentage of jobs with high memory demand.

V. EXPERIMENTS CHARACTERIZATION

A. Workloads

For the experiments we have used the cleaned [15] versions of the workloads SDSC Blue Horizon (SDSC-BLUE) and Cornell Theory Center (CTC) SP2. For the evaluation experiments explained in the following section, we have used the first 10000 jobs of each workload. Based on these workload trace files, we have generated three variations for each one with different memory bandwidth pressure:

- **HIGH:** 80% of jobs have high memory bandwidth demand, 10% with medium demand and 10% of low demand.
- **MED:** 50% of jobs have high memory bandwidth demand, 10% with medium demand and 40% of low demand.
- **LOW:** 10% of jobs have high memory bandwidth demand, 10% with medium demand and 80% of low demand.

B. Architecture

For each of the workloads used in the experiments we have defined an architecture with nodes of four processors, 6000 MB/Second of memory bandwidth, 256 MB/Second of Network bandwidth and 16 GB of memory. Additionally to the SWF traces with the job definitions we have extended the standard workload format to specify the resource requirements for each of the jobs. Currently, for each job we can specify the average memory bandwidth required (other attributes can be specified but are not considered in this work).

Based on our experience and the architecture configuration described above, we defined that a *low memory bandwidth demand* consumes 500 MB/Second per process; a *medium memory bandwidth demand* consumes 1000 MB/Second per process; and that a *high memory bandwidth demand* consumes 2000 MB/Second per process. As a future work we plan to consider other resources such as network bandwidth.

Based on the job runtime model used, the maximum penalty in the jobs runtime for a given shadow will take place in those cases that the amount memory bandwidth requested per processors would be 2000 MB/Sec. Thus the penalty would be $\frac{8}{6}$. Future research will include the effect of using functions with high rates of penalties or jobs with high resource requirements per processors against the available per node (i.e: 5500 MB/Second per process).

C. Experiments

For each of the different scenarios, the following configurations have been evaluated:

- 1) The Shortest-Jobs-First Backfilling scheduling policy.
- 2) Two different Resources Selection Policies (RSP) have been evaluated: **First Fit** (FF:jobs processes are allocated to the first available set of processor); **First Continuous Fit** (FC:jobs processes are allocated to the first set of continuous processors).

Center	RS	B-SLD	WT	KJ
CTC	FF	1.9	1390	30
	FC	1.7	3890	30
SDSC	FF	32	31008	2
	FC	64	44301	2

TABLE I: System performance metrics using runtime=trace file runtime

- 3) Using as execution time the runtime existing in the workload trace file (traditional modelization) and using our job runtime model (with resource sharing modelization).

VI. EVALUATION

Table I shows the 95_{th} Percentile for the bounded slowdown (B-SLD), the 95_{th} Percentile for the wait time (WT) and the number of killed jobs (KJ) ² for the simulations of the workloads using the trace file provided job runtime for each of the Resource Selection Policies (RSP). Clearly, the performance of both workloads depends on the resource selection policy used. Surprisingly, for the CTC workload the First Continuous policy (FC) achieves better performance than the First Fit policy(FF). Analyzing the simulation in detail, we stated that in a very loaded interval of time, some jobs that require a high number of processors are delayed in the FC due to no continuous processors are available. Thus all the short jobs are backfilled and the system becomes less loaded more quickly. On the other side, with the FF policy, these jobs start before and all these short jobs are substantially delayed. Thereby, the load of the system decreases slower than with the FC. The SDSC workload, shows the behavior that *could* be considered more normal. In this case the First-Fit policy achieves best performance in terms of wait time and bounded slowdown.

Table II shows the System performance metrics when executing using as job runtime model the mode proposed in this paper that includes the penalty estimation based on the conflicts generated by the resource sharing. Additionally to the information shown in Table I, we also show the average of the percentage of penalized runtime (%PRT).

The results show quantitative differences with the performance obtained when using the trace file runtime. In both workloads, incrementing the number of jobs with high memory bandwidth demand has resulted in an increment of all the performance metrics. In CTC evaluations, the performance metrics have shown a clear increment in all the experiments. For instance, in the LOW scenario for the CTC with FF, the B-SLD has grown from 1,9 (from the original model) to 2,2 (15% of increment). Furthermore, the BSLD in the HIGH scenario for the CTC with FF has grown from 1.9 until 4.2 (120% of increment). Similar effects can be appreciated in the SDSC workload. For example, using the FC policy, the wait time 95_{th} Percentile grows form against the 44301 sec. of the

²Killed due to the runtime was higher than the estimation provided by the user

Center	MT	RSP	B-SLD	WT	% PRT	KJ
CTC	High	FF	4.2	10286	8.8	428
	Med	FF	2.8	8963	4.8	247
	Low	FF	2.2	4898	0.92	64
	High	FC	3.5	4527	11.1	527
	Med	FC	3.1	3755	6.17	333
	Low	FC	2.1	2679	1.2	80
SDSC	High	FF	99.3	55667	11.8	475
	Med	FF	55.4	44346	6.7	255
	Low	FF	37.7	32730	1.5	51
	High	FC	197	96803	13.8	548
	Med	FC	149	62883	7.5	307
	Low	FC	85.9	44940	1.4	66

TABLE II: System performance metrics using runtime=job runtime model

traditional model until 44940 in LOW scenario, and until 96803 in HIGH scenario.

Another important effect of the resource sharing model is concerning the number of killed jobs. For instance they grow from 30 to a maximum of 527 jobs in the worse case for the CTC (HIGH scenario and FC policy), and from 2 to 548 in the worse case for the SDSC (HIGH scenario and FC policy). These numbers are also consistent with the fact that the percentage of the runtime penalty introduced when using a FC policy is higher than when using a FF policy. This conclusion seems to be reasonable because the probability that a job generates an *internal* conflict is higher when processors are consecutive.

Qualitative differences have also appeared in the results. The bounded slowdown, for the CTC-SP2 workload using the workload with a medium percentage of jobs with high memory demand concludes that the FC resource selection policy is has better performance than the FF resource selection policy. However, using the traditional model the bounded slowdown results state that the FF policy performance is better than the FC policy.

VII. CONCLUSIONS AND FUTURE WORK

In this paper we have evaluated the impact of considering the penalty introduced in the job runtime due to resource sharing (such as the memory bandwidth) in system performance metrics, such as the average bounded slowdown or the average wait time, in the backfilling policies. This evaluation has been done using simulation techniques with the Alvio simulator. In this paper we have described the different components and elements that are used in its simulation model. We have formalized the model that characterize how the resources are being shared by the allocated jobs. This model punishes the run time for the running jobs that are sharing a given resource in those cases that the resource demand is higher than the resource capacity.

In this paper we evaluate the effect of considering the memory bandwidth usage when simulating the SJF-Backfilling policy under several workloads and resource selection policies. Two different workloads have been used in the experiments. For each of them we have generated three different scenarios: with high (HIGH), medium (MED), and low (LOW) percentage of jobs with high memory demand.

The results have shown quantitative differences between the performance obtained using the presented model and not. In both workloads, incrementing the number of jobs with high memory bandwidth demand has resulted in an increment of all the performance metrics. In CTC evaluations, the performance metrics have shown a clear increment in all the experiments. Furthermore, the number of killed jobs has experimented an evident increment.

Future work will include more a more sophisticate job penalization model derived from a workload synthetic executions analysis. We also are designing resource allocation techniques that will try to avoid job penalization due to resource sharing.

REFERENCES

- [1] D. Tsafirir, Y. Etsion, and D. G. Feitelson, "Backfilling using system-generated predictions rather than user runtime estimates," *In the IEEE TPDS*, 2006.
- [2] D. Talby and D. Feitelson, "Supporting priorities and improving utilization of the ibm sp scheduler using slack-based backfilling," *Parallel Processing Symposium*, pp. pp. 513–517, 1999.
- [3] 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. vol. 949, pp. pp. 337–360, 1995.
- [4] D. G. Feitelson, "Workload modeling for performance evaluation," *In Performance Evaluation of Complex Systems: Techniques and Tools, Lect. Notes Comput. Sci. vol. 2459*, pp. pp. 114–141, 2002.
- [5] D. Tsafirir and D. G. Feitelson, "Instability in parallel job scheduling simulation: the role of workload flurries," *In 20th Intl. Parallel and Distributed Processing Symp*, 2006.
- [6] M. Calzarossa, L. Massari, , and D. Tessaera, "Workload characterization issues and methodologies," *In Performance Evaluation: Origins and Directions, Lect. Notes Comput. Sci.*, p. pp. 459482, 2000.
- [7] W. Cirne and F. Berman, "A comprehensive model of the supercomputer workload," *4th Ann. Workshop Workload Characterization*, 2001.
- [8] K. C. Sevcik, "Application scheduling and processor allocation in multiprogrammed parallel processing systems," *Performance Evaluation*, pp. pp. 107–140, 1994.
- [9] A. B. Downey, "A parallel workload model and its implications for processor allocation," *6th Intl. Symp. High Performance Distributed Comput.*, Aug 1997.
- [10] U. Lublin and D. G. Feitelson, "The workload on parallel supercomputers: Modeling the characteristics of rigid jobs," *J. Parallel and Distributed Computing*, vol. 11, pp. pp. 1105–1122, November 2003.
- [11] 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*, vol. Lecture Notes In Computer Science; Vol. 1162 archive, pp. 41 – 47, 1996.
- [12] 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*, vol. 3277 / 2005, p. 9, June 2004.
- [13] F. Guim, J. Corbalan, and J. Labarta, "The internals of the alvio-simulator: Simulator of hpc infrastructures (upc-dac-rr-cap-2007-2)," Architecture Computer Department - Technical University of Catalunya, Tech. Rep., 2005.
- [14] D. D. G. Feitelson, "Parallel workload archive," 2006.
- [15] D. Tsafirir and D. G. Feitelson, "Workload flurries," School of Computer Science and Engineering and The Hebrew University of Jerusalem, Tech. Rep.