On the Quality of Wall Time Estimates for Resource Allocation Prediction

Mehmet Soysal
mehmet.soysal@kit.edu
Steinbuch Centre for Computing,
Karlsruhe Institute of Technology
Eggenstein-Leopoldshafen, Germany

Marco Berghoff
marco.berghoff@kit.edu
Steinbuch Centre for Computing,
Karlsruhe Institute of Technology
Eggenstein-Leopoldshafen, Germany

Dalibor Klusáček
klusacek@cesnet.cz
CESNET a.l.e.
Prague, Czech Republic

Achim Streit
achim.streit@kit.edu
Steinbuch Centre for Computing,
Karlsruhe Institute of Technology
Eggenstein-Leopoldshafen, Germany

ABSTRACT

Today's HPC systems experience steadily increasing problems with the storage I/O bottleneck. At the same time, new storage technologies are emerging in the compute nodes of HPC systems. There are many ideas and approaches how compute-node local storage can be made usable for HPC systems. One consideration is to copy job data to the compute-node local disks in advance. To accomplish this, the allocated nodes must be known in advance. In this paper, we look at the node allocation behavior of a HPC batch scheduling system. Our goal is to determine whether it is possible to stage data in advance, based on scheduler predictions. We show that wall time estimates must be excellent to reliably predict node allocations. In reality, the required accuracy enabling advance data staging is hard to achieve. Therefore, the behavior of (standard) batch scheduler have to be modified in order to enable efficient advance data staging.

KEYWORDS

wall time prediction, scheduling, batch system, node allocation

ACM Reference Format:
Mehmet Soysal, Marco Berghoff, Dalibor Klusáček, and Achim Streit. 2019. On the Quality of Wall Time Estimates for Resource Allocation Prediction. In *48th International Conference on Parallel Processing: Workshops (ICPP 2019), August 5–8, 2019, Kyoto, Japan.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3339186.3339204

1 INTRODUCTION AND BACKGROUND

High-performance computing (HPC) systems show an increasing gap between computing power and the performance of storage systems. While the computing power has increased tremendously during the last years the performance of storage systems did not keep pace. For example, from Sequoia (Top 1 HPC System in 2012) to Summit (Top 1 - 2018) the computing power increased by a factor of ten, from ~20 to ~200 PFLOPS, whereas the storage throughput increased only by a factor of 2.5 from 1 to 2.5 TB/s.

Recently, many developments and innovations have improved the I/O throughput and metadata performance for global parallel file systems (FS). Hardware based innovation like NAND-based disks, e.g., SSDs [1], are replacing spinning disks on global parallel FS. Other approaches are using compute-node local storage as burst buffers [2] or as hardware write accelerator caches within IBM's Spectrum Scale [3]. These approaches try to transparently forward the non-contiguous and random I/O of the application as sequential I/O to the parallel FS. Other hardware-based approaches such as DDN's Infinite Memory Engine [4, 5] act as a transparent in-bound cache.

New storage technologies such as NVRAM [6] and NVMe [7] have emerged onto the market and are more frequently used in compute nodes of large HPC systems. Using these technologies as fast local storage in each node is an option, but it would be beneficial to use them for compute jobs by coupling their capacity and performance. The ADA-FS [8] project follows this approach: making compute-node local storages available to the users' node-parallel applications. The idea is to create a cross-node, ad-hoc and independent parallel file system (FS) using the node-local storages. After creating a temporary FS with the node-local storage, the required data for a job is staged to this temporary FS. This data staging is either done with in-bound methods such as Moab's staging method [9] or by reserving the allocated nodes for data staging.

Both methods cause inefficient utilization of nodes: the nodes either idle for a while or are only used to stage-in data. One way to reduce these wasted CPU cycles is to stage-in the data on the nodes in advance before the nodes are freed up (cmp. Fig. 1). To make this possible, the predicted job allocations of the scheduler must be reliable. Operational experience from various HPC sites proves that these job allocations are very inaccurate on real machines [10]. In this paper, we evaluate how good job wall time estimates have to be to allow the scheduler to make reliable node predictions.

The remainder of this paper is structured as follows: In Chapter 2, we give a brief overview of the related work. Our simulation methodology that is used to evaluate the efficiency of node allocation predictions is described in Chapter 3. Chapter 4 shows the

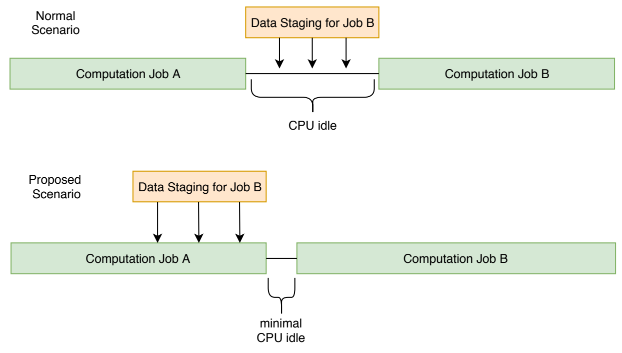


Figure 1: Normal Scenario for data staging is wasting CPU cycles during data staging (top). Proposed solution does advance data staging onto compute node(s) before job start, reducing idle CPU time to minimal (bottom).

results of our experimental evaluation and discusses the observed outcomes. Finally, we conclude our work in Chapter 5 and give an outlook to future work.

2 RELATED WORK

Batch system schedulers are responsible for resource planning and allocating the node(s) for a job [11]. Typically, resource utilization is improved by filling existing gaps in job schedule with sufficiently small jobs (so called back-filling) [12]. Similarly, job schedules can be used to either predict the start time or the node(s) for a given job. In both cases, the scheduler uses wall time estimates given by users to calculate job schedules. It is a well known problem that the user-provided estimates are far from being accurate [13–15].

There are several approaches to improve wall time estimates, and we can only give a brief overview in this section. Gibbons [16, 17], and Downey [18] used historical workloads to predict the wall times of parallel applications. They predict wall times based on templates. These templates are created by analyzing previously collected metadata and grouped accordingly. Smith et al. [19, 20] apply greedy and genetic search techniques to identify similar jobs. In the recent years, machine learning algorithms are used to predict resource consumption in several studies [21–25]. However, all the studies mentioned above do not evaluate the accuracy of node allocation predictions. Instead, most of the publications focus on observing the impact of improved estimates on, e.g., the system utilization or the reliability of calculated job start times.

In this paper, we focus on the node allocation prediction and how good wall time estimates have to be to reasonably predict which physical node(s) will be used for a job. This directly affects, whether a cross-node, ad-hoc, independent parallel FS can be deployed to stage-in data, or not.

3 SIMULATION METHODOLOGY

This section describes the simulation methodology to evaluate node allocation predictions. First, we define the metric used to evaluate the quality of predictions. Next, we present how the simulations were performed. Especially, we describe how the impact of imprecise wall time estimates has been studied. Finally, we briefly present the workloads used in this analysis and describe the job scheduling simulator used for our evaluation.

3.1 Applied Metric

The main goal of this paper is to study how node allocation predictions are reliable subject to variously accurate job run time estimates. Therefore, we define T_{NAP} as the *Node Allocation Prediction Time*. This metric measures the time from which the allocation did not change till the actual job start. In other words, T_{NAP} is the difference between job start time and the time when the last accurate prediction has been made. This time was taken as a metric to evaluate the results. In this context, T_{NAP} is a *per job* metric and, e.g., job parallelism does not influence the way it is computed. For example, if $T_{\text{NAP}} = 1$ hour then the actually used physical node has been correctly predicted one hour before job started its execution.

We categorized jobs into different groups depending on the T_{NAP} value. T_{NAP} below one second, T_{NAP} over one second and bellow ten minutes, and T_{NAP} over ten minutes (long-term predictions). A T_{NAP} below one second, indicates e.g.,

- that job started immediately after job submission due to free resources (we classify these jobs as “instant”),
- that job was picked up by a back-filling algorithm, or
- that job was later rescheduled on free resources (previously unavailable).

The reason why we choose ten minutes is based on our previous work. We have shown that parallel file systems can be created in a very short time. BeeGFS within few minutes on 256 nodes [26] and GekkoFS in less than 30 seconds on 512 nodes [27]. In the remaining time, the required data could be copied to the nodes accordingly. Whereby the 10 minute limit should be interpreted as an arbitrary guess. A meaningful value can only be determined by considering more factors. One is the maximum throughput of the global file system and the performance of the node-local storage. Especially the volume of copy-able data is limited by the sum of all allocated node-local storages. Of course, this time depends on how much data has to be copied, but 10 minutes seem to be a reasonable threshold based on real-world experience in a large HPC computing center.

3.2 The Design of the Experiment

We have modeled the system, scheduler and workload in a simulator and conducted several experiments where job run time estimates were subsequently improved. Starting with very imprecise wall time estimates as provided by real users of the system and continuing to fully accurate job run time estimates. For this purpose, we introduce $\tilde{T}_{\text{Req}}$, the “refined” requested wall time,

$$\tilde{T}_{\text{Req}} = T_{\text{Run}} + \lambda(T_{\text{Req}} - T_{\text{Run}}) \quad \text{with} \quad \lambda \in [0, 1], \tag{1}$$

where T_{Req} is the user-provided wall time and T_{Run} is the job run time. Each job in the workload is then adjusted by the same λ, effectively simulating different precision of provided wall time estimates. For example, if a user requests 100 hours but only needs 10, then for $\lambda = 1.0$ ($\lambda_{1.0}$) would be $\tilde{T}_{\text{Req}} = 100$ hours and for $\lambda = 0.2$ ($\lambda_{0.2}$) the refined wall time $\tilde{T}_{\text{Req}} = 10 + 0.2 \cdot (100 - 10) = 28$ hours. Clearly, for $\lambda = 0$ the $\tilde{T}_{\text{Req}}$ becomes a perfect estimate, i.e., $\tilde{T}_{\text{Req}} = T_{\text{Run}}$.

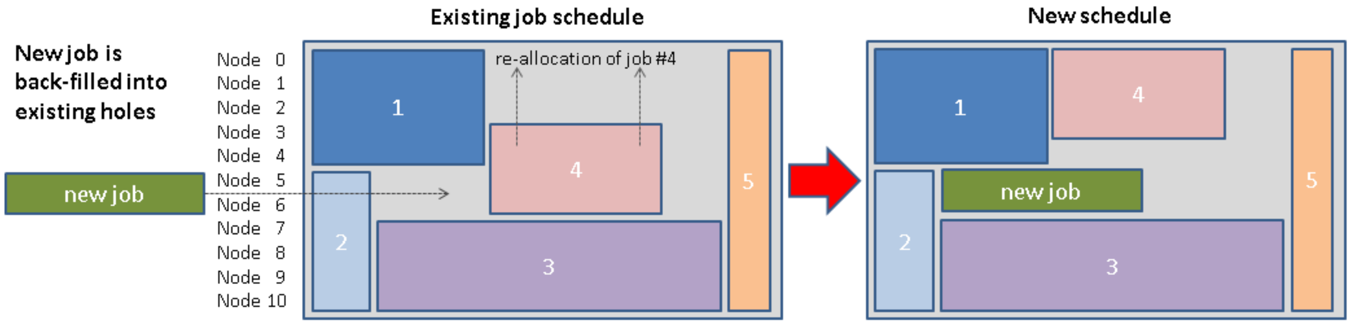


Figure 2: An example of back-filling distorting previous node allocations.

Of course, improving wall time estimates in this way is rather unrealistic as was shown repeatedly by Feitelson [28] and Tsafrir [29]. It is obvious, that users cannot improve their estimates in such a fine-grained fashion. But as mentioned before, we only want to evaluate the impact of accuracy on the node allocation prediction, thus such refinement of wall times is sufficient for our investigation.

Furthermore, beside these *synthetic refinements* we also used an existing *wall time predictor* (see Section 3.4.2) which uses historic job run time data to deliver improved wall time estimates. It provides a valuable comparison with aforementioned synthetic refinements and outlines possible outcomes of practical application of node predictions in everyday use.

These workloads with varying degree of wall time accuracy were then used as inputs for the simulator. The simulator uses the T_{NAP} metric to measure prediction accuracy of each such experiment. All experiments have been conducted using the Alea job scheduling simulator [30]. To emulate the system-level job batch scheduler we used two widely adopted scheduling algorithms to create job schedules — a simple First-Come-First-Serve (FCFS) algorithm and the Conservative back-filling algorithm [12].

3.3 Workload Traces

We used workload files from the Parallel Workload Archive [31] for evaluation. ForHLR II at KIT [32][1], SDSC[33], KTH[34] and CTC [35].

The workload from KIT has been modified to fit into the node-level simulation of ALEA:
The nodes have multi-core CPUs but due to the usage model of the system, the nodes are scheduled and allocated as whole nodes. Therefore, we changed the nodes from 20 cores per node to one, and divided also the requested number of cores accordingly. With this modification, a whole node scheduling is simulated. We also removed all jobs belonging to the partition of fat nodes, leaving approximately 112k jobs for ForHLR II. The workload covers a time span of 1.5 years (06/2016 – 01/2018).

3.4 Alea Simulator and Scheduling Algorithms

The Alea simulator used in this study has been further developed to enable detailed simulations of advanced node allocations. This covers several areas, which we describe here.

[1]The workload is not published yet. We intend to publish the workload with the publication of this paper.

First, Alea allows to built job schedules (a plan of future job execution) based on the provided estimated run time (either T_{Req} or refined $\tilde{T}_{\text{Req}}$) or using a built-in run-time predictor, which uses previous known run times of already completed jobs to establish a run-time prediction [36]. Either the Conservative back-filling or simple FCFS-based technique is used to construct such job schedules.

Second, Alea now records the time when a node allocation was created by the scheduling algorithm and measures the T_{NAP} metric. This extended version of the simulator that has been used in this study can be found at GitHub [37].

In the following paragraphs, we describe several interesting features of the used scheduling algorithms that influence the prediction accuracy of constructed plans. Next, we also briefly describe the built-in job run-time predictor.

3.4.1 Scheduling Algorithms. While the (in)accuracy of run-time estimates plays a major role in the reliability of constructed schedules, it is most important to focus also on the behavior of the used scheduling algorithm. The first used algorithm based on First-Come-First-Serve (FCFS) is very straightforward. It simply adds new jobs at the earliest free time slot at the end of the existing schedule. On the other hand, Conservative Back-filling tries to use existing "holes" in the schedule for the newly arriving jobs. The goal is to increase system utilization and to reduce average response time. However, this "hole filling" approach may have negative impact on prediction accuracy. Figure 2 illustrates a scenario, where a newly arriving job is placed into existing holes, but during this process it reshuffles node allocations of previously planned job. While this reshuffling has no negative impact on job start times (those are guaranteed to remain the same during back-filling), it hampers all efforts to efficiently stage job-related data in advance of actual computation. From this point of view a simple FCFS is a safer choice.

Similarly, when a job completes earlier than expected, a FCFS-built plan is updated simply by adjusting job start times with respect to the new situation, i.e., jobs are moved to earlier time slots that could have appeared as a result of the earlier job completion. Relative job ordering (defined by planned start times) is kept intact and jobs are thus less likely to end up on different nodes. On the other hand, Conservative Back-filling uses a more aggressive strategy called *schedule compression.* During compression, all jobs are removed from the schedule and then re-inserted one-by-one into the newly built schedule. It is quite common that after the compression the schedule looks rather different than the previous one [12]. As a result, although FCFS is less efficient by means of resource utilization, it is more likely to deliver more accurate predictions, thus providing better (higher) values of the T_{NAP} metric.

3.4.2 Run-time Prediction Technique. Alea implements a rather simple run-time prediction technique. It uses previous known run times of already completed jobs to establish a run-time estimate for a newly arriving job. It is working on a per-user basis, i.e., a new run-time estimate for a given job of a user is computed using information about previous jobs of that user[2].

The predictor works as follows. Each time a given user's job completes its execution, the predictor computes the ratio of the used wall time σ, the fraction of job's actual run time and user's estimate

$$\sigma(i) := \frac{T_{\text{Run}}(i)}{T_{\text{Req}}(i)}, \tag{2}$$

where i is the i-th job by the specific user, i.e., σ measures to what extent the estimated wall time was actually used. Since the user's estimate is the upper bound of job run time[3], σ falls between 0.0 and 1.0 representing the relative usage of requested wall time. In other words, the technique measures by how much a user overestimates job's run time. It is fair to mention, that a similar approach has been used in [38]. Once σ is computed, it is used to generate the final prediction. The predictor only uses the five last (most recently completed) jobs when computing the prediction in order to use "fresh" information. First, it computes the σ of those five recently completed jobs and then chooses their maximum. Next, it multiplies the job's wall-time estimate $T_{\text{Req}}(i)$ by this maximum. We define

$$T_{\text{Pred}}(i) := T_{\text{Req}}(i) \max_{i-5 \le k \le i-1} \{\sigma(k)\}, \tag{3}$$

the predicted run time.

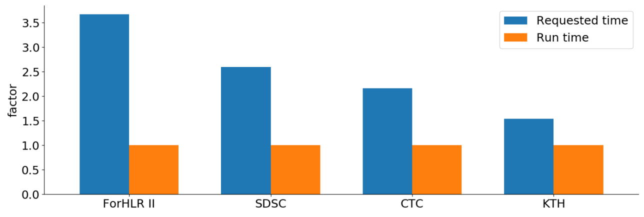
It represents a conservative strategy, where the prediction is calculated using the known relative accuracy of user's recent estimates. By choosing the maximum σ (i.e., choosing a job where the difference between actual and estimated run time was minimal), this technique aims to minimize the number of cases where the prediction will be underestimated. At the same time, by ignoring older jobs it reflects aging and orients itself more on the recent user's workload characteristics.

Of course, $T_{\text{Pred}}(i)$ may turn out to be overly optimistic, i.e., shorter than actual job run time. If this case is detected during simulation, $T_{\text{Pred}}(i)$ is gradually increased during the simulation (multiplied by a factor of 2), until it is either sufficient or it reaches the original user's estimated value $T_{\text{Req}}(i)$ [39].

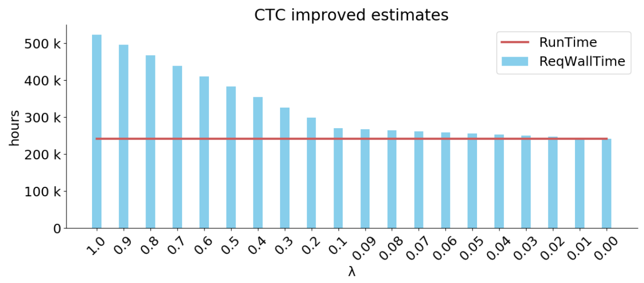
4 RESULTS

First, we take a look at the characteristics of the used workload traces. Figure 3a show the ratio between the requested and used wall time of the evaluated workloads in this paper.

Users of the ForHLR II request approximately ~3.8 more time than they use. The differences are smaller for the other workloads,



(a) Comparison of used workloads. The ratio of time requested to time used.



(b) CTC Workload with improved estimates. hours. Y-Axis: Sum of requested and used hours of all jobs

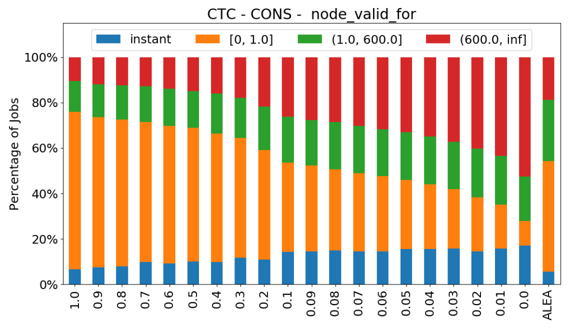
Figure 3: Comparison of requested wall time vs. used run time.

the smallest difference is for the KTH workload with a factor of ~1.5, CTC with ~2.1 and SDSC with a factor of ~2.6. The effect of the artificial improved wall times for CTC workload and different values of λ is shown in Figure 3b. The blue bars represent the sum of all requested job wall times for several λ. For comparison, the sum of the job run times is represented by the red line.

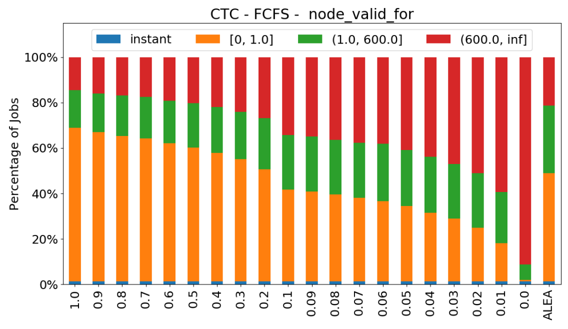
The simulation results are shown in Figure 4 and show the jobs categorized within four T_{NAP} classes (see Section 3.1). Each bar represents a simulation with a different λ value. The last bar, labeled "Alea", is the simulation using Alea's built-in wall time predictor. Each row shows the simulation of a specific workload with two different scheduling algorithms. The conservative back-filling algorithm (left column) and the simple FCFS algorithm (right column). The bars are categorized into four groups based on the T_{NAP}. The blue part represent the jobs that are started immediately after the job is submitted (instant). The orange part represent queued jobs with a T_{NAP} between 0 and 1 seconds. Jobs with a T_{NAP} from one second up to 10 minutes are represented by the green part. Red indicates a T_{NAP} value of more than 10 minutes (long term predictions) which is in our focus. This long-term predictions increases significantly only at very small $\lambda \le 0.1$ which already proves that very good run time estimates are needed.

Using back-filling, the improvement is not so steady, e.g., the long term predictions for the CTC workload increases from ~10% to ~26% of all jobs, for $\lambda_{1.0}$ to $\lambda_{0.1}$. From $\lambda_{0.1}$ to $\lambda_{0.01}$ this class improves from ~26% to ~52%. Even with perfect run time predictions, long-term predictions are not possible for the majority of jobs. With FCFS algorithm a more continuous prediction improvement is seen. When using FCFS with perfect run times estimates, (almost) only instant starting jobs (blue) or with long term predictions (red)

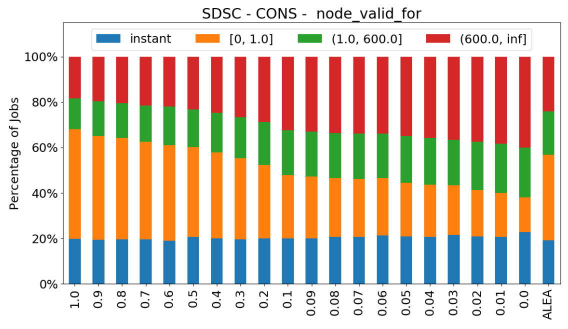
[2]In case that a given user has no completed jobs so far then such historic information is obviously missing, thus we use the user-provided estimate T_{Req} instead.
[3]The system is configured to kill a job if it exceeds user's wall-time estimate.



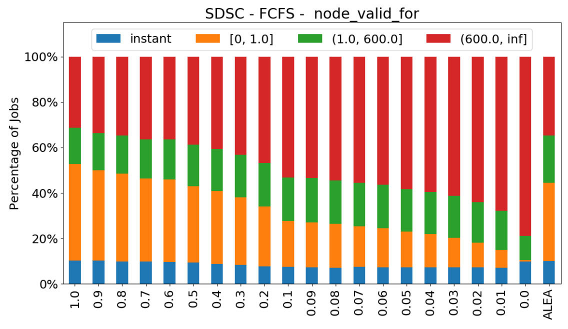
(a)



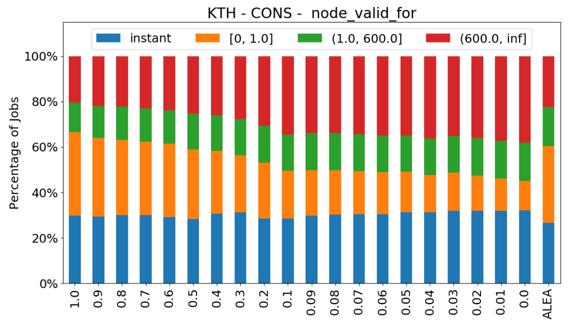
(b)



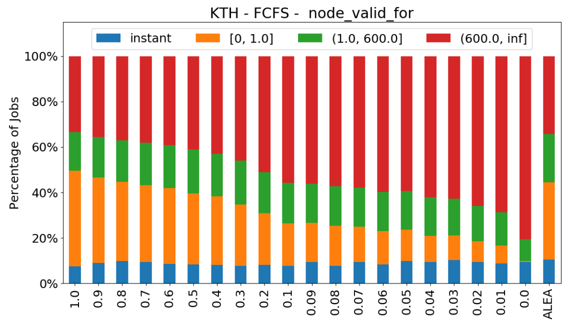
(c)



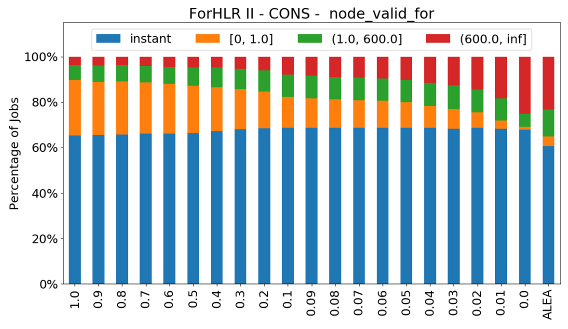
(d)



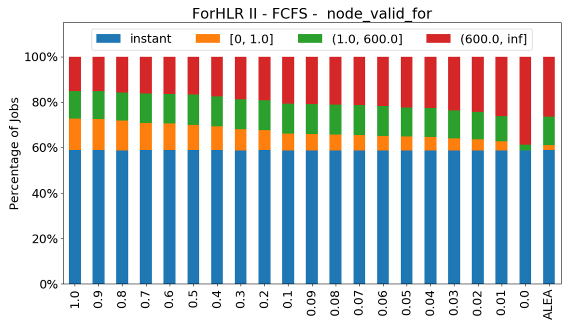
(e)



(f)

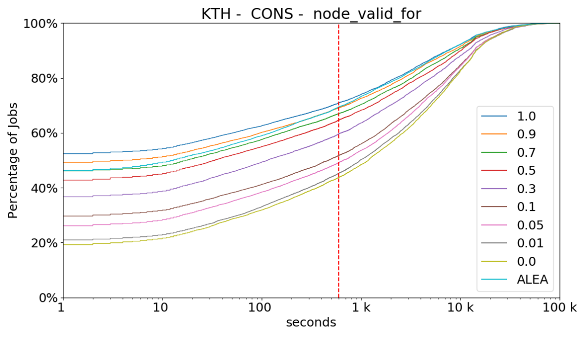


(g)

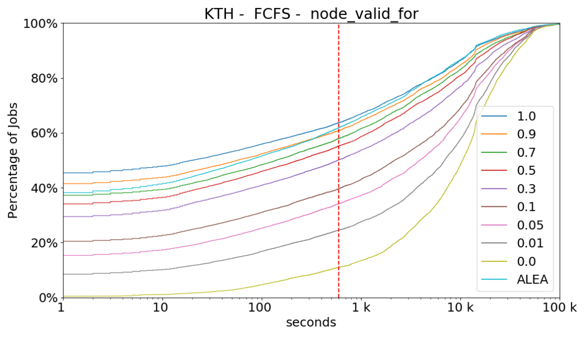


(h)

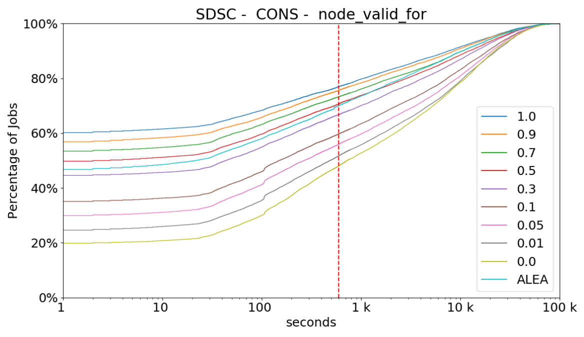
Figure 4: Job distributions according to the T_{NAP} class for all evaluated workloads with conservative back-filling (CONS) on the left and FCFS on the right. Blue color denotes instant jobs, orange color means job having $T_{\text{NAP}} \leq 1\,s$, green color denotes jobs with $1\,s < T_{\text{NAP}} \leq 600\,s$ and red color denotes long-term predictions ($600\,s < T_{\text{NAP}} \leq \infty$).



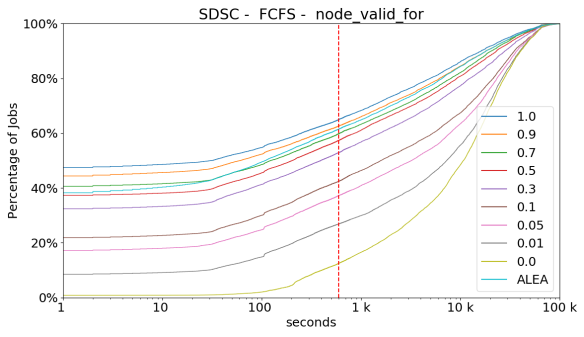
(a)



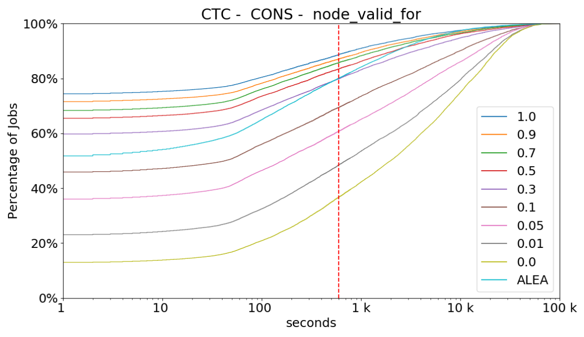
(b)



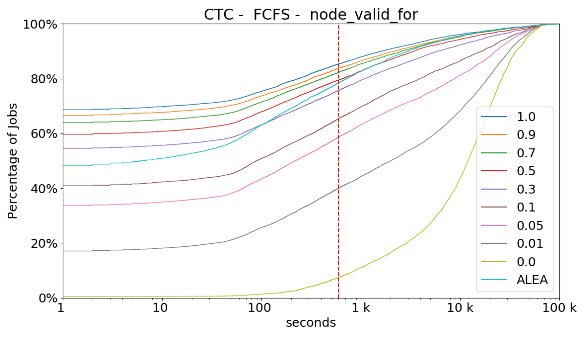
(c)



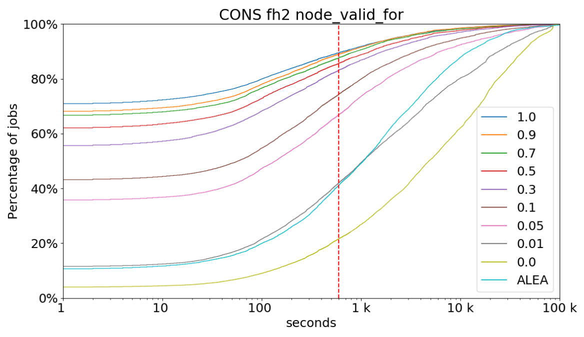
(d)



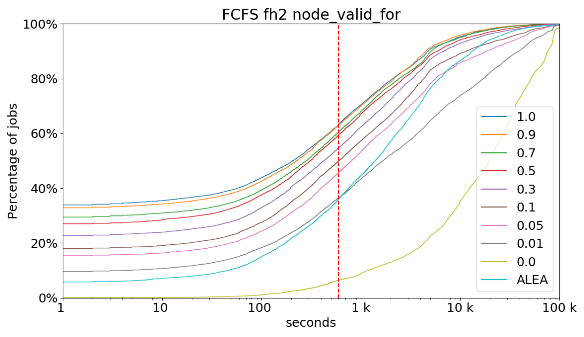
(e)



(f)



(g)



(h)

Figure 5: Cumulative distributions of T_{NAP} values for all considered workloads, with conservative back-filling (CONS) on the left and FCFS on the right. Fine dashed vertical line denotes the 10 minutes (600 s) T_{NAP} threshold, depicting the fraction of jobs that fail to obtain reasonably long-term predictions.

remain. Since operators like to keep the utilization of their system high, back-filling is often used which makes our approach more difficult. The culprit here is *the way back-filling uses existing holes in the schedule.* In back-filling, a newly arriving job is often placed into several existing holes. It improves utilization, but during this process it reshuffles node allocations of previously planned job, as was shown in Figure 2. This reshuffling then hampers all efforts to efficiently stage job-related data in advance of actual computation. From this point of view, simple FCFS is a safer choice.

The cumulative distributions of T_{NAP} for all considered workloads are shown in Figure 5. In these figures we removed the jobs which are started instantly after job submission (blue bars from Figure 4). These instant started jobs are not of our interest and we can focus on jobs which have to wait until their requests can be satisfied. X-Axis shows the T_{NAP} values while Y-Axis denotes the fraction of jobs having $T_{\text{NAP}} \leq x$. The dashed vertical line represents a valid prediction of 600 seconds for the node allocation. Everything to the right from this vertical line, represents jobs with a valid long term prediction. For all workloads it can be clearly seen that the FCFS algorithm performs better. Here the lines rise later, which means longer valid predictions. However, there is still a gap between $\lambda_{0.01}$ and the λ_0, which gives an indication that even a small deviation of the wall time estimate will reduce the accuracy accordingly.

The result when using Alea's run time estimate predictor are miscellaneous. With the ForHLR II workload the built-in predictor shows pretty impressive results, if the instant started jobs are ignored (cmp. Figs. 4g and 5g). The prediction accuracy is close to $\lambda_{0.01}$. This is caused by the gap between the requested and used wall time (see Figure 3a). With such a high difference, the effect of predictions on scheduling and thus on the node prediction is significant. For the remaining workloads with better user-provided estimates the line related to Alea's predictor is located, at least, behind the $\lambda_{0.3}$ line. Clearly, the simple Alea's predictor does not improve the accuracy very much in such cases. When comparing the cumulative distributions in Figure 5, it can be seen that only with very good estimates ($\lambda \leq 0.1$), the accuracy of predictions improves. Generally, the accuracy is better with the FCFS algorithm. This is because all jobs wait until their turn and cannot be brought forward in the schedule, causing disruptions for previously planned jobs (see Figure 2 for an example of such a side-effect).

4.1 Remarks

During the simulations we noticed small details, which should be mentioned.

In Figure 4 we can observe that when using the back-filling algorithm with improved λ values, the number of jobs which start immediately (blue) slightly increases. This has two reasons. First, the throughput improves with more accurate run times as more waiting jobs are started by back-filling. Second, newly submitted jobs may fit better into available free slots. On the other hand, these values hardly change for FCFS algorithm, since it does not use such back-filling optimization. Clearly, jobs that are started instantly cannot profit from any advance data stage-in method.

The results in Figure 4g-h show a high rate of jobs that are started immediately after the job submission. There are several reasons for this. First, the workload's time-frame is approximately 1.5 years and various maintenance slots have not been simulated, i.e., the queue is processed normally during maintenance. Second, a part of the nodes has been separated for a limited time for various campaigns whose consumption is not included in the workload.

5 CONCLUSION AND FUTURE WORK

In this work, we have analyzed whether advance data staging to computing nodes is achievable in nowadays systems. We have used detailed simulations using existing HPC workloads and measured the influence of (in)accurate wall time estimates on the prediction's reliability. Either artificially improved wall times or predictor-based estimates have been used to identify the threshold where the predictions become practically useful. Our result shows that very accurate wall time estimates are needed to deliver precise node allocation predictions. We have also shown, that the widely used back-filling algorithm may degrade the reliability of predictions as a side-effect of its mechanism to increase resource utilization. From this point of view, the vanilla FCFS algorithm produces more reliable results.

Still, our simulations demonstrate that without very accurate wall time estimates, current systems will not be able to guarantee good predictions needed for advance data staging. Also, on real production systems, there are of course, more parameters which could have an impact on the wall time of jobs. Parameters like processes-binding, NUMA locality, topology awareness, network congestion, or even power capping. The factors make the wall times more variant and thus increase the uncertainty.

Therefore, the goal to find a general approach for efficient advance data staging on the nodes remains open. We will need to investigate more aggressive strategies to fulfill this goal, e.g., consider modifications of the scheduling behavior or use reservations of nodes for data staging. A minimally invasive solution could involve introducing a special flag to the jobs. This flag could signal a request for a temporary on-demand FS and indicate which data should be staged. The scheduler could then reserve or pin down the required nodes for this job. This approach might reduce the cluster utilization, but also benefit from the reduced load on the global FS. This solution is going to be evaluated and the advantages and disadvantages weighed against each other in our future work.

What we did not consider in this paper is the fundamental question of whether an on-demand FS on the compute node makes sense. The resources of compute nodes are needed for the application, and these would have to be shared with an on-demand FS. Also the concurrent data-staging may interfere with the running application. These questions have already been evaluated, and we have shown that using an on-demand file system does not mean automatically, that an application is slowed down. It depends highly on the specific use-case and the I/O behavior of the use case, but the advantages of an on-demand file system outweigh the disadvantages in the evaluated use cases [40].

6 ACKNOWLEDGEMENT

We acknowledge the support of the National Grid Infrastructure MetaCentrum provided under the programme "Projects of Large Research, Development, and Innovations Infrastructures" (CESNET LM2015042) and the support provided by the project Reg. No.

CZ.02.1.01/0.0/0.0/16_013/0001797 co-funded by the Ministry of Education, Youth and Sports of the Czech Republic. We also appreciate the funding provided by the German Research Society (Deutsche Forschungsgemeinschaft – DFG) Priority Program "Software for Exascale Computing" (SPPEXA, SPP 1648). This work was performed on the computational resource ForHLR II funded by the Ministry of Science, Research and the Arts Baden-Württemberg and DFG.

REFERENCES

[1] E Kim. SSD Performance-A Primer: An Introduction to Solid State Drive Perfromance, Evaluation and Test. Technical report, Tech. rep., Storage Networking Industry Association, 2013.

[2] N. Liu, J. Cope, P. Carns, C. Carothers, R. Ross, G. Grider, A. Crume, and C. Maltzahn. On the role of burst buffers in leadership-class storage systems. In *012 IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–11, April 2012.

[3] IBM. GPFS - highly available write cache (hawc), 2018.

[4] DataDirect Networks. IME - Flash native cache, 2018.

[5] Eugen Betke and Julian Kunkel. *Benefit of DDN's IME-FUSE for I/O Intensive HPC Applications: ISC High Performance 2018 International Workshops, Frankfurt/Main, Germany, June 28, 2018, Revised Selected Papers*, pages 131–144. Springer, 06 2018.

[6] Sooyong Kang, Sungmin Park, Hoyoung Jung, Hyoki Shim, and Jaehyuk Cha. Performance trade-offs in using NVRAM write buffer for flash memory-based storage devices. *IEEE Transactions on Computers*, 58(6):744–758, 2009.

[7] Ji Jun Hung, Kai Bu, Zhao Lin Sun, Jie Tao Diao, and Jian Bin Liu. Pci express-based nvme solid state disk. In *Applied Mechanics and Materials*, volume 464, pages 365–368. Trans Tech Publ, 2014.

[8] ADA-FS - Advanced Data Placement via Ad-hoc File Systems at Extreme Scale, February 2019. http://ada-fs.github.io.

[9] Moab Workload Manager 9.1.0 Administrator Guide: Data Staging, February 2019. http://docs.adaptivecomputing.com/9-1-0/MWM/help.htm.

[10] Cynthia Bailey Lee, Yael Schwartzman, Jennifer Hardy, and Allan Snavely. Are user runtime estimates inherently inaccurate? In *Job Scheduling Strategies for Parallel Processing*, pages 253–263. Springer, 2004.

[11] Matthias Hovestadt, Odej Kao, Axel Keller, and Achim Streit. Scheduling in hpc resource management systems: Queuing vs. planning. In *Job Scheduling Strategies for Parallel Processing*, volume 2862, pages 1–20, 06 2003.

[12] Ahuva W. Mu'alem and Dror G. Feitelson. Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. *IEEE Transactions on Parallel and Distributed Systems*, 12(6):529–543, 2001.

[13] W. Cirne and F. Berman. A comprehensive model of the supercomputer workload. In *Fourth Annual IEEE International Workshop on Workload Characterization. WWC-4*, WWC '01, pages 140–148, Washington, DC, USA, 2001. IEEE Computer Society.

[14] Ahuva W. Mu'alem and Dror G. Feitelson. Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. *IEEE Trans. Parallel Distrib. Syst.*, 12(6):529–543, June 2001.

[15] Su-Hui Chiang, Andrea C. Arpaci-Dusseau, and Mary K. Vernon. The impact of more accurate requested runtimes on production job scheduling performance. In *Job Scheduling Strategies for Parallel Processing*, LNCS, pages 103–127, London, UK, UK, 2002. Springer-Verlag.

[16] Richard Gibbons. A historical profiler for use by parallel schedulers. *Master's thesis, University of Toronto*, 1997.

[17] Richard Gibbons. A historical application profiler for use by parallel schedulers. In *Job scheduling strategies for parallel processing*, pages 58–77. Springer, 1997.

[18] Allen B. Downey. Predicting queue times on space-sharing parallel computers. In *11th International Parallel Processing Symposium*, pages 209–218. IEEE, 1997.

[19] Warren Smith, Ian Foster, and Valerie Taylor. Predicting application run times using historical information. In *Job Scheduling Strategies for Parallel Processing*, pages 122–142. Springer, 1998.

[20] Warren Smith, Valerie Taylor, and Ian Foster. Using run-time predictions to estimate queue wait times and improve scheduler performance. In *Job Scheduling Strategies for Parallel Processing*, pages 202–219. Springer, 1999.

[21] Andréa Matsunaga and José AB Fortes. On the use of machine learning to predict the time and resources consumed by applications. In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, pages 495–504. IEEE Computer Society, 2010.

[22] Nirav H. Kapadia and José AB Fortes. On the design of a demand-based network-computing system: The purdue university network-computing hubs. In *The 7th International Symposium on High Performance Distributed Computing*, pages 71–80. IEEE, 1998.

[23] Farrukh Nadeem and Thomas Fahringer. Using templates to predict execution time of scientific workflow applications in the grid. In *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 316–323. IEEE Computer Society, 2009.

[24] Warren Smith. Prediction services for distributed computing. In *IEEE International Parallel and Distributed Processing Symposium*, pages 1–10. IEEE, 2007.

[25] Dan Tsafrir, Yoav Etsion, and Dror G Feitelson. Backfilling using system-generated predictions rather than user runtime estimates. *IEEE Transactions on Parallel and Distributed Systems*, 18(6), 2007.

[26] Sebastian Oeste, Michael Kluge, Mehmet Soysal, Achim Streit, Marc-André Vef, and André Brinkmann. Exploring opportunities for job-temporal file systems with ada-fs. *1st Joint International Workshop on Parallel Data Storage and Data Intensive Scalable Computing Systems*, 2016.

[27] M. Vef, N. Moti, T. Süß, T. Tocci, R. Nou, A. Miranda, T. Cortes, and A. Brinkmann. GekkoFS - a temporary distributed file system for HPC applications. In *2018 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 319–324, 2018.

[28] Dror G. Feitelson. *Workload Modeling for Computer Systems Performance Evaluation*. Cambridge University Press, New York, NY, USA, 1st edition, 2015.

[29] Dan Tsafrir. Using inaccurate estimates accurately. In *Job Scheduling Strategies for Parallel Processing*, pages 208–221. Springer, 2010.

[30] Dalibor Klusáček, Šimon Tóth, and Gabriela Podolníková. Complex job scheduling simulations with Alea 4. In *Ninth EAI International Conference on Simulation Tools and Techniques (SimuTools 2016)*, pages 124–129. ACM, 2016.

[31] Parallel workloads archive. http://www.cs.huji.ac.il/labs/parallel/workload/, February 2019.

[32] ForHLR II, KIT/SCC. https://www.scc.kit.edu/dienste/forhlr2.php.

[33] The San Diego Supercomputer Center (SDSC) SP2 log. http://www.cs.huji.ac.il/labs/parallel/workload/l_sdsc_sp2/.

[34] The Swedish Royal Institute of Technology (KTH) IBM SP2 log. http://www.cs.huji.ac.il/labs/parallel/workload/l_kth_sp2/.

[35] Steven Hotovy. Workload evolution on the Cornell Theory Center IBM SP2. In Dror G. Feitelson and Larry Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, pages 27–40, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[36] Dalibor Klusáček and Václav Chlumský. Evaluating the Impact of Soft Walltimes on Job Scheduling Performance. In Narayan Desai Dalibor Klusáček, Walfredo Cirne, editor, *Job Scheduling Strategies for Parallel Processing*, volume 11332 of *LNCS*, pages 15–38. Springer, 2018.

[37] Alea 4: Job scheduling simulator, February 2019. https://github.com/aleasimulator.

[38] Wei Tang, Narayan Desai, Daniel Buettner, and Zhiling Lan. Analyzing and adjusting user runtime estimates to improve job scheduling on the Blue Gene/P. In *IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, pages 1–11. IEEE, 2010.

[39] Soft walltime documentation, June 2019. https://pbspro.atlassian.net/wiki/spaces/PD/pages/42532871/PP-482+Soft+Walltime.

[40] Mehmet Soysal, Marco Berghoff, Thorsten Zirwes, Marc-André Vef, Sebastian Oeste, Andre Brinkman, Wolfgang E. Nagel, and Achim Streit. Using On-demand File Systems in HPC Environments. *Accepted @ The 2019 International Conference on High Performance Computing and Simulation (HPBench@HPCS),*, 2019.