Constraint Programming-based Job Dispatching for Modern HPC Applications

Cristian Galleguillos[1,2], Zeynep Kiziltan[2], Alina Sîrbu[3], and Ozalp Babaoglu[2]

[1] Pontificia Universidad Católica de Valparaíso, Valparaíso, Chile
cristian.galleguillos.m@mail.pucv.cl
[2] University of Bologna, Bologna, Italy
{zeynep.kiziltan,ozalp.babaoglu}@unibo.it
[3] University of Pisa, Pisa, Italy
alina.sirbu@unipi.it

Abstract. HPC systems are increasingly being used for big data analytics and predictive model building that employ many short jobs. In these application scenarios, HPC job dispatchers need to process large numbers of short jobs quickly and make decisions on-line while ensuring high Quality-of-Service (QoS) levels and meet demanding timing requirements. Constraint Programming (CP) is an effective approach for tackling job dispatching problems. Yet, the state-of-the-art CP-based job dispatchers are unable to satisfy the challenges of on-line dispatching and take advantage of job duration predictions. These limitations jeopardize achieving high QoS levels, and consequently impede the adoption of CP-based dispatchers in HPC systems. We propose a class of CP-based dispatchers that are more suitable for HPC systems running modern applications. The new dispatchers are able to reduce the time required for generating on-line dispatching decisions significantly, and are able to make effective use of job duration predictions to decrease waiting times and job slowdowns, especially for workloads dominated by short jobs.

1 Introduction

Easy access to massive data sets, data analytics tools and High-Performance Computing (HPC) have been fueling the trend towards *data-driven* computational scientific discovery [3], with big-data processing frameworks such as Hadoop and Spark increasingly integrated with HPC systems [2,34,31,17]. Workloads of HPC systems engaged in data-driven analytics tend to be a mix of many short jobs ($<$ 1h) with fewer longer jobs [32]. Hence, HPC job dispatchers need to rapidly process a large number of short jobs in making on-line decisions so as to minimize both waiting times and *slowdown* (the ratio between the total job duration including waiting time and the actual job duration during runtime). These measures of Quality-of-Service (QoS) are particularly important when HPC systems are used to provide real-time services, such as big-data visualization [33,39,29], where response times are critical for acceptable user experience.

While the on-line job dispatching problem in HPC systems is NP-hard [6], it can be formulated as a job scheduling and resource allocation problem for which

Constraint Programming (CP) has produced good results [4]. The first CP-based HPC dispatcher with job waiting times as a measure of QoS was introduced in [5] and shown to obtain better solutions compared to a Priority Rule-Based (PRB) dispatcher [10,21], which is widely adopted in commercial HPC workload management systems such as Altair PBS Professional [1] and SLURM Workload Manager [35]. The dispatcher was later embedded as a plug-in within the software framework of PBS professional [9]. Subsequently, another CP-based dispatcher with a similar measure of QoS with the additional feature of limiting system power consumption was presented in [8,7] and proved to outperform a PRB dispatcher on the instances with tight power capping values.

Despite the potential of these CP-based job dispatchers, certain limitations hinder their adoption for modern HPC systems. As reported in [9], the first dispatcher is not resilient to *heavy workloads* — workloads where resource requests greatly exceed available resources. The time spent by this dispatcher in generating a dispatching decision increases dramatically as more jobs requiring high system utilization arrive to the system. The second CP-based HPC dispatcher was initially employed in off-line mode [8], and later also in on-line mode [7] but on workloads of maximum 1000 jobs submitted in a time window of half an hour. A more realistic scenario where jobs arrive continuously and many of them end up waiting in a queue due to unavailable computational resources increases greatly the difficulty of generating dispatching decisions. Our experimental results confirm that both dispatchers are not resilient to heavy workloads that are present in real datasets, which is undesirable in the quest for fast response times.

Another limitation is related to the actual runtime duration of a job on a specific HPC system which is not known before it is executed and yet is crucial for generating dispatching decisions to guarantee high QoS levels. Dispatchers often use the expected job duration, which is the maximum time a job is allowed to execute on the system. In the above mentioned dispatchers, the expected duration is the default value assigned by the system, which is typically the default wall-time of the queue where the job is submitted, unless the job owner supplied her own expected duration. Even in the latter case, however, users tend to use the maximum wall-time and user estimations are acknowledged to be overestimated in general [16,13,27]. A dispatcher that relies on overestimated durations is likely to schedule fewer jobs than possible at dispatching time, and consequently, is likely to cause unnecessary delays. Prediction of actual runtime durations using simple heuristics or more sophisticated machine learning techniques is an active area of research [38,20,19,15]. Recent studies show that the use of job duration predictions when generating dispatching decisions can substantially improve QoS levels in backfilling-based dispatchers [37,20,15,19].

Our contribution is a class of novel CP-based dispatchers that are more suitable for HPC systems running modern applications. We build on [5,8] and redesign their main components. First, we revisit their model and search control mechanism so as to make them resilient to heavy workloads and applicable to on-line dispatching. Second, we study the use of job duration prediction, instead of the expected duration, when generating dispatching decisions. We discuss why

naively replacing the expected duration with a predicted duration may be ineffective, if not detrimental for QoS. Consequently, we adapt the model and search algorithm of our dispatchers to the use of job duration predictions to obtain high QoS levels in terms of job waiting times and slowdown. We conduct a simulation study on a workload trace collected from an HPC system containing large numbers of short jobs. We use predictions with different accuracy, underestimation and overestimation rates on the dataset. Our results demonstrate that with our approach, the CP-based dispatchers can: (i) significantly reduce the time required to generate dispatching decisions; and (ii) benefit from good job duration predictions and considerably decrease the waiting times and the slowdown of the jobs, especially for workloads dominated by short to medium jobs.

The rest of the paper is organized as follows. In Section 2, we introduce the on-line job dispatching problem in HPC systems and give an overview of the CP-based dispatchers introduced in [5,8]. In Sections 3 and 4, we describe our approach. In Sections 5 and 6, we detail our experimental study and present our results. We discuss the related work in Section 7 and conclude in Section 8.

2 Formal Background

2.1 On-line job dispatching problem in HPC systems

A user request in an HPC system consists of the execution of a computational application over the system resources. Such a request is referred to as *job* and the set of all jobs is known as *workload*. Each job in the workload is associated to a name, required resources (cores, memory, etc) to run the corresponding application, and its *expected duration* which is the maximum time it is allowed to execute on the system. An HPC system typically receives multiple jobs simultaneously from different users, placing them in a *queue* together with the other waiting jobs (if there are any). The time interval during which a job remains in the queue until its execution time is known as *waiting time*. At a given *dispatching time*, a job *dispatcher decides* when the jobs waiting in the queue can start executing and on which resources they can execute. The goal is to dispatch in the best possible way according a measure of QoS, such as by reducing the waiting times or the slowdown of the jobs, which is directly perceived by the HPC users. During execution, a job exceeding its expected duration is killed.

Formally, *on-line dispatching* in an HPC system takes place at a specific time t for (a subset of) the queued jobs Q. A typical HPC system is composed of N nodes, with each node $n \in N$ having a capacity $cap_{n,r}$ for each of its resource type $r \in R$, giving the total amount of available resource. Each job $i \in Q$ has the arrival time $q_i \leq t$ to the queue, which is unknown before the arrival, and a demand $req_{i,r}$ giving the amount of resources required from r. The *on-line dispatching problem* at time t consists in *scheduling* each job i by assigning it a start time $s_i \geq t$, and *allocating* i to the requested resources during its expected duration d_i, such that the capacity constraints are satisfied: at any time in the schedule, the capacity $cap_{n,r}$ of a resource r is not exceeded by the total demand $req_{i,r}$ of the jobs i allocated on it, taking into account the presence

of jobs already in execution. A typical objective is to minimize the sum of the waiting times $s_i - q_i$. Once the problem is solved, only the jobs with $s_i = t$ are dispatched. The remaining jobs with $s_i > t$ are queued again with their original q_i. It is the workload management system software that decides the dispatching time t and the subsequent dispatching times.

A solution to the problem (i.e., a *dispatching decision*) is obtained according to a policy using the current system status, such as the queued jobs, the running jobs and the availability of the resources. A sub-optimal solution could cause exceptional delays in the queue, hurting the QoS. While a (near-)optimal solution is a critical requirement in HPC systems, the on-line job dispatching problem is an NP-hard problem [6] and thus needs to be addressed with a dedicated approach. In [5,8], the first CP-based dispatchers for HPC systems are developed and tested on a workload trace collected from the Eurora system [11].

2.2 CP-based dispatchers for HPC systems

In the first dispatcher [5], the entire dispatching problem is modelled and solved using a CP solver. The second dispatcher [8] instead relies on a hybrid method. While the scheduling problem is modelled and solved in a CP solver, the allocation problem is solved separately using a heuristic search algorithm. We will refer to them as `PCP` and `HCP`, respectively, to mean the use of a Pure CP and a Hybrid CP method in their dispatching algorithms.

Scheduling In both `PCP` and `HCP`, the scheduling problem is modeled with Conditional Interval Variables (CIVs) [25]. A CIV $\tau_i \in \tau$ represents a job i and defines the time interval during which i runs. At a certain dispatching time t, there may already be jobs in execution which were previously scheduled and allocated. We refer to such jobs as running jobs. The scheduling model considers in the τ variables both the running jobs and the queued jobs in Q. The properties $s(\tau_i)$ and $d(\tau_i)$ correspond respectively to the start time and the duration of the job i. Since the actual runtime duration d_i^r of a running or queued job i is unknown at the modeling time, `PCP` and `HCP` rely on an estimation and use the expected duration d_i for $d(\tau_i)$. Thus we have $d(\tau_i) = d_i$ for the queued jobs and $d(\tau_i) = s(\tau_i) + d_i - t$ for the running jobs. While the start time of the running jobs have already been decided, the queued jobs have $s(\tau_i) \in [t, eoh]$, where eoh is the end of the worst-case makespan calculated as $t + \sum_{\tau_i} d(\tau_i)$. Expected durations d_i are supplied by the users. In the absence of this information, the dispatchers use the default wall-time of the queue. It is important to note that even user-supplied values tend to be equal to the wall-time of the queue, which is indeed the maximum allowed value for d_i. We will refer to the use of such d_i to define $d(\tau_i)$ as the *wall-time approach*.

Unlike `PCP`, `HCP` searches for a start time for the first m jobs in Q (referred to as $\bar{Q}$). The remaining jobs in $Q \setminus \bar{Q}$ are still in the model, but they are postponed to the end of the makespan by fixing their start time as $s(\tau_i) = eoh - d(\tau_i)$. The capacity constraints in `PCP` are enforced via a `cumulative` constraint for all $n \in N$ and for all $r \in R$, ensuring that at any given time in the makespan

the total $req_{i,r}$ of the jobs i using r does not exceed $cap_{n,r}$. In HCP, resources of the same type across all nodes are considered as a pool of resources, hence the cumulative constraints are posted for each $r \in R$ with the total capacity $Cap_r^T = \sum_{n \in N} cap_{n,r}$. Any infeasibility that may be introduced due to this modelling choice is fixed during the allocation phase. HCP considers also power as a resource type, allowing to restrict the total power consumption of the jobs. We here omit this feature as it is not relevant to our study.

We consider the objective function which minimizes the sum of the waiting times of the jobs. In PCP it is formalized as $\sum_{\tau_i} \max(0, \frac{s(\tau_i)-q_i-ewt_i}{ewt_i})$. It is a weighted sum so as to give priority to the jobs that stay in the queue longer than their ewt_i. The ewt_i value is the average waiting time of the queue where i is submitted, and is obtained by analyzing the Eurora workload data which was collected by the PBS dispatcher [22]. In the objective function of HCP, the weights are slightly different, giving priority to the jobs of the queues with lower expected waiting times: $\sum_{\tau_i} \frac{\max(ewt_i)}{ewt_i} * (s(\tau_i) - q_i)$. We will explain later how the corresponding scheduling models are solved by PCP and HCP.

Allocation In PCP, the allocation problem is modelled via an `alternative` constraint [25] for all $\tau_i \in \tau$, which ensures that the requested resources $req_{i,r}$ are satisfied by selecting a subset of the alternative possibilities for allocating i. An alternative possibility is an optional CIV, which may or may not be present in the allocation decision, and represents an individual allocation to the resources of a given node n. Instead in HCP, it is solved by a PRB algorithm for the jobs which have $s(\tau_i) = t$ after the scheduling model is solved. This heuristic algorithm iteratively tries to allocate each scheduled job using the best-fit allocation strategy. The jobs are chosen based on their priority. The jobs that have been waiting the longest at time t have the highest priority. Such a priority is calculated in line with the priority of the jobs in the objective function: $\frac{\max(ewt_i)}{ewt_i} * (t - q_i)$. As a tie breaker, job demand is used, which is the job's resource requirements multiplied by job duration $d(\tau_i)$. Hence, among the high priority jobs, those that have requested fewer resources and have shorter durations have further priority. Since the scheduling decision may contain some inconsistencies due to considering the resources of the same type as a pool, a job may not be allocated, in which case it is postponed to the next dispatching time.

Search To solve the scheduling and the allocation model altogether, PCP uses the self-adapting large neighborhood search algorithm [24] which is the default search available in the solver where PCP is implemented [26]. HCP instead uses a custom search algorithm derived from the schedule-or-postpone algorithm [30] to solve the scheduling model. The criteria used to select a job among all the available ones at each decision node follows the priority rule used in the PRB allocation algorithm, thus preferring the jobs that can start first and whose priority are highest. Note that the priorities are calculated once statically at the dispatching time t before search starts. Due to problem complexity, search in both PCP and HCP is bounded by a time limit δ. Thus, the best solution found within the limit is the dispatching / scheduling decision. If, however, no solution

is found within the limit, the search is restarted with an increased time limit $2 * \delta$. This procedure continues while no solution is found and $\delta \geq \delta_{max}$, where δ_{max} is the maximum time available to generate a decision.

3 Resiliency to Heavy Workloads

In this section, we *reduce the model size and improve the search control* of the dispatchers in an effort to make the dispatchers resilient to heavy workloads and applicable to on-line dispatching.

At a dispatching time t, `PCP` searches for a solution for all the jobs in Q which can be very time consuming when many jobs are waiting. While this problem is tackled in `HCP` by searching for a solution for the jobs in $\bar{Q}$ and postponing the remaining jobs in $Q \setminus \bar{Q}$ to the end of the makespan, there raises another issue: when many jobs are postponed in the same way, they are likely to overlap and create excess demand for the system resources at a given time in the schedule. It may therefore be not be possible to find a feasible solution that satisfies the resource constraints, consequently the entire Q may be postponed to the next dispatching time $t + 1$. To address this problem, we remove the remaining jobs jobs in $Q \setminus \bar{Q}$ from the model and place them in the queue with their original q_i.

During the typical operation of an HPC system, job submission by users has a stochastic nature and actual runtime durations are known only when jobs terminate. Additionally, at a dispatching time t, only the jobs with $s(\tau_i) = t$ are dispatched. Thus, it is not fruitful to generate a dispatching decision for the entire schedule makespan $[t, eoh]$. We therefore remove from the model all the jobs requiring more amount of resources than available at time t and queue them again with their original q_i. In addition to reducing the model size in terms of decision variables, we also eliminate the unnecessary variables and constraints in the model of a given problem instance. Specifically, for a given resource type r (in a node n), if none of the jobs in the model require it, we remove the corresponding `cumulative` constraint from the model. Moreover, in `PCP`, if there is no availability to allocate i in the system resources, we remove i and its corresponding `alternative` constraint from the model, and queue it again with its original q_i. Note that removing jobs from the model and putting them back in the queue does not cause any starvation problem. As we will argue in Section 4 and confirm experimentally in Section 6, their priority grow with their slowdown and eventually they are all dispatched.

During search for a solution, both solvers of `PCP` and `HCP` use a time limit δ to interrupt the search and return the best solution found. If, no solution is found within the limit, the search is restarted with an increased time limit $2 * \delta$. In the latter case, the dispatchers cannot distinguish an unsatisfiable problem instance from a difficult instance that is not solved yet. This has the consequence of searching for a solution again and again for an instance known to be unsatisfiable. To address this problem, we add the solver state to the search control. Consequently, if the solver proved unsatisfiability, this will be known when the search is interrupted by the time limit, and the subsequent restart will be avoided

by placing the jobs in the queue for the next dispatching time. Finally, we avoid a restart if the solution quality did not change after k consecutive restarts.

In the following, we refer to the versions of PCP and HCP whose model and search control are built as described here as $\texttt{PCP}_1$ and $\texttt{HCP}_1$.

4 Incorporation of Job Duration Prediction

A straightforward way to incorporate the duration prediction d_i^d of a job i into our dispatchers is to use it for defining the duration $d(\tau_i)$ as $d(\tau_i) = d_i^d$ for the queued jobs and $d(\tau_i) = s(\tau_i) + d_i^d - t$ for the running jobs, without any other changes to the dispatchers. In this section, we argue that this naive use may be ineffective, if not worsen the QoS, thus we adapt the model and search algorithm of both dispatchers to the use of job duration predictions in order to obtain high QoS levels in terms of job waiting times and slowdown.

A duration prediction d_i^d of a job i may be perfectly accurate ($d_i^d = d_i^r$), underestimated ($d_i^d < d_i^r$), or overestimated ($d_i^d > d_i^r$). If a running job i is underestimated, at a certain dispatching time t, we will have $s(\tau_i) + d_i^d < t$ and thus $d(\tau_i) = s(\tau_i) + d_i^d - t < 0$. That is, the duration of a running job will have a negative value even if the job is still running. A negative $d(\tau_i)$ for a running job directly affects the calculation of the makespan $\sum_{\tau_i} d(\tau_i)$ of the queued jobs. With a reduced makespan, it may not be possible to find a schedule and/or allocation for the queued jobs, consequently they may all be postponed to the next dispatching time $t+1$, worsening the QoS. If instead, a running job is overestimated at t, we will surely have $s(\tau_i) + d_i^d > t$ and $d(\tau_i) > 0$, thus the makespan will not be shorter than necessary.

To *address the problem of duration prediction underestimation*, we extend the duration $d(\tau_i)$ of a running job i which has $d(\tau_i) < 0$ at time t. Specifically, we redefine it as $d(\tau_i) = 1$, assuming that the job i needs at least one more unit of time as of t. This value is necessary and sufficient. It is the minimum value necessary to prevent a feasible problem instance from turning into an unfeasible one, as the makespan will be large enough to fit all the queued jobs in a schedule. To show that it is sufficient, we remind that at t, only the jobs for which the dispatcher decides that $s(\tau_i) = t$ are dispatched (the remaining are queued again). The allocation decision made for such jobs is valid until the next dispatching time $t+1$ and is not affected by the actual runtime durations of the running jobs even if they are underestimated. By using the minimum possible value for the duration of the underestimated running jobs, we keep the search space size compact. Our initial experiments confirm that higher values of $d(\tau_i)$ make the problem more difficult. In the following, we refer to this version of $\texttt{PCP}_1$ and $\texttt{HCP}_1$ as $\texttt{PCP}_2$ and $\texttt{HCP}_2$.

Even if the job duration prediction is accurate, resulting in $d_i^d \sim d_i^r$ for all jobs, the dispatchers may still not be able to exploit them fruitfully for targeting low job waiting time $s(\tau_i) - q_i$ and slowdown $(s(\tau_i) - q_i + d_i^r)/d_i^r$. As we saw in Section 2.2, both dispatchers assign a priority to the jobs that should not wait long. Then the jobs with higher priority are forced to be scheduled first via the

Enhancement	PCP$_1$	HCP$_1$	PCP$_2$	HCP$_2$	PCP$_3$	HCP$_3$
Reduced model size, improved search control.	✓	✓	✓	✓	✓	✓
Addressing duration prediction underestimation.			✓	✓	✓	✓
Job durations in the obj. function and search.					✓	✓

Table 1: Dispatcher versions.

objective function, as well as in the custom search of the scheduling problem and in the heuristic search of the allocation problem of the HCP dispatcher. However, job duration $d(\tau_i)$ is ignored in the priority. It is used only as a tie breaker among the jobs having the same priority during the search of the scheduling and the allocation problems of HCP. The priority instead focuses on a relation between the current waiting time $t - q_i$ of the job i and its expected waiting time ewt_i. The problem is that ewt_i is not a job specific feature that can be decided on-line at the time of dispatching. It is a feature of the queue where the job is submitted and is calculated offline. Such a value may not be informative on the current job submission status so as to generate a dispatching decision of high quality.

We tackle this limitation by involving *job durations in the objective function and in the search* of the scheduling and allocation, via the use of job slowdown as job priority. Thus, the new objective function and the priority of a job i at a dispatching time t become $\sum_{\tau_i} \frac{s(\tau_i)-q_i+d(\tau_i)}{d(\tau_i)}$ and $(t - q_i + d(\tau_i))/d(\tau_i)$, respectively. This is the normalization of the job waiting time, which has a higher value for jobs waiting more than their duration than for jobs waiting less than their duration. We foresee the following benefits. First, since it gives priority to short jobs, the dispatcher will aim at lowering both the total job waiting times and the total job slowdown, as required by modern HPC applications. Our experimental results in Section 6 show that by giving priority to short jobs, we never penalize the medium and long jobs. Second, it prioritizes the jobs based on a job specific feature $d(\tau_i)$ which can be calculated on-line and which can reflect better the current job submission status. Finally, integrating $d(\tau_i)$ in the objective function and search of the dispatchers paves the way to exploit job duration predictions.

In the following, we refer to the versions of PCP$_2$ and HCP$_2$ whose model and search algorithms are adapted as described here as PCP$_3$ and HCP$_3$. Table 1 summarizes all the dispatcher versions. We note that, similar to HCP, the HCP$_3$ dispatcher uses the job priorities in the custom search of the scheduling problem and in the heuristic search of the allocation problem, and calculates the priorities once statically at the dispatching time t before search starts. Our initial experiments revealed that updating them dynamically during search is not beneficial. As we described in Section 2.2, the search of PCP relies on the default search of the underlying solver and does not exploit priorities. We observed in our initial experiments that the custom search of the scheduling model in HCP is valuable also for PCP to solve the entire scheduling and allocation problem, hence we adopt that kind of search and exploit priorities also in PCP$_3$.

5 Experimental Study

To evaluate the significance of our approach, we conducted an experimental study, by simulating on-line job submission to an HPC system.

HPC system and its workload dataset Our study is based on a workload trace collected from the Eurora system [11], with (the portions of) which the original CP-based dispatchers were tested. [5,8,9,7]. We repeated the same study using another workload trace collected from the Gaia system [14] and obtained similar results which we omit in the paper due to space restrictions. The Eurora system was hosted by CINECA [12], the largest Italian datacenter. Eurora occupied the first place in the Green500 list of June 2013, and was in production until August 2015. It consisted of 64 nodes, each equipped with 2 8-core GPUs, 16GB of RAM memory, and 2 accelerators: GPUs and MICs. The workload, collected by the PBS dispatcher between March 2014 and August 2015, consists of logs for over 400,000 jobs submitted to one of its four queues, including job duration and detailed resource usage. The workload is dominated by short jobs (under 1 hour), making up 93.14% of all jobs, while the remaining 6.10% are medium jobs (between 1 and 5 hours) and 0.75% are long jobs (over 5 hours).

Job duration prediction To derive job durations, we used three prediction methods with varying accuracy levels, and underestimation and overestimation rates: (i) the wall-time approach, (ii) a data-driven prediction heuristic [19] which is simple to implement and has a low overhead, and as a baseline (iii) the actual runtime (real) durations. In [19], the authors have applied the heuristic prediction to the Eurora dataset. The mean absolute error (MAE) of the heuristic and the wall-time approach with respect to the real duration were shown to be 40 mins and 225 mins, respectively. The heuristic prediction shows thus an improvement of 82% over the wall-time approach. In Figure 1, we show the empirical cumulative distribution function (ECDF) of the prediction accuracy $A = d_i^r / d(\tau_i)$, the ratio between the real and the predicted duration of a job, of all the three methods. The empirical ECDF shows the proportion of scores that are less than or equal to each score of A on Eurora. When $A = 1$, the duration $d(\tau_i)$ matches the real duration d_i^r. We have underestimation when $A > 1$, overestimation when $A < 1$. In theory, we should not have underestimation with the wall-time approach because in a real system a job is killed if it takes longer than its d_i. However, a system requires extra time after a job is killed or completed to bring the resources on-line again and this extra time is reflected to the dataset. Therefore, in some cases we have $A > 1$ in Figure 1. We have $0.75 \leq A \leq 1.25$ for about 50% of the workload with the heuristic, and for less than 10% with the wall-time approach. On the other hand, the heuristic introduces considerable underestimation. The exact under and overestimation rates are 3.6% and 96.3% for the wall-time and 25.8% and 53.7% for the heuristic, respectively.

Experimental setup We used the open-source discrete event simulator AccaSim [18] to simulate the Eurora system with its workload dataset. Each job submission is simulated by using its available data, for instance, the owner, the

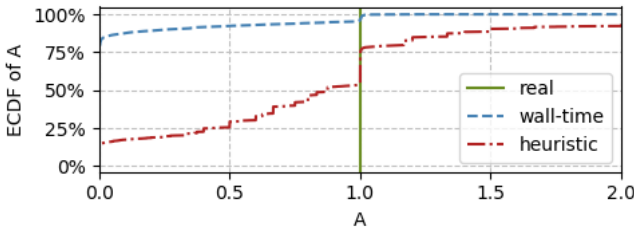

Fig. 1: The distribution of the accuracy of the three prediction methods.

requested resources, and the real duration, the execution command or the name of the application executed. AccaSim uses the real duration to simulate the job execution during its entire duration. Therefore job duration prediction errors do not affect the running time of the jobs with respect to the real workload data. The dispatchers under study are implemented using the AccaSim directives to allow them to generate the dispatching decisions during the system simulation.

With the heuristic prediction, as opposed to calculating the predictions off-line as in [19], we calculate them on-line during the simulation and update the knowledge base upon job terminations. The accuracy of the heuristic thus depends on the generated dispatching decisions. As a CP modelling and solving toolkit, we used Google OR-Tools[4] version 6.7 and ported it to Python 3.6 to implement the dispatchers in AccaSim. The `PCP`, $\texttt{PCP}_1$, and $\texttt{PCP}_2$ dispatchers use the default search algorithm of OR-Tools for CIVs, which is the schedule-or-postpone algorithm. As explained in Sections 2.2 and 4, all the other dispatchers use the custom search derived from schedule-or-postpone. In terms of the dispatcher parameters, we set $\delta = 1s$, $k = 2$, and $\delta_{max} = 16s$ to small values to keep the dispatcher overhead low. We keep $m = 100$ as in `HCP`. Both dispatchers need in some of their versions the estimated waiting time $ewt_Q = \frac{1}{|Q|}\sum_{i \in Q} s_i - q_i$ of each queue Q in the system. These values were calculated for the Eurora workload in [5,8] and reused here. All experiments were performed on a dedicated server with a 16-core Intel Xeon CPU and 8GB of RAM, running Linux Ubuntu 16.04. The source code of the CP-based dispatchers is available at `https://git.io/fjia1`.

6 Experimental Results

In this section, we report our experimental results. While the best and the final versions of the dispatchers are $\texttt{PCP}_3$ and $\texttt{HCP}_3$, all the previous versions (`PCP`, $\texttt{PCP}_1$, $\texttt{PCP}_2$, `HCP`, $\texttt{HCP}_1$, $\texttt{HCP}_2$) appear in the experiments in order to evaluate each of our contributions. To refer to a dispatcher using a certain job duration prediction method, we append `-W`, `-D` or `-R` to the name of the dispatcher for the Wall-time approach, the Data-driven heuristic and the Real duration, resp.

[4] `https://developers.google.com/optimization/`

Dispatcher	PCP	PCP_1-W	PCP_3-R	PCP_3-D	HCP	HCP_1-W	HCP_3-R	HCP_3-D
Avg. disp. time [ms]	∞	743	692	701	1,014	703	523	575
Total pred. time [s]	-	-	-	289	-	-	-	308
Total sim. time [s]	∞	262,436	261,985	262,764	374,788	245,663	201,223	215,814
Avg. # of intervals	-	145	94	115	379	100	51	63
Avg. # of req. res.	-	853	142	584	6,267	1,292	258	571
Avg. # of avl. res.	-	1,476	1,471	1,473	1,487	1,477	1,473	1,474

Table 2: Times and problem sizes.

6.1 Dispatcher performance and problem size

We first assess the impact of reducing the model size and improving the search control of the dispatchers for resiliency to heavy workloads. Following the original dispatchers PCP and HCP, we use the wall-time approach in PCP_1 and HCP_1 for job duration prediction, and compare the performance of and the problem size in PCP and PCP_1-W, as well as HCP and HCP_1-W. We report in Table 2 the mean CPU time spent in generating a dispatching decision over all dispatcher invocations, including the time for modeling the dispatching problem instance and searching for a solution. We also report the total simulation time from the first job submission until the last job completion, and the average problem size: number of intervals, number of requested resources, number of available resources.

PCP crashes before the completion of the entire workload, demonstrating that it is not resilient to heavy workloads. We therefore underline the improvement reached by the PCP_1-W dispatcher which is now able to process the workload. Compared to HCP, the HCP_1-W dispatcher reduces the total time by around 34% and reduces the problem size and time required for dispatching significantly. These results demonstrate that our approach has significantly better performance, making the dispatchers applicable to heavy workloads and paving the way to the use of CP-based dispatchers for HPC on-line dispatching.

6.2 Quality of the dispatching decisions

Next, we evaluate the value of adapting the model and search algorithm of the dispatchers to the use of job duration predictions by comparing the quality of the decisions made by PCP, PCP_1, PCP_2, PCP_3, as well as by HCP, HCP_1, HCP_2, HCP_3. Since we are aiming at reducing both the slowdown and waiting time of jobs, we consider both of these metrics. We first study the effectiveness of PCP, PCP_1, PCP_2, and PCP_3 with each job duration prediction method. Then, we analyze HCP, HCP_1, HCP_2, and HCP_3. We show the results of PCP_2 and HCP_2 only in conjunction with the data-driven heuristic. This is because on our workload the heuristic has a considerable underestimation rate while the other prediction methods have negligible or no underestimation, so the behaviour was very similar to PCP_1 and HCP_1. We also compare the various dispatchers with the performance of PBS in the original system, by calculating the slowdown and waiting time from the workload data.

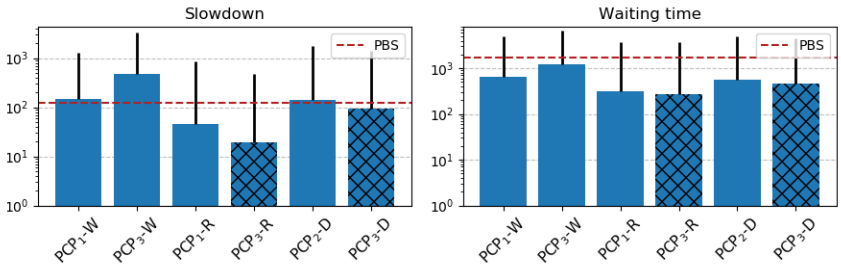

Fig. 2: Average and error bars showing one standard deviation of slowdown and waiting times [s] using the PCP dispatchers.

PCP results Figure 2 shows the slowdown and waiting times obtained by various versions of the dispatchers, compared to PBS. PCP is missing from the plot due to the fact that it is not able to process the workload, hence we consider PCP$_1$-W as a baseline, which is the enhancement most similar to the original algorithm. Additionally, we do not report the results of PCP$_1$-D because the simulation was too heavy and did not terminate in more than two weeks, so we interrupted it. We believe the long simulation time is due to the fact that PCP$_1$-D does not deal with underestimation, so it tends to use the maximum time limit for the instances in which jobs are underestimated, generating long queues.

A first observation is that, our best dispatcher coupled with the best duration predictor (PCP$_3$-R) and the heuristic predictor (PCP$_3$-D) always outperform PBS. PCP$_3$-W has lower performance compared to PCP$_1$-W. This is probably because the wall-time approach has a high overestimation rate, which is not beneficial when the dispatcher involves job durations in dispatching decisions. However, if we look at the dispatchers using real durations, we observe a significant increase in performance compared to PCP$_1$-W but also when moving from PCP$_1$-R to PCP$_3$-R. The reduction in the slowdown and waiting time from PCP$_1$-R to PCP$_3$-R reach up to 58% and 13%. This is due to the accuracy of the prediction method which does not present any underestimation nor overestimation. This proves that our approach is essential when a good quality prediction is available.

On a more realistic prediction, the results confirm that great care needs to be taken when integrating predictions. A straightforward integration of the predictions in previous algorithms is not helpful at all: PCP$_1$-D takes too long. By handling underestimation as in PCP$_2$-D, we are able to improve the results compared to PCP$_1$-W. Further improvement is observed when moving to PCP$_3$-D, demonstrating again the benefits of including predictions, albeit imperfect, into the model and search algorithm. Specifically, we observe 37% and 29% reduction in the average slowdown and the average waiting time.

HCP results Figure 3 shows the performance of HCP, HCP$_1$, HCP$_2$ and HCP$_3$ compared to PBS. Unlike the PCP case, here the original dispatcher HCP is able to process the entire workload so we can compare our results directly with the state-of-the-art method, besides PBS. We observe that in general, if we include

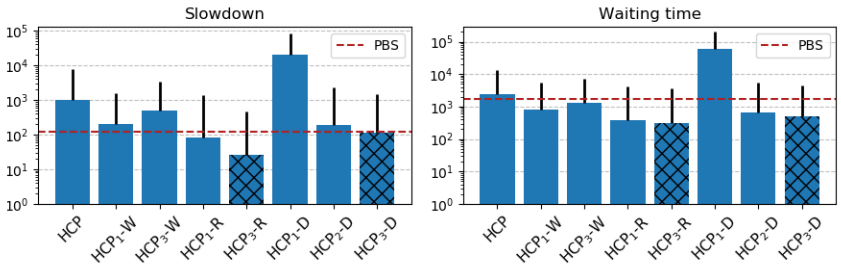

Fig. 3: Average and error bars showing one standard deviation of slowdown and waiting times [s] using the HCP dispatchers.

predictions with good accuracy and take into account also the underestimation problem, our algorithms can improve the quality of the dispatching decisions significantly (see $\texttt{HCP}_3\texttt{-D}$ and $\texttt{HCP}_3\texttt{-R}$ compared to HCP and PBS).

In more detail, we observe that simply moving from HCP to $\texttt{HCP}_1\texttt{-W}$, with an approach aimed at reducing the CPU time for dispatching, we also improve the quality of the solutions. $\texttt{HCP}_3\texttt{-W}$ does not improve $\texttt{HCP}_1\texttt{-W}$, since the accuracy of predictions using wall-time is rather low. We observe the most significant improvements over HCP with $\texttt{HCP}_3\texttt{-R}$, proving again the importance of our approach when a good quality prediction is available. The decreased performance of $\texttt{HCP}_1\texttt{-D}$ compared to all other algorithms confirms again that naively including predictions can be detrimental. The gains obtained by $\texttt{HCP}_2\texttt{-D}$ with respect to $\texttt{HCP}_1\texttt{-D}$ support again the need of dealing with underestimated jobs. We note that, while $\texttt{HCP}_1\texttt{-D}$ performs worse than the original HCP, $\texttt{HCP}_2\texttt{-D}$ becomes better than HCP and $\texttt{HCP}_3\texttt{-D}$ further improves $\texttt{HCP}_2\texttt{-D}$, demonstrating again the benefits of including predictions, albeit imperfect, into the model and search algorithm.

Discussion We conclude that suitable incorporation of job duration predictions in PCP and HCP, such as $\texttt{PCP}_3$ and $\texttt{HCP}_3$, can lead to significantly higher levels of QoS especially for workloads dominated by short jobs. To benefit from this potential, durations should rely on predictions with acceptable levels of accuracy, going beyond the standard wall-time approach. The quality of the decisions generated by $\texttt{PCP}_1\texttt{-W}$ and $\texttt{HCP}_1\texttt{-W}$ is much worse than $\texttt{PCP}_3\texttt{-R}$ and $\texttt{HCP}_3\texttt{-R}$. On the other hand, $\texttt{PCP}_3\texttt{-D}$ and $\texttt{HCP}_3\texttt{-D}$ offer valid alternatives to $\texttt{PCP}_1\texttt{-W}$ and $\texttt{HCP}_1\texttt{-W}$ with further reductions in problem size (as reported in Table 2) and with QoS measures closer to those of $\texttt{PCP}_3\texttt{-R}$ and $\texttt{HCP}_3\texttt{-R}$. Table 2 shows also the time cost of this gain. While $\texttt{PCP}_3\texttt{-D}$ and $\texttt{HCP}_3\texttt{-D}$ come each with a cost of prediction, the total simulation times of $\texttt{PCP}_3\texttt{-D}$ and $\texttt{PCP}_1\texttt{-W}$ are similar, and $\texttt{HCP}_3\texttt{-D}$ reduces notably the time with respect to $\texttt{HCP}_1\texttt{-W}$. The fact that the new dispatchers give priority to short jobs does not penalize the medium and long jobs, as can be witnessed in Figure 4. Finally, our approach does not affect the system utilization. We did not observe any major differences between the various dispatchers (results not shown due to space limitations). This is probably because all the dispatchers are using the best-fit allocation strategy.

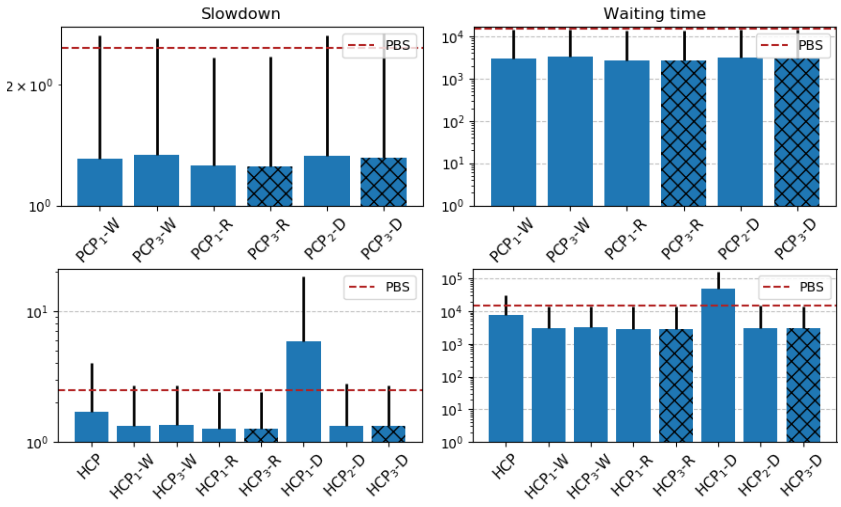

Fig. 4: Average and error bars showing one standard deviation of slowdown and waiting times [s] on medium and long jobs using all the dispatchers.

7 Related Work

Job duration prediction has been used to optimize job dispatchers. In [20], a simple linear model can improve the slowdown of backfilling techniques by 28%. On an IBM Blue Gene/P machine, adjusted user estimates were able to improve up to 20% the performance of the dispatchers favoring short jobs [37], while a predictive heuristic was shown to double the performance of a backfilling-based dispatcher [38]. The heuristic prediction that we employ here is similar to [38], however it considers more complex job profiles. When the prediction underestimates the job duration, [38] apply a correction step, to keep the job alive, similar to the adjustment that we make. However, they adjust the duration to define a new backfilling window whereas in our approach it is intended for defining valid CP models. Recently, machine learning methods were applied to predict job duration [23,36], including metadata such as job names as features. In fact, the heuristic method we employ also relies on job metadata, however it is much simpler, being an heuristic that does not require model training. Neither of the methods is integrated within a dispatcher for testing. An adaptive on-line machine learning method based on state space models is used in [28] to predict job duration. The authors show that their predictions allow for reducing waiting times by 25% in backfilling-based dispatchers.

Underestimation of job duration is a problem that appears often in the literature, since it negatively affects dispatcher performance, more than overestimation. Recently, [15] proposed a predictive method based on a censored regression model, which could minimize underestimation. Although promising, it requires heavier computations compared to the heuristic prediction we adopted here.

None of these works combine job duration prediction with a CP-based job dispatcher. Recently, [19] attempted to do that with `HCP`. However, it was done naively by replacing the expected durations with predicted durations, without adapting the model and search to deal with duration underestimation and to the use of predictions, as we did here. Moreover, the predictions were calculated off-line, as opposed to on-line, as we did here. Indeed, the results were not satisfactory, leading to worse performance compared to the wall-time approach.

8 Conclusions

We have argued that, while Constraint Programming (CP) is an effective approach in tackling the job dispatching problem, the-state-of-the-art CP-based dispatchers [5,8] are unable to satisfy the challenges of on-line dispatching and they are unable to take advantage of job duration predictions, which impede their adoption in HPC systems. We have introduced a class of novel CP-based dispatchers by building on [5,8] and redesigning their main components. We made them resilient to heavy workloads and applicable to on-line dispatching, as well as adapted them to the use of job duration predictions to obtain high QoS levels in terms of job waiting times and slowdown. We evaluated the significance of our approach on a workload trace collected from an HPC system, using predictions with different accuracy and underestimation and overestimation rates on the dataset. The experimental results are excellent. Compared to the original dispatchers, the time spent by the new dispatchers in generating decisions on a heavy workload is significantly reduced. Moreover, the new dispatchers can benefit from job duration predictions and generate decisions of higher QoS levels on a workload dominated by short jobs. The new dispatchers are thus more suitable for HPC systems running modern applications that employ short jobs. To benefit from this potential, the durations should rely on predictions with acceptable levels of accuracy, going beyond the standard wall-time approach. While the heuristic prediction considered in the paper is not the best, we have shown that it is a valid alternative to the wall-time approach, despite its simplicity.

In future work, we will include the allocation problem in the search of the new `PCP` dispatcher, which currently focuses only on the scheduling problem. We also plan to test the dispatchers with other, more sophisticated, duration prediction methods, as well as to integrate dedicated allocation strategies in the dispatchers so as to enhance system utilization.

Acknowledgements

We thank A. Bartolini, L. Benini, M. Milano, M. Lombardi and the SCAI group at Cineca for providing the Eurora data, and A. Borghesi and T. Bridi for sharing the original implementations of the dispatchers. We thank the IT Center of the University of Pisa and M. Marzolla for providing computing resources. C. Galleguillos has been supported by Postgraduate Grant PUCV 2018. A. Sîrbu has been partially funded by the SoBigData EU project (grant agreement 654024).

References

1. Altair: Altair PBS professional (2019), http://www.pbsworks.com
2. Anderson, M.J., Smith, S., Sundaram, N., Capota, M., Zhao, Z., Dulloor, S., Satish, N., Willke, T.L.: Bridging the gap between HPC and big data frameworks. PVLDB **10**(8), 901–912 (2017)
3. Ashby, S., Beckman, P., Chen, J., Colella, P., Collins, B., Crawford, D., Dongarra, J., Kothe, D., Lusk, R., Messina, P., et al.: The opportunities and challenges of exascale computing–summary report of the advanced scientific computing advisory committee (ASCAC) subcommittee. US Department of Energy Office of Science pp. 1–77 (2010)
4. Baptiste, P., Laborie, P., Pape, C.L., Nuijten, W.: Chapter 22 - constraint-based scheduling and planning. In: Handbook of Constraint Programming, Foundations of Artificial Intelligence, vol. 2, pp. 761–799. Elsevier (2006)
5. Bartolini, A., Borghesi, A., Bridi, T., Lombardi, M., Milano, M.: Proactive workload dispatching on the EURORA supercomputer. In: Proc. of Principles and Practice of Constraint Programming - 20th International Conference, CP 2014. LNCS, vol. 8656, pp. 765–780. Springer (2014)
6. Blazewicz, J., Lenstra, J.K., Kan, A.H.G.R.: Scheduling subject to resource constraints: classification and complexity. Discrete Applied Mathematics **5**(1), 11–24 (1983)
7. Borghesi, A., Bartolini, A., Lombardi, M., Milano, M., Benini, L.: Scheduling-based power capping in high performance computing systems. Sustainable Computing: Informatics and Systems **19**, 1 – 13 (2018)
8. Borghesi, A., Collina, F., Lombardi, M., Milano, M., Benini, L.: Power capping in high performance computing systems. In: Proc. of Principles and Practice of Constraint Programming - 21st International Conference, CP 2015. LNCS, vol. 9255, pp. 524–540. Springer (2015)
9. Bridi, T., Bartolini, A., Lombardi, M., Milano, M., Benini, L.: A constraint programming scheduler for heterogeneous high-performance computing machines. IEEE Transactions on Parallel and Distributed Systems **27**(10), 2781–2794 (2016)
10. Buddhakulsomsiri, J., Kim, D.S.: Priority rule-based heuristic for multi-mode resource-constrained project scheduling problems with resource vacations and activity splitting. European Journal of Operational Research **178**(2), 374–390 (2007)
11. Cavazzoni, C.: EURORA: a European architecture toward exascale. In: Proc. of Future HPC Systems - the Challenges of Power-Constrained Performance. pp. 1–4. ACM (2012)
12. CINECA: The Italian Interuniversitary Consortium for High Performance Computing (2019), https://www.cineca.it/
13. Cirne, W., Berman, F.: A comprehensive model of the supercomputer workload. In: Proc. of the Fourth Annual IEEE International Workshop on Workload Characterization. pp. 140–148 (Dec 2001)
14. Emeras, J., Varrette, S., Guzek, M., Bouvry, P.: Evalix: Classification and prediction of job resource consumption on HPC platforms. In: Proc. of 19th and 20th Workshops on Job Scheduling Strategies for Parallel Processing, JSSPP 2015 and JSSPP 2016. LNCS, vol. 10353, pp. 102–122. Springer (2015)
15. Fan, Y., Rich, P., Allcock, W.E., Papka, M.E., Lan, Z.: Trade-off between prediction accuracy and underestimation rate in job runtime estimates. In: Proc. of IEEE International Conference on Cluster Computing, CLUSTER 2017. pp. 530–540. IEEE Computer Society (2017)

16. Feitelson, D.G., Weil, A.M.: Utilization and predictability in scheduling the IBM SP2 with backfilling. In: Proc. of First Merged International Parallel Processing Symposium and Symposium on Parallel and Distributed Processing, IPPS/SPDP 1998. pp. 542–546 (1998)
17. Fox, G.C., Qiu, J., Jha, S., Ekanayake, S., Kamburugamuve, S.: Big data, simulations and HPC convergence. In: Proc. of 6th and 7th International Workshop Big Data Benchmarking, WBDB 2015. LNCS, vol. 10044, pp. 3–17. Springer (2015)
18. Galleguillos, C., Kiziltan, Z., Netti, A., Soto, R.: AccaSim: a customizable workload management simulator for job dispatching research in HPC systems. Cluster Computing (2019)
19. Galleguillos, C., Sîrbu, A., Kiziltan, Z., Babaoglu, Ö., Borghesi, A., Bridi, T.: Data-driven job dispatching in HPC systems. In: Proc. of Machine Learning, Optimization, and Big Data - Third International Conference, MOD 2017. LNCS, vol. 10710, pp. 449–461. Springer (2017)
20. Gaussier, É., Glesser, D., Reis, V., Trystram, D.: Improving backfilling by using machine learning to predict running times. In: Proc. of International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2015. pp. 1–10. ACM (2015)
21. Haupt, R.: A survey of priority rule-based scheduling. Operations-Research-Spektrum **11**(1), 3–16 (1989)
22. Henderson, R.L.: Job scheduling under the portable batch system. In: Proc. of Workshop on Job Scheduling Strategies for Parallel Processing, JSSPP 1995. LNCS, vol. 949, pp. 279–294. Springer (1995)
23. II, M.R.W., Herbein, S., Gamblin, T., Moody, A., Ahn, D.H., Taufer, M.: PRIONN: predicting runtime and IO using neural networks. In: Proc. of 47th International Conference on Parallel Processing, ICPP 2018. pp. 1–12. ACM (2018)
24. Laborie, P., Godard, D.: Self-adapting large neighborhood search:application to single-mode scheduling problems. In: Proc. of 3rd Multidisciplinary International Conference on Scheduling : Theory and Applications, MISTA 2007. pp. 276–284 (2007)
25. Laborie, P., Rogerie, J.: Reasoning with conditional time-intervals. In: Proc. of Twenty-First International Florida Artificial Intelligence Research Society Conference, FLAIRS 2008. pp. 555–560. AAAI Press (2008)
26. Laborie, P., Rogerie, J., Shaw, P., Vilím, P.: IBM ILOG CP Optimizer for scheduling. Constraints **23**(2), 210–250 (2018)
27. Lee, C.B., Schwartzman, Y., Hardy, J., Snavely, A.: Are user runtime estimates inherently inaccurate? In: Proc. of 10th Workshop on Job Scheduling Strategies for Parallel Processing, JSSPP 2004. LNCS, vol. 3277, pp. 253–263. Springer (2004)
28. Naghshnejad, M., Singhal, M.: Adaptive online runtime prediction to improve HPC applications latency in cloud. In: Proc. of 11th IEEE International Conference on Cloud Computing, CLOUD 2018. pp. 762–769. IEEE Computer Society (2018)
29. Nonaka, J., Sakamoto, N., Shimizu, T., Fujita, M., Ono, K., Koyamada, K.: Distributed particle-based rendering framework for large data visualization on hpc environments. In: 2017 International Conference on High Performance Computing Simulation (HPCS). pp. 300–307 (2017)
30. Pape, C.L., Couronne, P., Vergamini, D., Gosselin, V.: Time-versus-capacity compromises in project scheduling. AISB Quartetly pp. 19–31 (1995)
31. Qiu, J., Jha, S., Luckow, A., Fox, G.C.: Towards HPC-ABDS: an initial high-performance big data stack. Building Robust Big Data Ecosystem ISO/IEC JTC **1**, 18–21 (2014)

32. Reuther, A., Byun, C., Arcand, W., Bestor, D., Bergeron, B., Hubbell, M., Jones, M., Michaleas, P., Prout, A., Rosa, A., Kepner, J.: Scalable system scheduling for HPC and big data. J. Parallel Distrib. Comput. **111**, 76–92 (2018)
33. Rückemann, C.: Using parallel multicore and HPC systems for dynamical visualisation. In: 2009 International Conference on Advanced Geographic Information Systems Web Services. pp. 13–18 (2009)
34. Singh, D., Reddy, C.K.: A survey on platforms for big data analytics. Journal of big data **2**(1), 8 (2015)
35. SLURM: SLURM workload manager (2019), `http://slurm.schedmd.com`
36. Soysal, M., Berghoff, M., Streit, A.: Analysis of job metadata for enhanced wall time prediction. In: Proc. of 22nd Workshop on Job Scheduling Strategies for Parallel Processing, JSSPP 2018. LNCS, vol. 11332, pp. 1–14. Springer (2018)
37. Tang, W., Desai, N., Buettner, D., Lan, Z.: Analyzing and adjusting user runtime estimates to improve job scheduling on the Blue Gene/P. In: Proc. of 24th IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2010. pp. 1–11. IEEE (2010)
38. Tsafrir, D., Etsion, Y., Feitelson, D.G.: Backfilling using system-generated predictions rather than user runtime estimates. IEEE Transactions on Parallel and Distributed Systems **18**(6), 789–803 (2007)
39. Vivodtzev, F., Bertron, I.: Remote visualization of large scale fast dynamic simulations in a HPC context. In: Proc. of 4th IEEE Symposium on Large Data Analysis and Visualization, LDAV 2014. pp. 121–122. IEEE (2014)