Constraint Programming-based Job Dispatching for Modern HPC Applications

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

[1] Pontificia Universidad Catlica de Valparaso, Chile
[2] University of Bologna, Italy
[3] University of Pisa, Italy

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 response times to generate dispatching decisions. Constraint Programming (CP) has been shown to be an effective approach for tackling job dispatching problems. State-of-the-art CP-based job dispatchers are unable to satisfy the challenges of on-line dispatching and they are unable to take advantage of job duration predictions. Both of these limitations jeopardize achieving high QoS levels, and consequently impede the adoption of CP-based dispatchers in HPC systems. In this paper, we propose a class of CP-based dispatchers that are more suitable for HPC systems running modern applications. The dispatchers we propose 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 [2], with big-data processing frameworks such as Hadoop and Spark increasingly integrated with HPC systems [1,25,22,15]. This modern computing has been powered by continuous and significant technology advances and achieved through innovations in computer architecture, programming models, and needs of end-user goals that could only be addressed by computational means.

An HPC system is a complex set of technologies which involves multiple disciplines of the Information and Communications Technology domain, such as hardware, software, infraestructure, communications, to take advantage of the computing hardware through the correct use of the available software large applications are executed in parallel in order to reduce their runtime and to process workloads as fast as possible. According to TOP500 [4], in 2020 is expected to reach the exascale computing (10^{18} float operations per second), such capacity represents a thousandfold increase over the current HPC systems. The current top #1 HPC system reaches a peak performance of ≈150 teraFLOPS (10^{15}) under the High Performance Linpack benchmark. The peak performance is calculated considering all of its computing units being used, i.e. 2,414,592 CPU cores. Therefore, such performance may be reached if the system is highly busy.

On a typical usage of HPC systems, users submit jobs which are self-contained scripts with several information such as resource request, walltime, command to execute, etc. to run without supervision. As the demand for HPC technology continues to grow, a typical HPC system receives a large number of jobs by its end users. However the maximum peak performance may not be reached even if the system is busy and with a high queue to process. This is due to starvation, which is caused by its management software due to an incorrect load balancing. This calls for the efficient management of the submitted workload and system resources. This critical task is carried out by the Workload Management System (WMS) software component, specially by its dispatcher which has the key role of deciding when and on which resources to execute the individual jobs.

Workloads of HPC systems engaged in data-driven analytics tend to be a mix of many short jobs that run for less than one hour, together with fewer longer jobs [23]. Hence, HPC job dispatchers need to rapidly process a large number of short jobs in making on-line decisions so as to improve the Quality-of-Service (QoS) which is particularly important when HPC systems are used to provide real-time services, such as big-data visualization [24,29,21], where response times are critical for acceptable user experience. As to improve response times the minimization of the waiting times is a typical metric to improve. However, this metric affects differently depending of the job size class. Waiting time gives greater emphasis on long jobs, as opposed to short jobs, which are much more common. The normalization of the waiting time is a solution to this problem, and is called slowdown which is the ratio between the actual job duration plus its waiting time and the actual job duration.

Our study consider an on-line analysis of workloads, assuming all jobs arrive over a period of time, then dispatchers must handle jobs in “real time” without knowing about future job arrivals. While the on-line job dispatching problem in HPC systems is NP-hard [5], it can be formulated as a job scheduling and resource allocation problem for which Constraint Programming (CP) has produced good results [3]. Indeed, two dispatchers has been proposed [4,7,6] showing good results on restricted test cases.

Despite the potential of these state-of-the-art CP-based job dispatchers, certain limitations hinder their adoption for modern HPC systems. As reported in [8], the first dispatcher is not resilient to heavy workloads, which are present in real scenarios, these are 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

[4] TOP500 Supercomputer sites https://www.top500.org/

arrive to the system. The second CP-based HPC dispatcher was initially employed in off-line mode [7], and later also in on-line mode [6] 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. These dispatchers were tested under different conditions, each one using different routines as to simulate specific workloads on specific system configurations with specific job attributes. However, to evaluate dispatchers the simulation of an HPC system is essential, and the current available HPC simulators do not allow to carry out experiments across different workload sources, resource types, and dispatching algorithms; neither are scalable to large workload datasets nor provides support for easy customization, such as Alea [19] and BatSim [12] simulators. This makes it difficult for users to develop novel advanced dispatchers by exploiting information regarding the current system status. 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 for this purpose, 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 [14,11,20]. 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 [28,18,16,13]. Recent studies show that the use of job duration predictions when generating dispatching decisions can substantially improve QoS levels in backfilling-based dispatchers [27,18,13,16].

The main contributions of our work are (i) a Simulator capable to easily carry out dispatching research and be able to conduct simulation on large datasets, (ii) a job duration predictor to cope with inaccurate user predictions, (iii) a class of novel CP-based dispatchers that are more suitable for HPC systems running modern applications and employ job duration prediction to achieve higher QoS levels.

2 HPC Systems

An HPC system provides a larger compute capability than is possible to achieve in a personal computer. Its architecture is typically an aggregation of several computers, each one of which can look pretty similar to a personal computer. HPC systems are usually shared among many users, where each user has a dedicated portion of the computers resources for a period of time to run her jobs. A job encapsulates an application software which usually tries to solve a problem and give answers to research questions. How fast a problem can be solved depends on how complex is itself and how fast is the machine where the application is executed.

A WMS is an important software of an HPC system, being the main access for the users to exploit the available resources for computing. Examples for WMS are PBS[5]and Slurm[6]. A WMS manages user requests and the system resources through critical services. A user request 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 are known as workload. The jobs are tracked by the WMS during all their states, i.e. from their submission time, to queuing, running, and completion. Once a job is completed, the results are communicated to the respective user.

A WMS offers distinct ways to users for job submission such as a GUI and/or a command line interface. A job is a self-contained script, which includes commands, arguments, input file paths, and the resource requirements; thus it doesn't require user intervention. A WMS periodically receives job submissions. It may receive submissions all day long; however, most of the submissions are received during working hours. Some jobs may have the same computational application with different arguments and input files, referring to the different running conditions of the application in development, debugging and production environments. When a job is submitted, it is placed in a queue together with the other pending jobs (if there are any). The time interval during which a job remains in the queue is known as waiting time. The queued jobs compete with each other to be executed on limited resources.

A job dispatcher decides which jobs waiting in the queue to run next (scheduling) and on which resources to run them (allocation) by ensuring high system utilization and performance. The dispatching decision is generated according to a policy using the current system status, such as the queued jobs, the running jobs and the availability of the resources. A suboptimal dispatching decision could cause resource waste and/or exceptional delays in the queue, worsening the system performance and the perception of its users. A (near-)optimal dispatching decision is thus a critical aspect in a WMS.

3 Job Duration Prediction

Duration of jobs is an important consideration in dispatching decisions and knowing them at job submission time clearly facilitates dispatching algorithms. Such algorithms are often developed with the assumption that job durations are known [30,10]. Even if this is not practical, in some cases it may be possible to rely on user-provided estimates of job duration [30,7]. 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 can be considered a crude prediction of job duration.

[5] Altair PBS: `https://www.pbsworks.com/`

[6] Slurm Workload Manager: `https://slurm.schedmd.com/`

In general user estimations are not reliable [30], 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. We propose 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 [16] 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 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. 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 have also used machine learning to predict job duration. However, results were not satisfactory, 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.

We remark a difficulty to carry out dispatching research, and specifically to test this job duration prediction method, because we need to access to the actual workload data — which is possible in a real environment, without any preprocessing that could remove important information such as the name of jobs, to make better predictions. We predict the job duration over all the workload from the Eurora system [7]. The exact under and overestimation rates are 3.6% and 96.3% for the user defined walltime, and 25.8% and 53.7% for our job duration prediction, respectively. Our job duration prediction introduces a considerable rate of underestimation, which is common in job duration predictions, despite this, we improve the accuracy in the expected job duration, 50% of the jobs are around ±25% of the actual duration, instead the user walltime is less than 10%. In [16], we tested the integration of our prediction with five state-of-the-art dispatching algorithms, and concluding that using prediction is advantageous and the main beneficiaries are the short jobs. Given the prominent presence of short jobs in typical HPC system [26] workloads, this benefit should apply to large-scale computational infrastructures in general.

4 HPC simulator

One of the challenges of job dispatching research is the intensive experimentation necessary for evaluating and comparing various dispatchers in a controlled envi-

[7] https://www.cineca.it/en/content/eurora

ronment. The experiments differ under a range of conditions with respect to the workload, the number and the heterogeneity of resources, and the dispatching algorithms. However, using a real HPC system for experiments is not realistic. Therefore, simulating an HPC system is essential for conducting controlled dispatching experiments. To do so, we developed AccaSim [17], a library for simulating the Workload Management System in an HPC system, which offers to the researchers an accessible tool to facilitate their job dispatching research. The library is open-source, implemented in Python, which is freely available for any major operating system, and works with dependencies reachable in any distribution. It is executable on a wide range of computers thanks to its lightweight installation and light memory footprint. AccaSim is scalable to large workload datasets and provides support for easy customization, allowing to carry out experiments across different workload sources, resource types, and dispatching algorithms. Moreover, AccaSim enables users to develop novel advanced dispatchers by exploiting information regarding the current system status, which can be extended for including custom behaviors such as energy and power consumption and failures of the resources. Last but not least, AccaSim aids users in their experiments via automated tools to generate synthetic workload datasets, to run the simulation experiments and to produce plots to evaluate dispatchers. Researchers can thus use AccaSim to mimic a wide range of real systems, including those possessing heterogeneous resources, develop advanced dispatchers using for instance power and energy-aware, fault-resilient algorithms, and test and evaluate them in a convenient way over a wide range of workload sources.

We compared AccaSim with other similar simulators [17]. From the results, we obtained AccaSim uses up much less memory than the other simulators. Indeed, 19 MB of memory in average was used by Accasim, where other simulators, Batsim and Alea, average 6,421 MB when simulating the Metacentrum 2 workload dataset [8]. In addition, AccaSim can process big workload datasets without degrading its performance, whereas other simulators showed issues with them. From the experimental study, the time to process the biggest dataset by AccaSim took 383s, where Batsim and Alea average 1,158s, all of then using a rejection policy. In general, we can say Accasim is scalable to large workload datasets, and overall it performs much better than other similar simulators.

5 Constraint Programming-based Job Dispatchers

The on-line job dispatching problem in HPC systems 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

[8] The MetaCentrum 2 log, which includes 5,731,100 jobs: `https://www.cse.huji.ac.il/labs/parallel/workload/l_metacentrum2/index.html`

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. In our experimental study, t belongs to certain events, specifically, when a job is queued or is completed.

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. 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.

In [4,7], the first CP-based dispatchers for HPC systems are developed and tested on a workload trace collected from the Eurora system [9]. In the first dispatcher [4], the entire dispatching problem is modelled and solved using a CP solver. The second dispatcher [7] 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. Both methods consider the objective function which minimizes the sum of the waiting times of the jobs. `PCP` uses a weighted sum so as to give priority to the jobs that stay in the queue longer than their Estimated Waiting Time (EWT). The EWT value is the average waiting time of the queue where a job is submitted, and is obtained by analyzing the workload data. Instead in `HCP`, the weights are slightly different, giving priority to the jobs of the queues with lower expected waiting times.

Our current contribution is redesign the main components of `PCP` and `HCP`. First, we revisit their model and search control mechanism so as to make them resilient to heavy workloads and applicable to on-line dispatching. To do so, we consider only jobs which can be dispatched at the current time, with a maximum number of jobs, remaining jobs are postponed. In addition, we add the solver state to the search control. Consequently, if the solver proved unsatisfiability, the dispatcher will avoid further restart until a new time point. `PCP` and `HCP` restarts until a maximum time is reached. Second, we study the use of job duration prediction, instead of the user expected duration, when generating dispatching decisions. We incorporate the job duration prediction in a more active way in the dispatcher by replacing the objective function, which uses a EWT based metric, with the minimization of the job slowdowns. The EWT based metrics are 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. In addition, we involve the job

Dispatcher	Total sim. time [s]	Avg. waiting time [s]	Avg. slowdown
PCP	-	-	-
PCP' + h. prediction	262,764	462	95
PCP' + real duration	261,985	275	39
HCP	374,788	2,449	987
HCP' + h. prediction	215,814	508	111
HCP' + real duration	201,223	301	26

Table 1: Dispatcher results

duration prediction in the search of the scheduling and allocation, via the use of job slowdown as job priority.

Table 1 demonstrate that with our redesign (PCP' and HCP') we significantly reduce the time required to generate dispatching decisions; and the dispatchers can benefit from good job duration predictions and considerably decrease the waiting times and the slowdown of the jobs.We also used the real duration of jobs as predictor, so as the accuracy increase the dispatching decision are better with `PCP` and `HCP`. In general to benefit from this potential, job durations should rely on predictions with acceptable levels of accuracy, going beyond the standard user defined walltime. While the heuristic prediction considered is not the best, we have shown that it is a valid alternative to the walltime, despite its simplicity.

6 Conclusions

So far, we have dealt with different aspects of dispatching research. First, we developed an HPC simulator capable to mimic any real system, to develop, test and evaluate advanced dispatchers. Second, we developed a data-driven approach to predict job duration prediction showing a more effective predictions than the user estimates. Results showed that short jobs are the main beneficiaries, and given the features of the majority of workloads, this benefit should apply to large-scale computational infrastructures in general. Last, we introduced new CP-based dispatchers built on top of [4,7] 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. Results show that the new dispatchers compared to the original ones reduced the time spent in generating decisions on a heavy workload. Moreover, the new dispatchers can benefit from job duration predictions and generate decisions of higher QoS levels on workloads dominated by short jobs. The new dispatchers are thus more suitable for HPC systems running modern applications that employ short jobs.

We will finalize our work by including 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.

References

1. 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)
2. 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)
3. 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)
4. 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)
5. 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)
6. 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)
7. 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)
8. 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)
9. 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)
10. Chandio, A.A., Xu, C., Tziritas, N., Bilal, K., Khan, S.U.: A comparative study of job scheduling strategies in large-scale parallel computational systems. In: TrustCom/ISPA/IUCC. pp. 949–957. IEEE Computer Society (2013)
11. 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)
12. Dutot, P., Mercier, M., Poquet, M., Richard, O.: Batsim: A realistic language-independent resources and jobs management systems simulator. In: JSSPP'16. LNCS, vol. 10353, pp. 178–197. Springer (2016)
13. 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)
14. 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)

15. 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)
16. 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)
17. Galleguillos, C., Kiziltan, Z., Netti, A., Soto, R.: AccaSim: a customizable workload management simulator for job dispatching research in HPC systems. Cluster Computing pp. 1–16 (2018)
18. 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)
19. Klusácek, D., Rudová, H.: Alea 2: job scheduling simulator. In: SimuTools. p. 61. ICST/ACM (2010)
20. 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)
21. 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)
22. 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)
23. 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)
24. 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)
25. Singh, D., Reddy, C.K.: A survey on platforms for big data analytics. Journal of big data **2**(1), 8 (2015)
26. Sîrbu, A., Babaoglu, Ö.: A holistic approach to log data analysis in high-performance computing systems: The case of IBM blue gene/q. In: Euro-Par Workshops. Lecture Notes in Computer Science, vol. 9523, pp. 631–643. Springer (2015)
27. 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)
28. 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)
29. 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)
30. Weil, A.M., Feitelson, D.G.: Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. IEEE Trans. Parallel Distrib. Syst. **12**(6), 529–543 (2001)