Online Runtime Prediction Method for Distributed Iterative Jobs

Xiaofei Yue[1], Lan Shi[1], Yuhai Zhao[1(✉)], Hangxu Ji[1], and Guoren Wang[2]

[1] School of Computer Science and Engineering, Northeastern University, Shenyang 110819, China
zhaoyuhai@mail.neu.edu.cn

[2] School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China

Abstract. Predicting the runtime of distributed iterative jobs can help reduce the deployment cost of clusters and optimize their resource allocation and scheduling strategies, but the runtime depends on various factors which are difficult to be acquired before execution. In this paper, we propose a generalized online prediction method for the runtime of distributed iterative jobs, which is centered on a series of online machine learning models. The method consists of three phases: 1) estimating the number of iterations for the current iterative job. 2) predicting the runtime metrics of each iteration by an online polynomial regression model. 3) Runtime metrics sequence is analyzed using an LSTM trained with online learning to predict the runtime of each iteration. We conducted experiments on typical Flink iterative jobs, and the experimental results show that our method improves the accuracy by 4.79% compared to the state-of-the-art methods, while for the improvement in accuracy for delta iterative jobs is even more than 15%.

Keywords: Iterative job · Online runtime prediction · LSTM · Polynomial regression · Flink

1 Introduction

In the past few years, we have seen a rapid growth of large-scale analytic jobs, especially in the field of artificial intelligence, such as in the direction of distributed image recognition [1], etc. Moreover, the core algorithms of these operations are often iterative, i.e., one or more steps are executed repeatedly until convergence criteria are met. Current big data computing platforms like Apache Flink [2] allow users to perform distributed iterative computation on clusters to analyze large-scale dataset.

Most of the distributed iterative jobs need to comply with service level objectives (SLO) in real production [3], because external applications usually need timely feedback of analysis results, too high execution latency can violate service level agreements (SLA) with users [4]. However, in poor cluster environments, even using all the remaining computational resources can cause the job to take

C. Xing et al. (Eds.): WISA 2021, LNCS 12999, pp. 156–168, 2021.
https://doi.org/10.1007/978-3-030-87571-8_14

much longer to complete than the user expects, so runtime prediction is a very useful mechanism for solving the job feasibility analysis problem. In addition, based on the predicted runtime, the allocation of cluster resources and deployment costs can be rationalized and optimized [5–7]. Unfortunately, the runtime of iterative jobs depends on factors such as dataset characteristics, system configuration and algorithm parameters, there are also involve inter-iteration dependencies for delta iterative jobs, these information is difficult to obtain before job execution. Previous works tend to be a single model designed for a specific distributed framework and requires specialized isolated training that does not meet the user's needs for response time and prediction accuracy [8–10].

In this paper, we proposes a method for predicting the runtime of iterative jobs in distributed environment, which centers on fine-grained prediction using a series of online machine learning models. For distributed frameworks that support resource monitoring and pluggable components, our online model is generic, and once the model is built, it only needs to be fine-tuned to adapt to changes in the dataset and cluster.

The main contributions of this work can be summarized as follows:

- A generic approach to online predict the runtime of iterative jobs is proposed, which performs fast and fine-grained prediction through multiple phases.
- The runtime prediction of iterative jobs is enhanced by monitoring fine-grained resource consumption data and constructing runtime metrics sequences based on the order of iteration steps.
- The accuracy of the proposed method is validated by performing experiments on four typical Flink iterative jobs.

The rest of the paper is organized as follows. Section 2 presents the background on which our work is based. Section 3 elaborates our runtime prediction method. Section 4 validates the effectiveness of our method and analyzes the experimental results. Section 5 presents related work, and finally, Sect. 6 describes the conclusions and future research directions.

2 Preliminaries

2.1 Problem Definition

Definition 1 (Container). In resource management systems such as YARN [11], containers are requested by different frameworks as the basic unit of resource allocation and applied to the distributed jobs, and basic computing resources such as CPU cores, memory are encapsulated in containers, and the monitoring of system resources in this paper is carried out through containers.

Definition 2 (Runtime Prediction Model). In a distributed iterative job, assuming rsc is the current input and cluster resource state, $x^{(i)}(rsc)$ denotes the execution characteristics of the i-th iteration of the iterative job in that

state, and $y^{(i)}$ is the time consumed by that iteration, then building a runtime prediction model is to find the functional relationship as shown below.

$$runtime = \sum_i f : rsc \rightarrow x^{(i)}\,(rsc) \rightarrow y^{(i)} \tag{1}$$

An iterative job consists of a series of steps $S = (S_1, \cdots, S_t, S_{t+1}, \cdots S_n)$ with certain dependencies between adjacent iterative steps S_t and S_{t+1}, and the execution of S_{t+1} will start only after the execution of S_t is completed. Our runtime prediction model does its work by analyzing multiple sequences consisting of execution features from iterative steps with absolute sequential order that from the resource management platform's monitoring of the consumption of resource utilization in the container assigned to the iterative job.

2.2 Background

Distributed Iterative Computation. There are many distributed systems that support users to build iterative programs using the provided iterative operators. The algorithmic logic of an iterative program is mainly composed of Multiple operators combined in different degrees of parallelism.

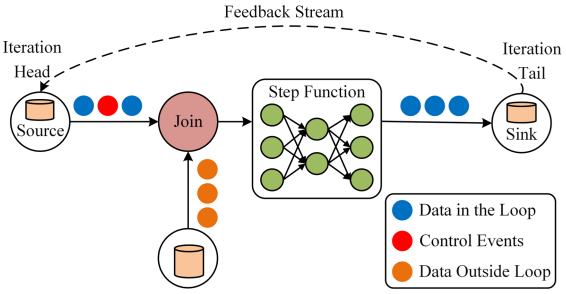

Fig. 1. The structure of iterative computing model in Apache Flink.

The Flink supports iterative computation by defining step function and embedding it in a special operator. As shown in Fig. 1, to maintain the DAG-based runtime and scheduler, Flink constructs iterative channels by creating iteration "head" and "tail" tasks and interconnected feedback edges [2].

Online Machine Learning. Online machine learning has gained wide attention from academia and industry [12]. Because it can stream training samples and incrementally update models, under the problem explored in this paper, the use of online learning methods can quickly incorporate data generated by periodically operating iterative jobs into the learning of models, and then give a near real-time [13]. Therefore, the online learning model is useful for creating a portable and general-purpose runtime prediction system.

3 Approach

In this section, we present an experimental method of online runtime prediction for distributed iterative jobs, as shown in Fig. 2, which consists of the following three key steps: 1) collecting iterative jobs and data required for training the model, 2) estimating the number of iterations for the current job, and 3) identifying the key features and building an online prediction model.

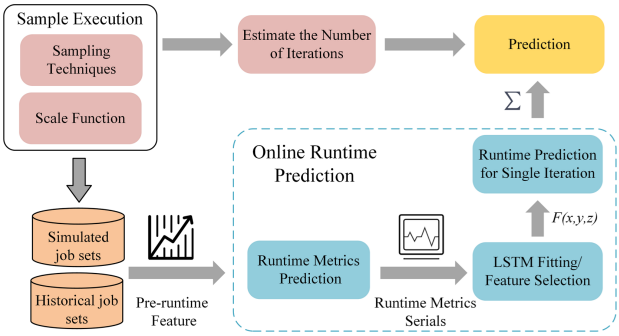

Fig. 2. The process of online runtime prediction for iterative jobs.

3.1 Data Collection

The iterative job sets collected for subsequent training of the runtime prediction model consist of two main types.

1) historical jobset. Since iterative jobs are typically executed periodically to analyse batches of data, we mainly use the relevant statistics and execution times of historical iterative jobs as training data.
2) Simulated jobset. For the case where historical data is scarce, we use sample execution to quickly generate simulated data, i.e., different iterative jobs are run on sample dataset generated with different sampling ratios.

Sampling Technique. Specifically, the convergence of the graph iteration algorithm is essentially determined by the vertices with higher out-degree, so we use the biased random jump (BRJ) sampling method proposed by Daniel et al. [9] which is more biased towards the vertices with higher out-degree when performing random wandering to preserve the key properties of the original graph.

3.2 Key Input Features

As shown in Tables 1 and 2, these key features are classified into two categories, namely, pre-runtime features and runtime features.

Pre-runtime features can be obtained statically before the execution of iterative jobs, and include job type, dataset size, and system configuration, which

reflect the basic properties of the job and cluster environment. Runtime features, as the name implies, can only be determined when the iterative jobs are actually running, and they can be used to directly reflect the performance differences of the cluster environment during the iterative process and the dependencies between iterations, including the fine-grained resource usage of each iteration.

Table 1. Description of the pre-runtime features of an iterative job.

Type	Name	Description
Iterative job	id	Execution ID of the iterative job
	input	Relative size of the input dataset for the iterative job
Cluster status	memory	Available memory capacity of the cluster
	disk	Available disk capacity for the cluster
	vcpu	Number of CPU cores in the cluster
Time point	startTime	Start time of iterative job execution
	endTime	End time of iterative job execution

Table 2. Description of the runtime features of an iteration.

Resource	Name	Description
CPU	procs	Number of processes during this iteration
	uTime	Time spent in user mode of the CPU during this iteration
	threads	Number of threads during this iteration
	kTime	Time spent in kernel mode of the CPU during this iteration
Memory	rsSize	The size of the resident memory occupied during the iteration
	vmSize	The size of the virtual memory occupied during the iteration
IO	ioWait	Time spent waiting on IO during this iteration
	rChar	Number of bytes read using read-like syscall during this iteration
	rBytes	Number of bytes read from disk during this iteration
	wChar	Number of bytes written using write-like syscall during this iteration
	wBytes	Number of bytes written to disk during this iteration

3.3 Estimate the Number of Iterations

Our method for estimating the number of iterations relies on the sample execution described in Sect. 3.1.

Scale Function. For iterative algorithms where the convergence threshold changes according to the size of the dataset, let sr is the sampling ratio, the scale function T has the following form:

$$T = (Conf_S = Conf_G, Conv_S = Conv_G/sr) \tag{2}$$

where $Conf_S => Conf_G$ and $Conv_S => Conv_G$ denote the mapping of configuration parameters and convergence parameters.

By combining the sampling technique and scale function, it is possible to make the sample execution and the real execution invariant in terms of the number of iterations, thus perform the estimation of the number of iterations.

Example. PageRank is a classical iterative algorithm and converges when the change in the result of two adjacent iterations is less than a threshold τ. For example, the graph G shown in Fig. 3 and its three arbitrary samples S1-S3 (with a sampling ratio of 50%), samples S1 and S2 are sampled using BRJ, preserving the diameter $D = 2$ of the original graph G. However, the opposite is true for sample S3, but none of the above samples maintains the invariance of the number of iterations. Because after the first iteration, the average delata of their PageRank values change as: $\bar{\Delta}(\text{s1}, 1) = d/8$, $\bar{\Delta}(\text{s2}, 1) = d/8$, $\bar{\Delta}(\text{s3}, 1) = 3d/16$, $\bar{\Delta}(G, 1) = d/16$, where $d = 0.85$ is the damping factor. Assuming the threshold $\tau = d/16$, then the original graph G will converge after one execution, while the rest of the samples will continue to execute, but by applying the scale function $T = (Conf_S = Conf_G, Conv_S = Conv_G * 2)$ on samples S1 and S2 the corresponding convergence threshold can be adjusted to $d/16$, thus maintaining the same number of convergence as the graph G.

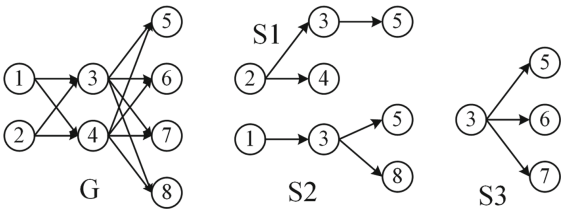

Fig. 3. Maintaining invariants for the number of iterations on sample graphs.

3.4 Online Runtime Prediction Model

The values of these runtime features cannot be determined before the execution, so a two-stage online machine learning model is built in this section, and the specific process is shown in Algorithm 1.

In the first stage, a model is generated for each runtime feature by iteration round, and the purpose of these models is to learn the impact of different inputs and cluster environments on the iterative process, and in the second stage, the final prediction of the runtime of each iteration step is made based on the runtime metric model generated in the first stage.

Algorithm 1: Online Runtime Prediction Algorithm

Input: J (Iterative job), S (Available resources), P (Input data scale)
Output: α (Runtime prediction value for Job J with resource S and input P)

$\tilde{N} \leftarrow IterNumEstimate\,(J)$; // Estimate number of iterations
$\Phi \leftarrow \emptyset$; // Initialize the set of pre-runtime features
for i=1:n **do**
  $\Phi \leftarrow \Phi \cup \Gamma_i\,(J, S, P)$; // Collect the pre-runtime features
end
for i=1:$\tilde{N}$ **do**
  if $isEmpty(r_i(J, S, P))$ **then**
    $R \leftarrow R \cup runtimeFeaturePredict\,(\Phi, i)$;
  else
    $R \leftarrow R \cup \mathrm{r}_i\,(J, S, P)$; ; // Collect the runtime features
  end
end
Rh $\leftarrow serialize\ (R) = (r_1, \cdots, r_{\tilde{N}})$; // Build the runtime metrics sequence
$\alpha \leftarrow 0$
for i=1:$\tilde{N}$ **do**
  if $i < now$ **then**
    $\hat{t_i} = realTime(J, i)$; ; // This iteration has been executed
  else
    $\hat{t_i} = LSTM(R, i)$; ; // Predict the runtime of every iteration
  end
  $\alpha += t_i$;
end
return α

First Stage. Let $P = \{P_1, ..., P_n\}$ be the set of pre-runtime features, and for each pre-runtime feature P_i, $1 \le i \le n$, let Φ_i denote the set of possible values of that feature in the set of training jobs, then Φ is uniquely determined for each iterative job that has completed or is about to run. Therefore, we can assume that there exists a function Γ_i to represent the fetch values of the pre-runtime feature P_i for some iterative job in some cluster environment.

$$\Gamma_i : Job \times Input \times Resource \rightarrow \Phi_i, \quad \forall 1 \le i \le n \tag{3}$$

Then an iterative job can be represented by a vector $P \in \mathbb{R}^n$, where n denotes the number of pre-runtime features. For each of the values of the runtime features shown in Table 2, we make predictions by building an online linear regression model with L2 regularization.

Specifically, considering the dependencies among the runtime features, we use a modified quadratic polynomial regression model (QPR) with a regression function of the form:

$$f\,(w, P) = w^T K\,(P)\,, \quad w \in \mathbb{R}^{1+2n+\binom{n}{2}} \tag{4}$$

where w is the weight vector to be learned and $K(\bullet)$ is the basis function of the polynomial regression model:

$$K(P) = \left(1, p_1, \cdots, p_n, p_1 p_2, \cdots, p_{n-1} p_n, p_1^2, \cdots, p_n^2\right)^T \tag{5}$$

Let $r_i^{(j)}$ denote the true value of the runtime feature x_j of the iterative job i, then the final form of the online regression model on the runtime feature x_j is

$$\underset{w}{argmin} \sum_{\mathrm{i}=1}^{N} \eta\left(w^T K(P_i), \mathrm{r}_i^{(j)}\right) + \lambda \|w\|_2^2 \tag{6}$$

where $\eta\left(w^T K(P_i), \mathrm{r}_i^{(j)}\right)$ is the loss function and λ is the L2 regularization parameter. We choose the mean square error (MSE) as the loss function, so once online regression model finds the w that minimizes the current cumulative loss, the runtime features for each iteration can be predicted by Equation (4).

Second Stage. After the first stage, we can obtain a set of sequences of runtime indicators in the order of iterations. Based on the above considerations, the long and short term memory network (LSTM) becomes a suitable method to predict the runtime of each iteration based on runtime features, and the LSTM implemented in Keras supports online learning, and when the value of the *batchSize* parameter is 1, it means that an online learning method is used.

The runtime feature records $R_t = \left\{r_t^{(1)}, \cdots, r_t^{(n)} | t \in \left[1, \tilde{N}\right]\right\}$ for each iteration can be obtained, where $\tilde{N}$ denotes the predicted value of the number of iterations of the current iterative job. These records are combined in the order of iterations to obtain a runtime metrics sequence $Rh_{current} = (R_1, ..., R_t, R_{t+1}, ..., R_{\tilde{N}})$. In particular, we assume that the iteration counts of training job set and current iterative job are same or similar. If the obtained sequence length is greater than $\tilde{N}$, the sequence is intercepted from the end of the sequence to the same length, and if the sequence length is insufficient, the end of the sequence is filled with zeros. Then we can obtain the matrix Rh consisting of the runtime metrics sequences of all iterative jobs in the training job set as shown in Fig. 4.

$$Rh = \begin{bmatrix} R_{1,1} & R_{1,2} & \cdots & R_{1,\tilde{N}} \\ R_{2,1} & R_{2,2} & \cdots & R_{2,\tilde{N}} \\ \vdots & \vdots & & \vdots \\ R_{M,1} & R_{M,1} & \cdots & R_{M,\tilde{N}} \end{bmatrix} \begin{matrix} \rightarrow Rh^{(1)} \\ \rightarrow Rh^{(2)} \\ \vdots \\ \rightarrow Rh^{(M)} \end{matrix}$$

Fig. 4. Overview of the matrix composed of runtime metrics sequence.

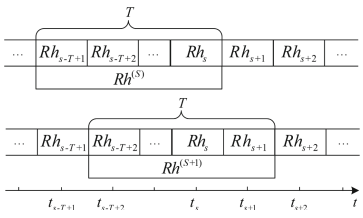

Fig. 5. Schematic diagram of sliding window input.

The model inputs are updated by means of a sliding window, as shown in Fig. 5. The LSTM is used to learn a sequence of runtime features for each iteration step within a window of length T to predict the runtime of the relative $(T+1)$-st iteration, and then the prediction is cycled to the end of the iteration.

4 Evaluation

This section describes the experimental setup, which evaluates the accuracy of our method and compares it with the current state-of-the-art solutions.

4.1 Experimental Environment

Cluster Setup. We run experiments on a 7-nodes OMNISKY cluster (1 JobManager & 6 TaskManagers), and all nodes are connected with 10-Gigabit Ethernet. Each node has two Intel Xeon Silver 4210 CPUs @ 2.20 GHz (10 cores $\times$ 2 threads, 20 TaskNumberSlots), 128 GB memory, and 1 TB SSD. Hadoop version 2.7.0 (for storing data on HDFS) and Flink version 1.8.0 are chosen.

Evaluation Metrics. To validate the performance of our method, we use the relative absolute error (RAE) as an evaluation metric. Let e_{ij} denotes the predicted value of our method for r_{ij}, RAE is calculated as shown below.

$$RAE = \frac{\sum_{i=1}^{n}\sum_{j=1}^{m}|r_{ij} - e_{ij}|}{\sum_{i=1}^{n}\sum_{j=1}^{m}|r_{ij} - \frac{1}{n}\sum_{i=1}^{n} r_{ij}|} \tag{7}$$

where n is the number of predictions and m denotes the number of iterations.

Baselines. We compare our model with the most advanced baseline as follows.

- SMiPE [10]: It continuously estimates the progress of the current iterative dataflow based on the matched historical execution.
- Two-stage [14]: Pham et al.'s two-stage machine learning-based task offline runtime prediction model, which combines input data features with the cluster environment to predict task runtime.

4.2 Dataset

In the experiment, multiple datasets are used for the typical Flink iterative jobs, as detailed in Table 3.

The LiveJournal dataset is friendship relationship data between users from the LiveJournal online community, the Wiki dataset is an encyclopaedic page graph from the English Wikipedia, and the RoadNet-CA dataset is from a road network data in California, including intersection and road start information. The Point dataset is a two-dimensional dataset generated by sampling from five Gaussian mixture models with random clustering centres and equal variances.

Table 3. Benchmark jobs and datasets.

Benchmark Job	Dataset	Size	Parameter
PageRank	LiveJournal	1 GB	d = 0.85, 500 iterations
ConnectedComponent (CC)	Wiki	5.74 GB	500 iterations
SSSP	RoadNet-CA	6.9 GB	500 iterations
K-Means	Points	10 GB	5 clusters, 500 iterations

4.3 Parameter Selection

For the LSTM, we used sigmoid as the activation function, 10 hidden layers for the initial training of the model, and added dropout layers to prevent overfitting. Let the number of training *epochs* = 100 and the dropout ratio was 0.2, we use the RAE of the samples before and after prediction and the time required for the LSTM to complete one round of training as indicators to compare sliding windows of different lengths of time to select the optimal window size.

As shown in Fig. 6, the sliding window size fluctuates due to the large variability of different iterations. Considering the prediction limitation and accuracy, the final sliding window size is 5% of the number of iterations.

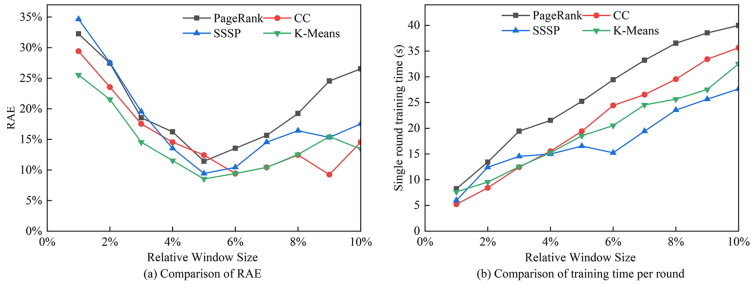

Fig. 6. Comparison of sliding window size under different iterative jobs.

4.4 Prediction Accuracy

We calculate the average RAE of the currently predicted iteration steps when the job runs to a certain time point, we use the relative iteration progress (RIP) to denote this time point. We implemented four iterative jobs with the bulk (Flink uses by default) and delta iterative operator to conduct experiments.

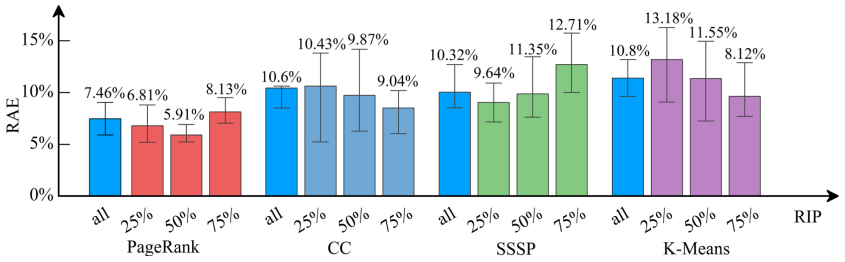

Fig. 7. Summary of prediction errors for runtime of bulk iterative job.

In Fig. 7 and 8, it can be seen that our method maintains between 5.91-12.71% and 10.16–15.03% throughout the prediction process for the bulk and delta iterative jobs, but the error at a RIP of 25% for the iterative jobs is generally larger than the other moments, which is due to the fact that little resource consumption information can be captured at the beginning of the iteration.

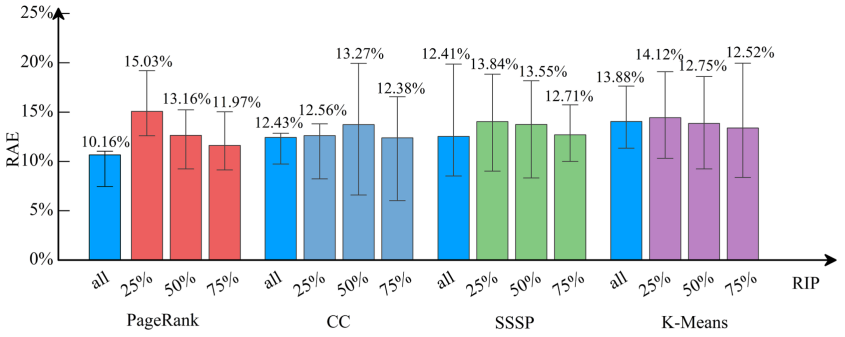

Fig. 8. Summary of prediction errors for runtime of delta iterative job.

4.5 Performance Comparison

In Table 4, the k indicates the size of the training data used and it is clear that our method outperforms the two-stage scheme when using 80% of the data, and when using the full amount of data, the error of our online prediction method is reduced by 4.79% compared to the superior two-stage scheme. In particular, the error of our method on the delta iterative job is reduced by about 15% compared to the two comparison schemes, which prefer to predict for the entire job.

Table 4. Comparison results of RAE for different prediction methods (%).

Type	Job	SMiPE	Two-stage	Our online prediction				
				k = 20%	k = 40%	k = 60%	k = 80%	k = 100%
Bulk	PageRank	23.57	18.26	31.51	28.63	19.54	13.57	10.72
	CC	27.13	17.63	35.33	32.58	21.66	18.52	12.87
	SSSP	19.54	12.87	29.88	24.62	18.51	14.35	9.83
	K-Means	17.88	14.54	27.44	22.92	14.23	13.47	10.85
	Average	22.03	15.86	31.04	27.19	18.49	14.98	**11.07**
Delta	PageRank	43.57	40.67	50.43	42.95	31.43	25.63	19.27
	CC	32.66	25.54	41.27	33.42	27.52	19.13	14.33
	SSSP	30.81	31.86	33.54	29.33	23.45	20.92	17.45
	K-Means	26.77	21.32	30.82	21.63	17.87	14.65	12.34
	Average	33.45	29.85	39.02	31.83	25.07	20.08	**15.85**

5 Related Work

There has been a lot of research such as [14–16] on how to predict the runtime of distributed jobs, as well as work for iterative jobs like [8–10].

Ellis [8] proposes the idea of modeling and predicting the runtime of Spark iterative jobs in stages. PREDIcT [9] captures the input features of each iteration and then build a cost model to predict the runtime of each iteration. SMiPE [10] is a system that captures runtime features to estimate the progress of iterative data flow by matching running jobs with previous executions.

In contrast to the above work based on statistics, Pham et al. [14] proposed a two-stage machine learning model-based prediction approach to predict the runtime of a continuous workflow task, and our approach is similar to it, which build a two-stage model to balance the response time and prediction accuracy.

6 Conclusion and Future Work

In this paper, we propose a online runtime prediction method for distributed iterative jobs, which core is a series of machine learning models that combine online learning and fine-grained resource consumption sequence data of each iteration to build a runtime prediction model by deeply analyzing the execution features of iterative jobs. In order to verify the effectiveness of the proposed method, we conduct many experiments to prove the superiority of our work. It is also worth considering how to improve job scheduling and resource allocation strategies based on runtime prediction in the future.

Acknowledgments. This research was supported by the National Key R&D Program of China under Grant No. 2018YFB1004402; and the National Natural Science Foundation of China under Grant No. 61772124.

References

1. Dean, J., et al.: Large scale distributed deep networks. In: Advances in Neural Information Processing Systems, pp. 1232–1240 (2012)
2. Carbone, P., et al.: Apache flink^{TM}: stream and batch processing in a single engine. IEEE Data Eng. Bull. **38**(4), 28–38 (2015)
3. Tumanov, A., et al.: TetriSched: global rescheduling with adaptive plan-ahead in dynamic heterogeneous clusters. In: Proceedings of the Eleventh European Conference on Computer Systems, pp. 35:1–35:16 (2016)
4. Wolf, J.L., et al.: FLEX: a slot allocation scheduling optimizer for mapreduce workloads. In: 11th International Middleware Conference, vol. 6452, pp. 1–20 (2010)
5. Thamsen, L., et al.: Selecting resources for distributed dataflow systems according to runtime targets. In: 35th IEEE International Performance Computing and Communications Conference, pp. 1–8 (2016)
6. Lama, P., Zhou, X.: AROMA: automated resource allocation and configuration of mapreduce environment in the cloud. In: 9th International Conference on Autonomic Computing, pp. 63–72 (2012)
7. Renner, T., et al.: Adaptive resource management for distributed data analytics based on container-level cluster monitoring. In: Proceedings of the 6th International Conference on Data Science, Technology and Applications, pp. 38–47 (2017)
8. Thamsen, L., et al.: Ellis: dynamically scaling distributed dataflows to meet runtime targets. In: IEEE International Conference on Cloud Computing Technology and Science, pp. 146–153 (2017). https://doi.org/10.1109/CloudCom.2017.37
9. Popescu, A.D., et al.: Predict: towards predicting the runtime of large scale iterative analytics. Proc. VLDB Endow. **6**(14), 1678–1689 (2013)
10. Koch, J., et al.: SMiPE: estimating the progress of recurring iterative distributed dataflows. In: 18th International Conference on Parallel and Distributed Computing, Applications and Technologies, pp. 156–163 (2017)
11. Kumar, V., et al.: Apache Hadoop YARN: yet another resource negotiator. In: ACM Symposium on Cloud Computing, pp. 5:1–5:16 (2013)
12. Hilman, M.H., et al.: Task runtime prediction in scientific workflows using an online incremental learning approach. In: 11th IEEE/ACM International Conference on Utility and Cloud Computing, pp. 93–102 (2018)
13. Gao, M., et al.: Online anomaly detection via incremental tensor decomposition. In: Ni, W., Wang, X., Song, W., Li, Y. (eds.) WISA 2019. LNCS, vol. 11817, pp. 3–14. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-30952-7_1
14. Pham, T., et al.: Predicting workflow task execution time in the cloud using A two-stage machine learning approach. IEEE Trans. Cloud Comput. **8**(1), 256–268 (2020). https://doi.org/10.1109/TCC.2017.2732344
15. da Silva, R.F., et al.: Online task resource consumption prediction for scientific workflows. Parallel Process. Lett. **25**(3), 1541003:1–1541003:25 (2015)
16. Pumma, S., et al.: A runtime estimation framework for ALICE. Future Gener. Comput. Syst. **72**, 65–77 (2017). https://doi.org/10.1016/j.future.2017.02.040