Correcting Job Walltime in a Resource-Constrained Environment

Jessi Christa Rubio^(✉), Aira Villapando^(✉), Christian Matira,
and Jeffrey Aborot

Department of Science and Technology, Advanced Science and Technology Institute,
Quezon City, Philippines
{jessi,aira.villapando,christianmatira,jep}@asti.dost.gov.ph
<http://asti.dost.gov.ph>

Abstract. A resource-constrained HPC system such as the Computing and Archiving Research Environment (COARE) facility provides a collaborative platform for researchers to run computationally intensive experiments to address societal issues. However, users encounter job processing delays that result in low research productivity. Known causes come from the limited system capacity and the relatively long and rarely modified default walltime. In this study, we selected and characterized real HPC workloads. Then, we reviewed and applied the recommended runtime or walltime-based predictive-corrective scheduling techniques to reduce long job queues and scheduling slowdown. Using simulations to determine walltime scheduling performances on environments with limited capacity, we proved that our proposed walltime correction, especially its simple version, is enough to increase scheduling productivity. Our experiments significantly reduced the average bounded scheduling slowdown in COARE by 98.95% with a predictive-corrective approach, and 99.90% with a correction-only algorithm. Systems with large job diversity as well as those comprising of mostly short jobs significantly lowered delays and slowdown, notably with walltime correction. These simulation results strengthen our recommendation to resource-constrained system administrators to start utilizing walltime correction even without prediction to eventually increase HPC productivity.

Keywords: Resource-constrained environment · Walltime · Prediction · Correction · Job scheduling

1 Introduction

High-performance computing (HPC) systems comparable to the Computing and Archiving Research Environment (COARE) [1] of the Department of Science and Technology - Advanced Science and Technology Institute (DOST-ASTI) cater to

Supported by the Department of Science and Technology - Advanced Science and Technology Institute.

© The Author(s) 2020
D. K. Panda (Ed.): SCFA 2020, LNCS 12082, pp. 118–137, 2020.
https://doi.org/10.1007/978-3-030-48842-0_8

data scientists and researchers who have growing demands for computing power. In particular, COARE HPC users work on computationally intensive research to address societal issues such as rice genome analysis for securing public health nutrition [2] and flood hazard mapping for disaster preparedness [3]. These studies are relevant and applicable to highly urbanized areas. On the path to modernizing solutions to these pressing concerns, initiatives from several research institutes in the country (in partnership with COARE) encounter hindrances in their resources. COARE and similar facilities have capacity limitations because of policy and budgetary constraints in operational and capital expenditures. Especially for government research institutes, it may take a while to enact new policy changes [4, 5] and to add more compute and storage servers that must undergo a notoriously long procurement process [6]. Thus, a shared environment among researchers where they can collaborate and concurrently calculate solutions to complex computing tasks is essential. Practicing resource management or job scheduling enables this sharing. Improving job scheduling performance then becomes crucial to optimizing the usage of resource-constrained HPC systems.

Such systems practice default job scheduling configuration with the *walltime request* (WTR) set to the maximum. Depending on the demand for longer simulation, COARE sets a default WTR to 7 or 14 days for all jobs. System administrators approximated this WTR setting based on the runtimes of the first few jobs that had been submitted to the facility when its operations began. The scheduler reads this walltime or *kill time* as the hard limit to process a job to give way for other jobs to run. However, not all jobs take as long as the estimated walltime to finish; most jobs need just under a day to complete. Furthermore, COARE users rarely adjust their WTRs (see Sect. 3.1) and instead use the default settings. This situation introduces inaccurate scheduling, which hampers processing more jobs. Moreover, setting the walltime to the maximum disables the backfilling of small and lower priority jobs [7]. From our analysis of jobs submitted to COARE (detailed in Sect. 2), most of these encounter long job queues to give way to higher priority ones that have huge computational resource requirements to finish. This processing delay has been one of the most pressing critical complaints of the COARE HPC users.

For years, several researchers continue to analyze and perform thorough development of walltime-based scheduling to improve scheduling accuracy [8–10]. In this work, we analyzed how COARE and other resource-constrained HPC systems (defined in Sect. 3) can take advantage of existing predictive-corrective WTR-based scheduling algorithms (expounded in Sect. 4). Specifically, we contribute the following:

- walltime corrective algorithms *even without prediction* can reduce scheduling slowdown and eventually eliminate unwanted job delays, and
- a *simple* version of walltime correction, a more practical approach than existing corrective algorithms that systems like COARE could immediately utilize.

We additionally developed a regression-based walltime prediction that considers job size diversity and accounts more features not limited to the recommended

CPU and walltime [11] to ensure finer predictions. After performing scheduling simulations (Sect. 5) on identified real HPC workloads, our results (Sect. 6) proved a significant increase in scheduling productivity. We conducted this study to gain useful insights to revise current HPC operations policies not only for COARE but to guide similar resource-constrained environments as well.

2 Productivity in Resource-Constrained HPC Systems

2.1 Job Delays

Of the total responses on DOST-ASTI COARE’s client satisfaction survey, 30.5% have reservations on the performance and reliability of the service with the current system. Not only is there a high demand for faster and larger computational power, but there were also helpful comments on long job queues. In particular, an end-user raised his concern about experiencing a one-week waiting time for one of his jobs. A week’s time is the default walltime in COARE, which primarily contributed to the long waiting time. About 99.5% of the total jobs from COARE have waiting times of less than 3 days. These 198,386 jobs, however, should not discount the 142 jobs that queued for more than 7 days, as depicted in Table 1. These queued jobs generally had large CPU and memory requirements that could not necessarily fit available nodes. If these large jobs have short run times, more than 7 days of waiting could really be frustrating especially if the user’s experiment is highly relevant in creating social impact.

Table 1. Distribution of jobs in COARE grouped according to their waiting time.

Waiting time (days)	No. of jobs
<3	198,386
[3, 7)	526
[7, 14)	115
≥ 14	27

Upgrading the facility’s computational capacity faces challenges that require careful planning to meet growing demands and adhere to existing policies. Because of operational and capital cost restrictions for each fiscal year, the length of time to acquire such equipment could render longer productivity delays or, worse, obsolete hardware by the time it operates at the production level. Capacity management, though recommended [6], is still an ongoing process and is yet to be established in COARE. Given these limitations, it is imperative to find ways to maximize usage with existing resources such as shortening job queues. Reducing job delays requires implementing an accurate walltime scheduling.

2.2 Walltime Accuracy Effect on Productivity

Since walltime is user-specified, walltime accuracy may depend on its closeness to the actual runtime in terms of underestimates or overestimates. A job with *underestimated* runtime gets paused or killed if the WTR is less than its actual duration. Meanwhile, *overestimation* hinders the scheduler to correctly organize jobs because the compute nodes are already reserved for other jobs. This situation is particularly evident when most or all WTRs are set to the maximum default timelimit [7], which resembles the case in COARE. Alike resource-constrained systems suffer from inaccurate scheduling that leads to long queues. Because of the limited hardware capacity and an increase in the number of users, more jobs need to be processed in the same period, subsequently resulting in even longer queues. These job delays imply an irony of performing high-speed calculations, which defeats productivity. The length of a job correlates to the user acceptance of the waiting time [12]. Hence, if most jobs have small resource requirements and shorter runtimes get stuck in a long queue, their waiting time consequently increases. This scenario then becomes unacceptable.

3 Understanding Real HPC Workloads

3.1 Walltime Characteristics

If we look closely, Fig. 1 shows that COARE is mostly comprised of fixed walltime requests at either 7 or 14 days, which demonstrate overestimates with the maximum timelimit. Sizeable wide gaps between the actual runtime and the WTR are observable. These differences cause scheduling walltime inaccuracies. Though real and large computer systems from the Parallel Workloads Archive [13] may not fully represent resource-constrained facilities, we used several of these workloads in comparison to COARE's that depict real-world scenarios for reproducibility. Alternatively, we could use the simulated results from these workloads to find out if correction-only WTR scheduling is sufficient and applicable for large HPC systems. We selected workloads from the archive that are similar to COARE, which comprise jobs with diverse or *heterogeneous* geometries [14]. This heterogeneity is currently an architectural trend in HPC systems [15].

A comparable workload is from the University of Luxembourg Gaia Cluster [16], which also portrays differences between runtime and WTR but at minimal distinction. MetaCentrum2 [17] has larger WTR and runtime gaps similar to COARE's but more accurate WTRs at the latter part. The CEA Curie system [18] primarily consists of jobs having runtime and WTR difference slightly distinguishable and within a day's length. To extend our analysis to other possible HPC setups beyond the small heterogeneous systems, we consider the large Curie workload and the homogeneous HPC2N Seth workload [19]. Further, with the Gaia workload primarily composed of specialized biological and engineering computing experiments and the homogeneous Seth workload, these systems may represent resource-constrained environments dedicated to specific scientific

Fig. 1. Daily average job walltime requests and daily average job runtimes in various real HPC systems.

applications. Less variation in the job sizes in a homogeneous workload could mean similar experiments. As observed in Fig. 1, the Seth workload depicts an ideal case of walltime estimates that are comparable to job duration.

3.2 Job Diversity and Walltime Scheduling

Job geometry or size refers to a combination of compute and walltime resources [14]. Jobs with small geometry may consist of a few compute requirements and short walltime while large ones may be composed of hundreds of CPUs and may span for days. In Fig. 2, we characterized the jobs of each selected HPC workload to give context on their job geometry distribution and to learn how this variation in job sizes influences scheduling performance. We applied hexbin plotting of the workloads, where each bin constitutes the number of counts for each number of CPU and runtime combination as represented in the color bar for guidance. These plots require logarithmic scaling of the bins to easily differentiate the small jobs from the large ones. To elucidate further, small jobs take the bottom left corner of the plot while longer jobs occupy the right side. This representation allowed us to understand the implications of workload heterogeneity among HPC clusters with respect to scheduling policies presented in Table 2.

The COARE workload (Fig. 2a), as well as the MetaCentrum2 (Fig. 2c), consists of predominantly small jobs and notably long jobs with small CPU requirements and long runtime. With a relatively wide distribution of large or long jobs, we can say that the COARE workload is highly heterogeneous or diverse. Also heterogeneous, the Gaia workload (Fig. 2b) has a good concentration of jobs at

Fig. 2. Job size distribution of HPC workloads from (a) ASTI COARE, (b) UniLu Gaia, (c) MetaCentrum2 and (d) HPC2N Seth where the color bar represents a scaled count n of each hexbin, given by $\log_{10}(n)$.

Fig. 3. Job size distribution of workload from CEA CURIE.

Table 2. Scheduling scenarios of predictive and corrective policies.

WTR policy	Prediction	Correction
<i>user-estimate</i>	none	user-estimates
<i>EASY++</i>	AVE_2	simple
		power
<i>regression</i>	<i>AdaBoost</i>	simple
		power
<i>simple</i>	none	simple
<i>power</i>	none	power

the bottom and the left corner depicting small jobs mainly with a dispersed set of long jobs. Similar to Gaia, the homogeneous Seth (Fig. 2d) has mostly short jobs concentrated at the bottom left corner. On one hand, the expansive Curie workload may have short jobs with less than a day's duration but these jobs have huge CPU requirements (Fig. 3). Given these workloads, we also must note that results may vary from one workload to another [20].

Heterogeneity in a workload may decrease the job waiting time predictability: the more diverse the job geometries, the harder it is to determine when jobs would finish [14]. In an environment with high job diversity such as those in

resource-constrained systems, there must be a way to refine scheduling requirements such as having accurate WTRs. To reduce long queues, accurate walltime will enable the scheduler to precisely assign jobs to allocated nodes [21]. Thus for heterogeneous workloads, developing accurate walltime prediction becomes relevant in the scheduling performance.

4 Walltime-Based Scheduling

4.1 Walltime Prediction and Correction

The goal of WTR-based prediction is to generate walltime values close to the actual duration for efficient scheduling. *Prediction* techniques along with *correction* and *backfilling* algorithms form a *heuristic triple* in walltime-based scheduling [11]. Scheduling performance varies depending on the combination of algorithms in the triple. As an illustration, if the runtime should reach the predicted walltime and the job is not yet done, correcting the kill time will prevent premature job termination. Instead of letting the scheduler kill jobs based on user-estimates, corrective techniques will automatically extend the walltime of jobs either incrementally or by doubling its value before the kill time.

We derived combinations of predictive and corrective algorithms and compared these to a user-estimate walltime request-based scheduling (see Table 2 for a summary of scheduling scenarios). We define *user-estimates* as user-specified approximates of their jobs' runtime. The existing practice in HPCs similar to COARE is to set user-estimate walltime request as the kill time.

The *user-estimate* scheduling has no predictive algorithm, but it allows the user to indicate the job walltime. In the case of COARE, user-estimates are generally the default WTR values. This prevents the continuation of jobs with duration more than the walltime. If set too high relative to the mean duration of all jobs, the scheduler will fail to accurately estimate the length of jobs. This will cause jobs to pile up leading to a long queue. To counter this inefficiency, a walltime-based predictive approach empowers the scheduler to have better foresight of each job's probable duration and thus precisely assigns jobs to appropriate resources. Prediction comes best with correction when avoiding underestimated walltime.

In the third part of the triple, the scheduler backfills queued jobs to available nodes. An efficient strategy is to backfill the shortest job first as it is with the EASY++-SJBF [7]. Along with backfilling, the EASY++-SJBF implements averaging the runtime of the previous two jobs of the same user to predict the walltime and automatically increases the time limit to correct underestimates. Because backfilling is already in effect in the COARE's scheduler, we focused on analyzing the triple's prediction and correction parts (as in Table 2) and set the backfilling configuration as fixed. We did this to differentiate and isolate the scheduling improvements brought by walltime prediction and correction.

4.2 User-Based Prediction

Another prediction method uses *soft* walltime estimates by taking as a factor the most accurate walltime with respect to the previous job duration of the same user [9]. If the posted walltime becomes underpredicted, the soft method then kills the job once its runtime reaches the user estimate. Setting the predictor to use the past 2 jobs as a reference would suffice [7] compared to considering all past jobs' duration.

While both EASY++ and soft techniques employ prediction, the accuracy of the prediction becomes limited due to its user-based only characteristics. These methods are dependent on the historical job duration of the same user. Predicting the walltime on the assumption that users consecutively run the same experiment fails to recognize that these jobs may have different lengths. Illustratively, if the user sequentially runs a 2-h job and another at 16 h, how are we certain that the next job is within their 9-h average? Correction (detailed in Sect. 4.4) becomes helpful at this point as it extends the walltime should there be an underestimation. This leads us to another question, how often does this case of the same user with different jobs occur?

Upon inspecting the distribution of runtime per user in COARE, numerous users have jobs of different lengths. Dissecting this distribution aids in analyzing how runtime varies for every user.

Fig. 4. Runtime distribution of user37 and user67 in COARE.

Looking closely at the job runtimes of user67 in Fig. 4, around 80% of its jobs were largely varying from 15 to 165 h. On the contrary, 65% of jobs submitted by user37 had runtimes ranging at a narrow 15 to 25 h. From these observations, we deduced that the prediction in EASY++ will be ineffective in user67 but will yield more accurate WTR in user37. We cannot say the EASY++ prediction would work properly if the user67 and the like scenario happens frequently. For this study, we are curious on how effective predictive algorithms are in the

scheduling process because if correction would always take place then this would be enough and we no longer need to implement prediction.

4.3 User and Job-Based Prediction

The same user's jobs that have similar characteristics may also have comparable runtime though not necessarily submitted consecutively. To incorporate both user and job-based prediction, an existing scheduling algorithm alerts users of potential underestimates wherein the jobs are patterned on the runtime behavior of other jobs from the same scientific application [10]. The premise of this algorithm approximates the duration of jobs meant for solving a particular type of differential equation problem the same runtime as future jobs of similar nature. But this algorithm focuses on walltime underestimates only and would require a large database of experiments and their runtime behavior that may result in inefficient scheduling. Further, scientific applications vary from one computer system to another. Other resource-constrained environments collaborating with COARE either have specific patterns of experimental calculations or cater to experiments that are as diverse as research from different scientific fields [2]. A numerical modeling type of problem has a myriad of resource requirements and the extent of this variation must be carefully considered when adopting this job-based prediction to actual HPC systems.

Narrowing the job-based prediction to available standard workload logs [13] instead of depending on the jobs' scientific application, a *regression* model can consider CPU resource and walltime requests of each job. This method is distinctly relevant for those with large geometries as predicting this type of job properly will lead to better scheduling performance [11]. Because most schedulers rely on the CPU requirement, gauging other job features, such as burst buffers when it comes to I/O intensive processing, can lead to improved performance [22]. With the available parameters from the workload logs in mind, we disregarded burst buffer then we accounted for other features such as memory size. As recommended [11], we developed a regression-based prediction of runtime estimates suited for COARE with more features considered other than CPU and WTR to ensure finer prediction accuracy.

We implemented our version of this prediction using the established AdaBoost algorithm [23] in conjunction with decision trees to extract potential runtime in a *regression* manner (see Algorithm 1). Instead of utilizing both squared and linear error functions as suggested [11], we applied the closely comparable AdaBoost, a commonly implemented and relatively accurate regression model for prediction [24]. For every learning iteration m , the model equally weighs WTR predictions made by fitting the decision tree regressor, $y_m(x)$, regardless of accuracy to minimize the linear error function

$$J_m = \sum_{i=1}^N w_i^{(m)} I[y_i(x_n) \neq h_i(x_m)], \quad (1)$$

where h is an output hypothesis. AdaBoost works by tweaking these weights resulting from the first learner depending on the error of prediction. The larger

Algorithm 1. *regression*-based WTR policy using AdaBoost

Input: dataset of size n with feature space X_n and runtime Y_n , weak learner *decision tree*

Output: regression model $F(x)$

- 1: Initialize the data weights w_n .
 - 2: **for** $m = 1$ to M **do**
 - 3: fit decision tree regressor $y_m(x)$ by minimizing weighted error function J_m
 - 4: compute for the weighted training error ϵ_m
 - 5: evaluate coefficient $\alpha_m = \log(\frac{1-\epsilon_m}{\epsilon_m})$
 - 6: update data weights $w_n^{(m+1)}$
 - 7: **end for**
 - 8: predict using the final model $F(x) = \text{sign}(\sum_{m=1}^M \alpha_m y_m(x))$
-

the prediction error ϵ_m , the smaller and more negative the weight w_n becomes. The predicted WTR is the weighted median prediction by the learners. In this writing, we considered working on historical data from X_n features comprising *user*, *job runtime*, and *requested CPU* and *memory* resources. In the instance that a job continues to run within 60s of the estimated WTR, the scheduling invokes a corrective algorithm. A potential downfall of this regression technique lies in the large historical data that the prediction has to always check which could lead to an even greater slowdown.

If the same user adjusted the compute requirements of the same experiment say requested for 60 CPUs instead, then the length of the new job's duration will most probably be different. The regression approach assumes that the same user can run different experiments at various points in time, contrary to EASY++. If the same user has another experiment with the same compute requirements, specifically numerical modeling this time compared to last time's statistical analysis, and the duration becomes 10 h, then prediction in the regression should still be effective because duration is one of the assumed features and will correctly classify the change as an entirely different experiment.

4.4 Correction

As indicated in Table 2, the predictive-corrective EASY++ [7] engages a technique to predict the runtime and then increments the walltime before the premature termination of jobs with underestimated walltime. Correction can be in the form of user-estimates or doubling WTRs [11]. Another form is the *power* function $15 \times 2^{i-2}$, where $i = 2, \dots, n$ minutes, as exercised in EASY++ and proven to deliver more accurate WTRs than the other correction methods. Aside from the power correction, we considered a simpler approach to correct the underestimated walltime. We invoked our version of an incremental walltime correction called *simple* as soon as the current runtime of a job reaches 60s before the set walltime (see Algorithm 2).

The simple corrective method basically checks if a job is still running within a minute of its set time limit. If it is, the scheduler will automatically extend the

Algorithm 2. *simple* walltime correction

Input: job object instance *job*, user-estimate walltime WTR, job start time *s*, current running time *t*

Output: updated walltime WTR

```

1: Query current running job by user job.user_id.
2: while job.user_id is RUNNING do
3:   if  $\text{WTR} - t < 60$  then
4:     if  $0 \leq t - s \leq 604800$  then
5:        $\text{WTR} = \text{WTR} + 3600$ 
6:     end if
7:   end if
8: end while

```

walltime limit to 1 h. It continues to check and update the time limit until the job completes or the hard time limit of 7 days is reached, whichever comes first.

Figure 5 details a comparison of the two corrective techniques and how fast their correction would reach a job’s actual duration. Correction stops as soon as the corrected walltime is greater than or equal to the runtime. At iteration 0, WTRs of jobs are arbitrarily initialized to 2 h and 6 h. This headstart represents the set walltime prediction before correction takes place. For an 8-h job, a 6-h headstart is a closer prediction than 2 h. If prediction is more accurate, then the simple method will approach the runtime sooner and will produce accurate walltime scheduling. Conversely, if prediction is bad, the power method converges faster with the actual duration.

Fig. 5. Walltime iteration comparison of the two correction methods with respect to actual runtimes arbitrarily given 2-h (in black) and 6-h (in white) headstart.

To determine how correction-only algorithms perform compared to predictive-corrective ones, we implemented the proposed simple and the power correction WTR policies and assumed a default prediction of 10 min. Setting prediction to this value invokes underestimation that should trigger correction. Based on the runtime distribution of our identified workloads (see Fig. 6 and

Sect. 3.1 for more information), most jobs are approximately less than 1 h. Thus, to ensure underprediction takes place, we kept the default to 10 min instead of 1 h or 10 h or more.

Fig. 6. Cumulative distribution of jobs with respect to runtime of various real HPC workloads.

Walltime underestimation from coarse-grained user-estimates is seen as the primary source of scheduling inaccuracies [9–11, 25]. Predicting walltime is likewise prone to error that correction could address. This strategy avoids lost scheduling opportunities in overestimation and reduces the gap between the user-estimates and the actual runtime. If the correction part is always taking place, then we can disregard prediction and implement correction-only in the scheduling. Moreover, if correction even without prediction improves scheduling performance in resource-constrained facilities, then we can therefore solidify our recommendation to adopt this policy to other similar HPC environments.

5 Experimental Setup

5.1 Workload Preparation

To demonstrate our idea, we performed simulations on the WTR policies (Table 2) using several real HPC workloads. If we repeat the same test scenario, simulation results will not converge [20]. Hence, to conduct reliable experimentation, we tested the reproducibility of our assumptions by comparing job traces from DOST-ASTI COARE to other computer systems from the Parallel Workloads Archive [13]. Specifically, we implemented our theories by simulating workloads from the UniLu Gaia (2014-2 logs [16]) and MetaCentrum2 (2013-3 logs [17]). We sampled the MetaCentrum2 workload to one month period of the most recent jobs to simplify our simulations. These workloads are from heterogeneous systems similar to COARE’s. We additionally examined the workload from

the CEA Curie (2011-2.1 cleaned logs [18]), likewise heterogeneous, which comprises of more than 93,000 CPUs that may not accurately represent a resource-constrained environment. We regard the Curie system in our experiments since it utilizes the same scheduler as COARE’s called Slurm (to be discussed in Sect. 4.2). Including this workload in our experiments will help us understand how large HPC systems influence predictive-corrective walltime scheduling performance. Further, we considered the HPC2N Seth (using the 2002-2.2 clean version [19]) to include homogeneous systems, expanding our simulations to other probable HPC setup in terms of job size distribution. Table 3 illustrates selected features of each computer system.

Table 3. Generic composition of real HPC workloads used in the experiments.

	DOST-ASTI COARE	UniLu Gaia	MetaCentrum2	CEA Curie	HPC2N Seth
No. of jobs (cleaned)	199,054	51,834	197,368	11,268	45,333
No. of nodes	48	151	495	5,544	240
No. of CPUs	2,304	2,004	8,412	93,312	240
Period (month)	12	3	1	1	12

There were specific workload anomalies that must be filtered [8] depending on the system. For instance, the Curie workload log portion considered contains jobs submitted only after February 2012. This takes into account the changes made in the infrastructure design since 2011. In HPC2N Seth, we removed flurry of very high activity by a single user, which constitutes more than 55% of the whole log. Finally, we disregarded all jobs that ran for more than 7 days across all HPC systems. These filters were generally applied to remove occurrences of flurries that could introduce unwanted biases.

In aggregating our simulation results, we removed the first 1% of the simulated jobs as prescribed [7]. This would help reduce the warm-up effects brought about by the learning period at the start of the prediction algorithms.

As of this writing, the COARE workload log in SWF format as prescribed [20] as well as other relevant scripts used in this work are available online [26].

5.2 Scheduling Simulator

The Slurm Workload Manager (Slurm) [27] is a widely adopted open-source workload manager for various HPC environments. This tool facilitates the concurrent running of multiple experiments or jobs through a scheduling algorithm to assign jobs to server nodes. Because COARE uses Slurm as its scheduler, we performed our experiments using a Slurm simulator [28].

We adjusted the Slurm simulator source code to carry out either simple or power correction algorithms, as required (see Table 2). Likewise, we modified the Slurm configuration scripts to suit system setup such as nodes, processors and memory specifications for each workload [1, 16–19]. We selected appropriate data fields from the SWF that are in congruence with the Slurm simulator and then

converted it into CSV format. A pre-processing tool from the Slurm simulator package would read the CSV file and then translate it into a binary equivalent that the simulator will process.

The SWF, in an attempt to create a generalized workload format, considers only bare minimum parameters. This discrepancy means that the Slurm simulator would require parameters, particularly the number of nodes (n-nodes) and the number of tasks (n-tasks), that are not explicitly specified in workloads in SWF. Closely related to these data fields, the SWF consists of number of CPUs per job only and each workload generalizes the total system node count information instead of node count per job. To supply the simulator with these missing SWF parameters, we modeled a decision tree regressor (apart from the one discussed in Sect. 4.3) according to COARE users' behavior and fitted it to relevant parameters in other workloads.

The decision tree regressor, which is a machine learning model, predicts values based on a logic-based tree structure [29], and is inherently non-parametric. This means prediction would still be reliable even if the distribution is not normal. Compared to a single source of learning input with the traditional linear regression, we utilized the decision tree logic to consider several features of the workload format compatible with the simulator. Because the raw COARE workload log follows through Slurm accounting, it provides data fields congruent to what the Slurm simulator requires. In this case, our input training data were from the COARE workload, where we used the *number of CPUs, required memory, WTR*, and *runtime* as predictor variables. The regressor would learn from the trend of the input training workload parameters based on the decision tree logic to produce predictions of n-nodes and n-tasks.

Because COARE has a maximum of 48 CPUs per node, systems of less capacity could still encounter inaccurate regressor predictions should the requested number of nodes exceed the system's limit. Based on the 48 CPU per node in COARE, the regressor model would return 2 nodes only for a 96 CPU request which a 36 CPU per node system would insufficiently service 72 CPUs at most. To address this error, we divided the 96 CPUs by the system's CPU per node limit and used its ceiling result of 3 nodes instead.

Before running a simulation, we recompiled the simulator to read changes in the source code, and then repopulated the database with the new configuration. Upon generating a job trace file, we can then run an experiment to simulate the scheduling process. Also, simulation time lags with respect to increasing node count [28]. Therefore, we limited our analysis to shorter time ranges for some workloads (as in Sect. 5.1).

5.3 Performance Metric

We utilized the average wait time metric to evaluate the effectiveness of each scheduling policy, as is recommended to have better convergence [20]. The wait time ω is equivalent to the absolute difference between the submit time (T_{submit}) of a job to the time it starts running (T_{start}) or $|T_{\text{submit}} - T_{\text{start}}|$. Note that this wait time is exclusive of the runtime and returns results in seconds. To better

realize the delay effect on productivity, instead of seconds, we converted the results into minutes in our analysis. Greater ω value means more job processing delays, which consequently entails lowered user and system productivity.

To validate our results further, we used the average bounded slowdown (*avgBSLD*) of the scheduler as practiced [7, 11, 20]. We defined this performance metric as

$$avgBSLD = \frac{1}{n} \sum \max \left(\frac{\omega + R}{\max(R, \tau)}, 1 \right), \quad (2)$$

where R is the actual runtime, and τ is a threshold value, set to 10 s as generally practiced. The max function guarantees that each job's *BSLD* should be greater than or equal to 1 to ensure boundedness in the results. The *avgBSLD* would then result to a factor such that the greater its value, scheduling slows down even more and likewise impedes overall HPC productivity.

6 Simulation Results Analysis

In this section, we consolidated the results of our simulations and analyzed the WTR policies' impact on resource-constrained HPC systems. Again, in this paper, our goal is to evaluate the effectiveness of implementing predictive-corrective scheduling to resource-constrained systems like COARE. We used this evaluation to support our proposal that correction can independently reduce job processing delays in terms of wait time and slowdown metrics. Because of the accurate WTRs in Gaia, Seth, MetaCentrum2 and Curie, using the user-estimate as the baseline for the other policies would lead to incomparable improvement results. Instead, we compared these WTR policies from one another in all workloads.

6.1 Wait Time Performance

Upon comparing workloads from one another (Table 4), we can immediately observe that the wait time performances among scheduling policies are not discernible enough to differentiate any improvement in COARE. We associate this discrepancy from getting the mean of jobs which are mostly with less than 3 days wait time (in Table 1). To resolve this inconsistency, we considered looking into the *avgBSLD* metric (to be discussed in Sect. 6.2).

Again, the sampled MetaCentrum2 workload used in the simulation (Sect. 5.1) has more accurate user-estimates than COARE. Thus the improvement in the other policies are at a minimum. All predictive-corrective policies have less than 30 min of ω . A 30-min wait time is generally not deterrent to user productivity compared to COARE's waiting time range as depicted in Table 1. The EASY++-simple ω of around 21 min considerably reduced waiting time at most 7 min in comparison to the other WTR policies.

Also, keep in mind that Gaia and Seth workloads have similar job diversity (Fig. 2) as well as closer WTRs to the actual duration compared to COARE (Fig. 1). The predictive-corrective ω performance of the Gaia workload exhibited

Table 4. Average wait time (in minutes) among workloads for each WTR policy.

	DOST-ASTI COARE	UniLu Gaia	MetaCentrum2	CEA Curie	HPC2N Seth
<i>user-estimate</i>	4.89	260.96	28.21	313.94	93.38
<i>EASY++ (power)</i>	5.01	289.54	28.09	234.94	111.13
<i>EASY++ (simple)</i>	4.88	185.93	21.50	237.62	108.66
<i>regression (power)</i>	4.95	247.22	25.68	183.37	53.09
<i>regression (simple)</i>	5.16	243.91	25.75	171.31	52.61
<i>power</i>	4.85	1,148.76	28.05	148.20	70.00
<i>simple</i>	4.90	1,137.67	26.93	151.41	59.63

interesting results where the EASY++ prediction produced a closer headstart and elicited speedier approach to the actual duration with simple and lagged with power. Regression is at a point where its prediction is not as accurate as EASY++ but close enough to the actual runtime compared to those of the plain corrective methods. The extremely negative simple and power correction results in this workload showed the necessity for a more accurate walltime prediction. We can observe the same pattern in the Seth’s wait time results but the EASY++ yielded inaccurate predictions compared to regression and even in correction. The EASY++ predictions cannot go below where a correction-only algorithm started and thus resulted in walltime overestimation. Even if the jobs are mostly short in these workloads, regression works best when the same user runs consecutive jobs of different sizes. For resource-constrained systems with job geometry distribution as Gaia and Seth, the simple correction along with an appropriate prediction method can promise favorable results.

Meanwhile, the large Curie workload generated a sizeable wait time reduction for all policies. Again, regression produced more accurate predictions than EASY++ because of the high variation among job sizes in the workload. Simple and power correction methods in Curie were almost the same. Across all HPC clusters in our experiments, the correction-only method in a large system primarily composed of short jobs though big ones garnered the most distinguishable improvement with a notable 160 min or 2.7 h reduced waiting time compared to the user-estimate in the Curie workload.

6.2 Scheduling Slowdown

The wait time experiments generated varying outcomes across HPC systems. At this point, the wait time metric is still insufficient to discern the differences from one policy to another, particularly for COARE-like systems. Hence, we derived the *avgBSLD*. Table 5 showcases the performance of predictive-corrective algorithms across workloads and at a glance, simple and power corrective algorithms produced the same outputs.

All workloads suffer heavy slowdown with walltime set to user-estimate compared to predictive-corrective algorithms. Scheduling slowdown in COARE is gravely around 29,447 *avgBSLD*. The implemented predictive-corrective policies

consistently eliminated this severe scheduling slowdown in COARE by 98.95%, and by 99.90% with a correction-only approach.

The heterogeneous workloads from COARE, Gaia and Curie measurably improved with predictive-corrective scheduling. These workloads performed noteworthy slowdown reduction with EASY++ and regression compared to user-estimates. Between prediction algorithms, these HPC systems obtained an increase in slowdown with regression compared to EASY++. Because of the large number of jobs and the job size diversity in these workloads, prediction may take much time and thereby contribute to a slowdown of less than 567 in regression. Correction-only policy remarkably removed scheduling slowdown on these workloads. In large HPC systems like Curie, we recommend using a corrective-only scheduling as this yielded more beneficial results than those with prediction.

Wait time and slowdown results from the regression and corrective walltime policies both agreed in Gaia. We can observe the same pattern in Seth. Though EASY++ contributed the largest *avgBSLD*, we note that the user-estimate policy in the Seth workload also has accurate WTRs. A 650 slowdown factor is better than COARE's extreme 29,447. For systems like Gaia and Seth that are focused on specific scientific applications and composed of mostly short jobs, applying correction techniques along with an appropriate prediction strategy could greatly reduce scheduling slowdown and job processing delays.

The sampled portion of the heterogeneous MetaCentrum2 (described in Sect. 5.1) had accurate user-estimates that are as comparable to those of Seth and Gaia (refer to Fig. 1). Slowdown results in the user-estimate and predictive-corrective approaches for MetaCentrum2 were almost the same. Correction outperformed the other slowdown results from more than 739 to an improved 6 *avgBSLD*. Particularly for resource-constrained systems with large job diversity, a correction-only walltime scheduling consistently guarantees reduced performance slowdown.

Table 5. Average bounded slowdown among workloads for each WTR policy.

	DOST-ASTI COARE	UniLu Gaia	MetaCentrum2	CEA Curie	HPC2N Seth
<i>user-estimate</i>	29,447	1,182	806	958	637
<i>EASY++ (power)</i>	160	53	813	53	655
<i>EASY++ (simple)</i>	160	53	822	53	655
<i>regression (power)</i>	259	412	739	567	2
<i>regression (simple)</i>	259	412	739	567	2
<i>power</i>	30	5	6	19	16
<i>simple</i>	30	5	6	19	16

6.3 Results Synthesis

Understanding the characteristics of a resource-constrained system should lead to effective implementation of walltime-based scheduling. While job dependen-

cies [9] and interarrival times [30] may have an effect on WTR accuracy, we leave this for future work. Instead in this study, we focus on analyzing the workloads along with the wait time and slowdown metrics to shed light on which WTR policy works best. Performing any predictive-corrective technique could immensely reduce delays from using user-estimates only. Resource-constrained environments similar to COARE, especially those with highly varying job size distribution, should start practicing walltime correction in their scheduling. The same goes for workloads consisting of mostly short jobs that could be catering to specific research applications. Adding prediction in these types of systems would slightly reduce scheduling slowdown up to 822 across all workloads. Thus, we suggest that walltime correction without prediction, particularly our simple version, is enough and more practical to implement in the production level.

7 Conclusion

Challenges in resource-constrained HPC systems such as COARE could be initially addressed with proper implementation of appropriate walltime prediction-correction scheduling. In this study, scrutinizing workload characteristics is essential to discern system-appropriate walltime policies. Systems with large job size diversity such as COARE produced desirable scheduling slowdown reduction with predictive-corrective algorithms and remarkably in our proposed simple corrective-only approach. We can now apply a walltime corrective-only scheduling policy in the upcoming production release of COARE's new HPC cluster. Resource-constrained environments similar to COARE can correspondingly follow through the evaluation process in this paper to fit their workload conditions and more importantly practice walltime correction even without prediction.

Acknowledgments. We thank the Department of Science and Technology - Advanced Science and Technology Institute (DOST-ASTI) for supporting us in conducting this research to improve COARE's HPC facility and user experience. We would also like to thank Arvin Lamando and most especially Jay Samuel Combinido for their invaluable contributions during the first stages of this research. Additionally, we acknowledge real HPC workload contributors from the Parallel Workloads Archive namely: Joseph Emeras for Gaia and Curie, Dalibor Klusáček for MetaCentrum2, and Ake Sandgren for Seth.

References

1. DOST-ASTI: Computing and Archiving Research Environment (COARE) (2019). <https://asti.dost.gov.ph/projects/coare>
2. COARE stakeholders, collaborations, and partnerships. <https://asti.dost.gov.ph/coare/wiki/Main/other-info/stakeholders/>
3. DOST-ASTI: DATOS remote sensing and data science help desk. <https://asti.dost.gov.ph/projects/datos/>
4. Morton, A.L.: Assessing policy implementation success: observations from the Philippines. *World Dev.* **24**(9), 1441–1451 (1996)

5. Quah, J.S.: Public bureaucracy and policy implementation in Asia: an introduction. *Southeast Asian J. Soc. Sci.* **15**(2), vii–xvi (1987)
6. Navarro, A., Tanghal, J.: The promises and pains in procurement reforms in the Philippines (2017). <https://pidswebs.pids.gov.ph/CDN/PUBLICATIONS/pidsdps1716.pdf>
7. Tsafirir, D., Etsion, Y., Feitelson, D.G.: Backfilling using system-generated predictions rather than user runtime estimates. *IEEE Trans. Parallel Distrib. Syst.* **18**(6), 789–803 (2007)
8. Feitelson, D.G., Tsafirir, D., Krakov, D.: Experience with using the parallel workloads archive. *J. Parallel Distrib. Comput.* **74**(10), 2967–2982 (2014)
9. Klusáček, D., Chlumský, V.: Evaluating the impact of soft walltimes on job scheduling performance. In: Klusáček, D., Cirne, W., Desai, N. (eds.) *JSSPP 2018. LNCS*, vol. 11332, pp. 15–38. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-10632-4_2
10. Guo, J., Nomura, A., Barton, R., Zhang, H., Matsuoka, S.: Machine learning predictions for underestimation of job runtime on HPC system. In: Yokota, R., Wu, W. (eds.) *SCFA 2018. LNCS*, vol. 10776, pp. 179–198. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-69953-0_11
11. Gaussier, E., Glesser, D., Reis, V., Trystram, D.: Improving backfilling by using machine learning to predict running times. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, p. 64. ACM (2015)
12. Schlagkamp, S., Renker, J.: Acceptance of waiting times in high performance computing. In: Stephanidis, C. (ed.) *HCI 2015. CCIS*, vol. 529, pp. 709–714. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21383-5_120
13. Feitelson, D.G.: Parallel workloads archive. <https://www.cse.huji.ac.il/labs/parallel/workload>
14. Rodrigo, G.P., Östberg, P.O., Elmroth, E., Antypas, K., Gerber, R., Ramakrishnan, L.: Towards understanding HPC users and systems: a NERSC case study. *J. Parallel Distrib. Comput.* **111**, 206–221 (2018)
15. Flórez, E., Barrios, C.J., Pecero, J.E.: Methods for job scheduling on computational grids: review and comparison. In: Osthoff, C., Navaux, P.O.A., Barrios Hernandez, C.J., Silva Dias, P.L. (eds.) *CARLA 2015. CCIS*, vol. 565, pp. 19–33. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-26928-3_2
16. Emeras, J.: The University of Luxembourg Gaia cluster log. https://www.cse.huji.ac.il/labs/parallel/workload/Lunilu_gaia/index.html
17. Klusáček, D., Tóth, Š., Podolníková, G.: Real-life experience with major reconfiguration of job scheduling system. In: Desai, N., Cirne, W. (eds.) *JSSPP 2015-2016. LNCS*, vol. 10353, pp. 83–101. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-61756-5_5
18. Emeras, J.: The CEA Curie log. https://www.cse.huji.ac.il/labs/parallel/workload/Lcea_curie/index.html
19. Sandgren, A., Jack, M.: The HPC2N Seth log. <https://www.cse.huji.ac.il/labs/parallel/workload/Lhpc2n/index.html>
20. Feitelson, D.G.: Metrics for parallel job scheduling and their convergence. In: Feitelson, D.G., Rudolph, L. (eds.) *JSSPP 2001. LNCS*, vol. 2221, pp. 188–205. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45540-X_11
21. Chiang, S.-H., Arpaci-Dusseau, A., Vernon, M.K.: The impact of more accurate requested runtimes on production job scheduling performance. In: Feitelson, D.G., Rudolph, L., Schwiegelshohn, U. (eds.) *JSSPP 2002. LNCS*, vol. 2537, pp. 103–127. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-36180-4_7

22. Fan, Y., et al.: Scheduling beyond CPUs for HPC. In: Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing, pp. 97–108. ACM (2019)
23. Freund, Y., Schapire, R.E.: A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.* **55**(1), 119–139 (1997)
24. Bühlmann, P., Hothorn, T.: Boosting algorithms: regularization, prediction and model fitting. *Stat. Sci.* **22** (2008). <https://doi.org/10.1214/07-STS242>
25. Tsafir, D.: Using inaccurate estimates accurately. In: Frachtenberg, E., Schwiegelshohn, U. (eds.) JSSPP 2010. LNCS, vol. 6253, pp. 208–221. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-16505-4_12
26. COARE workload in SWF. https://github.com/erangvee/slurm_sim_vanilla/blob/master/ASTI-COARE-2018-cln.swf
27. Yoo, A.B., Jette, M.A., Grondona, M.: SLURM: simple Linux utility for resource management. In: Feitelson, D., Rudolph, L., Schwiegelshohn, U. (eds.) JSSPP 2003. LNCS, vol. 2862, pp. 44–60. Springer, Heidelberg (2003). https://doi.org/10.1007/10968987_3
28. Simakov, N.A., et al.: A Slurm simulator: implementation and parametric analysis. In: Jarvis, S., Wright, S., Hammond, S. (eds.) PMBS 2017. LNCS, vol. 10724, pp. 197–217. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-72971-8_10
29. Lewis, R.J.: An introduction to classification and regression tree (CART) analysis. In: Annual Meeting of the Society for Academic Emergency Medicine in San Francisco, California, vol. 14 (2000)
30. You, H., Zhang, H.: Comprehensive workload analysis and modeling of a petascale supercomputer. In: Cirne, W., Desai, N., Frachtenberg, E., Schwiegelshohn, U. (eds.) JSSPP 2012. LNCS, vol. 7698, pp. 253–271. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-35867-8_14

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.