Don't Hurry be Happy: a Deadline-based Backfilling Approach[⋆]

Tchimou N'takpé[1] and Frédéric Suter[2,3]

[1] Université Nangui Abrogoua, Abidjan, Côte d'Ivoire
tchimou.ntakpe@gmail.com / ntakpeeul_sfa@una.edu.ci
[2] Centre de Calcul de l'IN2P3 / CNRS, Lyon-Villeurbanne, France
frederic.suter@cc.in2p3.fr
[3] Inria, Lyon, France

Abstract. Computing resources in data centers are usually managed by a Resource and Job Management System whose main objective is to complete submitted jobs as soon as possible while maximizing resource usage and ensuring fairness among users. However, some users might not be as hurried as the job scheduler but only interested in their jobs to complete before a given deadline.
In this paper, we derive from this initial hypothesis a low-complexity scheduling algorithm, called Deadline-Based Backfilling (DBF), that distinguishes regular jobs that have to complete as early as possible from deadline-driven jobs that come with a deadline before when they have to finish. We also investigate a scenario in which deadline-driven jobs are submitted and evaluate the impact of the proposed algorithm on classical performance metrics with regard to state-of-the-art scheduling algorithms. Experiments conducted on four different workloads show that the proposed algorithm significantly reduces the average wait time and average stretch when compared to Conservative Backfilling.

1 Introduction

To ensure a fair access to resources among users while maximizing resource utilization, most data-centers rely on a *Resource and Job and Management System* (RJMS). Many different systems exist, be they commercial or Open Source, in the High Performance Computing (HPC) [1,2,3] or Big Data [4,5,6] worlds. However, they all follow some common principles while the specifics of the managed workloads differ. For instance, they rely on simple yet efficient scheduling algorithms to ensure scalability. Most schedulers in HPC thus adopt a *First-Come-First-Served* (FCFS) policy [7], usually combined with some *backfilling* techniques to minimize resource idle times. This choice to keep the complexity of the scheduling algorithm as low as possible implies that managing fairness and optimizing the resource utilization are usually done through external mechanisms such as quotas, priorities, or queues. Finally, the common goal of most

[⋆] Work partially supported by the MOEBUS ANR project (13-ANR-INFR- 01).

resource and job management systems is to serve, and thus complete, the submitted jobs as soon as possible. This allow them to increase both the system throughput and the users' satisfaction.

In this paper, we aim at studying the impact of a simple assumption on the management of a given workload by a resource and job management system: "*what if some users were not interested in their jobs to complete as soon as possible, but only before a given deadline?*". For instance, some users may not need their results before the next day. Some periodic jobs may also exist that are important but not urgent, the only constraint being to execute them within the defined period. Then, it makes no difference whether such jobs complete as soon as possible, from the resource management system point of view, or just before when the user said s/he would need his/her results. The same reasoning can even be applied to longer time periods. Such a choice would obviously depend on some voluntary users, but we believe that this slack given by some users for the execution of their jobs could offer an extra degree of freedom to the scheduler. Delaying such deadline-driven jobs would give space to more urgent jobs that would thus start and complete earlier. Note that urgency does not necessarily implies importance. In this work, we only consider the urgency, or absence of, of a job as an optimization lever.

Following this initial hypothesis, we design a low-complexity scheduling algorithm, called *Deadline-Based Backfilling (DBF)*, that distinguishes *regular jobs* that have to complete as early as possible from *deadline-driven jobs* that come with an ultimate deadline before when they have to finish. We also propose a generic scenario in which various proportions of deadline-driven jobs are submitted to assess the impact of the proposed strategy on the scheduling of more urgent jobs. Experiments, conducted on four different workloads from the *Parallel Workloads Archive* (PWA) [8] show that DBF significantly reduces the average wait time and stretch when compared to Conservative Backfilling [9,10].

The remainder of this paper is organized as follows. In Section 2 we recall the principle of the most popular algorithms used in RJMS. Then in Section 3 we describe the workloads and platforms used for our evaluation and how we prepared data for our study. Section 4 details the principle of the proposed Deadline-Based Backfilling scheduling algorithm. In Section 5, we explain how we do select deadline-driven jobs and assign them deadlines. We evaluate the impact of the proposed algorithm on classical performance metrics and compare it to state-of-the-art scheduling algorithms in Section 6. Finally, we discuss related work in Section 7 before concluding this paper in Section 8.

2 Background on Job Scheduling

The scheduling algorithm is not the only component of a RJMS that influences resource utilization. Ordering policies, or priorities, are defined according to the characteristics of the jobs, their resource requirements, or the previous usages of the users submitting them. Another important component are queues that define a set of constraints on jobs, resources, or user profiles, e.g., job lasting between a day and a week, requesting up to 16 nodes, and belonging to users from a certain scientific collaboration. Queues can also be configured to only have access to a certain pool of resources. The combination of ordering policies and queues defines the final order in which jobs are presented to the scheduler. A common setting is to a apply one or more ordering policies within each queue and then define the browsing order of these queues. While these two components of a RJMS can be key to performance [11], in this work we focus on the scheduling algorithm and assume that the list of jobs has already been formed.

Most of the scheduling algorithms underlying RJMS handle jobs following a FCFS policy. However, the different requests for resources of the jobs usually lead to resource fragmentation and idle times. To increase resource usage this basic policy is often completed by a *backfilling* mechanism. Backfilling consists in moving jobs forward in the queue in order to fill "holes" in the schedule.

The Extensible Argonne Scheduling sYstem (EASY) [12] algorithm has been designed for the IBM SP2 supercomputer and is a popular variant of backfilling. In this algorithm, only the first waiting job is considered for allocation, with a guaranteed starting time. When this first job cannot start right away because its requested number of processors is not available, the algorithm browses the list of waiting jobs to find candidates for backfilling. These candidates are jobs that can start immediately, but without delaying the first job of the list.

Conservative Backfilling (CBF) is a less aggressive alternative to EASY with similar performance. It determines an allocation for each job when it enters the system. Then a job can be a candidate to backfilling if and only if it can begin its execution immediately without delaying *any* of the other pre-allocated jobs.

These two backfilling approaches increase the utilization of the resources and decrease the average waiting time of jobs with regard to FCFS alone, but the order in which jobs are scheduled may differ from the submission order. These dynamic modifications of the schedule prevent the more aggressive EASY algorithm to provide users with some guaranteed upper bound on the starting (and thus estimated completion) time of a given job. By design, the more conservative CBF algorithm gives such an upper bound right after the submission of a job, as backfilling can only make jobs start earlier than initially planned.

3 Workloads and Platforms

In this study, we consider different workload logs, i.e., traces of job submissions, extracted from the *Parallel Workloads Archive* (PWA). More precisely, we selected four workloads whose characteristics in terms of distributions of allocations, i.e., the number of processors used to execute a given job, and execution times per job cover a broad and representative range. Unfortunately, none of these publicly available workloads comprises information on the respective urgency of the submitted jobs. These workloads are:

SDSC-BLUE, in its cleaned 4.2 version, contains information on the submission of 250,440 jobs from April 2000 to January 2003 on the IBM SP Blue Horizon of the San Diego Supercomputer Center, that is made of 144 8-way nodes.

SDSC-DS covers a year of activity from March 2004 through March 2005 on the DataStar cluster of the San Diego Supercomputer Center. It is composed of 96,089 jobs executed on 184 nodes. This cluster is made of two kinds of nodes, 176 8-way and 8 32-way SMP nodes for a total of 1,664 processors. We used the cleaned version of the log as recommended by the maintainers of the PWA.
HPC2N covers three and a half years of activity from July 2002 through January 2006 on the Seth Cluster of the High-Performance Computing Center North in Sweden. This cluster is composed of 120 dual processor nodes. The original log comprised 527,371 jobs but 324,500 jobs from a burst submission by a single user were removed, leaving 202,871 jobs in the cleaned log.
ANL-Intrepid accounts for the submission of 68,936 jobs on the IBM Blue Gene/P Intrepid of the Argonne Leadership Computing Facility at Argonne National Laboratory from January 2009 to September 2009. This machine has 40 racks of 1,024 quad-cores nodes for a total of 163,840 cores. However, due to the specificity of the Blue Gene/P system, nodes are grouped into partitions. Eight racks are partitioned in groups of 64 nodes (or 256 cores) while the remaining racks group nodes by 512 (or 2,048 cores). Note that the number of processors requested by jobs are rounded up to the closest multiple of the partition size.

In this work, we only use a subset of the fields that describe a job in the Standard Workload Format (SWF) as we only aim at scheduling jobs and not optimizing their execution with regard to their memory or network usage. Then a job can be only modeled by its *submission time*, the *requested number of processors*, and the *requested time* or walltime. The selected workloads all comprise a certain number of anomalies that were detected and documented by the maintainers of the PWA during their conversion to the SWF format. Table 1 summarizes the anomalies we consider relevant for our study.

Table 1. Summary of anomalies in workloads from the Parallel Workloads Archive.

Workload	**Runtime < 0**	**CPU < 0**	**Used CPU $>$ Req. CPU**	**Runtime $>$ Walltime**	**Runtime $>$ Walltime $+$ 1'**
SDSC-BLUE	10,770	19,516	458	23,434	8,115
SDSC-DS	11,176	0	0	10,658	1,043
HPC2N	0	0	729	14,817	6,170
ANL-Intrepid	0	0	30,948	12,241	9,096

The first two columns correspond to invalid entries in the workloads as the corresponding jobs either logged a negative execution time or were allocated a negative number of processors. Such jobs usually have a "canceled" status in the logs. In the experiments presented in Section 6, we decided to discard these jobs for all the considered algorithms.

The third column shows that, in most workloads, there are jobs that got more processors than requested. This is especially true for ANL-Intrepid where requests are rounded up to fit the partition requirements. In this workload, the number of cores allocated to jobs are rounded up to multiples of either 256 or 2,048. We adapted the descriptions of the clusters accordingly to represent a set of the smallest allocable number of cores. We also make a simplifying yet not impacting assumption about the platforms. We consider the clusters, or partitions, to be fully homogeneous. This means that a job requesting four nodes can indifferently be allocated a contiguous set of nodes (e.g., $\{\mathrm{p}_1, \mathrm{p}_2, \mathrm{p}_3, \mathrm{p}_4\}$) or a disjoint set of nodes (e.g., $\{\mathrm{p}_1, \mathrm{p}_6, \mathrm{p}_8, \mathrm{p}_{22}\}$).

Finally the last two columns indicate that a fair amount of jobs report an execution time longer than the expressed walltime. For most of them, the extra time is less than a minute and can be explained by the time needed by the system to kill a job when it reaches its walltime. However, the last column shows that many jobs continue their execution despite the expiration of their walltime. For all these jobs, we chose to stop them when the walltime is reached.

The SDSC-DS and SDSC-BLUE workloads comprise a non-negligible number of interactive jobs. Such jobs correspond to submissions from users who need a direct and immediate access to the machine. This access mode is thus orthogonal to the idea of letting the scheduler delay jobs. Moreover these jobs are usually scheduled on a limited and distinct subset of the available resources. Consequently, we decided to remove these jobs from the original traces.

All these alterations of the original logs prevent us to compare simulation results to their contents. However, in this study we compare the results of our proposal to those achieved by state-of-the-art algorithms. As long as we use the same input workloads for all scheduling algorithms, results remain comparable.

4 A Deadline-based Backfilling Algorithm

Scheduling a job J_i amounts to find its place in the resource usage profile, i.e., a list of sets of available resources at a given time, maintained by the scheduler. Selecting a specific slot for a job determines the starting date $start_i$ of its execution. The exact set of resources used for the execution of a job is only determined when the job is about to start. In our deadline-based scheduling proposal, we consider two types of jobs. A regular job is a job that once submitted, at time $submit_i$, will be definitely scheduled in a way to minimize its completion date $completion_i$. Conversely, a deadline-driven job is associated to a *deadline* d_i such that the job can be scheduled at any time as long as its execution within a walltime $walltime_i$ can be completed before the deadline expires. Such jobs can be scheduled as regular jobs but their tentative allocations can be reconsidered if new regular jobs enter the system.

Our algorithm is an *online* scheduling algorithm, as CBF or EASY are. As these algorithms do we privilege a low-complexity in our design to ensure the applicability of the resulting algorithm in large-scale production systems. Allocation decisions are taken either when some jobs complete or some new jobs enter the system. These two kinds of events trigger a new scheduling round. The completion of a job, especially if it happens before the expiration of its walltime, makes nodes available that might be used by waiting jobs. New coming jobs, be they regular or not, may also impact the currently planned schedule for different reasons, e.g., candidate for backfilling, priority, tight deadline, . . .

Algorithm 1 Determination of a definitive allocation for a regular job J_r.

```
1: for all $J_i \in L_1$ do
2:     Cancel current allocation of $J_i$
3: end for
4: $L_{tmp} \leftarrow J_r$
5: $Get\_Allocation(J_r)$
6: for all $J_i \in L_1$ do
7:     $Get\_Allocation(J_i)$
8: end for
9: while $\exists J_i \in L_1 \mid ct_i > d_i$ do
10:     $L_{tmp} \leftarrow L_{tmp} \cup J_i$
11:     $L_1 \leftarrow L_1 \setminus J_i$
12:     for all $J_i \in L_{tmp} \cup L_1$ do
13:         Cancel current allocation of $J_i$
14:     end for
15:     for all $J_{tmp} \in L_{tmp}$ do
16:         $Get\_Allocation(J_{tmp})$
17:     end for
18:     for all $J_j \in L_1$ do
19:         $Get\_Allocation(J_j)$
20:     end for
21: end while
22: if $\{J_i \in L_{tmp} \mid ct_i > d_i\} \neq \emptyset$ then
23:     $S_{max} \leftarrow \max(submit_j | J_j \in L_{tmp} \wedge ct_j > d_j)$
24:     $L_{tmp} \leftarrow L_{tmp} \cup \{J_j \in L_1 \mid submit_j < S_{max}\}$
25:     $L_1 \leftarrow L_1 \setminus \{J_j \in L_1 \mid submit_j < S_{max}\}$
26:     for all $J_i \in L_{tmp} \cup L_1$ do
27:         Cancel current allocation of $J_i$
28:     end for
29:     for all $J_{tmp} \in L_{tmp}$ do
30:         $Get\_Allocation(J_{tmp})$
31:     end for
32:     for all $J_j \in L_1$ do
33:         $Get\_Allocation(J_j)$
34:     end for
35: end if
36: for all $J_i \in L_{tmp}$ do
37:     if $start_i = current\_time$ then
38:         Start execution of $J_i$
39:     else
40:         $L_0 \leftarrow L_0 \cup J_i$
41:     end if
42:     $L_{tmp} \leftarrow L_{tmp} \setminus J_i$
43: end for
44: for all $J_i \in L_1$ do
45:     if $start_i = current\_time$ then
46:         Start execution of $J_i$
47:     end if
48: end for
```

From its submission to the beginning of its execution, a job is in a *waiting* state. Our algorithm proposes to store the waiting jobs in two lists. The former, L_0, contains all the jobs whose allocations are definitively determined. It comprises regular jobs but also deadline-driven jobs that either come close to their deadline or improve the backfilling. The latter, L_1, contains only *deadline-driven jobs*, whose allocations can be modified in another scheduling round.

When a *deadline-driven job* is submitted, we determine its allocation according to the CBF algorithm. This allocation takes all the allocations, be they tentative or definitive, of the other waiting jobs into account. If the deadline associated to the job is large enough to prevent its violation from submission, the job is inserted into L_1. On the contrary, the job is considered as regular and inserted into L_0 to be scheduled as early as possible.

When a *regular job* J_r enters the system, we apply Algorithm 1 not only to determine its definitive allocation, but also to reconsider the allocations of waiting deadline-driven jobs. First, if such jobs exist in L_1, we cancel their current allocations (lines 1-3). Second, we build a temporary list L_{tmp} into which J_r is inserted (line 4), and get an allocation for this job. Then, we fill this list with jobs from L_1 whose deadline would be violated because of the allocation of the new regular job J_r. The algorithm proceeds as follows. A new allocation which takes the current allocation of J_r into account is determined for all the jobs in L_1 (lines 6-8). Then, while there is a job J_i in L_1 that does not respect its deadline, we move it from L_1 to L_{tmp} (lines 10-11) and recompute the allocations of both L_{tmp} (lines 15-17) and L_1 (lines 18-20). Note that the allocations for the jobs in L_{tmp} are determined by considering the jobs in an increasing order of submission time. This approach allows us to ensure that if a job can respect of the deadline when it is submitted, none of the modifications of its tentative allocation made by Algorithm 1 would lead to a deadline violation.

At the end of this step, all the deadline-driven jobs are allocated, some of them having been moved forward to avoid deadline violations. However, some jobs in L_{tmp} may still not be able to respect their deadlines. This may come from a different resource fragmentation that appears as we skip some deadline-driven jobs while building L_{tmp}. We thus add an extra step (lines 22-35), in which we move from L_1 to L_{tmp} all the deadline-driven jobs submitted before the last job in L_{tmp} unable to respect its deadline, before recomputing all the allocations.

The next step consists in determining which jobs in L_{tmp} can start their execution in this scheduling round (line 38). Those which cannot are now considered as regular jobs and moved to L_0 (line 40). Finally, our algorithm also starts the execution of some deadline-driven jobs from L_1 (lines 44-48).

5 On the Determination of Deadlines

The main concept underlying the proposed approach is that of *deadline*. This concept raises two important questions: "*Which jobs are considered deadline-driven?*" and "*What are the deadlines associated to these deadline-driven jobs?*".

In this section, we propose to consider a broad and generic scenario in which deadline-driven jobs can be submitted at any time of the day, for instance by adding an extra submission flag to indicate when a job has to be completed at last. This scenario allows us to answer another question: "*what would users in a hurry gain if other users allowed X% of the jobs to be delayed?*". Estimating the gain for different values of X will guide the experimental evaluation of our approach given in Section 6. For the jobs randomly selected to become deadline-driven, we define the associate deadline as a date that is a maximum between 24 hours and 10 times the expressed walltime of the job after the job submission.

The rationale for a delay of at least 24 hours comes from an analysis of the daily (and weekly) job arrival pattern in number of jobs submitted every hour, for the four studied workloads. Fig. 1 shows a similar, and expected, job arrival pattern for all workloads, with a period (gray area) during which the arrival rate is greater than the daily average arrival rate (horizontal line).

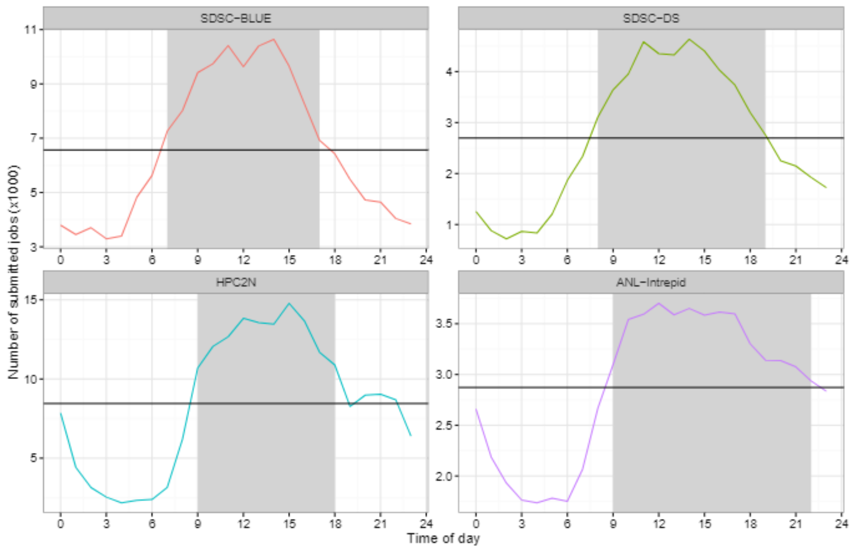

Fig. 1. Daily job arrival for four supercomputers. Gray area depicts a peak period when the arrival rate is greater than the daily average (horizontal line).

This peak period roughly corresponds to business hours from 9AM to 6PM for all workloads. We note that for SDSC-BLUE and SDSC-DS, this peak period is slightly shifted and starts earlier. This might be explained by the location of the corresponding supercomputers on the U.S. West Coast and submissions from users on the East Coast. Such a usage spanning over several time zones may also explain why the peak period ends later on the ANL-Intrepid machine. The sharp decrease in the arrival rate after business hours is likely to correspond to a lower competition for resources as less jobs are submitted. Weekly arrival shows a similar pattern with a sharp decrease of the number of job submission over the weekend. Then, with a deadline of at least twenty four hours after the submission, we ensure that every deadline-driven job can benefit of a period of lower load to be scheduled earlier than its deadline.

We also propose to set the deadline to be proportional to the expressed walltime as a way to favor the shortest jobs. Not only they will be less delayed than jobs with a larger walltime but they are also better candidates for backfilling. The chosen factor of 10 comes from the observed average stretch, i.e., how much a job is impacted by its waiting time, in the different workloads.

To favor the adoption of the proposed approach, incentives for users to submit deadline-driven jobs have to be provided. On most platforms managed by a RJMS, the submissions of a given user are often limited by different quotas (e.g., per user, group, resource type) and influenced by earlier submission pattern. A simple incentive would be to loosen these limitations, hence giving a better admission rate, for users accepting to see their jobs delayed by the scheduler. On platform where users have to pay to access resources, we can easily imagine a discount offered to users who set a deadline as part as their SLA.

6 Experimental Evaluation

6.1 Evaluation Metrics

To evaluate the impact of allowing the execution of certain jobs to be delayed provided they end before a given deadline on the complete workload, we use several performance metrics. First we consider the *wait time* of a job, defined as the difference between the starting and submission dates of a job:

$$wait_i = start_i - submit_i. \tag{1}$$

A second classical metric is to compute the *stretch* experienced by a job. This metric quantifies the relative impact of the wait time on the execution of a job and is defined as:

$$stretch_i = (wait_i + walltime_i)/walltime_i. \tag{2}$$

One of the objectives of our proposal is to reduce the *average wait time* and *average stretch* of the regular jobs. This would indicate how much these jobs benefit of the delayed executions of deadline-driven jobs. We also analyze these two metrics over the whole workload to quantify the potential gain offered by deadline-based scheduling.

We also consider performance metrics related to the deadline-driven jobs. First we measure the number of jobs for which the proposed algorithm is not able to respect the deadline. Second we estimate how the deadline has effectively been used by our algorithm. To this end we define a notion of *deadline usage* as:

$$usage_i = (completion_i - submit_i)/(deadline_i - submit_i). \tag{3}$$

Analyzing the average usage over the entire set of deadline-driven jobs will provide insight about our approach. High values will indicate that regular jobs were scheduled uninterruptedly in the interval left by delaying deadline-driven jobs. Conversely, smaller values will mean that deadline-driven jobs were able to exploit period of lower load before the expiration of their deadlines.

6.2 Simulation Environment

We resort to simulation for our experimental evaluation. Instead of developing an *ad-hoc* simulator, we opted for using an existing simulation framework. Several such tools have been used in the literature to simulate the replay of workloads from the Parallel Workloads Archive. The Alea[4] job scheduling simulator [13] is based on the GridSim toolkit and allows to compare queue-based scheduling algorithms. Alea separates the implementation of the algorithms, defined as independent Java classes, from that of the discrete event simulation itself. However, new algorithms have to be coded in this specific language and embedded into the code base of the tool. Alea also offers an interesting dynamic scheduling feature allowing jobs to be submitted during the simulation and support the management of priority queues [11]. In the early stage of this work, we used the SimBatch tool [14] for our evaluations. This framework, whose maintenance and evolution are no longer supported, was based on the SimGrid toolkit [15]. As for Alea, new algorithms had to be included to the code base, in C, of the tool.

In this work we decided to rely on another recent and promising SimGrid-based tool. Batsim[5] [16] is developed by the team that develops and maintains the OAR RJMS [1]. It decouples the simulation of the resources and the execution of a schedule from the decisions that led to this schedule. Then Batsim can leverage the different network and computing models of SimGrid to adapt the level of realism of the simulation to the needs of the users. In our experiments, we simulate jobs as simple delays defined as the minimum between the execution time as logged in the workload and the expressed walltime. Batsim exposes a simple message interface between the simulation engine and scheduling algorithms written as plugins in various programming languages.

For our experiments we use the latest version of Batsim shipped in a container as recommended by the development team. This container relies on the latest stable version of SimGrid (3.14.159) at the time of writing. We coded the proposed DBF algorithm as a scheduler plugin of Batsim in Python. We also implemented two state-of-the-art algorithms, CBF and EASY, that are used as references to evaluate the performance of our algorithm. To ensure the reproduction and further investigation of the presented results, and thus favor Open Science, these algorithm implementations, the scripts used to prepare and convert the workloads into the Batsim input format and analyze the outcomes of the simulations, as well as the sources of this paper are made available online [17].

[4] Alea web site: `https://github.com/aleasimulator/alea`
[5] Batsim web site: `https://github.com/oar-team/batsim`

6.3 Results

We begin the evaluation of the proposed deadline-based backfilling approach by assessing its impact on regular jobs. For each workload, we randomly select a number of jobs to become deadline-driven and assign deadlines to these jobs as described in Section 5. Table 2 summarizes the respective numbers of regular jobs when the percentage of deadline-driven jobs varies from 20% to 80%.

Table 2. Number of regular jobs impacted by deadline-based backfilling when the number of randomly selected deadline-driven jobs varies from 20% to 80%.

		Percentage of deadline-driven jobs			
Workload	Total	20%	40%	60%	80%
SDSC-BLUE	157,604	126,084	94,564	63,043	31,522
SDSC-DS	64,715	51,772	38,829	25,886	12,943
HPC2N	202,871	162,297	121,723	81,150	40,576
ANL-Intrepid	68,936	55,149	41,363	27,575	13,789

In all the subsequent analyses, we filter out regular jobs whose wait time (resp. stretch) was 0 (resp. 1) simultaneously for all the three algorithms. Such jobs were lucky enough to obtain the requested resources right on submission independently of the scheduling algorithm used. Keeping them would modify the perception of the actual performance of a given algorithm.

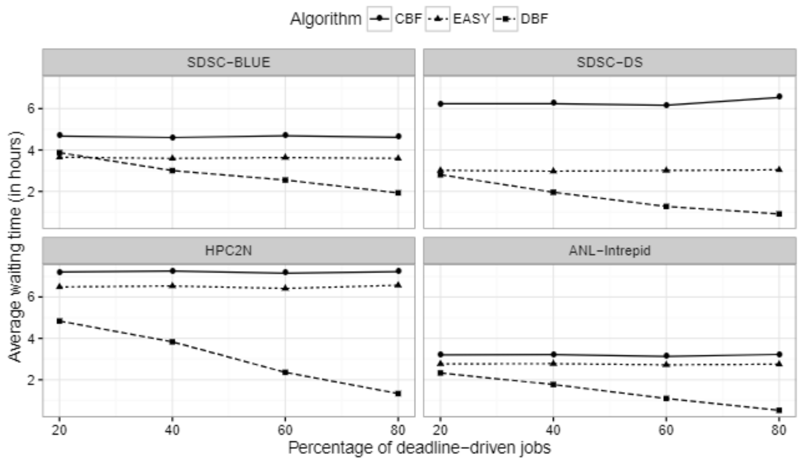

Fig. 2. Evolution of the average wait time experienced by regular jobs with the percentage of deadline-driven jobs.

First, we study the evolution of the *average wait time* experienced by the regular jobs shown by Fig. 2. A first observation is that this average wait time remains stable for both the CBF and EASY algorithms when we increase the number of deadline-driven jobs. This indicates that the decreasing number of jobs under consideration does not impact this metric. We also note that EASY consistently leads to a smaller average wait time than CBF, which comes from its more aggressive backfilling strategy. The proposed DBF algorithms outperforms its two contenders in all configurations except for the SDSC-BLUE workload with 20% of deadline-driven jobs where EASY is slightly better.

We also observe that our algorithm leads to a linear decrease of the average wait time of regular jobs when we increase the share of deadline-driven jobs. The best improvement is obtained for SDSC-DS where DBF already reduces the average wait time by more than a factor of two when there are only 20% of deadline-driven jobs. However, the results obtained by EASY in this configuration indicate that CBF obtains poor performance for this workload. It may be explained by a higher resource fragmentation for this workload that EASY can better exploit with its more aggressive backfilling strategy. It is also interesting to note that, even with only 20% of deadline-driven jobs, DBF is at least on par with EASY or reduces the average wait time up to 25% (for the HPC2N workload) but is also able to provide guarantees on job completion times that EASY would not give. Indeed, the scheduling of regular jobs is based on CBF and then the first tentative allocation of regular jobs gives them a completion time than can only be reduced afterwards. Moreover, DBF ensures that a deadline-driven job completes before its deadlines, which is another kind of upper bound.

Fig. 3 shows a more detailed view of the wait time experienced by the regular jobs. Each line corresponds to a workload while each column corresponds to a given percentage of deadline-driven jobs. Each panel presents the wait time as an Empirical Cumulative Distributive Function for the three considered algorithms.

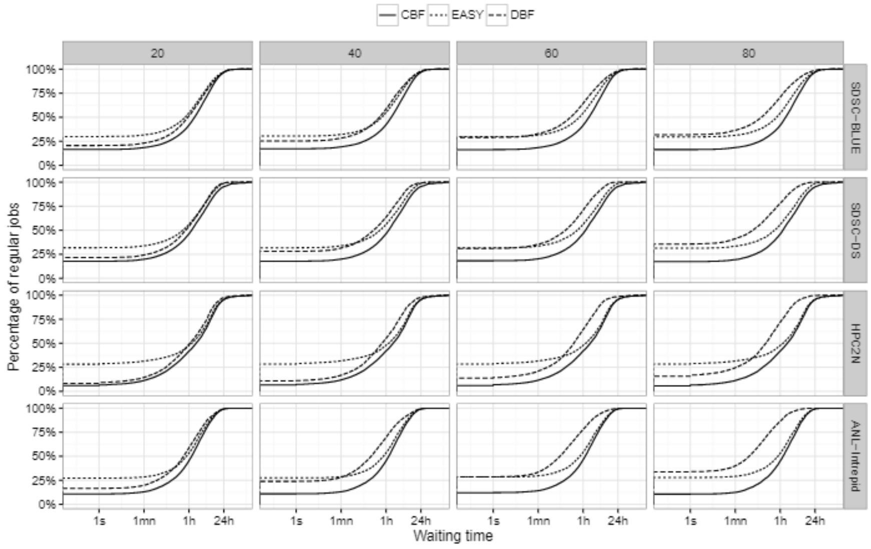

Fig. 3. Wait times experienced by regular jobs.

These more detailed results globally confirm the trends shown in Fig. 2 but also give us some interesting extra information. For instance, the top-left panel corresponds to the selection of 20% of deadline-driven jobs in the SDSC-BLUE workload. It is the configuration in which EASY leads to a slightly better average wait time than DBF. We observe that this comes from a greater number of jobs (above 25%) that can start immediately with EASY thanks to the aggressive backfilling. However, the first quartile for DBF is only of two and a half minutes. We also note a difference of less than half an hour for the third quartile in favor of EASY, but DBF is able to reduce the maximum wait time of about twelve hours. Again, this is explained by the design of EASY that causes extra wait time for jobs that cannot benefit of backfilling. We observe similar distributions for all the workloads when there are 20% of deadline-driven jobs. When the percentage of deadline-driven jobs increases, DBF competes with EASY with wait times close to zero for at least 25% of the jobs (with 40% of deadline-driven jobs on SDSC-BLUE and SDSC-DS, and 60% for ANL-Intrepid), while the performance of CBF remains unchanged. A noticeable exception is the HPC2N workload, which is the largest in terms of number of jobs. There we observe a uniform reduction of the wait time when the share of deadline-driven jobs increases.

Fig. 4 presents similar results as Fig. 2 but for the entire workload, i.e., regular and deadline-driven jobs combined. Note that this figure also includes the jobs whose wait time is zero with the different algorithms. Moreover and for the sake of clarity, we express the average wait time in minutes.

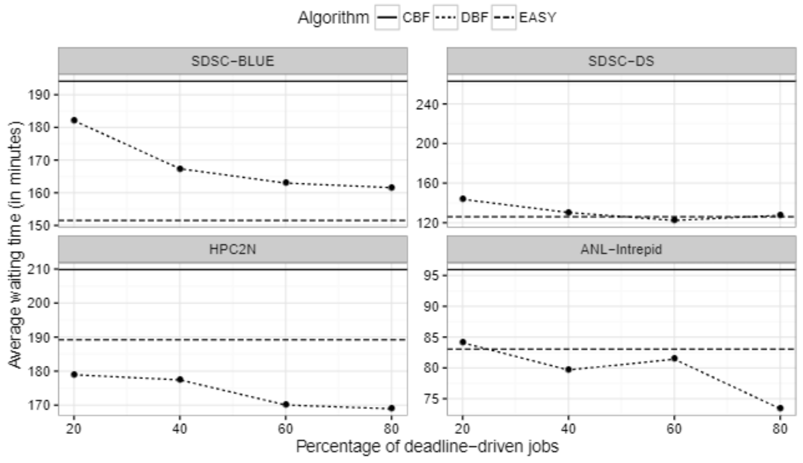

Fig. 4. Evolution of the average wait time experienced by all jobs with the percentage of deadline-driven jobs.

We can see that, in addition to reducing the average wait time of regular jobs, the proposed DBF algorithm also globally reduces the average wait time of the whole workload with regard to CBF. The improvement over this algorithm, upon which DBF is based, also increases with the proportion of deadline-driven jobs. This means that allowing the scheduler to delay some jobs (as defined in Section 5) to favor some others that are more urgent does not come at the price of a global degradation of the schedule quality but actually improves it.

The comparison with EASY does not show a clear winner, even though DBF leads to similar or lower average wait times for most workloads with at least 40% of deadline-driven jobs. We also recall that DBF provides users with an upper bound on job completion time, as CBF does. This valuable information that EASY cannot give may justify a slightly larger average wait time.

We continue our evaluation with the analysis of our second performance metric: the stretch, or slowdown, experienced by jobs when scheduled with the different algorithms. The evolution of the average stretch with the percentage of deadline-driven jobs and the relative performance of the three algorithms are very similar to those in Fig. 2 for the average wait time. This means that, on average, the respective execution time of deadline-driven does not influence this metric which is thus mainly driven by the wait time. Then we also analyze the *maximum stretch* which is a typical indicator of fairness in the literature. Indeed a small maximum stretch, ideally close to the average stretch indicates that the scheduling algorithm does not disfavor some jobs too much in order to reduce the completion time of others. Fig. 5 presents the evolution of the maximum stretch for regular jobs with the percentage of deadline-driven jobs.

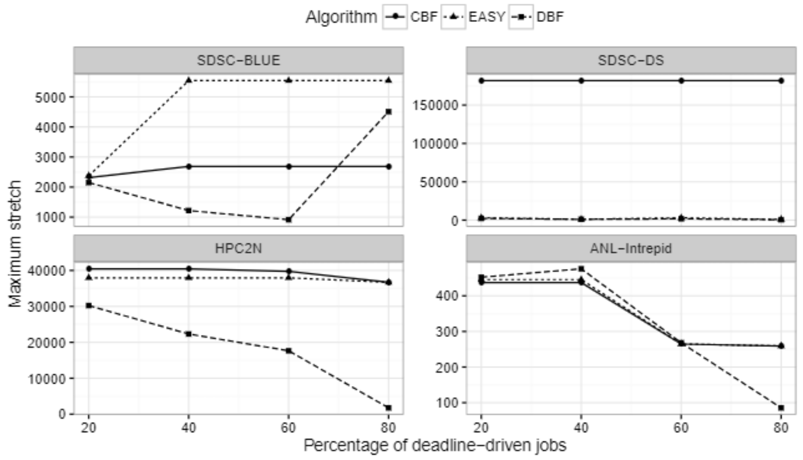

Fig. 5. Evolution of the maximum stretch for regular jobs with the percentage of deadline-driven jobs.

There is a great difference between the maximum and average stretches for all algorithms, which was expected as none of them aims at optimizing fairness among jobs. We also observe important variations of the maximum stretch for all the three algorithms when the percentage of deadline-driven jobs varies. This indicates that this value strongly depends on the subset of jobs that have been selected to become deadline-driven. For instance, the regular job that has the maximum stretch for CBF with 40, 60, and 80% of deadline-driven jobs on the SDSC-BLUE workload belongs to the set of 20% of deadline-driven jobs, hence a smaller maximum stretch. Then we can just comment on the general trends but not on specific values. We can however say that DBF either leads to similar (for SDSC-DS and ANL-Intrepid) or better (for SDSC-BLUE and HPC2N) maximum stretches than those achieved by EASY.

The DBF algorithm has been designed to ensure the respect of the deadlines associated to the jobs. As explained in Section 4, deadline violations can only occur when the first tentative allocation determined for a job already fails to respect the deadline, due to heavy load or the occupation of most of the resources by long lasting jobs. Fig. 6 shows how many deadlines were not respected for each workload depending on the proportion of deadline-driven jobs. For each configuration, we distinguish short jobs whose deadline was set to 24 hours from those whose deadline is proportional to the expressed walltime.

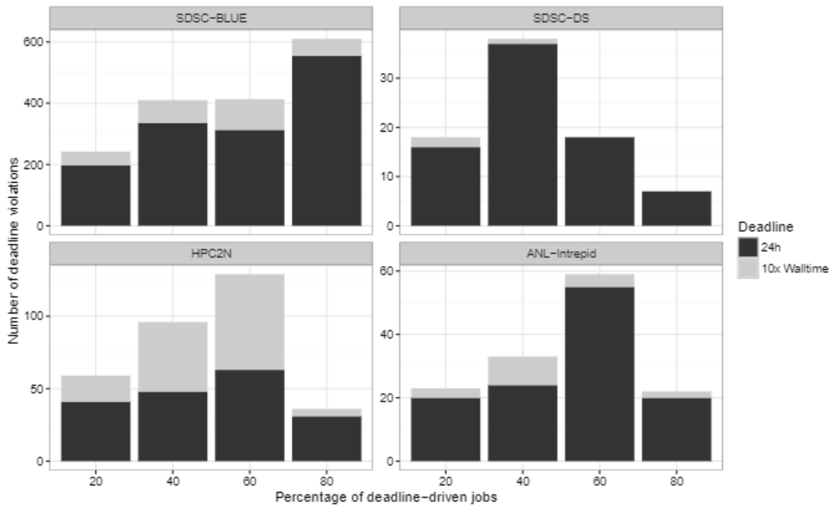

Fig. 6. Evolution of the number of deadline violations for the DBF algorithm with the percentage of deadline-driven jobs.

A first comment is that the number of deadline violations is extremely small compared to the number of deadline-driven jobs in the system and, not surprisingly, higher for the two largest workloads. These results show that DBF is able to guarantee the completion of almost all the deadline-driven jobs before their deadline. Moreover the evolution with the proportion of deadline-driven jobs does not indicate a direct correlation. For each of the presented experiment, a growing set of jobs to become deadline-driven jobs is randomly selected. However, these sets are not inclusive, e.g., all the jobs in the 20% set are not necessarily in the 80% set. Moreover, a majority of the jobs that cannot respect their deadlines were submitted in heavily loaded period. We also observe that a vast majority of these violations are for jobs with a 24-hour deadline, i.e., short jobs with an expressed walltime of less than three hours, which confirms a relation with a heavy load at submission time for these jobs. The HPC2N workload exhibits a different pattern with more violations for longer jobs. A further analysis shows that a few set of jobs experience similar deadline violations which indicates that all these jobs had to wait for the completion of a single job.

When analyzing how the deadlines associated to the jobs were exploited by the DBF algorithm we found that, all simulations combined, almost half of the deadline-driven jobs were executed immediately after their submission. These jobs are uniformly distributed over the workloads and scenarios. Fig. 1 showed large periods of lower load every night. As deadline-driven jobs were randomly selected it would not surprising that a large fraction of them were submitted during lower load periods. We decided to removed these jobs from the computation of the average deadline usage shown by Fig. 7.

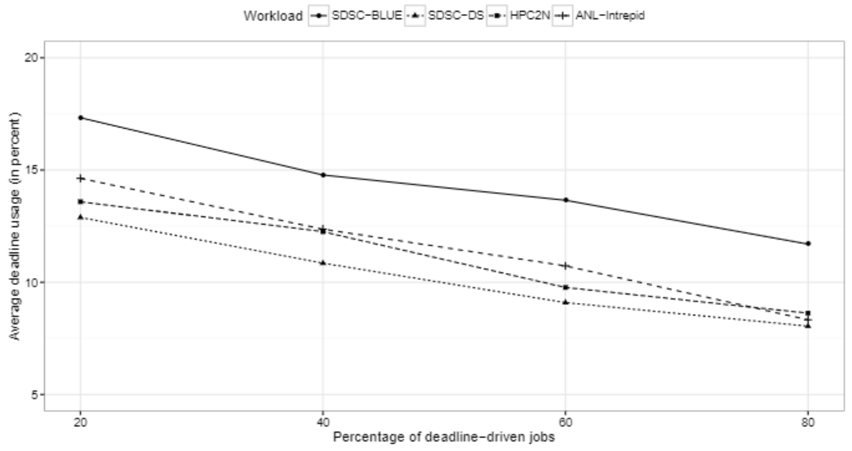

Fig. 7. Evolution of the utilisation of the deadlines of the deadline-driven jobs by the DBF algorithm with the percentage of deadline-driven jobs.

This graph shows a very similar trend for all the workloads: the more deadline-driven jobs are submitted the less they use their deadlines. Moreover, the percentages of deadline usage are pretty low, with a maximum of 17.3% for SDSC-BLUE with 20% of deadline-driven jobs. This tends to indicate that the chosen deadlines might have been too lazy and could be shortened. Note also that the deadlines were determined from the expressed walltime that are typically and largely overestimated by users. However, these results also show that associating a (very) large deadline to a job does not necessarily mean that the job will be delayed for a (very) long time. It just gives more freedom to the scheduler which will exploit it only when needed. For instance, less than 1% of the deadline-driven jobs used more than 80% of their deadlines.

We conclude this evaluation by discussing the time needed to simulate the scheduling of the different workloads by the three considered algorithms, as summarized in Tab. 3. All the simulations were run on a notebook (8-core 2.40GHz Intel i7-4700MQ CPU) using Batsim in a Docker container hosted by an Ubuntu 16.04 LTS Operating System.

Table 3. Time to simulate the scheduling of four workloads with three algorithms.

	DBF					
	20%	40%	60%	80%	CBF	EASY
SDSC-BLUE	4m33s	4m34s	04m38s	5m13s	3m35s	2m18s
SDSC-DS	2m	2m13s	2m07s	2m02s	1m49s	58s
HPC2N	23m03s	29m51s	34m26s	30m27s	7m21s	3m03s
ANL-Intrepid	2m20s	2m25s	2m21s	2m04s	1m52s	1m10s

We first observe that varying the percentage of deadline-driven jobs handled by the proposed DBF algorithm does not have any significant impact on the time needed to schedule a full workload. We also note that the time needed to schedule the HPC2N workload is much larger than for the other workloads. It can partially be explained by the greater number of jobs to execute on a relatively small number of processors (202,871 jobs on 240 processors), but also by a large overestimation of walltimes for a fair amount of jobs, i.e., up to 1,000 times greater. This implies a lot of extra rescheduling steps that directly impact the simulation time. However, the average time to schedule a job for this workload remains reasonable in less than 10 milliseconds, and is less than 2 milliseconds for the three other workloads. Compared to CBF, the management of deadline-driven jobs, and the benefits they bring, by DBF induces an affordable overhead of 25%. Finally, EASY that computes less tentative allocations is about twice as fast as DBF but does not provide the same guarantees on job completion times.

7 Related Work

In the scheduling literature, deadlines usually express a Quality of Service requirement. For instance, in (hard) real-time systems the Earliest Deadline First (EDF) policy [18] is a preemptive scheduling algorithm that puts jobs in a priority queue and selects the job with the closest deadline for execution when a scheduling event occurs. Similarly on big data analytics clusters, some jobs require guarantees on their completion time and thus can be seen as deadline-sensitive jobs [19,20]. In these two areas, deadlines act as a constraint the scheduler has to respect, while in our work we primarily see the deadline as an extra degree of freedom for the scheduler. Another main difference is that in both real-time and big data systems, jobs are usually executed on a single compute node and often periodic while in HPC systems jobs are mainly parallel and independent. The associated scheduling challenges and the definition and usage of deadlines are then completely different.

The distinction between urgent (but necessarily important) jobs and less latency-sensitive (but often important) jobs can also be found and characterized in big data workloads and RJMS. For instance the Google's Borg system [5] distinguishes "production" services used for end-user-facing products that show a diurnal usage pattern from "non-production" batch jobs that are less sensitive to short-term performance fluctuations.

In [21], the authors define a concept of flexible backfilling to schedule jobs on heterogeneous HPC resources. They use deadlines to increase the priority of jobs when they are coming close to their deadlines and decrease it when the deadlines expire. In this paper, we use deadlines in a different way, not to increase the priority of a job but on the contrary to further delay its execution.

While the backfilling strategies implemented by CBF and EASY are popular in production systems, they may cause important resource fragmentation. In [22,23], the authors rely on meta-heuristics, i.e., tabu-search and random selection, to periodically reorganize the schedules. While these modifications improve the average wait time and stretch, they do not preserve one of the most interesting feature of CBF which is to provide an upper bound of job completion time on submission. The proposed Deadline-based backfilling algorithm also builds upon and improves the seminal CBF algorithm, but conserves this feature for both regular and deadline-driven jobs. Finally, in [24] the authors modify the way candidates for backfilling are selected in the list of waiting jobs. As our proposed solution, this approach improves CBF with regard to the performance metrics used in Section 6 but differs in the selection criterion. They rely on new priority criteria while we use the deadlines associated to the jobs.

8 Conclusion and Future Work

A common and fundamental principle of Resource and Job Management Systems is to build schedules aiming at making jobs complete as soon as possible, hence minimizing their response time. Backfilling approaches participate to this effort, with a interesting side effect of improving resource usage, by moving jobs ahead in the schedule to fill resources left idle by other jobs.

In this paper, we followed the simple hypothesis that some users may not be willing to get their results as soon as possible to design an original algorithm called *Deadline-based Backfilling* (DBF). If some job is submitted along with a deadline for its execution, this algorithm can delay it up to the expiration of this deadline to leave room to more urgent jobs. We design this algorithm while aiming at keeping its complexity as low as possible to favor its adoption and at preserving the capacity to provide an upper bound on job completion time from the submission. We evaluate the performance of this algorithm in terms of average wait time and average stretch on four workloads extracted from the Parallel Workload Archive. Experimental results shown a clear improvement on both metrics when compared to the classical Conservative Backfilling and EASY scheduling algorithms.

Experiments presented in Section 6 study the impact of the proportion of deadline-driven jobs on the quality of the schedule using simple deadline determination rules. Our main future work will be to derive some principles to set the most beneficial deadlines from the characterization of jobs composing the workloads. Patterns in terms of duration, number of processors, periodicity will be investigated. Then we will complete this study by fixing the proportion of deadline-driven jobs and analyzing the impact of the derived deadline on the schedule. Finally we aim at defining policies and incentives to motivate users to be less eager to get their results and give some extra freedom to the scheduler.

References

1. Capit, N., Da Costa, G., Georgiou, Y., Huard, G., Martin, C., Mounié, G., Neyron, P., Richard, O.: A Batch Scheduler with High Level Components. In: Proc. of the 5th International Symposium on Cluster Computing and the Grid (CCGrid), Cardiff, UK (May 2005) 776–783
2. Yoo, A., Jette, M., Grondona, M.: SLURM: Simple Linux Utility for Resource Management. In: Proc. of the 9th International Workshop on Job Scheduling Strategies for Parallel Processing. Volume 2862 of Lecture Notes in Computer Science., Springer (June 2003) 44–60
3. Staples, G.: TORQUE - TORQUE Resource Manager. In: Proc. of the ACM/IEEE SC2006 Conference on High Performance Networking and Computing, Tampa, FL (November 2006) 8
4. Boutin, E., Ekanayake, J., Lin, W., Shi, B., Zhou, J., Qian, Z., Wu, M., Zhou, L.: Apollo: Scalable and Coordinated Scheduling for Cloud-Scale Computing. In: Proc. of the 11th USENIX Symposium on Operating Systems Design and Implementation, (OSDI), Broomfield, CO (October 2014) 285–300
5. Verma, A., Pedrosa, L., Korupolu, M., Oppenheimer, D., Tune, E., Wilkes, J.: Large-Scale Cluster Management at Google with Borg. In: Proc. of the 10th European Conference on Computer Systems, (EuroSys), Bordeaux, France (April 2015)
6. Hindman, B., Konwinski, A., Zaharia, M., Ghodsi, A., Joseph, A.D., Katz, R.H., Shenker, S., Stoica, I.: Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center. In: Proc. of the 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI), Boston, MA (2011)
7. Schwiegelshohn, U., Yahyapour, R.: Analysis of First-Come-First-Serve Parallel Job Scheduling. In: Proc. of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, CA (January 1998) 629–638
8. Feitelson, D., Tsafrir, D., Krakov, D.: Experience with using the Parallel Workloads Archive. Journal of Parallel and Distributed Computing **74**(10) (2014) 2967–2982
9. Feitelson, D.G., Weil, A.M.: Utilization and Predictability in Scheduling the IBM SP2 with Backfilling. In: Proc. of the 12th International Parallel Processing Symposium (IPPS). (1998) 542–546
10. Mu'alem, A.W., Feitelson, D.G.: Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling. IEEE Transactions on Parallel and Distributed Systems **12**(6) (June 2001) 529–543

11. Klusáček, D., Tóth, S.: On Interactions among Scheduling Policies: Finding Efficient Queue Setup Using High-Resolution Simulations. In: Proc. of the Euro-Par 2014 International Conference on Parallel Processing. Volume 8632 of Lecture Notes in Computer Science., Porto, Portugal, Springer (August 2014) 138–149
12. Lifka, D.A.: The ANL/IBM SP Scheduling System. In: Proc. of the Workshop on Job Scheduling Strategies for Parallel Processing. Lecture Notes on Computer Science, Springer-Verlag (1995) 295–303
13. Klusáček, D., Rudová, H.: Alea 2 – Job Scheduling Simulator. In: Proc. of the 3rd International ICST Conference on Simulation Tools and Techniques (SIMUTools 2010), Malaga, Spain (2010)
14. Caniou, Y., Gay, J.: Simbatch: An API for Simulating and Predicting the Performance of Parallel Resources Managed by Batch Systems. In: Euro-Par 2008 Workshops - SGS 2008, Revised Selected Papers, Las Palmas de Gran Canaria, Spain (August 2008) 223–234
15. Casanova, H., Giersch, A., Legrand, A., Quinson, M., Suter, F.: Versatile, Scalable, and Accurate Simulation of Distributed Applications and Platforms. Journal of Parallel and Distributed Computing **74**(10) (2014) 2899 – 2917
16. Dutot, P.F., Mercier, M., Poquet, M., Richard, O.: Batsim: a Realistic Language-Independent Resources and Jobs Management Systems Simulator. In: Proc. of the 20th Workshop on Job Scheduling Strategies for Parallel Processing, Chicago, IL (May 2016) Preprint available at: http://www.cs.huji.ac.il/~feit/parsched/jsspp16/p2-dutot.pdf.
17. N'takpé, T., Suter, F.: Companion of the Don't Hurry be Happy: a Deadline-based Backfilling Approach article (2017) Available at: https://doi.org/10.6084/m9.figshare.4644466.
18. Liu, C.L., Layland, J.: Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. Journal of the ACM **20**(1) (1973) 46–61
19. Jyothi, S.A., Curino, C., Menache, I., Narayanamurthy, S.M., Tumanov, A., Yaniv, J., Mavlyutov, R., Goiri, I., Krishnan, S., Kulkarni, J., Rao, S.: Morpheus: Towards Automated SLOs for Enterprise Clusters. In: Proc. of the 12th USENIX Symposium on Operating Systems Design and Implementation, (OSDI), Savannah, GA (November 2016) 117–134
20. Lucier, B., Menache, I., Naor, J., Yaniv, J.: Efficient Online Scheduling for Deadline-Sensitive Jobs. In: Proc. of the 25th ACM Symposium on Parallelism in Algorithms and Architectures, (SPAA), Montreal, Canada (July 2013) 305–314
21. Baraglia, R., Capannini, G., Pasquali, M., Puppin, D., Ricci, L., Techiouba, A.: Backfilling Strategies for Scheduling Streams of Jobs On Computational Farms. In: Proc. of the CoreGRID Workshop on Programming Models Grid and P2P System Architecture Grid Systems, Tools and Environments, Heraklion, Greece (June 2007) 103–115
22. Klusáček, D., Rudová, H.: Performance and Fairness for Users in Parallel Job Scheduling. In: Proc. of the 16th International Workshop on Job Scheduling Strategies for Parallel Processing. Volume 7698 of Lecture Notes in Computer Science., Shanghai, China, Springer (May 2013) 235–252
23. Klusáček, D., Chlumský, V.: Planning and metaheuristic optimization in production job scheduler. In: Proc. of the 20th Workshop on Job Scheduling Strategies for Parallel Processing, Chicago, IL (May 2016) Preprint available at: http://www.cs.huji.ac.il/~feit/parsched/jsspp16/p4-klusacek.pdf.
24. Lindsay, A., Galloway-Carson, M., Johnson, C., Bunde, D., Leung, V.: Backfilling with Guarantees Made as Jobs Arrive. Concurrency and Computation: Practice and Experience **25**(4) (2013) 513–523