Model for Online Prediction of Job Execution Time Based on Scheduling Historical Data

Authors: Xu Lunfan, Xiong Min, Yonghao Xiao

Date: 2019-01-03T00:00:00+00:00

Abstract

In backfilling-based scheduling systems, the estimated execution time of jobs is an indispensable parameter. Execution time estimates based on traditional user predictions typically suffer from poor accuracy. By combining classification and instance-based learning methods, and comprehensively utilizing both template similarity and numerical similarity approaches, similar jobs for the current job are identified from historical scheduling data, and their historical information is used to predict the execution time of the current job. Attributes from the scheduling history—including username, group name, queue name, application name, number of processors requested by the user, user-requested (estimated) execution time, and amount of memory requested by the user—are employed for training and prediction, while the parameters involved in the algorithm are determined using a genetic algorithm. Numerical experiments demonstrate that, compared to reference [1], and under the premise of using fewer parameters, a similar underestimation rate to that reported in the literature was achieved, along with a lower mean absolute error. On the HPC2N04 and HPC2N05 log datasets, the mean absolute error was reduced by 43% and 77%, respectively. The impact of using online predictions to replace user estimates on job scheduling was investigated, preliminary analysis of the results was conducted, and future directions for improvement were identified.

Full Text

Preamble

Vol. 37 No. 3
Application Research of Computers
ChinaXiv Partner Journal

On-line Prediction of Job Execution Time Using Schedule Historical Data

Xu Lunfan, Xiong Min, Xiao Yonghao†
(Institute of Computer Application, China Academy of Engineering Physics, Mianyang, Sichuan 621900, China)

Abstract: In scheduling systems based on backfilling strategies, estimated application runtimes are indispensable parameters. Traditional runtimes based on user estimates are typically inaccurate. This paper combines classification and instance-based learning methods, integrating template similarity and numerical similarity approaches to identify similar jobs for a given job from historical scheduling data, and uses their historical information to predict the current job's execution time. The algorithm utilizes seven attributes from scheduling history for training and prediction: username, group name, queue name, application name, number of processors requested, user-estimated (requested) runtime, and memory requested. Parameters involved in the algorithm are determined using a genetic algorithm. Numerical experiments demonstrate that, compared with reference [1], our approach achieves a similar underestimation rate while using fewer parameters and obtains a lower mean absolute error. On the HPC2N04 and HPC2N05 log datasets, the mean absolute error is reduced by 43% and 77%, respectively. We also investigate the impact of using online predictions to replace user estimates on job scheduling, provide a preliminary analysis of the results, and identify directions for future improvement.

Keywords: execution time prediction; job scheduling; genetic algorithm; K-nearest neighbor

0 Introduction

Backfilling-based job scheduling is one of the core scheduling algorithms in modern supercomputing centers. Traditional backfilling algorithms improve CPU utilization by backfilling smaller jobs into idle CPU slots on top of the First-Come-First-Served (FCFS) policy. The algorithm uses the number of CPUs requested by queued jobs as a condition, and for jobs meeting this condition, it further requires the estimated runtime of backfill candidates as a criterion. Only jobs satisfying both conditions can be backfilled, necessitating an estimated runtime for each job—referred to as the job' s estimated execution time.

In general scheduling systems, job execution time estimates are provided by users. However, studies in references [4,5] have shown that user-provided estimates differ significantly from actual execution times. Chiang et al. [10,11] noted that prediction accuracy substantially impacts scheduling performance when using aggressive backfilling strategies, implying that more accurate runtime estimates can significantly improve scheduling system performance [10]. Reference [16] investigated approaches for enabling users to accurately predict job execution times before submission in grid environments.

An alternative approach involves 系统自动生成预估时间. For instance, references

[1,3,6,8,9,13,15] propose prediction methods based on historical data mining, which are more practical to implement compared to approaches in references [6,7]. Tsafrir et al. [6] demonstrated that a naive prediction algorithm using system-generated estimates achieves substantially higher accuracy than direct user estimates. Reference [8] clustered user submission behavior patterns, improving prediction accuracy by 5.6% while reducing computational overhead to 3.8% compared to existing prediction algorithms. Reference [17] proposed a self-organizing map-based runtime prediction method that outperforms instance-based learning and statistical smoothing. Tran et al. [1] introduced a template-based prediction method that represents the state-of-the-art, reducing job underestimation rates while improving mean absolute error (MAE) by over 20% compared to reference [6].

Existing work [1,6,8,9,13] has primarily obtained predicted execution times through offline analysis of job logs. In contrast, this paper focuses on online prediction of job execution times based on historical data.

---

1 Related Work

Research on job execution time prediction based on scheduling history assumes that users repeatedly execute identical applications in supercomputing systems, operating under the premise that "similar jobs have similar execution times." By leveraging execution times of completed jobs, we can predict the runtime of similar jobs. From the perspective of finding similar jobs, prediction algorithms can be categorized into classification and instance-based learning approaches.

Classification-based prediction primarily uses templates to identify jobs similar to the target job and employs these similar jobs to estimate runtime. Downey [2] and Gibbons [9] utilized static templates for classification, including features such as user, job name, and system queue. After classification, the average execution time of each class predicts future job runtimes. Smith et al. [13] employed genetic algorithms to dynamically define templates by selecting job features that best characterize similarity. Tsafrir et al. [6] discovered that using only the two most recently submitted jobs from the same user could also provide reasonable predictions.

Instance-based learning algorithms for job runtime prediction [1,7,8,14] retain information from N recently completed jobs and estimate runtime by searching for K nearest neighbors (jobs most similar to the target job). The core challenges involve defining similarity between two jobs and determining how to use similar jobs' execution times for prediction. The similarity between jobs X and Y is measured by a distance function.

Template similarity determines whether two jobs are similar by checking if their attribute values match exactly under a given template. For example, when the template is {username, queue name}, two jobs are considered template-similar

only if both their usernames and queue names are identical. This paper uses four attribute values as template elements: username, group name, queue name, and application name, corresponding to User ID, Group ID, Queue Number, and Application Number in SWF, denoted as u, g, q, and j, respectively. Based on these four attributes, 16 different templates can be formed (including the empty template, under which all jobs are considered similar). Naturally, different datasets require different templates, with selection methods discussed later.

Numerical similarity calculates the similarity between two jobs based on several numerical attributes using a similarity function. This paper selects three attributes for numerical similarity calculation: number of processors requested, user-requested (estimated) runtime, and memory requested, corresponding to Requested Number of Processors, Requested Times, and Requested Memory in SWF, denoted as c, r, and m, respectively.

Before calculating numerical similarity, to avoid effects from different numerical scales, this paper employs a linear normalization function to scale each value to the [0,1] interval, defined as follows:

After normalization, Equation (3) calculates the similarity Sim(J1,J2) between jobs J1 and J2. Based on the three numerical attributes c, r, and m, Sim(J1,J2) is defined as:

where x and y represent feature vectors of jobs X and Y, respectively; wa is the weight; and da is the distance of the corresponding feature.

Tran et al. [1] combined classification and instance-based learning methods, defining a similarity function to measure job similarity and incorporating the impact of underestimation on predictions during estimation. Tran' s experimental results demonstrated that their hybrid approach outperformed individual methods in most cases. This paper draws inspiration from this hybrid algorithm with certain improvements.

---

2 Job Execution Time Prediction

Jobs in supercomputing systems are highly repetitive, as users frequently execute the same applications. Therefore, we can estimate job execution times based on information from previously completed jobs. Reference [1] posits that similar jobs have similar execution times, enabling predictions of future job runtimes using similar jobs' execution times. Under this premise, the prediction algorithm consists of two steps: (a) searching for similar jobs in scheduling history data, and (b) obtaining the predicted runtime for the current job based on similar jobs' actual execution times.

In this algorithm, scheduling history serves as the primary data source. Scheduling systems store information about completed jobs for analysis, typically including user ID, submission time, wait time, user-estimated runtime, actual runtime,

and number of processor cores used. References [18,19] introduced the Standard Workload Format (SWF), which defines 18 job attributes related to scheduling and their storage format, providing standardized job scheduling history data from multiple supercomputing centers for research. This paper' s research is also based on this standard.

2.1 Similar Job Search

The similarity between jobs J1 and J2 is calculated using Equation (3), where a smaller Sim(J1,J2) value indicates higher similarity. This paper combines both template similarity and numerical similarity methods: first using template similarity to identify a set of similar jobs, then applying numerical similarity to obtain the final similar job set. Specifically, given a new job J, the steps to select K similar jobs from N scheduling history jobs are: (a) select a template T; (b) based on template T, select jobs similar to J to form set SJ*; (c) within SJ*, use the K-nearest neighbor algorithm to select K jobs most similar to J, forming set SJ.

Limiting scheduling history to the N most recently completed jobs serves two purposes: improving algorithm efficiency and applying the principle of locality —similar jobs are likely among those recently scheduled. The determination of N and K is discussed in Section 2.3.

2.2 Execution Time Prediction

After obtaining the similar job set SJ for job J, we can predict J' s execution time using the actual runtimes of jobs in SJ. This paper employs the averaging method, using the mean execution time of jobs in SJ as J' s predicted runtime, calculated as:

where Ri is the actual execution time of the i-th job in SJ. Alternatively, normalized c, r, and m values could be used as attribute labels with linear regression or support vector regression to obtain J' s predicted runtime.

To minimize structural risk, a regularization term is added to Equation (4). This paper selects the standard deviation as the regularization term:

In reference [1], to reduce the likelihood of underestimation, the authors incorporated the user-provided runtime estimate. The job similarity is divided into template similarity and numerical similarity. Template similarity determines whether two jobs are similar by checking if their corresponding attribute values are identical under a given template. If the attribute values match, the two jobs are considered similar. The template consists of a set of job attribute values. The prediction formula for job J' s runtime is as follows:

where std(.) denotes standard deviation; rJ represents the user-requested (estimated) runtime; and and are weighting factors. Specific value determination methods are discussed in Section 2.3.

2.3 Online Parameter Training

As shown in Sections 2.1 and 2.2, several parameters must be determined to obtain job J' s execution time: (a) the optimal template in template similarity; (b) the size of the job set N; (c) the size of SJ, K; and (d) factors and .

The four template attribute values are represented as x1, x2, x3, and x4 in this paper. If xi = 1, the corresponding attribute is included in the template; conversely, if xi = 0, the attribute is removed from the template.

To better utilize local information from supercomputing logs and achieve optimal prediction performance, this paper adopts a batch processing approach for online training of multiple parameter sets. Each batch of logs is divided into a historical dataset HistSet (h job logs), a training set TrainingSet (t1 job logs), and a test set TestSet (t2 job logs). The historical dataset contains retrievable historical jobs for that batch, with earlier submissions invisible to the current batch. In the training set, a genetic algorithm is used to train the parameter set (x1, x2, x3, x4, N, K, ,), with the objective of reducing prediction error and minimizing the number of underestimated jobs. In the genetic algorithm, the fitness function is defined as:

where Pi and Ri are the predicted and actual execution times for the i-th job, respectively; and PU is the percentage of underestimated jobs relative to the total number of jobs. By setting appropriate population size and generations for the genetic algorithm, a set of optimal model parameters is produced upon convergence or termination. The execution time prediction algorithm mentioned in Section 2.2 then provides predicted runtimes for jobs in that batch.

As the next batch of jobs arrives, HistSet, TrainingSet, and TestSet all shift forward by t2 job logs in the log file, initiating the next batch of training and prediction.

When using the genetic algorithm to train job parameters, relevant parameters must be configured. Table 1 presents a set of optimal genetic algorithm parameters selected through multiple rounds of experiments. Compared with other parameter combinations (h, t1, t2) such as (5000, 2000, 1000) or (3000, 1500, 500), we found that increasing the size of HistSet and TrainingSet or reducing the number of jobs predicted per round does not help reduce underestimation rates or improve prediction accuracy. The genetic algorithm terminates training after 100 iterations because setting a smaller number of generations (generations=50) prevents parameter convergence to the global optimum, while a larger number (generations=200) provides no significant improvement in prediction performance and requires longer training time.

Figure 1 [Figure 1: see original paper] shows the cumulative distribution function (CDF) of user-estimated times and actual execution times across four datasets. This figure describes the probability that a job' s estimated duration and actual duration fall within any time interval. On each dataset, the actual runtime corresponding to the same probability density is far smaller than the

user-estimated time, indicating that users generally overestimate job runtimes. Due to large variations in job execution times, the time axis is log-scaled.

As shown in Figure 1, user estimates for SDSC02 and SDSC04 are highly inaccurate, with actual time distributions being smooth while user-estimated times exhibit a step-like distribution. This suggests that users provide coarse-grained estimates, with many selecting identical estimated times. The situation is slightly better for HPC2N04 and HPC2N05, though job execution times are still generally overestimated by users. Notably, actual execution times for SDSC02 and SDSC04 typically do not exceed 10^5 seconds (approximately 1 day), while those for HPC2N04 and HPC2N05 reach up to 10^6 seconds (about 12 days), suggesting potentially larger user estimation errors for the latter two datasets.

---

3 Numerical Experimental Results

To facilitate comparison with results from reference [1] (hereinafter referred to as Minh), this paper conducts experiments using scheduling history data from the HPC2N logs at the High Performance Computing Center North (Sweden) and SDSC logs at the San Diego Supercomputer Center, as provided in reference [18]. The HPC2N logs are divided by year into two historical datasets: HPC2N04 and HPC2N05. SDSC02 and SDSC04 correspond to data from all of 2002 and 13 months from March 2004 to March 2005, respectively. These four scheduling history logs reveal detailed job execution characteristics across different clusters and time periods. Table 2 shows basic information for the four logs. While reference [1] cleaned the four logs, this paper retains all valid jobs, thus processing more jobs than reference [1].

This paper uses Mean Absolute Error (MAE) to measure prediction accuracy, calculated for N jobs as:

where Pi and Ri are the predicted and actual execution times for the i-th job, respectively.

For the underestimation problem, since our algorithm aims to minimize the number of underestimated jobs, we measure underestimation using the percentage of underestimated jobs relative to all jobs. The underestimation rate is defined as:

where I(Pi<Ri) is defined as:

3.1 Underestimation Rate

In scheduling systems employing backfilling, underestimation adversely impacts system performance. Therefore, predictions should minimize the number of underestimated jobs.

As shown in Table 3 , our prediction results achieve underestimation rates similar to those in reference [1]. The literature notes that classifying tasks into large

and small tasks effectively reduces underestimation rates. However, this paper does not categorize tasks by size, and the larger number of predicted jobs likely contributes to slightly higher underestimation rates.

Reducing underestimated jobs inevitably increases the number of overestimated jobs. Nevertheless, in backfilling systems, overestimated jobs remain preferable to underestimated ones, as overestimated jobs are not killed by the system. The predictor must also prevent excessive overestimation from increasing prediction error. To this end, this paper uses the user estimate as an upper bound in Equation (6), effectively reducing training error.

3.2 Mean Absolute Error

Mean absolute error results are presented in Table 4 . As shown, our prediction results significantly outperform those of reference [1] in most cases. The primary reason is our adoption of a different distance function (Equation 3). Compared to the distance function in reference [1], Equation (3) better captures similarity between jobs. Based on this distance function, we also avoid classifying jobs by size at the cost of some underestimation rate, thereby reducing the number of parameters trained by the genetic algorithm. Additionally, our online batch processing approach generates predictions in the order of log arrival, better utilizing real-time information.

On the SDSC04 dataset, our method exhibits relatively larger prediction errors. As shown in Figure 1, the actual runtime distribution for SDSC04 is smooth, lacking regularity for learning true execution times from historical data. For our prediction algorithm, logs with step-shaped actual runtime distributions are more likely to achieve good prediction performance.

Although our results improve upon reference [1], the absolute error remains relatively large. We conducted a preliminary analysis of prediction absolute errors based on a subset of jobs from HPC2N04 to provide references for future algorithm improvements.

Figure 2 [Figure 2: see original paper] shows the cumulative distribution of absolute errors for 9,000 jobs. The figure reveals that 80% of absolute errors are less than 1,000 seconds, while approximately 20% are relatively large. Future work could analyze characteristics of these 20% of jobs to further improve the algorithm and reduce overall average error.

As shown in Figure 3 [Figure 3: see original paper], the mean absolute error generally increases with the number of predicted jobs. The jumps in absolute error observed in the figure may indicate the emergence of special jobs whose sudden increase in absolute error leads to higher average errors. Future improvements could involve analyzing characteristics of these jobs to reduce their prediction errors and thereby decrease overall error. On the other hand, the relatively low mean absolute error when the number of jobs is small (a phenomenon also observed in other job sets) suggests that the genetic algorithm trains parameters

only on a fixed amount of historical data. As the test set grows, new parameters may need to be trained to adapt to new jobs, representing another direction for future improvement.

3.3 Impact on Job Scheduling

The following experiments quantify the impact of our prediction algorithm on job scheduling by simulating the scheduling and execution of jobs from log datasets. We use the open-source batch scheduling simulator pyss [20] for log-based simulation. This paper evaluates the EASY and EASY-SJBF backfilling-based scheduling strategies on the four log datasets mentioned above. By analyzing experimental results from different scheduling strategies and systems, we can demonstrate the beneficial impact of our prediction algorithm on job scheduling.

For each set of scheduling history logs, we first conduct a scheduling simulation using user-provided runtime estimates in pyss, using the resulting average wait time and average bounded slowdown as baselines. We then perform simulations using our predicted times as estimates and compare the corresponding objective function values.

In backfilling scheduling systems, a backfilled job must guarantee that it will not delay the execution of the first job in the wait queue. EASY selects backfillable jobs based on the First-Come-First-Served principle, while EASY-SJBF prioritizes short jobs for backfilling.

Reference [21] notes that job scheduling performance can be measured using two objective functions: (a) Average Wait Time (AVEwait). For job j, user wait time is the period between job submission time submitj and start time startj, i.e., waitj = startj - submitj. Shorter wait times indicate earlier job execution and higher user satisfaction. The average wait time across all jobs reflects overall performance. (b) Average Bounded Slowdown (AVEbsld) [22,23]. This calculates job j' s response ratio, considering both wait time waitj and runtime runtimej. A constant = 10s is set to avoid excessively large response ratios for short jobs that would not reflect scheduling performance. This metric is called the bounded slowdown for job j, and the average bounded slowdown can be similarly defined. Reducing average bounded slowdown improves overall system performance.

These two objective functions evaluate scheduling performance from the user perspective. For the four given sets of scheduling history logs, we first use user-provided job runtime estimates for simulation in pyss, establishing baseline results for average wait time and average bounded slowdown. We then use our prediction results as estimated times for simulation, comparing the corresponding objective function values with the baseline.

Tables 5 and 6 show that using our predicted times significantly reduces average wait time and average bounded slowdown across both scheduling strategies and

all four log datasets, effectively improving scheduling system service quality and reducing resource idle time.

EASY-SJBF achieves lower AVEwait and AVEbsld than EASY due to its preference for backfilling short jobs, but both strategies yield similar relative improvements in AVEwait and AVEbsld. Using predicted times yields greater improvement in AVEbsld while consistently reducing AVEwait. SDSC04 shows the most significant relative improvement, with AVEbsld improvements exceeding 60% under both scheduling strategies.

The scheduling simulation results demonstrate that for backfilling-based scheduling systems, our predictions can effectively improve scheduling performance, confirming that accurate predictions benefit job scheduling.

---

4 Conclusion

Job scheduling performance in supercomputing centers is a critical factor affecting system utilization. Currently, backfilling-based scheduling is the primary job scheduling strategy employed by supercomputing centers. In backfilling scheduling, a job' s estimated runtime is a key criterion for determining whether it can be backfilled, and the accuracy of runtime predictions directly impacts scheduling performance. This paper presents an online job performance prediction algorithm that combines classification and instance-based learning. The algorithm first searches for similar jobs using seven different attribute values from job history data, then calculates estimated runtime using three numerical attributes. We employ a genetic algorithm to search for optimal values for the eight parameters used in the algorithm.

Numerical experiments show that compared with reference [1], our approach achieves a similar underestimation rate using fewer parameters while obtaining lower mean absolute error. We conducted a preliminary analysis of error results and identified several directions for improvement. Future work will incorporate machine learning to train and learn more detailed job characteristics from scheduling history data to achieve better prediction performance. This paper also used predicted times for scheduling simulation, demonstrating improved backfilling algorithm performance. The next step involves exploring the impact of online runtime prediction on scheduling in combination with other scheduling strategies.

References

[1] Tran N, Lex W. Using historical data to predict application runtimes on backfilling parallel system [C]// Proc of the 18th Euromicro Conference on Parallel, Distributed and Network-based Processing. 2010.

[2] Allen B. Predicting queue times on space-sharing parallel computers [C]// Proc of the 11th International Parallel Processing Symposium. 1997:209-218.

[3] Allen B. Using queue time predictions for processor allocation [C]//Lecture Notes in Computer Science, volume 1291. 1997:35-57.

[4] Ahuva W, Dror G. Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling [J]. IEEE Trans on Parallel and Distributed Systems, 2001,12:529-543.

[5] Dror G, Ahuva W. Utilization and predictability in scheduling the IBM SP2 with backfilling [C]// Proc of the 12th International Parallel Processing Symposium. 1998:542-546.

[6] Dan T, Yoav E, Dror G. Backfilling using system-generated predictions rather than user runtime estimates [J]. IEEE Trans on Parallel and Distributed Systems, 2007,18: 789-803.

[7] Li Hui, David L, Lex W. Mining performance data for metascheduling decision support in the grid [J]. Future Generation Computer Systems, 2007:92-99.

[8] Liang Feng, Liu Yunzhen, Liu Hai, et al. A Parallel job execution time estimation approach based on user submission patterns within computational grids [J]. International Journal of Parallel Programming, 2015, 43 (3): 440-454.

[9] Richard G. A historical application profiler for use by parallel schedulers [C]// Lecture Notes in Computer Science, volume 1291. 1997:58-77.

[10] Chiang S, Andrea A, Mary K. The impact of more accurate requested runtimes on production job scheduling performance [C]//Lecture Notes in Computer Science, volume 2537. 2002:103-127.

[11] Chiang S, Mary K. Production job scheduling for parallel shared memory systems [C]// Proc of the 15th International Parallel and Distributed Processing Symposium. 2001.

[12] Vasupongayya S, Chiang S. Performance problems of using system-predicted runtimes for parallel job scheduling [C]// Proc of the 19th Parallel and Distributed Computing and Systems. 2007.

[13] Smith W, Foster I, Taylor V. Predicting application run times using historical information [C]//Lecture Notes in Computer Science, volume 1459. 1998:122-142.

[14] Smith W, Wong P. Resource selection using execution and queue wait time predictions [R]. NAS Technical Report Number: NAS-02-003, NASA Ames Research Center, 2002.

[15] 余莹, 李肯立, 徐雨明. 计算集群中一种基于任务执行时间的组合预测方案. 计算机应用 [J], 2015, 35 (8): 2153-2157, 2163. (Yu Ying, Li Kenli, Xu Yuming. Combined prediction scheme for runtime of tasks in computing cluster [J]. Journal of Computer Applications, 2015, 35 (8): 2153-2157, 2163.)

[16] 蒋炎华. 网格环境下任务的执行时间预测技术研究 [J]. 计算机工程与设计, 2011, 32 (10): 3428-3430. (Jiang Yanhua. Research of task execution time prediction technology in grid computing environments [J]. Computer Engineering and Design, 2011, 32 (10): 3428-3430.)

[17] Park J, Kim E. Runtime prediction of parallel applications with workload-aware clustering [J]. Journal of Supercomputing, 2017, 73 (3): 938-958.

[18] Dror F. Parallel workloads archive [EB/OL]. (2005-12-08) [2018-08-10]. http://www.cs.huji.ac.il/labs/parallel/workload/.

[19] Steve J, Cirne W, Dror G, et al. Benchmarks and standards for the evaluation of parallel job scheduler [J]. Job Scheduling Strategies for Parallel Processing. 1999:67-90.

[20] pyss-the Python scheduler simulator [EB/OL]. (2018-08-10) [2015-10-01]. http://code.google.com/p/pyss/.

[21] Dror G, Larry R, Uwe S, et al. Theory and practice in parallel job scheduling [C]//Lecture Notes in Computer Science. 1997: 1-34.

[22] Oliner A, Sahoo R, Moreira J, et al. Fault-aware job scheduling for bluegene/l systems [C]// Proc of the 18th International Parallel and Distributed Processing Symposium. 2004: 64.

[23] Eric G, David G, Reis V, et al. Improving backfilling by using machine learning to predict running times [C]// Proc of IEEE High Performance Computing, Networking, Storage and Analysis. 2017: 64.

Note: Figure translations are in progress. See original paper for figures.