Practical Resource Usage Prediction Method for Large Memory Jobs in HPC Clusters

Xiuqiao Li[1(✉)], Nan Qi[1], Yuanyuan He[2], and Bill McMillan[3]

[1] IBM China Systems Laboratory, Beijing, China
{lxiuqiao,qinan}@cn.ibm.com
[2] IBM China Systems Laboratory, Xi'an, China
yyhe@cn.ibm.com
[3] IBM United Kingdom Limited, Hursley, UK
bill.mcmillan@uk.ibm.com

Abstract. Users in high performance computing (HPC) clusters normally face challenges to specify accurate resource estimates for running their applications as batch jobs. Prediction is a common way to alleviate this complexity by using historical job records of previous runs to estimate resource usage for new coming jobs. Most of existing resource prediction methods directly build a single model to consider all of the jobs in clusters. However, people in production usage tend to only focus on the resource usage of jobs with certain patterns, e.g. jobs with large memory consumption. This paper proposes a practical resource prediction method for large memory jobs. The proposed method first tries to predict whether a job tends to use large memory size, and then predicts the final memory usage using a model which is trained by only historical large memory jobs. Using several real-world job traces collected from large production clusters of IBM Spectrum LSF customer sites, the evaluation results show that the average prediction errors can be reduced up to 40% for nearly 90% of large memory jobs. Meanwhile, the model training cost can be reduced over 30% for the evaluated job traces.

Keywords: Resource usage prediction · Large memory jobs · Resource manager

1 Introduction

Nowadays high performance computing (HPC) clusters are not only deployed in large research centers, but also widely adopted by industries such as chip design and manufacture, life sciences, etc. This trend brings more diverse workload patterns to HPC clusters compared with the traditional scientific applications. As those clusters normally consists of thousands of nodes, it is common to use resource managers (e.g. IBM Spectrum LSF [1], Slurm [2], Moab [3]) to manage resources and make decisions to allocate proper resources for applications submitted by end users. Resource managers enable multiple users sharing massive cluster resources by scheduling applications as batch jobs in queue systems. However, end users generally have little knowledge of computing resources, while resource managers normally rely on accurate resource

D. Abramson and B. R. de Supinski (Eds.): SCFA 2019, LNCS 11416, pp. 1–17, 2019.
https://doi.org/10.1007/978-3-030-18645-6_1

requirements specified by users to scheduling and allocating resources. This conflict produces challenges for cluster administrators to achieve high resource utilization and job execution efficiency in their clusters. For example, when users tend to over-estimate the resource usage of their applications, resource manager will finally place fewer jobs to run in the cluster as the reserved additional resources cannot be currently used by other waiting jobs. Conversely, application may fail due to compete resources when users made under-estimation of resources usage. Another consequence of inaccurate memory requirement is wasting budget to apply excessive memory when bursting workloads to cloud, where the resources are charged by size over time [25].

Recent rapid progress on machine learning gives the opportunities to make resource managers smarter. Specifically, job resource usage together with the job submission options (e.g. submission queue, job command) are normally recorded by resource managers as accounting information after applications are completed. Applications in large production cluster are normally run repeatably. Therefore, it is possible to explore the relationship of resource usage and job patterns from historical job records. Previous work have been done for predicting job memory usage [4, 5], job runtimes [6, 7], etc. Most of those work focus on building models using all of the historical data, and comparing various machine learning algorithms on prediction accuracy.

Based on our customer experience, we found the special needs for resource prediction from with real world industry customers. Specifically, people only care about the memory usage for those large memory jobs, such as more than several or even hundreds of gigabytes. For small memory jobs, there is no need to know the memory usage in such fine-grained size compared with the massive available memory on the nodes. To satisfy this need, we propose a practical resource prediction method to improve the prediction accuracy for the large memory jobs. Considering the workload characteristics and customer needs, we mainly made the following contributions in this study:

- We analyzed the characteristics of workload traces collected from real customers, and found the number of jobs consuming large memory is smaller than small memory ones. Then we adopt the over-sampling method for large memory jobs to reduce the information loss.
- Considering both prediction accuracy and training cost, we propose a practical prediction method using two-stages prediction models. The method removes the noise of small memory jobs when predict the final memory usage. As training complexity is reduced with smaller number of jobs and class number, it is suitable for scenarios with high model updating frequency.
- We performed evaluation tests using the collected job traces, and analyzed the benefits of the proposed method in reducing prediction errors for large memory jobs and accelerating model training.

The rest of the paper starts with Sect. 2 which gives the motivation of this work by summarizing the characteristics of real-world job traces, and then lists the main design goals of this paper. In Sects. 3 and 4, we introduce the specific work need to be done during dataset preparation, and the proposed two-stage prediction method for large memory jobs. Evaluation results are analyzed in Sect. 5, and Sect. 6 summarized related work on job resource usage prediction. At last, we make the conclusions and introduce the future directions in Sect. 7.

2 Motivation

The work in this paper is motivated from real-world scenarios encountered by resource manager users. In this section, we will first introduce the analysis of job traces collected from our customers. Then the design goals of the resource prediction method is given based on the analysis results.

2.1 Real-World Traces Analysis

IBM Spectrum LSF is widely used in large scale computing centers from academic research centers to industrial datacenters and even on cloud. The largest clusters in those sites consists of several and even ten thousand of computing nodes with millions of jobs finished per day. We collected three job traces from those sites[1] and had surveys with the cluster administrators about the job patterns in those clusters. Based on our experiences and analysis of real job traces in their clusters, we have the following major observations:

- ***Large memory jobs are only small portions of the whole job records though contribute to most of the total memory consumptions.***

Table 1 shows that the statistics of total memory usage of large and small memory jobs. Take Trace A as an example, 99.49% of total memory usage are contributed by 37.3% of jobs consuming memory larger than 1 GB. Trace C has more large memory jobs but most of them use less than 16 GB memory, while the large memory jobs of the other two traces are scattered between 1 GB and 128 GB.

Table 1. Job and memory usage statistics of job traces by per-job memory usage (>1 GB as large memory jobs and others as small memory jobs).

Traces	#Jobs	Small memory jobs (%)	Small memory usage (%)	Large memory jobs (%)	Large memory usage (%)
Trace A	587k	62.7	0.51	37.3	99.49
Trace B	907k	77	3.3	23	96.7
Trace C	1m	43.4	12.1	56.6	87.9

- ***Cluster administrators care more about the accuracy of memory usage of large memory jobs***.

Figure 1(a) shows the job traces statistics of user specified errors compared with real memory usage consumption. The relative user specified error of a job is calculated as the following formula:

$$100 * |User\_Specified\_Mem - Real\_Mem\_Usage| / Real\_Mem\_Usage \quad (1)$$

[1] The customer related information in the job traces are hidden in this paper due to IBM data privacy policies.

As there are no user specified memory values recorded in Trace B, we just show the statistics of Trace A and C. The absolute value of memory loss has large impact on cluster resource utilizations. For example, it is a common case that a chip design simulation application can consume hundreds of gigabytes memory in maximum. Meanwhile, end users specify several times of real memory usage to guarantee application running correctly. That means hundreds of gigabytes are wasted and cannot be concurrently used by other jobs.

In contrast, Fig. 1(b) shows that the small memory jobs tend to be completed in short time even in a minute. It could be tolerable to have certain user specified errors for those small memory jobs. One exception is there are lots of small memory jobs run for over 1 h and less than 6 h in Trace C. Those jobs are possibly compute-intensive jobs, and we found the user specified errors are quite low for those jobs.

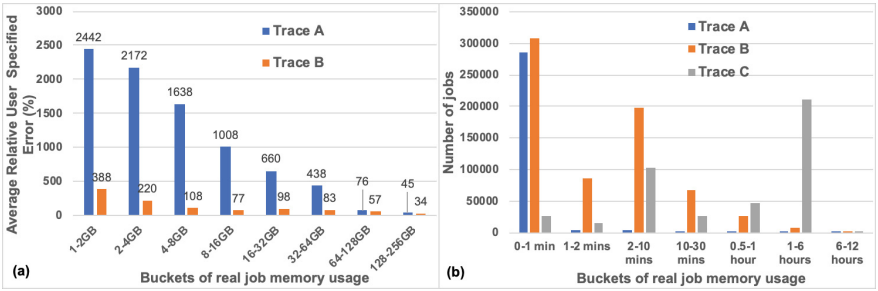

Fig. 1. Job traces statistics: (a) User specified errors for job memory usage; (b) Number of small memory jobs by their job runtime ranges

- ***Small memory jobs introduce noises in predicting memory usage for large memory jobs***

Figure 2 shows that the comparisons of memory usage prediction errors for large memory jobs using datasets with different amount of small memory jobs. It can be observed that the prediction is more accurate with a smaller number of small memory jobs in the training datasets. It is easy to be explained as the small memory jobs become noise data points when training model for large memory jobs.

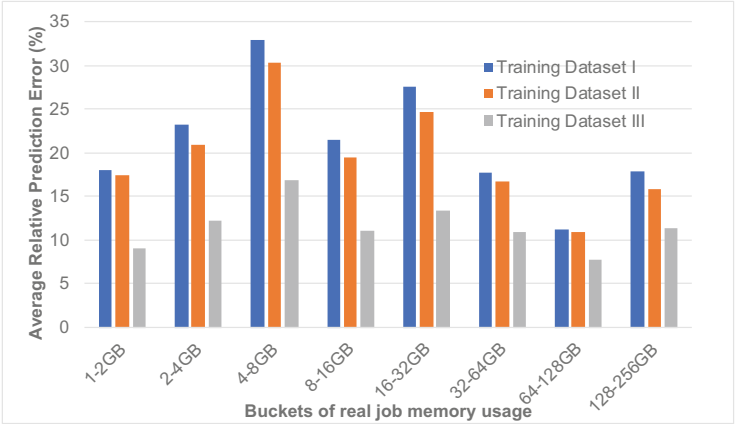

Fig. 2. Comparison of the average relative prediction errors of job memory usage by using training datasets (generated from job trace A) with various portions of small memory jobs: Dataset I consists of all jobs, Dataset II has large memory jobs and half part of small memory jobs, and Dataset III only contains of large memory jobs.

- ***Jobs with new job patterns keep updating during the cluster is running***

According to our analysis of those traces, there are always new job patterns cannot fit into the ones recognized using previous job records. Cluster administrators explain this observation as there are new projects started with new applications or people improve their work by changing their applications or scripts. That means the prediction model needs to be updated more frequently to include the new job patterns. Considering the large amount of historical jobs in the training datasets, the training model tasks are quite time-consuming. Resource prediction face challenges to have low training cost while keeping good prediction accuracy at the same time.

2.2 Design Goals

To satisfy the above real scenarios faced by resource managers, we target to design a practical resource prediction method using machine learning models for large memory jobs. The method can help resource manager administrators to better adjust over-estimated user specified memory values for their large memory jobs, improve the overall cluster utilization for memory resource and save budget of applying cloud resources to burst workloads.

Specifically, the method aims to achieve the following design goals: (1) to improve the memory usage prediction accuracy for large memory jobs with high coverage rate; (2) to reduce the model training cost to support frequent model updating; (3) to keep low prediction latency to reduce the impact on job submission performance.

3 Dataset Preparation

The collected job traces need to be well pre-processed before using for building prediction models. Enough number of job-related attributes should be extracted and mapped to input training features to train a good model. Besides the normal machine learning flow to prepare datasets, the following steps are specific to the requirements of large memory jobs prediction studied in this paper.

3.1 Biased Job Trace Sampling

It is a common case for many industrial production clusters to have millions of finished jobs per day. The training cost of building a daily updating model using months of data would be impossible. Data sampling is one of the feasible ways to reduce the training dataset size while keeps the job patterns as many as possible. Based on the job traces analysis, there are normally more small memory jobs though the total memory usage cannot be comparable with the one of large memory jobs. Therefore, the large memory jobs could be overwhelmed if we adopt uniform sampling method to extract training datasets. Instead, the large memory jobs need to be favored than other ones during sampling.

3.2 Job Attributes Extraction

The job traces are log based formats which contains the job submission options together with the job resource usage information. Some log fields are directly readable, while some other fields are encoded with rich information. Generally, resource managers provide public APIs to decode detail attributes from those encoded fields.

- The encoded job option fields are generally using bit flags to record multiple boolean options (e.g. whether a job is an interactive or urgent one).
- Some job fields need to be further processed to extract useful information. For example, the job submission time is normally recorded as Unix time which cannot be directly used as a feature. It is more meaningful to extract day or hour in a week or a day to recognize the time related job submission patterns, such as a user may always run his simulation application before leaving office every day.
- There are also more customized job fields which need to be processed as multiple features. For example, people may leverage job names or project names to 'tag' some job specific information with pre-defined formats.

We directly extracted 30+ features (e.g. queue, project, application, job group names, job command, requested resource names and their values) from the collected job traces used by the work in this paper. The features with non-integer data types are encoded into integers and normalized before used to training models. It is possible to further extract more useful information, but it is not the scope of the main problem addressed in this paper (Table 2).

Table 2. Training features extracted for the studies in this paper.

Feature	Data type	Feature	Data type
Submission User ID	*Integer*	*Attached SLA Name*	*String*
Submission User Group	*String*	*Submission Host Name*	*String*
Queue Name	*String*	*Input File Name*	*String*
Application Profile Name	*String*	*Output File Name*	*String*
Project Name	*String*	*Specified Job Begin Time*	*Date*
Job Command Name	*String*	*Specified Termination Time*	*Data*
Job Working Directory	*String*	*Job Name*	*String*
Resource Requirements	*String*	*Pre-execution Command Name*	*String*
Requested Number of Slots	*Integer*	*Job Group Name*	*String*
User Login Shell	*String*	*User Specified Memory Reservation Value*	*Integer*
Job Submission Working Directory	*String*	*Job Description*	*String*
Advanced Reservation Name	*String*	*Array Job or Single Job*	*Boolean*
License Project Name	*String*	*Post-execution Command Name*	*String*
Job Submission Day of Week	*Integer*	*Job Submission Hour of Day*	*Integer*
Job Options	*Integer*	*User Specified Runtime Limit or Estimation*	*Integer*

3.3 Abnormal Job Removal

Jobs could be terminated from the system before normal completion. For example, user or administrator may explicitly kill the job submitted with wrong options. Also, application may run into errors and terminate itself with certain exit codes. Those incomplete jobs surely become the noise points when training model. We remove the jobs with following abnormal cases from our job traces.

- Jobs with zero or minus memory usage
- Jobs with zero or minus runtimes
- Jobs finished with exit status or exit codes

Besides, we follow other standard feature engineering steps [18] to further refine the features, such as feature encoding (e.g. mapping the value of string type fields as integer values) and normalization.

4 Resource Prediction Method

In this section, we present the design of the proposed resource prediction method for large memory jobs. The key idea of our method is to separate the mode training as two stages: firstly, a binary classification model is built to determine whether a job will consume large or small memory; then a regression model is trained by just using the

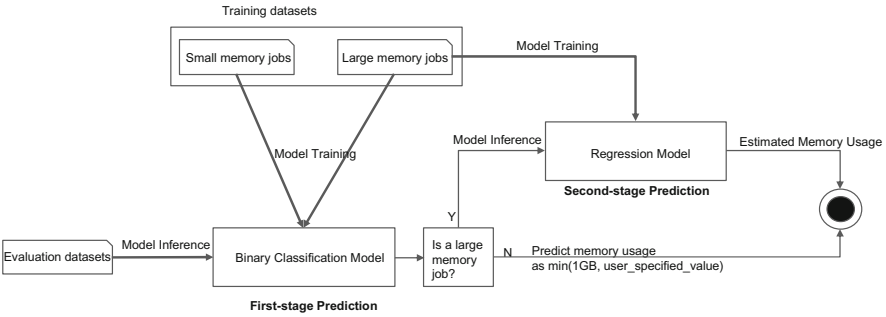

Fig. 3. Proposed prediction flow for job memory usage

historical large memory jobs to predict the final memory usage for the jobs which are predicted to use large memory. Figure 3 describes the overall prediction flow with the above method.

4.1 Predicting Job Memory Usage Type

Predicting the correct job memory usage is not easy, but the complexity of predicting whether a job tend to belong a large or small memory one can be significantly reduced. The latter one becomes a binary classification problem, while the former one is a multi-class classification problem or a regression problem. We treat the job with memory usage larger than 1 GB as large memory one in this paper. However, this boundary could be dynamically changed in production system based on the practical workload needs.

As shown in Fig. 3, the prediction accuracy of this binary classification model needs to be very high to reduce the number of wrongly predicted large memory jobs got requeued at runtimes. Besides adopting biased sampling to increase the number of large memory jobs in the training datasets, it is important to extract the user specified memory usage as an input feature. Though people tend to specify much bigger memory value for their jobs, this value gives some clues on whether a job will use large memory or not. For example, user may specify always 4 GB as the reserved memory size for their small memory jobs, while specify a few hundred GB of memory size for large memory jobs. Combining this value with the real job memory usage, the model could possibly infer another job from the same user with 4 GB memory requirements as a small memory job.

While there are standard metrics (e.g. precision, recall) to evaluate the classification performance, we define two evaluation metrics to better correlate with the context of our work. Specifically, we define ***coverage rate (CR)*** and ***incorrect coverage rate (ICR)*** to separately quantify the percentage of large memory jobs which got correct prediction (*Hit_LMEM_Jobs*), and the percentage of small memory jobs which are wrongly classified as large memory ones (*Miss_SMEM_Jobs*). The calculation formulas are as follows. Therefore, the prediction target is to achieve higher coverage rate while keeps incorrect coverage rate as lower as possible.

$$CR = \#Hit\_LMEM\_Jobs / \#Total\_LMEM\_Jobs \quad (2)$$

$$ICR = \#Miss\_SMEM\_Jobs / \#Total\_SMEM\_Jobs \quad (3)$$

The classification version of random forester method is chosen as the binary classification algorithm to train the first stage model. We also did basic hyper-parameter search using the public random search method provided by *scikit-learn* toolkits [19].

4.2 Predicting Large Job Memory Usage

As long as the large memory jobs have been identified with the first stage model, it becomes straightforward to train another model by just using the historical large memory jobs. The results given by Fig. 2 show that such model can remove the noises of small memory jobs and get better prediction accuracy. We use the regression version of random forest algorithm in *scikit-learn* toolkits to train the large memory model.

Meanwhile, as the number of jobs in the training datasets is reduced, the model training time can also be significantly shortened. Though now we need to build two models, the training time of the first stage model is much shorter than training a regression or multi-class classification model using all large and small jobs. The total time of training two models is also shorter than the one of training a single model.

Furthermore, as there is no dependency between the models in two stages, model training can be concurrently started with additional hardware resources. Then the training time can be further reduced by hiding the latency. We will show more comparison results in the following evaluation section.

4.3 Tolerate Incorrect Predictions

Prediction always have errors and the more important thing is how to minimize the impact of incorrect predictions. Specific to our predictions, we have the following prediction errors during the two different stages.

- *Predict a large memory job as using small amount of memory*: this job will not get memory usage prediction, and resource manager will reserve memory if user specified memory requirement. Otherwise, the job will compete memory resource with other jobs at the run time. The former one may waste memory but can be acceptable if the percentage of missing rate is low. The latter one may encounter memory failures, and can be requeued to run again depending on resource manager configurations.
- *Predict a small memory job as using large amount of memory*: this job will get a memory size which is larger than its consumption. As this kind of jobs tend to have short duration, there will be not much impact on overall memory utilization.
- *Memory prediction is under-estimated by the second stage model*: in practical, administrator can round up the predicted value to the memory bucket upper value. Then the impact introduced by this kind of under-estimation errors can be reduced.

4.4 Model Inference for New Jobs

Finally, the trained model will be used for predicting job memory usage for new coming jobs. Compared with single model approach, our method requires up to two inference operations to get final memory predictions. However, the time granularity of job submission latency is quite larger than model inference latency. It has little impact on the job submission throughput. In addition, one can also perform inference from the two models at the same time with certain probability. Then the inference latency can be hidden as one longest latency of the two models.

5 Evaluation Results and Analysis

In this section, we introduce the evaluation details of proposed method using the traces we described in Sect. 3. We will first describe the testing environments and strategy, then prove the benefits of our prediction method in terms of prediction accuracy, model training and inference efficiency.

5.1 Experimental Setup

IBM Spectrum LSF Predictor [21] is a tool to predict job resource metrics using LSF job traces. We used this tool to implement and evaluate proposed prediction method for large memory jobs in this paper. The tool is running as a Docker container and includes the steps of a complete machine learning flow. The testbed is a X86_64 machine with 44 Intel Xeon E5 CPUs, and 64 GB physical memory.

We used three real world job traces collected from IBM Spectrum LSF customers for the evaluation tests. Besides the job memory size statistics information in Sect. 2, more statistics information is listed in Table 3. Note we used biased sampling for large

Table 3. Statistics of job traces used for evaluation tests.

Metrics	Trace A	Trace B	Trace C
Time periods of job traces (days)	36	4	4
Num. of total jobs	587k	907k	1m
Num. of active users	1270	3085	506
Num. of projects	5589	278	23
Num. of jobs in training/evaluation datasets	469.6k/117.4k	700k/207k	750k/251.5k
Num. of large/small memory jobs in training dataset	175.3k/294.3k	538k/162k	428.8k/321.2k
Num. of large/small memory jobs in evaluation dataset	43.7k/73.7k	47k/115k	183.6/137.6k
Avg. large/small job memory usage (GB)	28.83/0.155	7.46/0.29	2.54/0.43
Avg. runtimes of large/small memory jobs (mins)	175.56/16.6	92/12	221/103

memory jobs to generate Trace A from the original traces, which can retain the job patterns over long periods.

We split roughly 80% of jobs as training datasets, while use the left jobs as inference datasets. The evaluation experiments do comparison tests between single model method and the two-stage model proposed in this paper. To make fair comparisons, we used the same hyper-parameter settings to predict the memory usage for both two kinds of prediction methods. Table 4 shows the configurations for the Random Forest algorithms used by the single model and our two-stages model approaches. To accelerate the model training, we configure *n_jobs* as 10 to run training in parallel across multiple CPUs. The following sections will give the detail analysis of comparisons in different metrics.

Table 4. Hyper-parameter settings for model trainings.

Hyper-parameter	Single model	1st stage model	2nd stage model
n_estimators	100	50	100
n_jobs	10	10	10
max_depth	Auto	Auto	Auto
random_state	2	2	2
max_features	Auto	Auto	Auto

5.2 Prediction Accuracy and Efficiency

This section first analyzes the prediction accuracy of first stage model in distinguishing large and small memory jobs. Then we evaluate the prediction errors of job memory usage for large memory jobs using the second stage model.

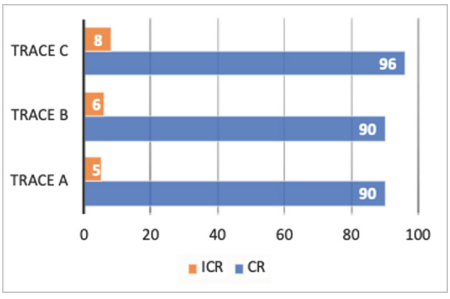

Fig. 4. Coverage rate (CR) and Incorrect coverage rate (ICR) of three job traces

Figure 4 shows the *CR* and *ICR* of three tested job traces produced by random forest binary classification model. We can see over 90% of large memory jobs can get correct prediction for their memory usage types, while only less than 8% of small

memory jobs are wrongly predicted. Trace C even can correctly predict 90% of large memory jobs. This result provides a good basis for the second stage prediction, which finally predict the memory usage for the large memory jobs recognized by the binary classification model in the first stage.

For those wrongly predicted large memory jobs, it is possible that those jobs may fail to get enough memory during runtime. Jobs may suffer from the overhead of memory swapping or even encounter execution failure. On one hand, the prediction quality of classification model can be further tuned to get better prediction accuracy. Note that we don't perform too much tuning efforts on the feature engineering and model hyper-parameter tuning in this paper. Therefore, it can be expected that the CR value can possibly be further increased, while the ICR value can be reduced to filter more small memory jobs from the second prediction stage. One the other hand, resource managers can make smart allocation adjustments once a job exceeds its predicted memory size. For example, the job can be checkpointed and migrated to other hosts or simply requeued to be rescheduled with the user specified memory size. As we can achieve pretty higher classification predictions, the overall memory utilizations still can be significantly improved even small number of jobs needs to be restarted.

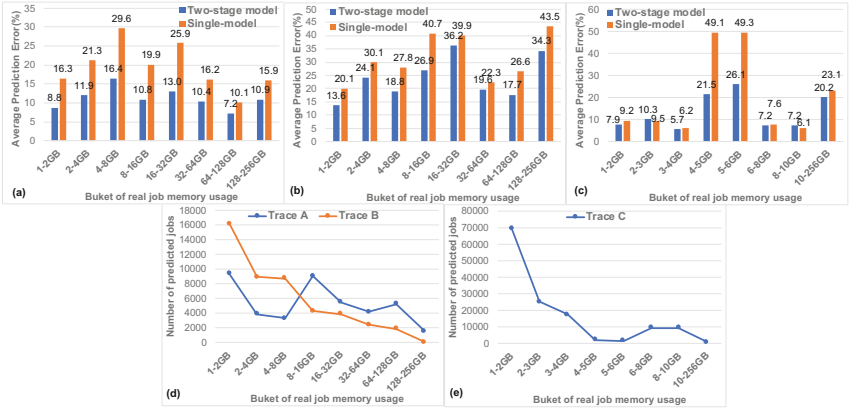

Fig. 5. Comparisons of prediction accuracy between single model (baseline) and two-stage model (proposed method): (a) Trace A; (b) Trace B; (c) Trace C, and distribution of predicted jobs by job memory usage buckets: (d) Trace A&B; (e) Trace C

Figure 5(a–c) shows the prediction accuracy comparisons between existing single model approach and proposed two-stages prediction method. To fairly evaluate the improvements, the figures draw the calculated average prediction errors of each pre-defined memory usage buckets for the large memory jobs recognized by the first-stage model. The prediction error numbers are marked on the top of each bar in the figure. We try to split the job memory usage buckets with roughly balanced number of jobs in each bucket. So the bucket distribution in Trace C is different with the other two traces as most of its jobs have memory usage less than 16 GB.

Figure 5(d) and (e) depicts the number of jobs in each bucket for the three traces. From the results, we can see obvious improvements on prediction errors for all of the three traces with the proposed method. The average percentages of improvements for Trace A, B and C are 40.7, 24.3 and 14.5 respectively. The results prove the advantages of removing the noises of small memory jobs from the training model. Although there are a little bit increments of prediction errors for the buckets 2–3 GB and 8–10 GB of Trace C, the prediction error numbers are less than 10% and can be tolerated in practice. It might be related the particular unknown job patterns in those two buckets, which can be further investigated. We don't see this phenomenon in other two traces.

Furthermore, the predictions improvements compared with single model approach can save corresponding percentage of bucket to use cloud resources when moving those workloads to public cloud. Take the jobs of Trace A with 1–2 GB requirements as an example, the average improvements of prediction errors can be reduced by nearly 50%. That means the cluster must apply a VM with 2 GB memory (e.g. *t3.small* from AWS [24]) to run the job, but now only needs to provision a VM with 1 GB memory (e.g. *t3.micro*) for the job. As the price shown by AWS, the cost reduces from $0.0208 per hour to $0.0104 per hour. As shown by Fig. 1, user specified errors of memory requirements could be hundreds to thousands of times compared with real usage. The proposed approach in this paper can significantly reduce the budget of moving workloads to cloud in large scale clusters.

In summary, the proposed two-stages prediction method is much better than building a single model to predict memory usage for large memory jobs. As the first stage model can get good coverage rate for identifying large memory jobs, it is practical and effective to use the second model with only historical large memory jobs for final prediction.

5.3 Model Training Cost Analysis

Figure 6 shows the evaluation results of model training efficiency. As there are two models needed to be built in our proposed method, the left figure gives the distribution of training time of the two models, and the training time numbers (seconds) are marked in the corresponding bars. Also, the green bars in the figure show the training time numbers of building a single model. We can see the cost of building the first stage binary classification model is much lower than building a regression model using the same datasets. Meanwhile, as the second stage model only needs to train the large memory jobs, the model building time can also be significantly reduced. That proves that the total training cost can be reduced even we split the model training into two stages. This observation can be further confirmed by the improvement percentage of training time compared with the training time of single model in Fig. 6(b).

As we introduced in Sect. 4.2, the two stage models have no building dependency and can be trained concurrently with additional computing resources. Figure 6(b) shows the improvements of training the two stage models in serial or parallel. The percentage numbers are marked above the corresponding bars. For both two cases, the proposed method requires less training cost to build those models. As the problem complexity is increased by the number of jobs in training datasets, we believe the

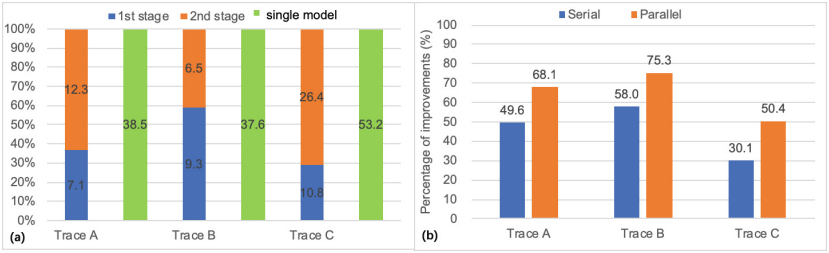

Fig. 6. Comparisons of model training cost: (a) training time distributions of two stages models and single model; (b) percentage of training time improvements compared between single model training (baseline) and two stages models training (including both total training time of running two models sequentially and in parallel)

improvements can be further enlarged when training large models containing long periods of historical job records.

5.4 Model Inference Performance

One potential side effect of using two-stage models is the inference latency may be increased compared with predicting using a single model. Tables 5 and 6 show the statistics of model inference latency and total inference time. We can see the inference latency from the second stage model is increased. The reason should be that there are more computing efforts of fitting a large memory value. However, as the inference latency is only microseconds level, it is pretty little impact on the job submission performance in practical. Generally, submitting a job to resource manager requires setup connections to remote master daemon, process requests and create persistence

Table 5. Comparisons of model inference latencies

Trace Name	Avg. model inference latency (microseconds)		
	1st stage	2nd stage	Single model
Trace A	2.38	7.28	4.88
Trace B	1.57	7.31	2.87
Trace C	1.76	4.49	3.04

Table 6. Comparisons of total time of model inference for jobs in the evaluation datasets

Trace Name	Total time of model inference (milliseconds)			Inference delay (percent)
	1st stage	2nd stage	Single model	
Trace A	279.6	318.2	597.8	4.38
Trace B	325.3	343.7	595.9	12.3
Trace C	442.8	618.2	765.7	38.6

entry by resource managers. So, the job submission latency of a single job is around several or dozens of milliseconds. Compared with the network and resource manager processing delay, the inference latency of predicting job resource usage can be totally ignored. In addition, there are lots of work on using GPUs to do model inference, which can significantly accelerate the floating computing operations [22, 23].

6 Related Work

Using machine learning algorithms to predict job resource usage is one of the hottest topics in recent years, and a lot of research work has been done on not only memory usage prediction, but also CPU usage, job wall time predictions, etc.

Many works have been done on directly predicting the value of job resource usage [4, 5]. The predicted value then can be integrated by resource managers to help users or administrators to adjust incorrect specified usage value. Taghavi [4] et al. from Qualcomm introduced their work on predicting job memory usage using various machine learning algorithms and tools. Their work shows the memory usage of prior jobs can be good guess for next jobs, and a simple linear regression model can be used for prediction. Another IBM research team [5] performed similar work on using various machine learning algorithms (e.g. k-NN, Random forest, SVM) to do memory usage prediction using LSF job trace records. Those work provides good basis for our study in this paper, and proves the feasibility of using machine learning models to predict further job memory usage. However, instead of building a single model using all of collected data, we target to only predict job memory usage more accurately for large memory jobs using two-stage prediction method. Administrator don't put too much attention on small memory usage due to the fact of its short job duration and massive memory on modern computing nodes. Therefore, our work is more practical for production usage and promote the prediction accuracy for those jobs really cared by resource manager users.

Besides memory usage prediction, a lot of other researchers focus on predicting more other job related metrics, such as job runtimes [6, 7, 10, 11], job queue-waiting time [12, 13], job start times [20], job completion time [14], power usage [16, 17], etc. Instead of using job submission records, some of the work are directly using the recent previous runs of the same applications to calculate the runtimes for next jobs, which limited to certain suitable scenarios. While our work focuses on predicting memory usage for large memory jobs, it could potentially be extended to more broader scenarios. For example, people may also care about the resource usage of long running jobs. Then it is also applicable to do similar two-stage prediction models for better prediction accuracy.

Differ with predicting the usage value, there are lots of work investigating how to binary classification can do some correctness judgements in cluster or job usage. Guo et al. [8] proposed to predict whether user specified runtime limits are under-estimated compared with actual job runtimes. Then user can be notified to correct their job limit and ensure job not being terminated unexpectedly. Andresen et al. [9] used classification algorithms to do similar predictions on whether a job will fail due to resource shortage. Their work also performed regression models to predict memory usage and

job runtimes. One innovation of their approach is that they introduced additional per-user features like average memory usage, requested memory and runtimes to train the models. In our paper, we also use binary classification models to identify the large memory jobs in the first stage prediction model.

7 Conclusions and Future Work

Resource managers can perform better job scheduling decisions and overall cluster resource utilization with more accurate job resource requirements. In hybrid cloud environments, accurate memory predictions can also save lots of budget to burst workloads to cloud, which is one of popular use patterns adopted by HPC sites. We analyzed the real customer scenarios and found that administrators have more emphasis on the memory usage of large memory jobs. Instead of building a single model using all of historical jobs, we conducted evaluation tests using real world customer job traces to demonstrate that a two-stage prediction approach can remove the noise of small memory jobs and promote the prediction accuracy of large memory jobs with a high coverage rate. The proposed prediction method is suitable for the practical usage to predict job memory usage in production systems.

In the future, we plan to collect more large-scale job traces to further evaluate the benefits on prediction quality and performance. Also, we believe the job memory usage prediction of large memory jobs is only one of the practical scenarios to adopt the two-stage prediction method. In the next steps, we will evaluate whether it is useful to have better prediction for the job runtimes of long running jobs in HPC clusters.

References

1. IBM Spectrum LSF. www.ibm.com/Storage/LSF. Accessed 01 Jan 2019
2. Slurm Workload Manager. https://slurm.schedmd.com/. Accessed 01 Jan 2019
3. Moab Cloud HPC Suite. www.adaptivecomputing.com/moab-hpc-basic-edition/. Accessed 01 Jan 2019
4. Taghavi, T., Lupetini, M., Kretchmer, Y.: Compute job memory recommender system using machine learning. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2016), pp. 609–616. ACM, New York (2016)
5. Rodrigues, E.R., Cunha, R.L., Netto, M.A., Spriggs, M.: Helping HPC users specify job memory requirements via machine learning. In: Proceedings of the Third International Workshop on HPC User Support Tools, pp. 6–13. IEEE Press (2016)
6. Yang, L.T., Ma, X., Mueller, F.: Cross-platform performance prediction of parallel applications using partial execution. In: Proceedings of the 2005 ACM/IEEE Conference on Supercomputing (2005)
7. 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)
8. 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

9. Andresen, D., Hsu, W., Yang, H., Okanlawon, A.: Machine learning for predictive analytics of compute cluster jobs. In: The 16th International Conference on Scientific Computing (2018, accepted)
10. Carlos, F.G.: Improving HPC applications scheduling with predictions based on automatically-collected historical data. Master thesis, UPC (2014)
11. Matsunaga, A., Fortes, J.A.: 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, pp. 495–504. IEEE Computer Society (2010)
12. Carvalho, A., Belo, O.: Predicting waiting time in customer queuing systems. In: 2016 IEEE International Conference on Knowledge Engineering and Applications (ICKEA), pp. 155–159. IEEE Computer Society (2016)
13. Smith, W., Taylor, V., Foster, I.: Using run-time predictions to estimate queue wait times and improve scheduler performance. In: Feitelson, D.G., Rudolph, L. (eds.) JSSPP 1999. LNCS, vol. 1659, pp. 202–219. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-47954-6_11
14. Chen, X., Lu, C.-D., et al.: Predicting job completion times using system logs in supercomputing clusters. In: DSN Workshops. IEEE Computer Society (2013)
15. Storlie, C., Sexton, J., Pakin, S., et al.: AI Modeling and predicting power consumption of high performance computing jobs. preprint arXiv:1412.5247 (2014)
16. Borghesi, A., Bartolini, A., Lombardi, M., Milano, M., Benini, L.: Predictive modeling for job power consumption in HPC systems. In: Kunkel, J.M., Balaji, P., Dongarra, J. (eds.) ISC High Performance 2016. LNCS, vol. 9697, pp. 181–199. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-41321-1_10
17. Author, F., Author, S.: Title of a proceedings paper. In: Editor, F., Editor, S. (eds.) CONFERENCE 2016. LNCS, vol. 9999, pp. 1–13. Springer, Heidelberg (2016)
18. Zheng, A., Casari, A.: Feature Engineering for Machine Learning - Principles and Techniques for Data Scientists. O'Reilly Media, Sebastopol (2018)
19. scikit-learn: machine learning in Python. https://scikit-learn.org. Accessed 10 Nov 2018
20. Li, H., Groep, D.L., et al.: Predicting job start times on clusters. In: IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid), pp. 301–308. IEEE Computer Society (2004)
21. A Crystal Ball for HPC. https://www.hpcwire.com/solution_content/ibm/cross-industry/a-crystal-ball-for-hpc. Accessed 10 Dec 2018
22. GPU-Based Deep Learning Inference. https://www.nvidia.com/content/tegra/embedded-systems/pdf/jetson_tx1_whitepaper.pdf. Accessed 13 Oct 2018
23. Gardner, J.R., Pleiss, G., Bindel, D., et al.: GPyTorch: Blackbox Matrix-Matrix Gaussian Process Inference with GPU Acceleration. NeurIPS 2018. preprint arXiv:1809.11165 (2018)
24. Amazon EC2 Pricing. https://aws.amazon.com/ec2/pricing/on-demand/. Accessed 07 Feb 2019
25. IBM Spectrum LSF Hybrid Cloud. https://github.com/IBMSpectrumComputing/lsf-hybrid-cloud. Accessed 07 Feb 2019