Data-driven job dispatching in HPC systems

Cristian Galleguillos[1,2], Alina Sîrbu[3], Zeynep Kiziltan[1], Ozalp Babaoglu[1], Andrea Borghesi[1], and Thomas Bridi[1]

[1] Department of Computer Science and Engineering, University of Bologna, Italy
{zeynep.kiziltan, ozalp.babaoglu, andrea.borghesi3, thomas.bridi}@unibo.it
[2] Escuela de Ing. Informática, Pontificia Universidad Católica de Valparaíso, Chile
cristian.galleguillos.m@mail.pucv.cl
[3] Department of Computer Science, University of Pisa, Italy
alina.sirbu@unipi.it

Abstract. As High Performance Computing (HPC) systems get closer to exascale performance, job dispatching strategies become critical for keeping system utilization high while keeping waiting times low for jobs competing for HPC system resources. In this paper, we take a data-driven approach and investigate whether better dispatching decisions can be made by transforming the log data produced by an HPC system into useful knowledge about its workload. In particular, we focus on job duration, develop a data-driven approach to job duration prediction, and analyze the effect of different prediction approaches in making dispatching decisions using a real workload dataset collected from Eurora, a hybrid HPC system. Experiments on various dispatching methods show promising results.

1 Introduction

High Performance Computing (HPC) systems have become fundamental "instruments" for doing science much like microscopes and telescopes were during the previous century. The race towards exascale (10^{18} operations per sec.) HPC systems is in full swing with several efforts underway. Pushing current HPC systems to exascale performance requires a 50-fold increase in their speed and an order of magnitude increase in their energy efficiency [6]. While we can expect progress in hardware design to be a major contributor towards these goals, rest of the increase has to come from software techniques and from massive parallelism employing millions of processor cores. At these scales, job *dispatching* strategies become critical for keeping system utilization high while keeping waiting times low for jobs that are competing for HPC system resources.

HPC systems produce large amounts of data in the form of logs tracing resource consumption, errors and various other events during their operation. Data science can transform this raw data into knowledge through models built from historical data capable of anticipating unseen or future events. We believe that predictive computational models obtained through data-science tools will be indispensable for the operation and control of future HPC systems.

In an HPC system, a *dispatcher* decides which jobs to run next among those waiting in the queue (*scheduling*) and on which resources to run them (*allocation*). Ideally, dispatching decisions should complete all jobs in the shortest amount of time possible while keeping the system utilization high. This goal is achievable only with complete *a priori* knowledge of the workload which is rarely available at dispatching time. We therefore rely on predictive models to obtain useful knowledge about the workload from the log data of an HPC system, with the purpose of making better *dispatching decisions*. In particular, we focus on job duration, investigate an approach to job duration prediction based on the data available at job submission, and subsequently study the power of different approaches to prediction in making *dispatching decisions*. Our data-driven approach is based on a real workload dataset collected from Eurora [2], a hybrid HPC system equipped with CPU, GPU and MIC technologies to deliver high power efficiency. Experimental results on various dispatching methods show that job duration prediction can significantly benefit dispatching decisions in general, and specifically our simple data-driven approach can offer a valid alternative.

The contributions of this paper are twofold: (i) development of a simple yet effective data-driven approach for job duration prediction, (ii) analysis of the effect of different prediction approaches on various state-of-the-art dispatching methods.

The rest of the paper is organized as follows. In Sections 2, 3 and 4, we describe the dataset used, the prediction approaches and the dispatching methods in consideration, respectively. In Section 5, we evaluate the reliability of the predictions and then their impact on dispatching decisions. We discuss the related work in Section 6 and conclude in Section 7 with indications for future work.

2 Data description

The workload dataset used throughout this paper comes from the Eurora system which is hosted at CINECA[4], the largest datacenter in Italy, and was ranked first on the Green500 list in July 2013. Eurora has a modular architecture based on nodes (blades), each one having 2 octa-core CPUs and 2 expansion cards that can be configured to host an accelerator module. Of the 64 nodes, half of them host 2 powerful NVidia GPUs, meanwhile the other half are equipped with 2 Intel MIC accelerators. Each node has 16GB of RAM memory. These 64 nodes are dedicated exclusively to computation, with the user interface being managed by a separate node. Eurora has been used by scientists across Italy to perform simulation studies from different fields, hence the workload is heterogeneous.

The workload data includes logs for over 400,000 jobs submitted between March 2014 and August 2015. For each job, we have information on the submission, start and end times, queue, wall-time, user and job name, together with resources used and their allocation on the various nodes. The data has been collected through a dedicated monitoring system [6]. For our study, we selected

[4] The Italian Inter University Consortium for High Performance Computing (http://www.cineca.it).

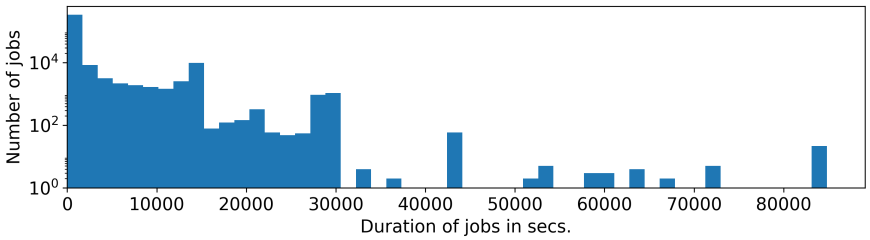

Fig. 1. Distribution of job durations on Eurora.

the 10 busiest months, resulting in a total of 372,321 jobs. Figure 1 shows the distribution of job durations of the selected workload. The maximum job duration is 24 hours. The figure demonstrates the existence of many short jobs and fewer longer jobs, with a long tailed distribution of job duration. As observed earlier, this is typical to HPC [21] and cloud systems [9], hence results on this system should apply to large scale computational infrastructures in general.

To evaluate the effects of prediction on different job types, we divided the jobs into classes: short jobs with duration of under 1 hour, medium jobs with duration between 1 and 12 hours, long jobs with duration over 12 hours. In terms of frequency, 93.15% of jobs fall into the short class (the vast majority), 6.82% into the medium class and only 0.03% into the long class. We also computed the CPU time used by jobs in each class. It is the medium class that uses most resources, with 87.63% of the total while short and long jobs use only 10.77% and 1.6%, respectively.

3 Job duration prediction

Duration of jobs is an important consideration in dispatching decisions and knowing them at job submission time clearly facilitates better algorithms. Dispatching algorithms are often developed with the assumption that job durations are known [17, 4]. Even if this is not practical, in some cases it may be possible to rely on user-provided estimates of job duration [17, 7]. Many HPC systems allow users to define a wall-time value, and use a default value when users fail to provide one. This wall-time, which in the case of Eurora is set on a per-queue basis, can be considered a crude prediction of job duration.

It has been shown that in general user estimations are not reliable [17], while predefined wall-times are inflexible to account for all user needs. In these conditions, prediction of job duration through other means may prove to be an important resource. Here, we describe a simple data-driven heuristic algorithm that relies on user histories to predict job duration. The data-driven approach is particularly useful when user data can be stored for longer periods of time, which is increasingly feasible through modern Big Data tools and techniques.

Our heuristic constructs job profiles from the available workload data. The profile includes job name, queue name, user-declared wall-time, and the number of resources of each type (CPU, GPU, MIC, nodes) requested. Each user is

analyzed separately. Prediction is based on the observation that jobs with the same or similar profiles have the same duration for long periods of time — there is a temporal locality of job durations. Then, at some point, the duration changes to a new set of values, which are again stable in time. This could be due for instance to changes in user behavior: a user first tests the code with short runs, then decides to run the real simulation which may last longer, then may decide to test again after having made changes, and so on. Another explanation could be switching between input datasets: the user performs repeated runs on one set of data, then moves to another. Hence, for each new job, our heuristic searches for the last job with a similar profile, and uses the duration of that job to predict the duration of the new one. We analyze users separately. The similar profile is identified using a set of consecutive rules. First, a full profile match is searched for, then if this does not exist in the user history, a profile where the job name has the same prefix is looked up. This follows from the observation that users often name jobs with similar durations with the same job name followed by a number (e.g. "job1","job2"). If this is unsuccessful, we allow for resources used to differ, as long as the full job name, queue and wall-time are the same. If also this search fails, we look for the same match but with the name prefix rather than the exact name. If none of these rules give a match, we look for the last job with the same name, or, as a last resort, the same name prefix. If all rules fail, then we take the wall-time as the predicted duration. In all cases, the prediction is capped by the wall-time.

We have also used machine learning to predict job duration. However, results were not satisfactory (not shown for space reasons), with our simple heuristic providing much better performance. We believe this is due to the temporal locality observed in the data, and also due to the fact that jobs with the same profile may have several different durations depending on when they were submitted. This means that a regular regression model would try to fit a wide range of values with the same features, resulting in an averaging of the observed durations.

4 Job dispatching methods

Job dispatching in HPC systems is an optimization problem which has been studied extensively [11, 15]. Since it is a hard problem [14], most of the proposed solutions are heuristic-based methods, which are fast but do not guarantee optimality. In this paper, we examine 5 of such methods reported in the literature. In the following, we give intuitions for the algorithms underlying these dispatching methods. We note that in the first three, we have adopted the *all-requested-computers-available* policy for resource allocation [24]. For each scheduled job, this policy searches sequentially the nodes in an attempt to find resources available for running the job, and if succeeds, it maps the job onto those nodes. The resource allocation policy of the remaining two are custom made and explained in their respective subsections. In all methods, the objective of the dispatcher is to minimize the total waiting time of the submitted jobs. The waiting time of a job is the time passed between its submission and its starting time.

Shortest job first, longest job first Shortest Job First (SJF) and Longest Job First (LJF) use the estimated duration at scheduling time, sorting all jobs that have to be scheduled in ascending (or descending) order, and then mapping the shortest job (or the longest job) to a resource [23]. Both algorithms continue moving through the sorted list until no available resources remain for allocating to the current job. The aim of SJF is to reduce the waiting time of the short jobs, thus causing delays for the execution of the long jobs. Conversely, LJF reduces the waiting time of the long jobs, causing slowdown for short jobs.

EASY-Backfilling A key element of many commercial dispatchers is the *backfilling* algorithm [24] which starts scheduling jobs stepping through a priority list such as SJF or LJF, or commonly (as also adopted here) using the jobs' submission order (first-in-first-out policy). If a job cannot be dispatched due to lack of available resources (blocked job), backfilling calculates the time in the future when enough resources will be released to run the blocked job, based on the estimated duration of running jobs. While the blocked job is waiting, the dispatcher maps other jobs in the queue over the available resources. If, however, the durations have been underestimated, the resources for the blocked job will not be available when needed, which can force termination of the running jobs. In such a case, EASY-Backfilling (EBF) [24] does not terminate the running jobs but instead delays the starting time of the blocked job. To keep all the jobs running until their termination, we have here adopted EBF.

Priority rule-based As an extension of the first-in-first-out policy, many dispatchers sort the set of jobs to be scheduled by certain priority, running those with higher priorities first. This algorithm is referred to as Priority Rule-Based (PRB) [18, 1] and is widely used in commercial HPC dispatchers[5,6]. In our work, the priority rules are based on [7] and sort the jobs to be scheduled in decreasing order of the jobs' urgency in leaving the queue. To determine if a job could wait in the queue, the ratio between the waiting time and the expected waiting time (assumed to be available for each queue the jobs are submitted) of the job is calculated. Then, jobs that are closer to surpass their expected waiting time have priority over the jobs that still could wait in the queue. As a tie breaker the "job demand" is used, which is the job's resource requirements multiplied by the estimated job duration. Hence, among the high priority jobs, those that have requested less resources and have shorter durations have further priority. The allocation process tries to assign each job to the nodes containing resources available for running the job. The nodes are also sorted by their current load (nodes with fewer free resources are preferred), thus trying to fit as many jobs as possible on the same node, to decrease the fragmentation of the system.

Hybrid constraint programming method One of the drawbacks of the heuristic methods is the limited exploration of the solution space. Recently, new ap-

[5] Altair PBS Works (http://www.pbsworks.com/).
[6] SLURM Workload Manager (https://slurm.schedmd.com/).

proaches have been proposed to improve the performance of traditional scheduling, without violating the real-time requirements. For example, Bartolini et al. [5] propose a HPC job dispatcher based on Constraint Programming (CP) that is able to outperform traditional PRB methods. To increase scalability, Borghesi et al. introduce a hybrid approach combining CP and a heuristic algorithm [7] (CPH). We adopted this last method in this paper.

CPH is composed of two stages. The first corresponds to scheduling the jobs using CP with the objective of minimizing the total waiting time. The schedule is generated using a relaxed model of the problem which considers each resource type as one unique resource, i.e., CPU availability corresponds to the sum of the available CPUs of all the computing nodes, memory availability corresponds to the sum of the memory availability of all the computing nodes, and so on. The model is solved with a custom search strategy guided by a branching heuristic using the scheduling policy of PRB. Due to the problem complexity, we do not insist on finding optimal solutions but impose a time limit to bound the search; the best solution found within the limit is the scheduling decision. The preliminary schedule generated in the first stage may contain some inconsistencies because of considering the available resources as a whole. During the second stage, which corresponds to the resource allocation, any inconsistencies are removed. If a job can be mapped to a node then it will be dispatched, otherwise it will be postponed. The second stage uses the allocation policy of PRB.

5 Experimental results

We have implemented a discrete event simulator for job submission and job dispatching, named AccaSim[7], and used it to simulate the Eurora system with the workload trace described in Section 2. AccaSim is a freely available Python library. At every time point, it checks if there are jobs to be dispatched. If so, it calls a dispatching method to generate a dispatching decision, and then simulates the running of the jobs on the system. AccaSim library already includes the implementations of the SJF, LJF and EBF dispatching methods. The PRB and CPH implementations are available for download in the AccaSim website. The experiments were ran on a CentOS machine equipped with Intel Xeon CPU E5-2640 Processor and 15GB of RAM.

The simulation study considered the five dispatching methods described in Section 4 together with three estimations of job duration: prediction based on wall-time (W), data-driven prediction presented in Section 3 (D) and real duration (R). The real duration was included to provide a baseline to which the other two predictions are compared. Therefore, for each of the five dispatching methods, there are three estimations of job duration, resulting in 15 combinations (e.g., for the SJF method we have SJF-W, SJF-D and SJF-R corresponding to wall-time prediction, data-driven prediction and real duration, respectively).

To compare the quality of the dispatching decisions of the 15 combinations, we have selected two criteria. The first is *job slowdown*, a common met-

[7] https://sites.google.com/view/accasim

ric for evaluating job scheduling algorithms[16], which quantifies the effect of each method on the jobs themselves and is directly perceived also by the HPC users. Slowdown of a job j is a normalized waiting time and is defined as $slowdown_j = (T_{w,j} + T_{r,j})/T_{r,j}$ where $T_{w,j}$ is the waiting time and $T_{r,j}$ is the duration of job j. A job waiting more than its duration has a higher slowdown than a job waiting less than its duration. The second criterion is the *number of queued jobs* at a given time. This metric is a measure of the effects of dispatching on the computing system itself, being directly related to system throughput: the lower the number of waiting jobs, the higher the throughput.

Next, we present the performance of our data-driven prediction of job duration and then continue with the evaluation of the various dispatching methods.

5.1 Prediction performance

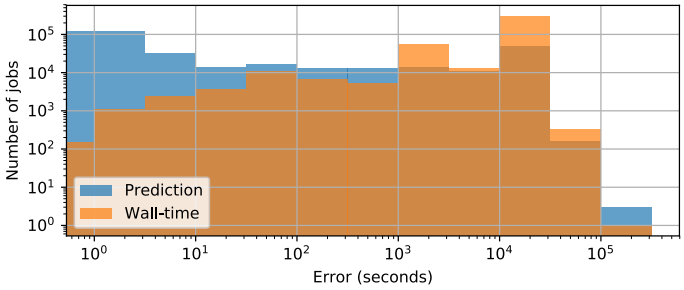

Fig. 2. Absolute data-driven prediction error, compared to wall-time prediction.

To evaluate the performance of our data-driven prediction of job duration, we compute the absolute error of the prediction and compare with that of the wall-time prediction. Over all the jobs in the system, our algorithm obtains a mean absolute error of 38.9 minutes. Using the wall-time, on the other hand, results in a mean absolute error of 225.11 minutes. Figure 2 shows the distribution of the absolute errors in the two cases, showing that in the data-driven case, these are concentrated towards small values, while in the case of the wall-time the distribution peaks at errors over 1 hour. The plot shows clearly that our data-driven prediction produces much better results compared to wall-time prediction.

5.2 Dispatching performance using prediction

To analyze the effects of prediction on job dispatching, we plot the distribution of our evaluation criteria for all 15 combinations of the dispatching methods and duration predictions. For easy visualization of distributions, we use box-plots that show the minimum and maximum values (top and bottom horizontal lines), the range between the 1st and 3rd quartiles (the colored box), the median (horizontal line within the box) and the mean (the triangles). Note that with the logarithmic scale on the vertical axis, some of these elements may be missing from the plots, meaning their value is zero.

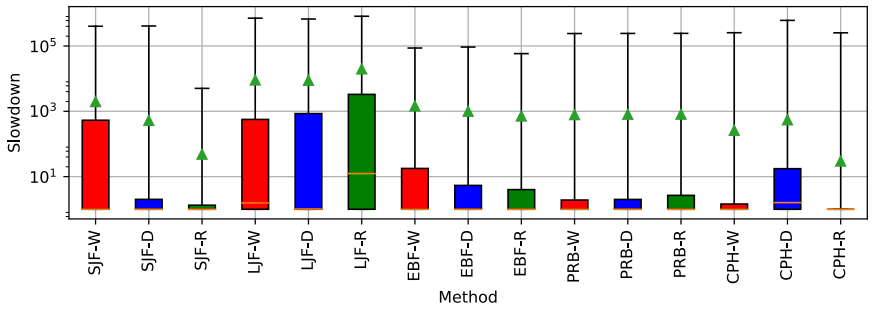

Fig. 3. Distribution of job slowdown for each method.

Effects of prediction on jobs. The first analysis looks at job slowdown for all 372,321 jobs dispatched. Figure 3 shows the distribution of slowdown achieved by each dispatching method with each prediction type. For better visualization, we plot only the jobs where slowdown is different from 1 in at least one method-prediction combination. The removed jobs are those that are dispatched immediately as they arrive in the system, so are not relevant for our comparison. As the figure shows, the dispatching methods displaying best performance when the most basic and least effective prediction is used (wall-time) are PRB and CPH, while the methods performing worst are LJF and SJF. This is understandable since the latter methods are quite simple while the former employ more sophisticated reasoning.

An interesting effect when using real duration is that not all dispatching methods show a clear benefit. While we observe a clear decrease in slowdown in SJF, EBF and CPH, for LJF a significant increase in slowdown is present, while for PRB no change is observed. We understand that prediction does not always help the dispatching methods. One possible explanation is that the incomplete nature of the dispatching methods tends to lead to suboptimal decisions which can sometimes be compensated by underestimation of job durations, which will not be possible anymore with a (perfect) prediction.

When using our data-driven prediction in the dispatching methods, we expect the performance to stay between the wall-time prediction and the real job duration. Figure 3 shows that this is true for most methods. In the cases of SJF and EBF, the real job duration improves the results, so does our prediction, albeit less effectively. In the case of LJF, real job duration worsens the results, so does our prediction, but less severely. PRB, which already does not benefit from real job duration, does not benefit from our prediction either. The only dispatching method where the performance improves with perfect prediction but decreases with our prediction is CPH. We believe this is because our data-driven prediction may sometimes underestimate job duration, which is never the case for wall-time and the real duration. CPH is not resilient to job duration underestimation, hence an imperfect prediction can actually be detrimental.

Even if PRB and CPH provide the best overall results, we observe that SJF comes in very close, with comparable slowdown, when adding prediction. However, the first two methods are more sophisticated and incur an overhead

when building the dispatching decisions, while SJF is a very simple strategy. Hence, in the presence of predictions, one may prefer to use a simple method such as SJF over the heavier methods such as PRB and CPH.

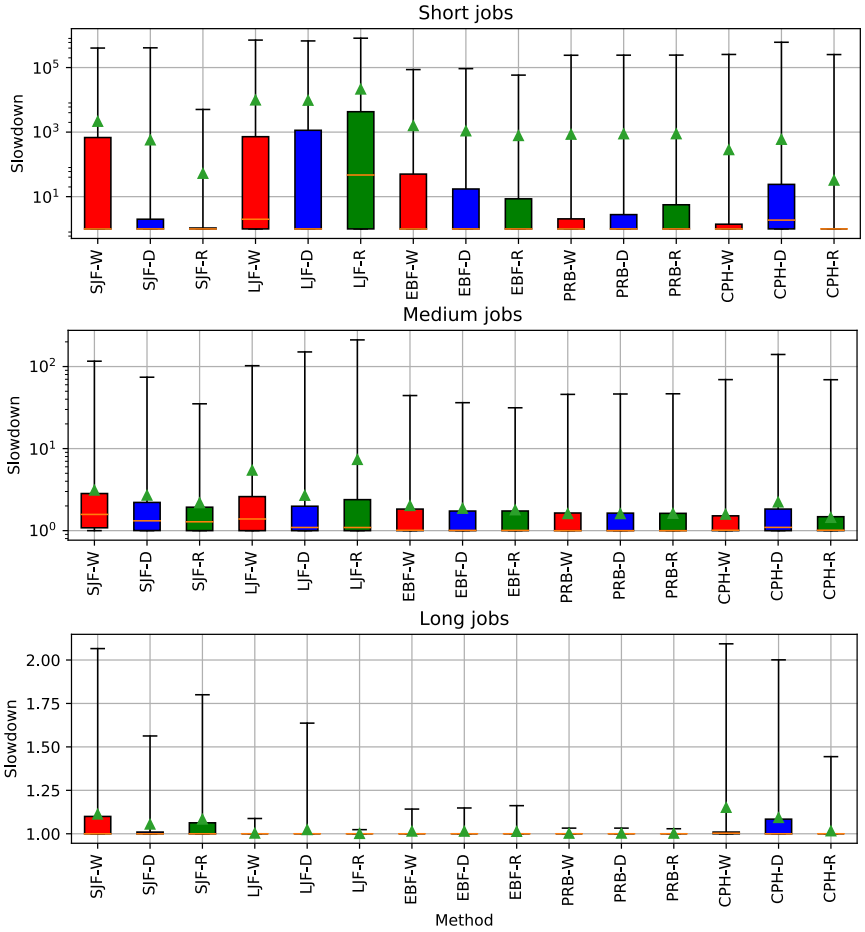

Fig. 4. Distribution of job slowdown for short, medium and long jobs for each method.

To better understand the effects of prediction, we also look at the different job classes. Figure 4 shows box-plots of slowdown distributions for short, medium and long jobs. When prediction is beneficial, we see that the jobs that benefit most are the short ones. This is good news, given that a large number of our jobs are short, as we saw in Section 2. Some smaller differences are also visible on medium jobs, while on long jobs the methods seem to be quite comparable, with slightly larger slowdown values in CPH and SJF compared to the rest.

Effects of prediction on the system. Besides effects on individual jobs, it is important to understand prediction's role in improving system-level behavior.

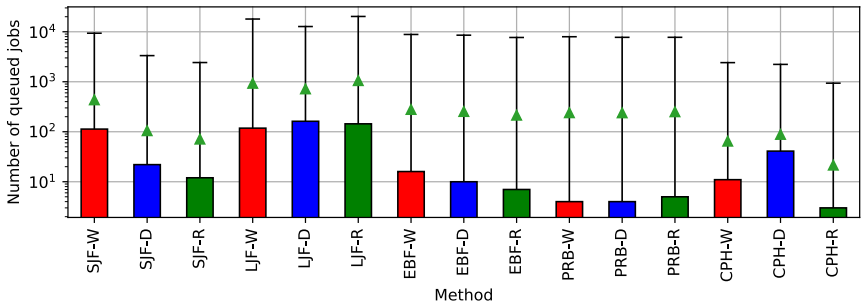

Fig. 5. Distribution of number of jobs waiting at every second, for each method.

For this we look at the size of the waiting queue. Figure 5 shows the distribution of the number of jobs in the queue at every second. We removed from the plot those time points where there were no jobs in the queue for any of the 15 combinations, because these corresponded to low system utilizations and have no value for our comparison. The figure shows that the effect on the system is similar to the performance measured by the slowdown. In particular, SJF and EBF are improved by prediction (both data-driven and real durations). PRB shows no difference, however queue size is already the shortest among all dispatching methods. LJF does not benefit from prediction, while CPH seems to be improved only by perfect and not by our data-driven prediction.

We also looked at the distribution of resource utilization (amount of CPUs, GPUs, MICs and memory used at each second), but we observed minor differences in the total use of resources between methods, so results are not shown.

6 Related work

A number of previous efforts have developed techniques for predicting interesting aspects of workloads such as power consumption and job duration [20, 10]. Borghesi et al. [8] propose a machine learning approach to forecast the mean power consumption of HPC applications using only information available at scheduling time, such as the resources requested, the maximum duration, the user, etc. Sirbu et al. [22] present a support vector machine model to predict the power consumption of jobs, taking also into account their variability.

Predicting the durations of HPC jobs have also been considered in previous research works, especially in relation to job dispatching [19, 3]. Tsafrir et al. [12] propose a model that uses the run times of the last two jobs to predict the duration of the next job. This prediction is then used for scheduling purposes. Their approach is lightweight and efficient, however, the prediction accuracy can be improved using more complex techniques like the ones proposed in this paper. Gaussier et al. [13] show the importance of estimating the duration of HPC jobs with backfilling schedulers. Their results clearly suggest that a backfilling policy benefits from accurate duration predictions; the only limitation is that their work focuses exclusively on a particular scheduling algorithm.

7 Conclusions

We have presented an analysis of the effect of job duration prediction on HPC job dispatching decisions, based on a real workload dataset collected from Eurora, a hybrid HPC system. We implemented five state-of-the-art dispatching methods and studied their performance in the presence of predictions based on a data-driven heuristic and on estimates based on wall-time. These two approaches to prediction were compared among themselves and also against a baseline: perfect prediction using the real job duration from the data.

Our conclusions are severalfold. First, our data-driven approach results in more effective predictions than the estimates based on wall-time. Second, even a perfect prediction does not necessarily benefit dispatching methods. One possible explanation is that the incomplete nature of the dispatching methods tend to lead to suboptimal decisions which can sometimes be compensated by underestimation of job durations. Third, prediction is nevertheless beneficial in the majority of the methods we have considered, and in the presence of our data-driven prediction, a simple dispatching method can become a valid alternative to the sophisticated state-of-the-art methods. Finally, when using prediction is advantageous, the main beneficiaries are the short jobs. Given the prominent presence of short jobs in typical HPC [21] and cloud system [9] workloads, our conclusions should apply to large-scale computational infrastructures in general.

The dispatching methods presented here exploit prediction in their scheduling component. In future work, we will also develop allocation heuristics that can exploit prediction, especially in the case of hybrid HPC systems. Additionally, we plan to extend our job duration prediction heuristic to include resources shared with other jobs, similar to [22], to improve our prediction power. Finally, we plan to integrate power predictions [22] into the dispatchers, to optimize not only the system response, but also energy consumption, which is mandatory for building exascale systems.

Acknowledgments. We thank Dr. A. Bartolini, Prof. L. Benini, Prof. M. Milano and Dr. M. Lombardi for fruitful discussions on the work presented here and for providing access to the Eurora data, together with the SCAI group in Cineca. We acknowledge the Cineca PM-HPC award allowing access to HPC resources. C. Galleguillos has been supported by Postgraduate Grant PUCV 2017. A. Sîrbu has been partially funded by the E.U. project SoBigData Research Infrastructure — Big Data and Social Mining Ecosystem (grant agreement 654024).

References

1. J. Buddhakulsomsiri and D. S. Kim. 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), 2007.
2. C. Cavazzoni. EURORA: a european architecture toward exascale. In *FutureHPC@ICS*, pages 1:1–1:4. ACM, 2012.

3. X. Chen and et al. Predicting job completion times using system logs in supercomputing clusters. In *DSN Workshops*. IEEE Computer Society, 2013.
4. A. A. Chandio et al. A comparative study of job scheduling strategies in large-scale parallel computational systems. In *TrustCom/ISPA/IUCC*, pages 949–957. IEEE Computer Society, 2013.
5. A. Bartolini et al. Proactive workload dispatching on the EURORA supercomputer. In *CP*, volume 8656 of *LNCS*, pages 765–780, 2014.
6. A. Bartolini et al. Unveiling eurora - thermal and power characterization of the most energy-efficient supercomputer in the world. In *DATE*, pages 1–6. European Design and Automation Association, 2014.
7. A. Borghesi et al. Power capping in high performance computing systems. In *CP*, LNCS, 2015.
8. A. Borghesi et al. Predictive modeling for job power consumption in HPC systems. In *ISC*, volume 9697 of *LNCS*, pages 181–199, 2016.
9. C. Reiss et al. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *SoCC*, page 7. ACM, 2012.
10. C. Storlie et al. Modeling and predicting power consumption of high performance computing jobs. *preprint arXiv:1412.5247*, 2014.
11. D. G. Feitelson et al. Theory and practice in parallel job scheduling. In *JSSPP*, volume 1291 of *LNCS*, 1997.
12. D. Tsafrir et al. Backfilling using system-generated predictions rather than user runtime estimates. *IEEE Trans. Parallel Distrib. Syst.*, 18(6):789–803, 2007.
13. É. Gaussier et al. Improving backfilling by using machine learning to predict running times. In *SC*, pages 64:1–64:10. ACM, 2015.
14. J. Blazewicz et al. Scheduling subject to resource constraints: classification and complexity. *Discrete Applied Mathematics*, 1983.
15. J. Cao et al. A taxonomy of application scheduling tools for high performance cluster computing. *Cluster Computing*, 9(3), 2006.
16. D. G. Feitelson. Metrics for parallel job scheduling and their convergence. In *JSSPP*, volume 2221 of *LNCS*, pages 188–206, 2001.
17. D. G. Feitelson and A. Mu'alem Weil. Utilization and predictability in scheduling the IBM SP2 with backfilling. In *IPPS/SPDP*, pages 542–546, 1998.
18. R. Haupt. A survey of priority rule-based scheduling. *Operations-Research-Spektrum*, 11(1):3–16, 1989.
19. A. M. Matsunaga and J. A. B. Fortes. On the use of machine learning to predict the time and resources consumed by applications. In *CCGRID*, pages 495–504. IEEE Computer Society, 2010.
20. H. Shoukourian, T. Wilde, and et al. Predicting the energy and power consumption of strong and weak scaling hpc applications. *Supercomputing frontiers and innovations*, 1(2):20–41, 2014.
21. A. Sîrbu and O. Babaoglu. A holistic approach to log data analysis in high-performance computing systems: The case of IBM blue gene/q. In *Euro-Par Workshops*, volume 9523 of *LNCS*, 2015.
22. A. Sîrbu and O. Babaoglu. Power consumption modeling and prediction in a hybrid CPU-GPU-MIC supercomputer. In *Euro-Par*, volume 9833 of *LNCS*, pages 117–130, 2016.
23. A. Streit. Enhancements to the decision process of the self-tuning dynp scheduler. In *JSSPP*, LNCS, 2004.
24. A. K. L. Wong and A. M. Goscinski. Evaluating the easy-backfill job scheduling of static workloads on clusters. In *CLUSTER*. IEEE Computer Society, 2007.