Online Runtime Environment Prediction for Complex Colocation Interference in Distributed Streaming Processing

Fan Liu[1,2], Weilin Zhu[1,2], Weimin Mu[1] *, Yun Zhang[1], Mingyang Li[1], Can Ma[1], and Weiping Wang[1]

[1] Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
[2] School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China
{liufan,zhuweilin,muweimin,zhangyun,limingyang,macan,wangweiping}@iie.ac.cn

Abstract. To improve system resource utilization, multiple operators are co-located in the distributed stream processing systems. In the colocation scenarios, the node runtime environment and co-located operators affect each other. The existing methods mainly study the impact of the runtime environment on operator performance. However, there is still a lack of in-depth research on the interference of operator colocation to the runtime environment. It will lead to inaccurate prediction of the performance of the co-located operators, and further affect the effect of operator placement. To solve these problems, we propose an online runtime environment prediction method based on the operator portraits for complex colocation interference. The experimental results show that compared with the existing works, our method can not only accurately predict the runtime environment online, but also has strong scalability and continuous learning ability. It is worth noting that our method exhibits excellent online prediction performance for runtime environments in large-scale colocation scenarios.

Keywords: Complex Colocation · Interference · Runtime Environment Prediction · Fused Deep Convolutional Neural Networks · Distributed Stream Processing.

1 Introduction

The distributed stream processing systems (DSPSs) offer an effective means to analyze and mine the real-time value of data. They support many data stream processing applications (DSPAs), which are composed of various operators that implement specific computing logic. To improve system resource utilization, multiple operators are deployed and run in colocation. In the colocation scenario, the runtime environment and co-located operators affect each other [1–4]. Specifically, the competition for shared resources by co-located operators will interfere with the runtime environment of nodes. In turn, the runtime environment will lead to operator performance fluctuations.

* Corresponding author

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_7

Now many studies focus on the impact of the runtime environment on operator performance [5, 6]. Zhao et al. [7] proposes an incremental learning method to predict the performance of serverless functions under a partial interference environment. Patel et al. [8] studies whether the runtime environment can meet the performance requirements of multiple jobs when the latency-critical and background jobs are co-located. Few studies analyze the interference of operator colocation to the runtime environment [9, 10]. Xu et al. [11] weights each job's interference to the runtime environment and designs a linear summation model to predict the runtime environment. Li et al. [12] computes the mean and variance of the interference of all co-located games to represent the runtime environment. However, the above methods are too simple and rough, and lack in-depth research on the interference of operator colocation to the runtime environment. It will lead to inaccurate prediction of the performance of the co-located operators, and further affect the effect of operator placement. The overall interference of co-located operators is complex and time-varying. So it is impossible to accurately predict the runtime environment with simple statistical methods. Besides, these methods are offline methods based on various benchmarks, and cannot continuously learn online.

In this paper, we propose an online runtime environment prediction method based on the operator portraits for complex colocation interference. Our method can accurately quantify and predict the key runtime environment metrics. The contributions of our work are as follows:

- To the best of our knowledge, our work is the first to deeply study the interference of operator colocation to the runtime environment. We use the operator portraits and fused deep convolution neural networks to model the colocation interference. On this basis, we predict the runtime environment in complex colocation scenario online.
- We design a runtime environment prediction model that fuses the deep convolutional neural networks and spatial pyramid pooling strategy [13]. Compared with the traditional padding models, our model can theoretically adapt to the learning of any scale colocation. Therefore, our model has strong scalability in the context of the rapid development of hardware technology.
- The experimental results show that our method can accurately predict the runtime environment in complex colocation scenarios online. In addition, our method has strong scalability and online learning ability to continuously improve prediction performance.

We organize the rest of this paper as follows. Section 2 elaborates the motivation of our work. Section 3 presents the design and implementation of our method. Section 4 describes the experimental results. Section 5 concludes our work.

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_7

2 Motivation

2.1 Interference Characteristics of Operator Colocation to Runtime Environment

Non-additivity of Colocation Interference. In the DSPSs, due to different resource contention of operators, multiple operator instances, and time-varying input load, the interference of co-located operators to the runtime environment is more complicated. The overall interference of co-located operators is not the sum of each operator's interference. We illustrate with the numerical calculation operator *calculator* and the sorting operator *sorter* as an example to illustrate. As shown in Fig. 1, the average CPU utilization of operator *calculator* is 15.19%, and that of operator *sorter* is 17.57% from Epoch t to $t + 850$. The actual measured average CPU utilization is 35.63% when they are co-located. It is greater than the sum of each CPU utilization (i.e. 32.76%). This is because when two operators are co-located, the CPU resources are time-division multiplexed. In addition to the overhead of operator solo-run, the overhead of CPU resource contention, inter-process switching, and site reservation is also increased. Moreover, for runtime environment metrics such as the cache hit rate, CPU interrupt times, and context switch times, the colocation interference cannot be superimposed.

Instability of Interference. The interference of operators to the runtime environment is time-varying and unstable. We analyze the interference when eleven different types of operators solo or co-located run. Specifically, we describe the runtime environment characteristics with nine key metrics, including the load, CPU utilization, cache hit rate, and context switching times, etc. We collect the runtime environment metrics at regular intervals to form multivariate time series. Then, we use the *Unit Root Test* method [14] to test the stationarity of the time series. We find that for some combinations of co-located operators and some runtime environment metrics, the *ADF test statistic* value is greater than the *Test critical value under* 10% *level*, and the *P-value* is also large, as shown in Fig. 2. Experimental results show that the interference of operators to the runtime environment is time-varying and unstable. Therefore, it is unreasonable and inaccurate to describe the runtime environment with a fixed value or mean and variance.

2.2 Importance of Interference of Operator Colocation to Runtime Environment.

SLA Guarantee. In the DSPSs, the runtime environment analysis is the premise of operator performance prediction and operator placement. Besides, to process the input load in real time, it is necessary to predict the runtime environment before the operators are actually co-located. In addition, when the colocation changes, it is necessary to predict the runtime environment instantaneously. Therefore, if we can predict the runtime environment online, we can better meet the end-to-end latency requirements.

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_7

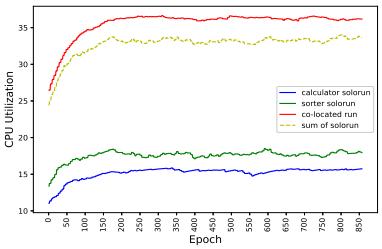


Fig. 1. The CPU utilization of the operator *calculator* and *sorter* when they solo or co-located run.

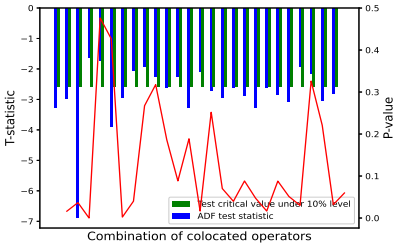


Fig. 2. The Stationarity Analysis of Interference of Different Operator Colocation to the CPU Utilization.

System Stability. If we can accurately predict the runtime environment online, we can effectively support operator placement decisions, avoid or reduce operator migration and adjust jitter, and thus improve system stability.

3 Design and Implementation

3.1 Overview

Fig. 3 describes the design architecture of our method, which contains three core modules: the operator interference profiler(OIProfiler), the colocation interference learner(CILearner), and the runtime environment predictor(REPredictor). We adopt the OIProfiler to build the portraits of single operators offline, and the CILearner to continuously learn the portraits of co-located operators online. Based on the portraits of single and co-located operators, we use the REPredictor to predict the runtime environment.

The work process consists of two main stages: the offline solo-run and the online colocation-run. In the offline solo-run stage, a single operator runs on an empty node, and the Metric Collector collects the key runtime environment metrics over some time at regular intervals. Then the OIProfiler analyzes the metrics and builds the portrait of the operator.

In the online colocation-run stage, multiple operators run on an empty node simultaneously, and the Metric Collector collects the key runtime environment metrics at regular intervals continuously. Then the CILearner builds or updates the portraits of the co-located operators. At last, the REPredictor refers to the portraits of single operators and co-located operators to predict the runtime environment interfered by arbitrary co-located operators. Besides, as the collected samples increase, our CILearner dynamically updates the built portraits and creates the portraits of new co-located operators. Our REPredictor also updates adaptively to realize online prediction.

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_7

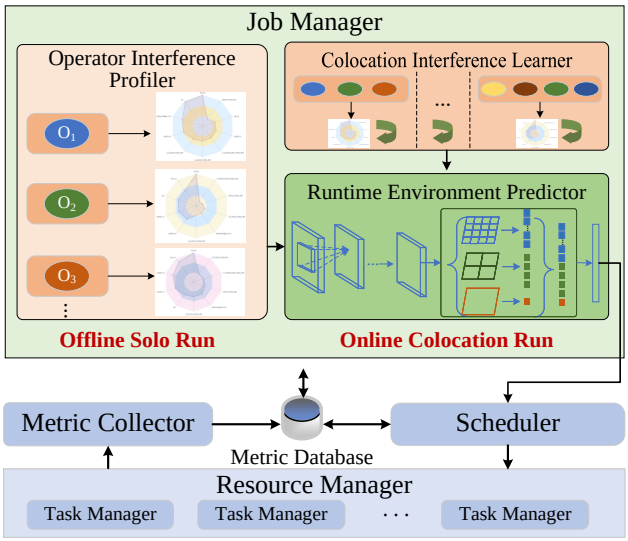


Fig. 3. Achitecture.

3.2 OIProfiler: Operator Interference Profiler

Key Runtime Environment Metrics. By summarizing previous work [7, 15, 16], we can collect the runtime environment metrics through the system performance analysis tools and benchmarks. The system performance analysis tools directly monitor the various runtime environment metrics of nodes. The benchmark method customizes benchmarks sensitive to different resources and indirectly quantifies the runtime environment by the performance degradation of benchmarks. The benchmarks occupy more node resources by comparison, which themselves affect the runtime environment. And the benchmark method cannot collect the runtime environment metrics online. Therefore, we adopt the system performance analysis tools to collect the runtime environment metrics.

In this paper, we use the *top*, *perf* [17], and *vmstat* tools to monitor 42 runtime environment metrics, including CPU, memory, bandwidth, L1 instruction and data cache, LLC cache, context switch, and branch prediction, etc. However, some metrics have no or low correlation with the operator performance. To avoid overfitting and improve prediction efficiency, we use the *Pearson Correlation Coefficient* [18] to measure the correlations between the runtime environment metrics and the performance of eleven operators with different resource contention characteristics. Finally, we select nine key metrics, including *load_average_1min*, *cpu_system*, *cpu_user*, *l1_dcache_load_misses_rate*, *llc_load_misses_rate*, *llc_store_misses_rate*, *branch_misses_rate*, *ipc*, and *system_cs*. The correlation between the key runtime environment metrics and the performance of eleven operators is shown in Fig. 4.

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_7

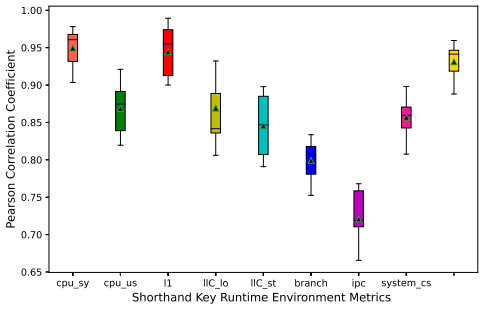


Fig. 4. The correlation between the Key Runtime Environment Metrics and the Performance of Eleven Operators.

Input and Output. We collect the time series of key runtime environment metrics as input when a single operator runs on an empty node. In detail, we use t to denote the start time and $se_o^i(t)$ to denote the environment metric i when the operator o is running at time t. The $SE_o(t) = \{se_o^1(t), se_o^2(t), ..., se_o^n(t)\}$ represents the key runtime environment metrics at time t. So we use the dataset $S_o = \{SE_o(t), SE_o(t+1), SE_o(t+2), ..., SE_o(t+w)\}$ as the input, in which w represents the collection duration.

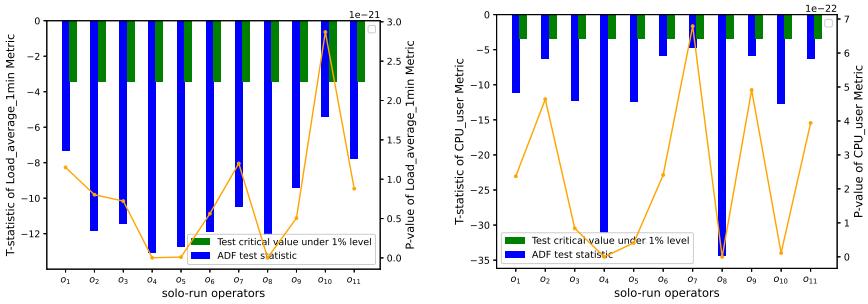
We build the operator portrait as output, denoted as $SP_o = \{sp_o^1, sp_o^2, ..., sp_o^n\}$. The sp_o^i represents the interference of the operator o to the runtime environment metric i, in which $i \in \{1, 2, ..n\}$.

Profiling Module. As explained in Section 2, the interference of a single operator to the runtime environment is time-varying and unstable. We utilize the $first\text{-}order\ difference$ method [19] to process the time series of key runtime environment metrics. We find that the time series of all metrics after the difference are stable. As shown in Fig. 5, we take the metrics of $load\_average\_1min$ and cpu_user as an example to illustrate. The $ADF\ test\ statistic$ value of each operator is less than the $Test\ critical\ value\ under\ 1\%\ level$. The $P\text{-}value$ is much less than the significance level of 0.01 and very close to zero. These show that the original hypothesis that the time series after differencing is not stationary can be strictly rejected. Therefore, we use the mean values of the time series after differencing to image the single operators.

3.3 CILearner: Colocation Interference Learner

Input and Output. We collect the time series of key runtime environment metrics as input when multiple operators co-located run on an empty node. In detail, we use O to denote the co-located operator set and $ce_O^i(t)$ to denote the environment metric i when the co-located operator set O is running at time t. The $CE_O(t) = \{ce_O^1(t), ce_O^2(t), ..., ce_O^n(t)\}$ represents the key runtime environment metrics at time t. So we use the dataset $C_O = \{CE_O(t), CE_O(t+1), CE_O(t+2), ..., CE_O(t+w)\}$ as the input, in which w represents the collection duration.

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_7



(a) (b)

Fig. 5. The Stationarity Analysis of Time Series of $load\_average\_1min$ and cpu_user Metrics after Difference.

We build the protrait of the co-located operators as output, denoted as $CP_O = \{cp_O^1, cp_O^2, ..., cp_O^n\}$. The cp_O^i represents the interference of the co-located operator set O to the runtime environment metric i, in which $i \in \{1, 2, ..n\}$.

Learning Module.

Observation. Co-located operators started in different orders interfere equally with the runtime environment.

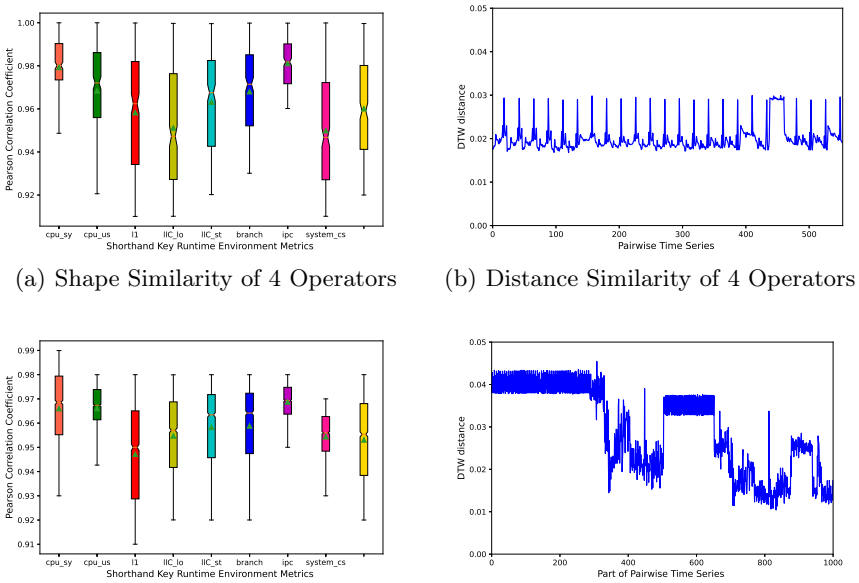
We start $2, 4, 8, 16, 24$ and 32 operators in different orders respectively. The startup interval between adjacent operators is randomly between one second and ten minutes. As the co-located operators run, we get the time series of key runtime environment metrics. We adopt the $Pearson\ Correlation\ Coefficient$ [18] to measure the shape similarity between pairwise time series, and the $Dynamic\ Time\ Warping\ (DTW)$ [20] to measure the distance similarity.

We take the colocation of four and twenty-four operators as an example to illustrate. We compare the correlation and distance between pairwise time series. As shown in Fig. 6, the minimum correlation of any metrics are is greater than 0.9 and the distances are less than 0.1. Therefore, we get that the runtime environment is independent of the operator startup order. Furthermore, we find that although the time series of key runtime environment metrics are unstable, those after the difference are stable. Therefore, we use the mean values of the time series after the difference to image the co-located operators. Moreover, after collecting new existing co-located samples, we update the mean value to realize continuous learning.

3.4 REPredictor: Runtime Environment Predictor

Input Processing. Our REPredictor predicts the runtime environment based on the portraits of each co-located operator. The colocation scale is variable. But the existing prediction models require the input size to be fixed. Therefore, the traditional method is to unify the input based on the maximum colocation capacity of nodes. If the colocation scale is less than the maximum colocation

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_7



(a) Shape Similarity of 4 Operators (b) Distance Similarity of 4 Operators

(c) Shape Similarity of 24 Operators (d) Distance Similarity of 24 Operators

Fig. 6. The Shape Similarity and Distance Similarity Analysis of Pairwise Time Series at Different Colocation Scales.

capacity, the method completes the input with zero padding. Our method organizes the input into scalable images, which can scale vertically with the change of colocation scale.

In Fig. 7, we vividly describe the input of traditional and our methods. We assume that the operator portrait contains three metrics, which are denoted by the gold, blue and green squares in the figure. The maximum colocation capacity is assumed to be ten. Therefore, the traditional method unifies the input to 30 dimensions, which is obtained by multiplying the operator protrait dimension by the maximum colocation capacity. The part less than the maximum capacity is padded zeros, as shown in the purple dotted box in the figure. Our method flexibly scales the input image vertically according to the colocation scale.

Input and Output. As mentioned above, we organize the portraits of each co-located operator into image format as input. We use o_i to denote the ith co-located operator, m to denote the colocation scale, and $O(o_i \in O, i = \{1, 2, ..., m\})$ to denote the co-located operator set. The input of our REPredictor is expressed

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_7

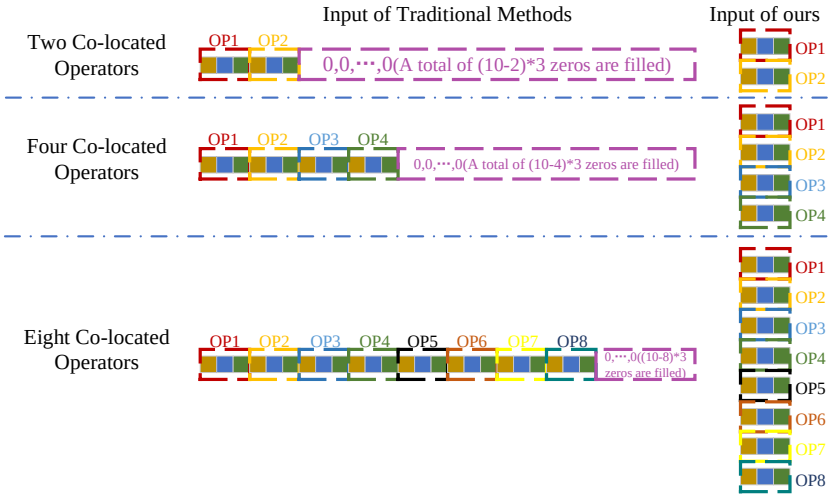


Fig. 7. The Input of Traditional Methods and Ours.

as:

$$P_O = \begin{bmatrix} sp^1_{o_1} & sp^2_{o_1} & \cdots & sp^n_{o_1} \\ sp^1_{o_2} & sp^2_{o_2} & \cdots & sp^n_{o_2} \\ \vdots & \vdots & \ddots & \vdots \\ sp^1_{o_m} & sp^2_{o_m} & \cdots & sp^n_{o_m} \end{bmatrix}$$

Our REPredictor predicts the final runtime environment as output. The output is expressed as: $CP_O = \{cp^1_O, cp^2_O, ..., cp^n_O\}$.

Predicting Module. As mentioned above, our REPredictor organizes the input into a scalable image. Because the colocation scale is variable, the image size is also not fixed. In this paper, we introduce the spatial pyramid pooling (SPP) Strategy [13] to remove the fixed-size constraint. Specifically, we design fused deep convolutional neural networks by adding an SPP layer between the last convolutional layer and the fully connected layer, as shown in Fig. 8. Firstly, the convolutional part runs in a sliding-window manner to learn features for co-located operator images of arbitrary size. The convolutional part consists of two convolutional layers, a pooling layer, and two convolutional layers. Their outputs are *feature maps*. After the last convolutional layer, the SPP layer pools the features and generates fixed-length outputs. We use the blocks of three different sizes to extract features, and put these three grids on the *feature maps* to get different spatial bins. Then we extract a feature from each Spatial bin to obtain fixed-dimensional feature vectors. At last, we feed the feature vectors to the fully connected layer to predict the final runtime environment.

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_7

Our predicting networks can generate a fixed-length runtime environment output regardless of the colocation scale. The networks use multi-level spatial bins and pool features extracted at variable scales to improve prediction accuracy, scalability, and robustness.

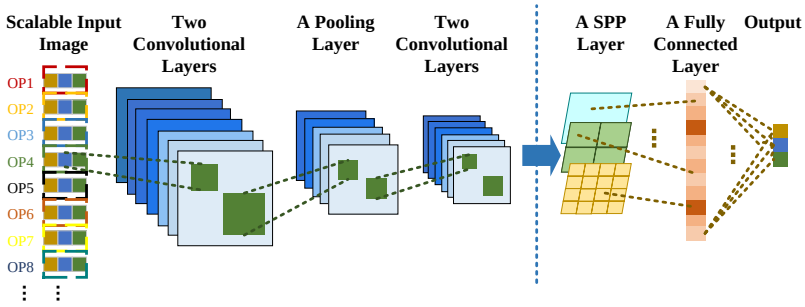


Fig. 8. The Structure Diagram of Our Fused Deep Convolution Neural Networks.

4 Experiments

4.1 Settings and Datasets

Settings. We experiment on a cluster of twelve servers. There are two kinds of servers in the cluster: two GPU servers and ten CPU servers. Each GPU server contains 36 cores Intel Xeon CPU E5-2697 v4 2.30 GHz, 256GB memory, two NVIDIA GeForce GTX 1060ti cards, and 500GB disks. We use one GPU server to run Job Manager, Scheduler, and MetricDatabase, and another GPU server to train and evaluate our proposed model. Each CPU server contains 10 cores Intel Xeon CPU Gold 5115 2.4GHz, 256GB memory, 480G SSD, and 2.4T SAS. We use the CPU servers to run Task Manager to execute operators. Besides, we train and evaluate our REPredictor with python 3.7 and tensorflow 1.15.0.

Datasets. We experiment on DataDock [21], our distributed stream processing system. We collect the key runtime environment metrics when the operators solo or co-located run, and build four databases, as shown in Table 1. The samples in the SRE database are collected at every second interval for 20 minutes offline. The SIP database is built by our OIProfiler with samples from the SRE database. The samples in the CRE database are continuously collected online at every second interval. The CIP database is built by our CILearner with samples from the CRE database.

We design eleven operators with different resource contention characteristics. These operators are implemented in C language to eliminate the impact of factors

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_7

such as garbage collection on the runtime environment. In the experiment, the operators run at full load, that is, the input rate of the operators is greater than their processing capacity. Besides, each operator can be copied into multiple instances, so that we can simulate the complex colocation scenarios.

Table 1. The Datasets of Our Experiments.

ID	Name	Description
1	SRE	The key runtime environment metrics collected when the operators run offline and solo.
2	SIP	The portraits of interference of single operators to the runtime environment.
3	CRE	The key runtime environment metrics when the operators run online and co-located.
4	CIP	The portraits of interference of co-located operators to the runtime environment.

4.2 Evaluation

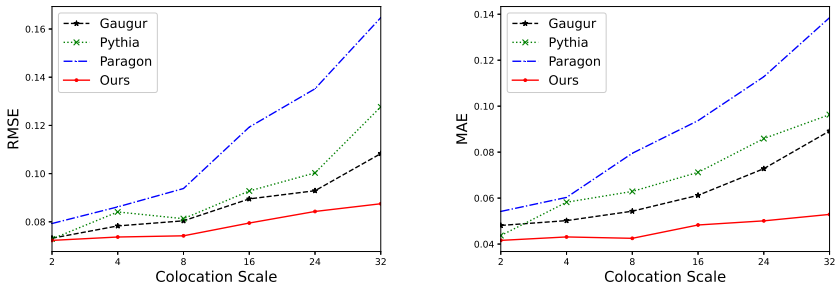
We evaluate the performance of runtime environment prediction from three aspects: accuracy, scalability, and continuous learning ability. We use the *Root Mean Square Errors*($RMSE$) and *Mean Absolute Errors*(MAE) as the evaluation metric. $RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y-\hat{y})^2}$, $MAE = \frac{1}{n}\sum_{i=1}^{n}|y-\hat{y}|$, where y is the actual runtime environment, and $\hat{y}$ is the predicted value. Besides, we repeat the experiments 100 times and compute the average results to eliminate outliers and reduce random errors.

Prediction Accuracy. To evaluate the accuracy of runtime environment prediction, we compare with the GAugur [12], PYTHIA [11], and Paragon [22]. For a set of co-located operators, the GAugur method computes the mean and variance of the portraits of all co-located games to characterize the runtime environment. The PYTHIA method computes the linear sum of the portraits of all co-located games to characterize the runtime environment. The Paragon method calculates the sum of the portraits of all co-located games to characterize the runtime environment. We compare the accuracy of four methods for runtime environment prediction when 2, 4, 8, 16, 24, and 32 operators are co-located.

As is shown in Fig. 9, Our method outperforms other methods. Especially when the colocation scale increases, the advantages of our method are more significant. This is because we can learn more complex hidden features of colocation interference by using the fused deep convolutional neural networks. Besides, with the development of hardware technology, the colocation scale is increasing. Our

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_7

method shows better prediction performance in large-scale colocation scenarios. In addition, our method can directly predict the runtime environment. It is particularly suitable for online prediction and solves the cold start problem well.



(a) (b)

Fig. 9. The Prediction Accuracy of Different Methods.

Prediction Scalability. The colocation scale is variable. To evaluate the scalability of runtime environment prediction, we compare our method with the MLP and CNN models. These models use the traditional padding method to unify the input.

The main parameters of the models are as follows. For the MLP and CNN networks, we set the maximum number of co-located operators to 32 and unify the input to 288 dimensions. In the MLP networks, we use *relu* as the activation function, *adam* as the optimizer, and *mse* as the loss, and set five hidden layers with sizes 512, 256, 128, 64, and 32. The CNN networks consist of two convolutional layers, a pooling layer, two convolutional layers, and a fully connected layer. And we use *relu* as the activation function, *adam* as the optimizer, *mse* as the loss, and set the six layers with sizes 288, 128, $(2, 2)$, 64, 32, 9. The composition of our model is the same with the CNN networks, only adding an SPP layer between the last convolutional layer and the fully connected layer. And we set the SPP layer with block sizes of $(1 * 1, 2 * 2, 4 * 4)$ and the other parameters are the same as the CNN networks.

As is shown in Table 2, the prediction performance of MLP model is the worst, and the CNN model is comparable to ours. Our method solves the scalability problem while ensuring good prediction performance.

Continuous Learning Ability. To verify the continuous learning ability of our method, we compare the prediction performance at different running times and sample capacities. As shown in Fig. 10, as the running time or sample capacity increases, the prediction accuracy of our methods continues to improve. This is because our method can collect samples online and learn new operator colocation scenarios.

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_7

5 Conclusion

In this paper, we propose an online runtime environment prediction method for complex colocation interference. It contains three core modules: the OIProfiler, the CILearner, and the REPredictor. Firstly, the OIProfiler builds the portraits of single operators in the form of offline solo-run. Then, the CILearner builds the portraits of co-located operators in the form of online colocation-run. At last, we use the REPredictor to model the portraits of single operators and co-located operators to predict the runtime environment with our fused deep convolutional neural networks. The experiments on the real-world datasets demonstrate that our method is better than the state-of-the-art methods. Our method can not only accurately predict the runtime environment online for complex colocation interference, but also has strong scalability and continuous learning ability.

Table 2. The Prediction Scalability of Different Methods.

Co-located Numbers	MLP		CNN		**Ours**	
	RMSE	MAE	RMSE	MAE	**RMSE**	**MAE**
2	0.0793	0.0473	0.0729	0.0437	**0.0723**	**0.0416**
4	0.0836	0.0502	0.0748	0.0429	**0.0737**	**0.0431**
8	0.0891	0.0543	0.0759	0.0441	**0.0742**	**0.0425**
16	0.0928	0.0612	0.0784	0.0473	**0.0795**	**0.0483**
24	0.1003	0.0685	0.0847	0.0538	**0.0843**	**0.0501**
32	0.1172	0.0783	0.0837	0.0574	**0.0825**	**0.0529**

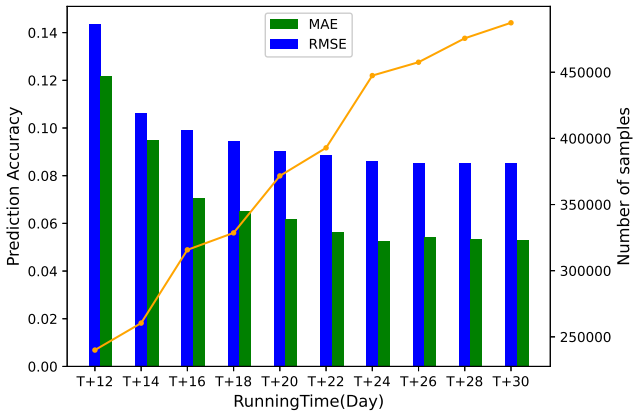


Fig. 10. The Continuous Learning Ability of Our Method.

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_7

References

1. H. Zhang, X. Geng, and H. Ma, "Learning-driven interference-aware workload parallelization for streaming applications in heterogeneous cluster," *IEEE Trans. Parallel Distributed Syst.*, vol. 32, no. 1, pp. 1–15, 2021.
2. Y. Gan, Y. Zhang, K. Hu, D. Cheng, Y. He, M. Pancholi, and C. Delimitrou, "Seer: Leveraging big data to navigate the complexity of performance debugging in cloud microservices," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2019, Providence, RI, USA, April 13-17, 2019* (I. Bahar, M. Herlihy, E. Witchel, and A. R. Lebeck, eds.), pp. 19–33, ACM, 2019.
3. M. HoseinyFarahabady, A. Y. Zomaya, and Z. Tari, "Qos- and contention- aware resource provisioning in a stream processing engine," in *2017 IEEE International Conference on Cluster Computing, CLUSTER 2017, Honolulu, HI, USA, September 5-8, 2017*, pp. 137–146, IEEE Computer Society, 2017.
4. T. Buddhika, R. Stern, K. Lindburg, K. Ericson, and S. Pallickara, "Online scheduling and interference alleviation for low-latency, high-throughput processing of data streams," *IEEE Trans. Parallel Distributed Syst.*, vol. 28, no. 12, pp. 3553–3569, 2017.
5. F. Romero and C. Delimitrou, "Mage: online and interference-aware scheduling for multi-scale heterogeneous systems," in *Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques, PACT 2018, Limassol, Cyprus, November 01-04, 2018* (S. Evripidou, P. Stenström, and M. F. P. O'Boyle, eds.), pp. 19:1–19:13, ACM, 2018.
6. S. Chen, C. Delimitrou, and J. F. Martínez, "PARTIES: qos-aware resource partitioning for multiple interactive services," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2019, Providence, RI, USA, April 13-17, 2019* (I. Bahar, M. Herlihy, E. Witchel, and A. R. Lebeck, eds.), pp. 107–120, ACM, 2019.
7. L. Zhao, Y. Yang, Y. Li, X. Zhou, and K. Li, "Understanding, predicting and scheduling serverless workloads under partial interference," in *International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2021, St. Louis, Missouri, USA, November 14-19, 2021* (B. R. de Supinski, M. W. Hall, and T. Gamblin, eds.), p. 22, ACM, 2021.
8. T. Patel and D. Tiwari, "CLITE: efficient and qos-aware co-location of multiple latency-critical jobs for warehouse scale computers," in *IEEE International Symposium on High Performance Computer Architecture, HPCA 2020, San Diego, CA, USA, February 22-26, 2020*, pp. 193–206, IEEE, 2020.
9. Y. Zhang, M. A. Laurenzano, J. Mars, and L. Tang, "Smite: Precise qos prediction on real-system SMT processors to improve utilization in warehouse scale computers," in *47th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2014, Cambridge, United Kingdom, December 13-17, 2014*, pp. 406–418, IEEE Computer Society, 2014.
10. C. Delimitrou and C. Kozyrakis, "Quasar: resource-efficient and qos-aware cluster management," in *Architectural Support for Programming Languages and Operating Systems, ASPLOS 2014, Salt Lake City, UT, USA, March 1-5, 2014* (R. Balasubramonian, A. Davis, and S. V. Adve, eds.), pp. 127–144, ACM, 2014.
11. R. Xu, S. Mitra, J. Rahman, P. Bai, B. Zhou, G. Bronevetsky, and S. Bagchi, "Pythia: Improving datacenter utilization via precise contention prediction for multiple co-located workloads," in *Proceedings of the 19th International Middleware*

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_7

Conference, Middleware 2018, Rennes, France, December 10-14, 2018 (P. Ferreira and L. Shrira, eds.), pp. 146–160, ACM, 2018.
12. Y. Li, C. Shan, R. Chen, X. Tang, W. Cai, S. Tang, X. Liu, G. Wang, X. Gong, and Y. Zhang, "Gaugur: Quantifying performance interference of colocated games for improving resource utilization in cloud gaming," in *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing, HPDC 2019, Phoenix, AZ, USA, June 22-29, 2019* (J. B. Weissman, A. R. Butt, and E. Smirni, eds.), pp. 231–242, ACM, 2019.
13. K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 9, pp. 1904–1916, 2015.
14. D. A. Dickey, "Dickey-fuller tests," in *International Encyclopedia of Statistical Science* (M. Lovric, ed.), pp. 385–388, Springer, 2011.
15. C. Courtaud, J. Sopena, G. Muller, and D. G. Pérez, "Improving prediction accuracy of memory interferences for multicore platforms," in *IEEE Real-Time Systems Symposium, RTSS 2019, Hong Kong, SAR, China, December 3-6, 2019*, pp. 246–259, IEEE, 2019.
16. Q. Chen, H. Yang, M. Guo, R. S. Kannan, J. Mars, and L. Tang, "Prophet: Precise qos prediction on non-preemptive accelerators to improve utilization in warehouse-scale computers," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2017, Xi'an, China, April 8-12, 2017* (Y. Chen, O. Temam, and J. Carter, eds.), pp. 17–32, ACM, 2017.
17. "perf." `https://perf.wiki.kernel.org/index.php/Tutorial`.
18. Y. Amannejad, D. Krishnamurthy, and B. H. Far, "Predicting web service response time percentiles," in *12th International Conference on Network and Service Management, CNSM 2016, Montreal, QC, Canada, October 31 - Nov. 4, 2016*, pp. 73–81, IEEE, 2016.
19. Y. Lan and D. Neagu, "Applications of the moving average of n^{th} -order difference algorithm for time series prediction," in *Advanced Data Mining and Applications, Third International Conference, ADMA 2007, Harbin, China, August 6-8, 2007, Proceedings* (R. Alhajj, H. Gao, X. Li, J. Li, and O. R. Zaïane, eds.), vol. 4632 of *Lecture Notes in Computer Science*, pp. 264–275, Springer, 2007.
20. Z. Geler, V. Kurbalija, M. Ivanovic, M. Radovanovic, and W. Dai, "Dynamic time warping: Itakura vs sakoe-chiba," in *IEEE International Symposium on INnovations in Intelligent SysTems and Applications, INISTA 2019, Sofia, Bulgaria, July 3-5, 2019* (P. D. Koprinkova-Hristova, T. Yildirim, V. Piuri, L. S. Iliadis, and D. Camacho, eds.), pp. 1–6, IEEE, 2019.
21. W. Mu, Z. Jin, J. Wang, W. Zhu, and W. Wang, "Bgelasor: Elastic-scaling framework for distributed streaming processing with deep neural network," in *Network and Parallel Computing - 16th IFIP WG 10.3 International Conference, NPC 2019, Hohhot, China, August 23-24, 2019, Proceedings* (X. Tang, Q. Chen, P. Bose, W. Zheng, and J. Gaudiot, eds.), vol. 11783 of *Lecture Notes in Computer Science*, pp. 120–131, Springer, 2019.
22. C. Delimitrou and C. Kozyrakis, "Paragon: Qos-aware scheduling for heterogeneous datacenters," in *Architectural Support for Programming Languages and Operating Systems, ASPLOS 2013, Houston, TX, USA, March 16-20, 2013* (V. Sarkar and R. Bodík, eds.), pp. 77–88, ACM, 2013.

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_7