Elastic Resource Allocation based on Dynamic Perception of Operator Influence Domain in Distributed Stream Processing

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

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

Abstract. With the development of distributed stream processing systems, elastic resource allocation has become a powerful means to deal with the fluctuating data stream. The existing methods either focus on a single operator or only consider the static correlation between operators to perform elastic scaling. However, they ignore the dynamic correlation between operators in data stream processing applications, which leads to lagging and inaccuracy resource allocation, increasing processing latency. To address these issues, we propose an elastic resource allocation method, which is based on the dynamic perception of operator influence domain, to perform resource allocation dynamically and in advance. The experimental results show that compared with the existing methods, our method not only guarantees that the end-to-end latency meets QoS requirements but also reduces resource utilization.

Keywords: Data stream processing · Dynamic correlation · Adaptive partition · Meta-learning.

1 Introduction

Distributed stream processing systems (DSPSs) can quickly analyze and mine the real-time value of data, which are powerful means to process continuous and massive data. In DSPSs, resource allocation determines the operator parallelism of data stream processing applications (DSPAs), which is the key to ensure the QoS of DSPAs. Because data streams exhibit the characteristics of dynamic fluctuation and mutation, elastic resource allocation has become a dominant method.

Many researchers have proposed elastic resource allocation methods. Zhang et al. [1] monitors the actual processing time of input data load for each operator, which is compared with the required time to identify the bottlenecks, and increases the parallelism of those operators to maximize the throughput of DSPAs

* Corresponding author

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08751-6_53

in shared memory multi-core architectures. Mu et al. [2] predicts the load of each operator and compares it with the processing performance to allocate resources quantitatively. These methods focus on a single operator to perform elastic scaling and do not consider the correlation between operators. Actually, the changes in the upstream operators may create ripple effects throughout DSPAs [3, 4], and some researchers utilize the correlation between operators. Lombardi et al. [5] calculates the load of an operator based on the input load of the DSPA and the average selectivity of its upstream operators, and then analyzes the CPU utilization of the operator under the load to determine whether to scale elastically. Wei et al. [6] calculates the load of an operator according to the DSPA's load, average selectivity, and average network bandwidth of its upstream operators, which is then compared with the performance to adjust the parallelism. However, these methods ignore the dynamic characteristic of correlation between operators. This will result in lagging and inaccurate resource allocation, which increases processing latency in DSPSs.

In this paper, we propose an elastic resource allocation method based on the dynamic perception of operator influence domain to adjust the operator parallelism dynamically and in advance. The contributions of our work are as follows:

- To the best of our knowledge, our work is the first to utilize the dynamic correlation between operators for resource allocation. We use the static selectivity metrics and dynamic selectivity statistic metrics to evaluate the influence domain of upstream operators. Accordingly, we divide the DSPA into partitions adaptively, and plan the parallelism of the operators in units of partitions.
- We use the random forest regression (RFR) [7] to model the correlation between operators within each partition online and update it dynamically. For the input load of each partition, we compute the optimal parallelism of each operator in this partition.
- We use the meta-learning method to predict the load of each partition online. To the best of our knowledge, this is the first to combine the strong expressive long short term memory networks (LSTM) meta-learner and the efficient multi-layer perceptron (MLP) base-learner to catch the fluctuations features of the data stream in real-time.
- The experimental results show that our work ensures that the end-to-end latency meets QoS requirements while improving resource utilization efficiency.

The rest of this paper is organized as follows. Section 2 introduces the motivation of our work. Section 3 describes the design and implementation. Section 4 shows the experimental results. Finally, Section 5 concludes our paper.

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08751-6_53

2 Motivation

2.1 Dynamic Correlation between Upstream and Downstream Operators

In DSPAs, the data stream is processed by the upstream operators and sent to the downstream operators. As mentioned above, when the load or parallelism of the upstream operator changes, the downstream operator will change accordingly. We take the word count application as an example. The directed acyclic graph (DAG) of the application is shown in Fig. 1. The operator *parser* parses each received article, *filtter* filters duplicate articles, *splitter* splits each article into words, and *counter* counts the frequency of each word. We illustrate our motivation with the operators *parser*, *filtter* and *splitter*, whose input loads are expressed in the number of articles received per second.

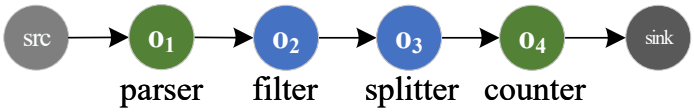


Fig. 1. The DAG of Word Count Application.

In Fig. 2(a), as the load of *parser* increases or decreases, the load of *filter* also increases or decreases. To handle the increased load, we increase the parallelism of *parser*, and subsequently the parallelism of *filter* also increases, as shown in Fig. 2(b). Thus, we get that the changes of load and parallelism of the upstream operator affect that of the downstream operator. Besides, we compute the Pearson Correlation Coefficient of load and parallelism between *parser* and *filter*, as shown in Table 1. We get that the correlation between the upstream and downstream operator is time-varying.

Table 1. Pearson Correlation Coefficient of Load and Parallelism between Upstream and Downstream Operators.

Epoch	CORR(*parser*,*filter*)		CORR(*parser*,*splitter*)	
	Load	Parallelism	Load	Parallelism
Epoch 0-9	0.9973	0.9489	-0.0626	0.01856
Epoch 10-19	0.9993	0.9836	0.9993	0.9972

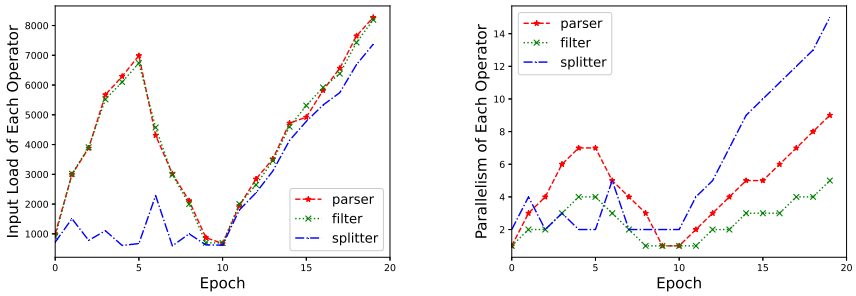
2.2 Dynamic Influence Domain of Upstream Operators

The operators in the DAG are not globally correlated but show the characteristics of local correlation. We find that there is little or no correlation between the upstream operator *parser* and downstream operator *splitter*. As shown in Fig. 2(a), from *epoch* 0 to 9, the load of *parser* increases first and then decreases, while that of *splitter* fluctuates. The reason is that *parser* receives a lot of duplicate articles during that period, which are filtered by *filter*.

So the correlation of load between *parser* and *splitter* is not obvious, as shown in Table 1. From *epoch* 10 to 19, the load of *parser* increases and that

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08751-6_53

of *splitter* also increases. The load between *parser* and *splitter* shows a strong correlation. In Fig. 2(b) we get the same results for the parallelism of *parser* and *splitter*. Thus, we get that the influence of upstream operators on downstream operators has a range, which is named the operator influence domain. Besides, the influence domain of operators is time-varying, as shown in Table 1.



(a) (b)

Fig. 2. The Load and Parallelism of Operators in the DAG.

2.3 Importance of Dynamic Perception of Operator Influence Domain.

QoS Guarantee. If we can accurately analyze the influence domain of upstream operators, we can adjust the parallelism of downstream operators within the domain in advance to satisfy the end-to-end latency requirements better.

System Stability. If we can refer to the upstream operators to adjust the parallelism of downstream operators in the influence domain, we can avoid adjustment jitter to improve system stability.

System Overhead. If we can model each influence domain instead of each operator, we will reduce the computational overhead and resource overhead.

3 Design and Implementation

3.1 Overview

We describe our model in Fig. 3. It contains three core modules: the online load predictor (OLPredictor), the adaptive operator partitioner (AOPartitioner), and the online partition-based operator parallelism planner (OPPlanner). We use the AOPartitioner to dynamically divide the DAG into partitions. The division is based on the influence domain of the upstream operators. Then we refer to the partition results and use the OPPlanner to determine the parallelism of operators in each partition. Besides, we use the OLPredictor to predict the load of each partition online to achieve proactive elastic scaling.

The work process of our model mainly contains five stages. Firstly, the Metric Collector collects the dynamic selectivity statistic metrics of theoretically

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08751-6_53

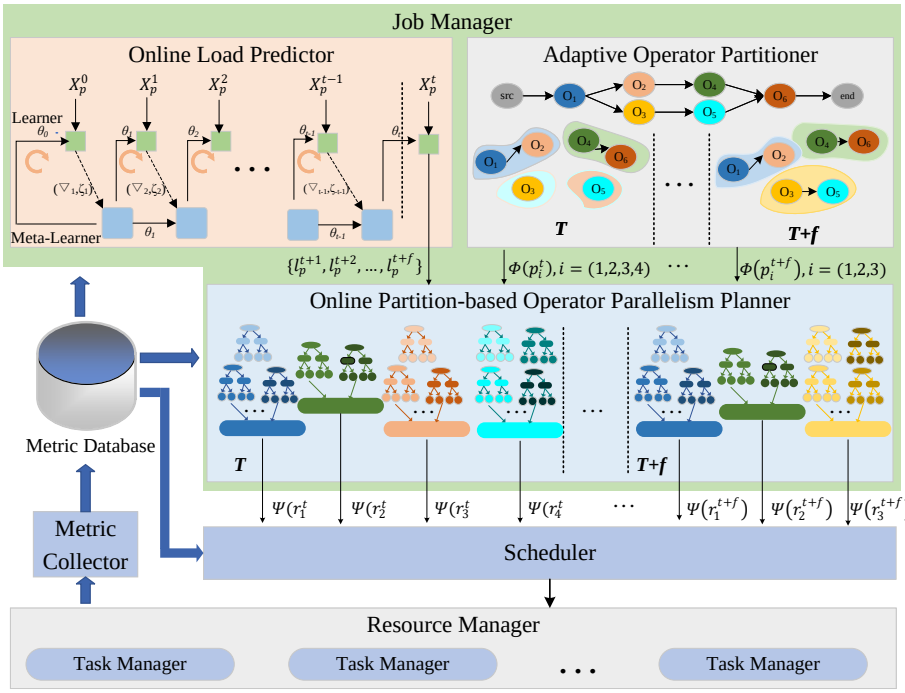


Fig. 3. Architecture.

unstable operators, the load metrics of each partition, the parallelism metrics of operators in each partition covering the load and the resource metrics of each node, and stores them in MetricDatabase [8]. Secondly, the AOPartitioner refers to the static selectivity metrics and dynamic selectivity statistic metrics to evaluate the influence domain of upstream operators and partitions the DAG adaptively. Thirdly, the OPPlanner uses the load and operator parallelism metrics to build the RFR to plan the optimal parallelism of operators in units of partitions. Fourthly, the OLPredictor uses the load metrics to accurately predict the load of each partition in the future online. And we input the predicted load into the OPPlanner to get the optimal operator parallelism of each partition in the future. Finally, we refer to the scheduler [9] in our previous work to place the instances of operators in each partition on appropriate nodes.

3.2 AOPartitioner: Adaptive Operator Partitioner

Operator Classification. In DSPSs, a DSPA is usually modeled as a directed acyclic graph (DAG), expressed as $G = (O, D)$. In a DAG, a vertex represents an operator o_i ($o_i \in O$) for data processing and an edge represents the data stream d_{ij} ($d_{ij} \in D$), which flows from operator o_i to o_j. While the DSPA is running, the total input rate of the operator o_u^+ is $r_{in}(u)$. Then it sends the processed data to its downstream operators. The data output rate to each downstream operator is $r_{out}(u, v)$. We use O_u^- to denote the set of downstream operators.

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08751-6_53

There are various operators in the DAG, such as transformation, union, filtering operators, and so on. The selectivity of an operator is defined as the ratio between the data output rate and the total input rate. We divide the operators into two categories: stable operators and unstable operators, based on their selectivity as follows.

$$\begin{cases} o_u^+ \textit{ is stable} & \forall o_v^- \in O_u^-,\ r_{out}(u,v)/r_{in}(u) \textit{ is constant} \\ o_u^+ \textit{ is unstable} & \exists o_v^- \in O_u^-,\ r_{out}(u,v)/r_{in}(u) \textit{ is variable} \end{cases}$$

When the operator is deployed to DSPSs, we can infer its theoretical selectivity based on the operator's data processing logic, called the static selectivity. For the theoretically unstable operator, we can get the actual selectivity when the operator is running, called the dynamic selectivity.

Input and Output. We use $s_s(o_i)$ to denote the static selectivity metrics of the operator o_i, and $s_d(o_i)$ to represent the dynamic selectivity statistic metrics of unstable operators from MetricDatabase. So we use the dataset $D_{s_s} = \{s_s(o_1), s_s(o_2), ...\}$ and $D_{s_d} = \{s_d(o_j), s_d(o_k), ...\}$ as the input. We use the operator partition results $\phi(p_i^t) = \{o_j, o_k, ...\}$ as the output, where p_i^t denotes the ith partition at time t.

Partitioning Module. We propose an adaptive operator partitioning algorithm, called AOPA, to partition the DAG during runtime. Our AOPA consists of two phases: the startup and running phase, as shown in Algorithm 1. In the startup phase, we use the static selectivity metrics of operators to solve the cold-start problem of partitioning. At first, we divide the operators into stable and unstable sets according to the static selectivity. Then, we cut off all the input edges of their downstream operators for the operators in the unstable set. At last, we find all connected sub-graphs based on the result of the second step and aggregate them into a partition. In the running phase, we analyze the dynamic selectivity statistic metrics of unstable operators during runtime and update stable and unstable sets to re-partition the DAG.

In AOPA, we use the trend of online statistics to judge the dynamic selectivity of operators and partition the DAG more accurately. We also present a solution for the cold-start problem.

3.3 OLPredictor: Online Load Predictor

Input and Output. We predict the multi-step load of partition in the future with the past multi-step load. We use the dataset $X_p^t = \{L_p^{t-W+1}, L_p^{t-W+2}, ..., L_p^t\}$ as the input, in which $L_p^t = (l_p^{t-h+1}, l_p^{t-h+2}, ..., l_p^t)$. In particular, we use t to represent the current time, h to represent the length of historical time window and W to represent the number of continuous load sequences. So we use l_p^t as the load of partition p at time t, L_p^t as the load sequence of partition p over the past h time period and X_p^t as continuous multi-step sequences of load with size of

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08751-6_53

W. Besides, we use the dataset $Y_p^t = \{l_p^{t+1}, l_p^{t+2}, ..., l_p^{t+f}\}$ as the output, which denotes the load of partition p in the future f time period. In addition, we use the Min-Max scaler to normalize all the load metrics to the range [0,1].

Algorithm 1 AOPA.

***Step* 1**

for *operator* ∈ *operatorSet* **do**
  if *operator* is theoretically stable **then**
    stableSet.put(*operator*)
  else
    unstableSet.put(*operator*)
  end if
end for

***Step* 2**

for *operator* ∈ *unstableSet* **do**
  for *downOperator* ∈ *operator*.allDownOperators() **do**
    downOperator.inactivateAllInputs()
  end for
  operator.inactivateAllOutputs()
end for

***Step* 3**

minimalClusters = NewEmptyClusterCollection
for *operator* ∈ *operatorSet* **do**
  cluster = *minimalClusters*.findOrBuildNewRelatedCluster(*operator*)
  cluster.add(*operator*.allActivatedDownOperators ())
  cluster.add(*operator*.allActivatedUpOperators ())
  for *cOperator* ∈ *cluster* **do**
    if *cOperator* ∈ *minimalCluster* **then**
      combine(*minimalClusters*.getBeforeCluster(*cOperator*),*cluster*)
    end if
  end for
end for

***Step* 4**

while the DSPA is running **do**
  for *operator* is theoretically unstable **do**
    if *operator* ∈ *unstableSet* AND *operator* is stable in the recent period **then**
      unstableSet.remove(*operator*), *stableSet*.put(*operator*)
    end if
    if *operator* ∈ *stableSet* AND *operator* is unstable in the recent period **then**
      stableSet.remove(*operator*), *unstableSet*.put(*operator*)
    end if
  end for
  if *unstableSet* is changed **then**
    repeat ***Step* 2** and ***Step* 3**
  end if
end while

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08751-6_53

Prediction Networks. The load of each partition is a time series that exhibits long-term trends and short-term fluctuation. In order to capture the short-term fluctuation characteristics and realize accurate prediction of future load online, the prediction method is required to update the model quickly and accurately to give the latest inference results after receiving new data at each moment .

Compared with the existing methods, the meta-learning method [10], combining the meta-learner and base-learner, can not only learn the long-term regular characteristics of data but also capture the short-term unique characteristics, showing powerful nonlinear generalization ability. The Meta-LSTM method [11] uses the LSTM model as the meta-learner to guide the convolutional neural network (CNN) base-learner for classification training and achieves good performance. In this paper, we use the Meta-LSTM method for online load prediction. However, the complexity of the CNN model is very high, and it is difficult to quickly update the model after receiving new data. So we use a simple and efficient MLP network as the base-learner. We propose a model combining the LSTM meta-learner with the MLP base-learner.

During the training process, the MLP base-learner trains the arriving small sample of data to learn the short-term fluctuation characteristics. The LSTM meta-learner summarizes the training results in the base learner and then provides the base learner with better initial values on the new data. Specifically, after receiving new data, we calculate the loss function value and loss function gradient value of the MLP base-learner and input them into the LSTM meta-learner to update the cell state. The meta-learner provides updated parameters to the base-learner.

3.4 OPPlanner: Online Partition-based Operator Parallelism Planner

Input and Output. We use the output of the OLPredictor $Y_p^t = \{l_p^{t+1}, l_p^{t+2}, ..., l_p^{t+f}\}$ and the output of the AOPartitioner $\phi(p_i^t) = \{o_j, o_k, ...\}$ as the input of our OPPlanner. We use $\psi(r_i^t) = \{(o_j, n_j), (o_k, n_k), ...\}$ to denote the optimal operator parallelism as the output, in which o_j is the operator in the partition p_i and n_j is the optimal number of operator instances.

Planning Module. For each partition in the DAG, we collect the metrics of load and the optimal parallelism of each operator through experiments. The optimal parallelism of an operator is the minimum number of instances that can handle the load while meeting the QoS requirements. We spend a long time collecting data and the dataset is relatively small. Then we build an operator parallelism planner for each partition. Because the partitions of the DAG change dynamically over time, the planners are also time-varying. Besides, we need to update our planning model as we collect more data.

The relationship between the load and the operator parallelism in a partition is complex and nonlinear. And the dataset we collected is relatively small. In order to improve the accuracy of modeling and avoid overfitting problems, we use

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08751-6_53

the ensemble learning method. The ensemble learning method integrates many weak models to improve the accuracy and robustness, which is effective for small sample learning.

Compared with the boosting models, such as the Adaptive Boosting (Adaboost) and the gradient boosting decision tree (GBDT) [12], the random forest regression (RFR) [7] adopts the bootstrap strategy and has strong anti-overfitting ability, which performs better. So in this paper, we use the RFR to build the relationship between the load and operator parallelism in each partition.

3.5 Scheduler

We use the scheduler from our previous work [9] to place the operators to the appropriate nodes. We build a cost model to evaluate the total cost of all elastic-scaling actions for all operators from the start time t_s to end time t_e. The cost is defined as:

$$W = \sum_{t \in [t_s, t_e]} w_t, \quad w_t = \sum_{o \in V} (c_o^r n_{ot}^r + c_o^u n_{ot}^u + c_o^d n_{ot}^d) \tag{1}$$

Where c_o^r denotes the cost of running an instance of operator o per unit time, n_{ot}^r denotes the number of instances of operator o running at time t, c_o^u denotes the startup cost of an instance of operator o, n_{ot}^u denotes the number of instances of operator o that started at time t, c_o^d denotes the stop cost of an instance of operator o, and n_{ot}^d denotes the number of instances of operator o that stopped at time t.

4 Experiments

4.1 Settings and Datasets

Settings. Our experiments run on a cluster with eight servers. There are two GPU servers and six CPU servers in the cluster. Both GPU servers are comprised of 36 cores Intel Xeon CPU E5-2697 v4 2.30 GHz, 256GB memory, two NVIDIA GeForce GTX 1060ti cards, and 500GB disks. All CPU servers are comprised of 36 cores Intel Xeon CPU E5-2697 v4 2.30 GHz, 256GB memory, and 500GB disks. One GPU server is used to run Job Manager and MetricDatabase, and the other is used to train and evaluate our proposed model. Six CPU servers are used as Task Manager to run the instances of operators on Kubernetes. The version of our Kubernetes is v1.22.0.

Datasets. We build four databases from our online DataDock system [8]. The first database collects the dynamic selectivity statistic metrics of unstable operators. The second dataset collects the load and the optimal operator parallelism metrics of each partition to build the OPPlanner. The third dataset collects the

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08751-6_53

load of each partition in 30 days to build the OLPredictor. We divide the dataset into two sets: a training set (from the beginning to the 20th day), and the test set (from the 21st day to the last day). The fourth dataset collects the resource metrics of each node.

We use a real preprocessing application to evaluate the performance of our algorithm. The DAG of the application is shown in Fig. 4, where the blue operators are unstable and the green are stable. The operators and edges cover the operator classification in Section 3.2.

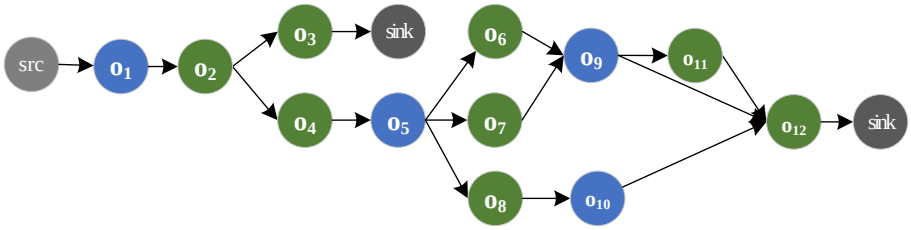


Fig. 4. The DAG of a Real Preprocessing Application.

4.2 Evaluation

To evaluate the performance of resource allocation, we compare our method with the BriskStream [1], ELYSIUM [5] and Pec [6]. The goal of resource allocation is to ensure that the latency meets the QoS requirements while reducing resource utilization. So we use the end-to-end latency guarantee and the total cost to evaluate the performance of our method.

End-to-End Latency Guarantee. We run the preprocessing application on the DataDock [8] and record the end-to-end latency of four models. The results are shown in Fig. 5(a). Compared with other methods, the end-to-end latency of our model is smaller and always stays stable. As a result of considering the dynamic correlation of operators in the DAG, we can accurately adjust the parallelism of operators in units of partitions before the load changes. So we can process data stream load in time. While the BriskStream adjusts the parallelism of an operator after its load changes, which is lagging and increases data processing latency. The ELYSIUM and Pec adjust the parallelism of operators in units of DAG before the load changes, but they refer to the average selectivity to compute the load of operators for parallelism adjustment, which is inaccurate and the error is always larger for the more downstream operators.

Total Cost. We calculate the cost of each operator at each epoch and get the total cost of all operators in a period of epoch. In our experiment, we set $c_o^r = 8000$, $c_o^u = 3000$ and $c_o^d = 3000$ and collect the times of startup and stop action of operators.

We take the baseline of BriskStream to get the relative cost. Fig. 5(b) represents the cumulative relative cost. Fig. 5(c) represents the cumulative startup and stop times of all operators.

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08751-6_53

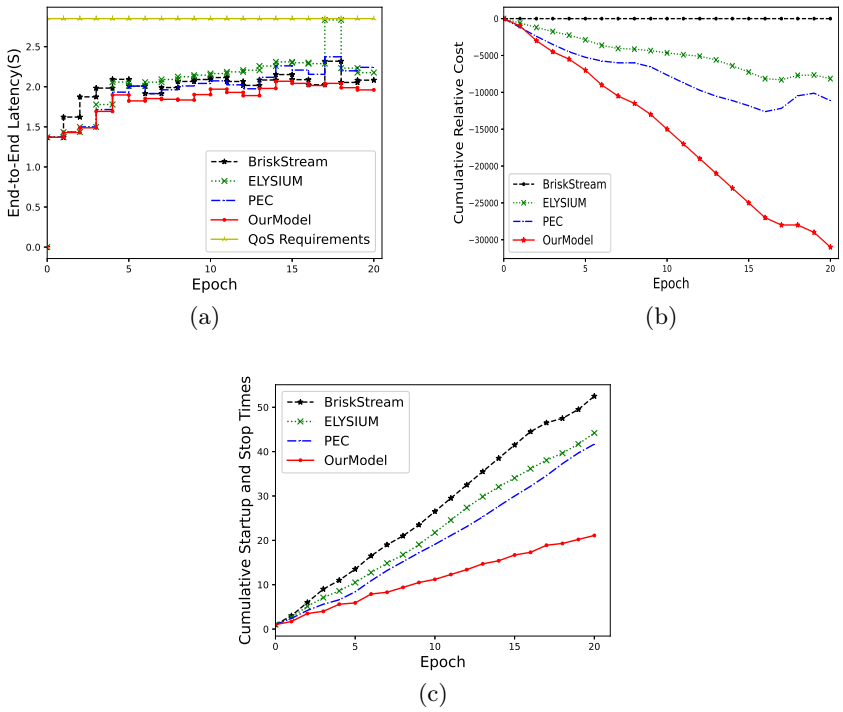


Fig. 5. The Overall Results of Different Methods.

Compared with other methods, our method costs and adjusts less. The reason is that we can accurately adjust the parallelism of operators and reduce the number of startup and stop. Besides, we take the cost of startup or stop into consideration to improve the cost-effectiveness of adjustment.

4.3 AOPartitioner Evaluation

The AOPartitioner refers to the static selectivity metrics to get the initial partitions, which address the cold-start problem of partitioning. The initial partition result is depicted in Fig. 6(b). Then we analyze the collected dynamic selectivity statistic metrics of theoretically unstable operators and find the selectivity of operator o_9 is constant from *epoch* 332 to 450, as shown in the red part of Fig. 6(a). So we re-partition the DAG and the result is depicted in Fig. 6(c).

4.4 OLPredictor Evaluation

To enhance the overall performance, we select the better prediction method in the OLPredictor. So we compare the prediction performance of our Meta-learning based OLPredictor with different methods, including separate LSTM, separate MLP, online support vector regression (SVR), and bidirectional gated

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08751-6_53

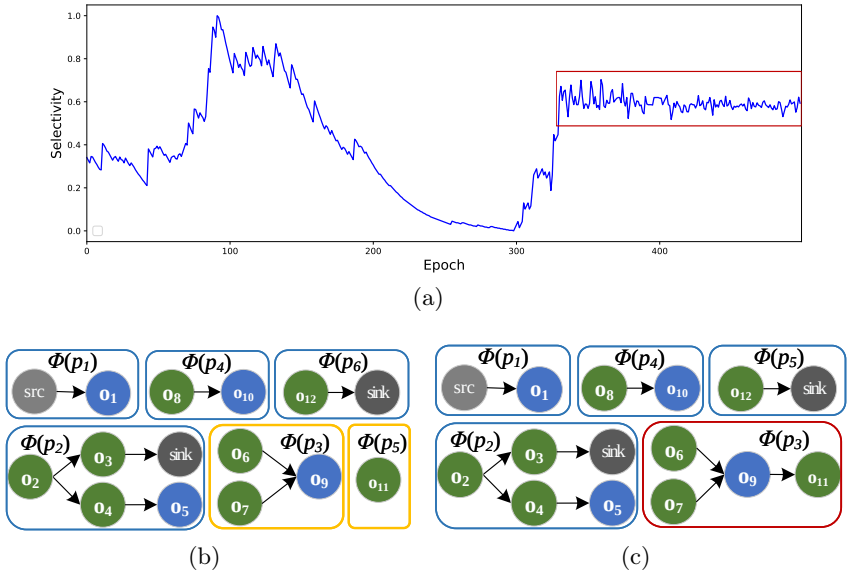


Fig. 6. The Adaptive Partition Results.

recurrent units (BiGRU) [13]. We use the RMSE and the MAE to evaluate the performance.

The main parameters in the compared methods are depicted as follows. In our Meta-learning, we use *adam* as the optimizer, 0.001 as the learning rate and 16 as the size of hidden layer for the meta-learner, and use *relu* as the activation function, *mse* as the loss, and 128 as the size of hidden layer for the base-learner. In the LSTM, MLP, and BIGRU model, we all use *relu* as the activation function, *adam* as the optimizer, *mse* as the loss, and 128 as the size of hidden layer. In the online SVR model, we use 200 as the window size, *rbf* as the kernel, *scale* as the gamma, and c is set to 100.0.

The experimental results of the five-step forward prediction of load are shown in Table 2. For the stable load, these models all get accurate prediction results with small RMSE and MAE. For the periodic load, the neural networks significantly outperform the traditional online SVR, because the neural networks can learn more latent features from historical data. For the fluctuating load, our OLPredictor performs better than other models, since our OLPredictor can not only learn the trend from the historical data, but also quickly capture the short-term fluctuation characteristics.

Table 2. The Five-step ahead Load Prediction Performance of Different Methods.

Method	Stable Load		Periodic Load		Fuctuating Load	
	RMSE	MAE	RMSE	MAE	RMSE	MAE
Online SVR	0.0348	0.0287	0.0456	0.0330	0.0541	0.0469
MLP	0.0339	0.0273	0.0439	0.0385	0.0515	0.0473
LSTM	0.0389	0.0302	0.0426	0.0353	0.0625	0.0513
BiGRU	0.0343	0.0272	0.0414	0.0324	0.0669	0.0549
OLPredictor	**0.0327**	**0.0258**	**0.0408**	**0.0312**	**0.0502**	**0.0406**

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08751-6_53

4.5 OPPlanner Evaluation

To enhance the overall performance, we select the better operator parallelism planning method in the OPPlanner. So we compare the planning performance of different methods, including the RFR model, the SVR model, the Adaboost model, and GBDT model.

The main parameters are depicted as follows. In our RFR based OPPlanner, we set $estimators = 100$. In the SVR model,we set $gamma = 'scale'$, $c = 1.0$, $kernel = 'rbf'$. In the Adaboost model, we set $estimators = 150$, $learning\ rate = 1.0$. In the GBDT model, we set $estimators = 100$, $loss = 'ls'$. Other parameters in the models are set to default values.

The performance of the above methods for partitions in Fig. 6(b) is shown in Table 3. The ensemble learning models, including the RFR, Adaboost, and GBDT, significantly outperforms the SVR model. The reason is that the ensemble learning models integrate many weak models to learn from a small sample effectively. Besides, our OPPlanner performs better than other ensemble learning models, since the bootstrap strategy of RFR avoids overfitting and improves robustness.

Table 3. The Planning Performance of Each Partition of Different Methods.

Partitions		SVR	Adaboost	GBDT	**OPPlanner**
p_1	RMSE	0.0649	0.0642	0.0641	**0.0467**
	MAE	0.0396	0.0370	0.0370	**0.0338**
p_2	RMSE	0.0764	0.0674	0.0634	**0.0591**
	MAE	0.0676	0.0362	0.0327	**0.0312**
p_3	RMSE	0.0758	0.0714	0.0628	**0.0602**
	MAE	0.0572	0.0357	0.0326	**0.0306**
p_4	RMSE	0.0562	0.0553	0.0548	**0.0523**
	MAE	0.0451	0.0214	0.0208	**0.0217**
p_5	RMSE	0.0559	0.0452	0.0541	**0.0437**
	MAE	0.0483	0.0143	0.0214	**0.0210**
p_6	RMSE	0.0794	0.0782	0.0714	**0.0549**
	MAE	0.0452	0.0426	0.0357	**0.0345**

5 Conclusion

In this paper, we present an elastic resource allocation method based on the dynamic perception of operator influence domain. It contains three core modules: the AOPartitioner, the OLPredictor, and the OPPlanner. Firstly, we use the AOPartitioner to adaptively partition the DAG based on the dynamic influence domain of upstream operators. Then we use the OLPredictor based on Meta-learning to get the online multi-step prediction result of load for each partition. At last, we use the OPPlanner to model the load and the optimal parallelism of operators in each partition with RFR. The experimental results illustrate our method is better than the state-of-the-art methods on the real-world datasets. We can ensure that the end-to-end latency meets the QoS requirements while reducing resource utilization.

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08751-6_53

References

1. S. Zhang, J. He, A. C. Zhou, and B. He, "Briskstream: Scaling data stream processing on shared-memory multicore architectures," in *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019* (P. A. Boncz, S. Manegold, A. Ailamaki, A. Deshpande, and T. Kraska, eds.), pp. 705–722, ACM, 2019.
2. W. Mu, Z. Jin, W. Zhu, F. Liu, Z. Li, Z. Zhu, and W. Wang, "Qescalor: Quantitative elastic scaling framework in distributed streaming processing," in *Computational Science - ICCS 2020 - 20th International Conference, Amsterdam, The Netherlands, June 3-5, 2020, Proceedings, Part I* (V. V. Krzhizhanovskaya, G. Závodszky, M. H. Lees, J. J. Dongarra, P. M. A. Sloot, S. Brissos, and J. Teixeira, eds.), vol. 12137 of *Lecture Notes in Computer Science*, pp. 147–160, Springer, 2020.
3. M. Borkowski, C. Hochreiner, and S. Schulte, "Minimizing cost by reducing scaling operations in distributed stream processing," *Proc. VLDB Endow.*, vol. 12, no. 7, pp. 724–737, 2019.
4. B. D. T. Hung, T. Omori, and A. Ohnishi, "Ripple effect analysis of data flow requirements," in *Proceedings of the 14th International Conference on Software Technologies, ICSOFT 2019, Prague, Czech Republic, July 26-28, 2019* (M. van Sinderen and L. A. Maciaszek, eds.), pp. 262–269, SciTePress, 2019.
5. F. Lombardi, L. Aniello, S. Bonomi, and L. Querzoni, "Elastic symbiotic scaling of operators and resources in stream processing systems," *IEEE Trans. Parallel Distributed Syst.*, vol. 29, no. 3, pp. 572–585, 2018.
6. X. Wei, L. Li, X. Li, X. Wang, S. Gao, and H. Li, "Pec: Proactive elastic collaborative resource scheduling in data stream processing," *IEEE Trans. Parallel Distributed Syst.*, vol. 30, no. 7, pp. 1628–1642, 2019.
7. L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.
8. 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*, pp. 120–131, 2019.
9. F. Liu, Z. Jin, W. Mu, W. Zhu, Y. Zhang, and W. Wang, "Droallocator: A dynamic resource-aware operator allocation framework in distributed streaming processing," in *Network and Parallel Computing - 17th IFIP WG 10.3 International Conference, NPC 2020, Zhengzhou, China, September 28-30, 2020, Revised Selected Papers* (X. He, E. Shao, and G. Tan, eds.), vol. 12639 of *Lecture Notes in Computer Science*, pp. 349–360, Springer, 2020.
10. S. Thrun, "Lifelong learning algorithms," in *Learning to Learn* (S. Thrun and L. Y. Pratt, eds.), pp. 181–209, Springer, 1998.
11. S. Ravi and H. Larochelle, "Optimization as a model for few-shot learning," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, OpenReview.net, 2017.
12. J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *The Annals of Statistics*, vol. 29, no. 5, pp. 1189–1232, 2001.
13. Y. Qin, D. Song, H. Chen, W. Cheng, G. Jiang, and G. W. Cottrell, "A dual-stage attention-based recurrent neural network for time series prediction," in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pp. 2627–2633, 2017.

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08751-6_53