QEScalor: Quantitative Elastic Scaling Framework in Distributed Streaming Processing

Weimin Mu[1,2], Zongze Jin[1(✉)], Weilin Zhu[1], Fan Liu[1,2], Zhenzhen Li[1,2], Ziyuan Zhu[1], and Weiping Wang[1]

[1] Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
{muweimin,jinzongze,zhuweilin,liufan,lizhenzhen,zhuziyuan, wangweiping}@iie.ac.cn

[2] School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China

Abstract. Recently, researchers usually use the elastic scaling techniques as a powerful means of the distributed stream processing systems to deal with the high-speed data stream which arrives continuously and fluctuates constantly. The existing methods allocate the same amount of resources to the instances of the same operator, but they ignore the correlation between the operator performance and resource provision. It may lead to the waste of the resources caused by the over-provision or the huge overhead of the scheduling caused by the under-provision. To solve the above problems, we present a quantitative elastic scaling framework, named QEScalor, to allocate resources for the operator instances quantitatively based on the actual performance requirements. The experimental results show that compared with the existing works, the QEScalor can not only achieve resource-efficient elastic scaling with lower cost, but also it can enhance the total performance of the DSPAs.

Keywords: Data stream processing · Elastic scaling · Random forest

1 Introduction

Recently, the distributed stream processing systems (DSPSs) [1–4] offer a powerful means to extract the valuable information from the data streams in time. We usually use the directed acyclic graph (DAG) [5] to model the data stream processing application (DSPA) in DSPSs. In the DAG, each vertex represents a kind of operations, named as the operator, and each edge represents a data stream between two operators. At the run time, the DSPS initiates a certain number of operator instances for each operator and deploy them on the runtime environment.

Considering the constant fluctuating data stream, we adopt the elastic scaling in the DSPSs, which adjusts the number of the operators dynamically, to satisfy the QoS requirements. There have been many researches on the elastic scaling.

V. V. Krzhizhanovskaya et al. (Eds.): ICCS 2020, LNCS 12137, pp. 147–160, 2020.
https://doi.org/10.1007/978-3-030-50371-0_11

Zacheilas et al. [6] adjusts the number of operators based on the state transition graph. Hidalgo et al. [7] evaluates the processing power of the operator through the benchmarking, and adjusts the number of the operator instances based on the threshold and the workload prediction. Wei et al. [8] only adjusts the CPU frequency of the virtual machines as the workload fluctuates to reduce the energy cost. Marangozova-Martin et al. [9] presents the method to allocate three levels of resources to the operator instances including virtual machines, processes and threads. The above methods allocate the same amount of resources to the instances of the same operator, but they ignore the correlation between the operator performance and resource provision. Actually, the unreasonable resources provision for the operator instance will cause some severe problems. For example, over-provision will result in a waste of resources. Besides, the under-provision means that the DSPSs will create a lot of instances to achieve high processing performance, which results in the huge overhead of the scheduling and the state transition.

In this paper, we present a quantitative elastic scaling framework, named QEScalor, to allocate the resources for the operator instances quantitatively based on the actual performance requirements. This framework firstly builds the operator performance and resource provision model (OPRPM), then it generates the low-cost elastic scaling plan based on the OPRPM. The contributions of this paper are as follows:

- We use the QEScalor, which first considers the correlation between the operator performance and resource provision, to enhance the performance of the resource provision.
- We propose an online algorithm, named DSA. It learns the correlation samples of the operator performance and resource provision based on the gradient strategy and re-sampling mechanism. Besides, we use the random forest regression model (RFR) [10] to build the OPRPM to get the suitable resource provision options for the operator performance requirement.
- We present a quantitative and cost-based elastic scaling algorithm, QCESA. It refers to the prediction of the workload [11] and the operator performance [12] to generate the low-cost scaling plan based on the OPRPM to achieve the resource-efficient elastic scaling to improve the performance.
- We implement the QEScalor as a key part of our DataDock [11]. The experiment results show that our QEScalor can enhance the performance with the lower scaling cost on the real-world datasets.

We organize the rest of our paper as follows. Section 2 describes the design of QEScalor. Section 3 shows the experimental results of our framework. Finally, Sect. 4 concludes our paper.

2 System Design

2.1 Overview

In this subsection, we describe our framework QEScalor in detail in Fig. 1, which contains three modules: the online operator performance sampler (OOPSer),

the operator performance and resource provision modeler (OPRPMer) and the quantitative elasticity controller (QECer). As is shown in Fig. 1, the QEScalor is an important module in the DataDock. In our previous work [11], we present our distributed stream processing system, DataDock, mainly aiming at processing the heterogeneous data in real-time.

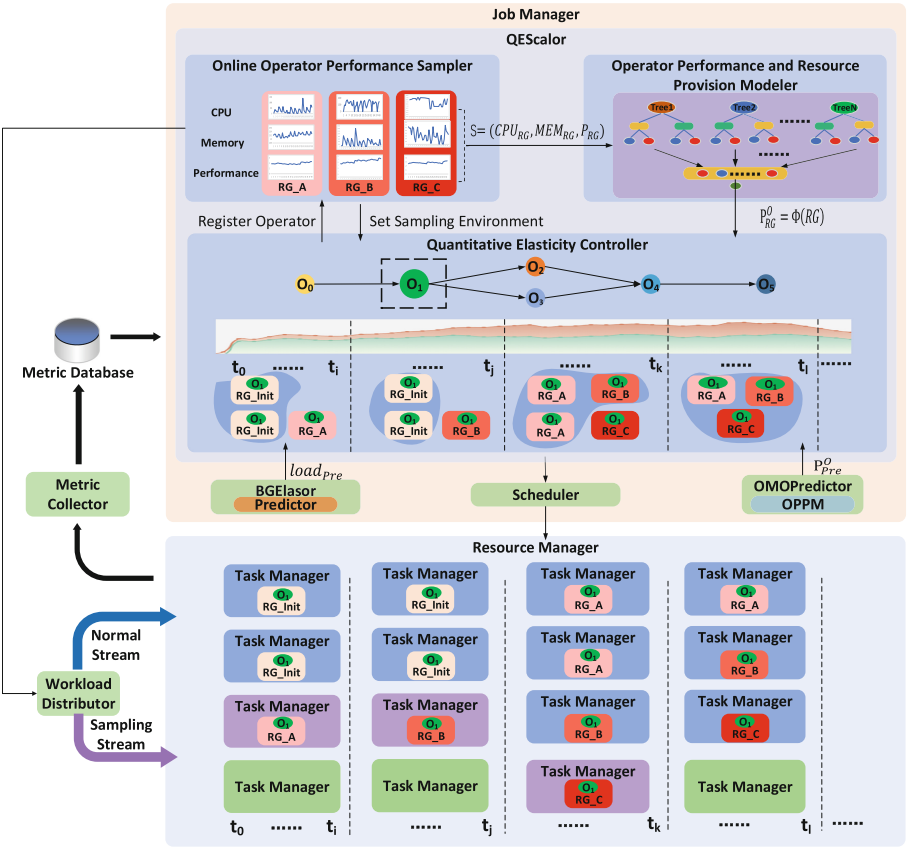


Fig. 1. QEScalor architecture

We use the OOPSer to learn the correlation samples of the operator performance and resource provision online. Then we use these samples as the input of the OPRPMer to build the operator performance and resource provision model (OPRPM). At last, we use the QECer to adjust the scaling plan according to the workload prediction of the BGElasor [11], the operator performance prediction of the OMOPredictor [12] and the OPRPM.

As is shown in Fig. 1, we take the DSPA including the new operator O_1 as an example to describe the work process of the QEScalor. The process contains three main stages.

- **Correlation Samples Online Learning.** When we use the QECer to execute the DSPA including the new operator O_1 at time t_0, it registers the O_1 on the OOPSer. The QECer starts enough instances of the O_1 with the default resources provision based on the cost-based elastic-scaling algorithm [11] and a single sampling instance with the RP_A depended on the OOPSer. Then the OOPSer interacts with the Stream Distributor (SD) to allocate the workload between the normal instances and the sampling instance. The OOPSer continues to collect the operator performance and resource utilization metrics until the current sampling process convergences at time t_i. The above sampling process will continue several rounds based on the gradient strategy until the operator performance no longer increases with the growth of resources. For example, the OOPSer learns the correlation samples of resource group RP_A, RP_B and RP_C from time t_0 to t_k.
- **Operator Performance and Resource Provision Modeling.** During the sampling process, we use the OOPSer to invoke the OPRPMer to build the OPRPM using the RFR, when it completes the learning process of one kind of source provision.
- **Quantitative Elastic Scaling.** We run the QECer periodically. It takes the workload prediction of the BGELasor and the operator performance prediction of the OMOPredictor as the input and makes the quantitative scaling decision based on the OPRPM. As is shown in Fig. 1, from time t_0 to t_j, the OPRPMer has learned the OPRPM of resource provision RP_A and RP_B. To process the workload from t_j to t_k, the QECer allocates two normal instances with the resource provision RP_A and one instance with the resource provision RP_B. And from t_k to t_l, the QECer allocates three instances with the resource provision RP_A, RP_B and RP_C respectively.

2.2 OOPSer: Online Operator Performance Sampler

We adopt the OOPSer to collect the correlation samples of the operator performance and resource provision to build the OPRPM. We focus on the two types of resources: the CPU and the memory. We do not consider the network bandwidth in our work, because the network bandwidth is more sufficient and cheaper than the CPU and the memory in the data center. Therefore, we only consider the correlation samples learning of two types of operators [12]: the compute-intensive Operator (COperator) and the compute-intensive operator mixed with the memory I/O (CMOperator).

We propose the Dynamic-Sampling-Algorithm (DSA) to collect the correlation samples online. We show it in Algorithm 1. The sampling process consists of three steps: Firstly, we create the sampling operator instance with specific resource provision. Secondly, we do not stress test on the sampling operator

instance until the performance of the operator instance converges. Thirdly, we continuously collect the correlation samples during the test. Since we spend some time in stress testing, it will take a long time to complete the sampling. In order to speed upsampling, we dynamically adjust the sampling step according to the gradient of the performance change of the sampling operator instance. Meanwhile, we use the re-sampling method to add sampling points when the operator performance fluctuates to improve accuracy.

We are processing the online workload while sampling with the normal running of the DSPAs. We dynamically allocate the workload between the normal instances and the single sampling instance. It can reduce the time and resource overhead obviously compared with running the sampling alone.

When we complete the sampling, we get the correlation sample set $CS_o = (cs_o^0, cs_o^1, ..., cs_o^i)$, where $cs_o^i = (cpu_o^i, mem_o^i, p_o^i)$ is the correlation sample of the resource provision (cpu_o^i, mem_o^i) and the corresponding operator performance p_o^i. We use the CS_o to build the OPRPM of the operator o. However, the performance of operators fluctuates during the life cycle of the DSPA. With the online correlation samples learning mechanism, we can continuously collect the samples and update the OPRPM when the performance of the operator is inconsistent with the OPRPM.

Algorithm 1. DSA

sampling(start, step, delta)

```
1: stopCount = 0
2: step = step.min
3: cur = start
4: sample(cur)
5: while  stopCount < maxStopCount do
6:    cur += step
7:    sample(cur)
8:    rate = (curload - lastload) * minStep / (cur - last) / lastload
9:    if abs(rate) < delta.stop then
10:      stopCount.increase()
11:   else
12:      stopCount = 0
13:   end if
14:   if rate < 0 or rate > upDelta then
15:      re-sample ( sample.last , cur )
16:   else
17:      step = min(step.min * delta.up / rate), step.max)
18:   end if
19: end while
```

2.3 OPRPMer: Operator Performance and Resource Provision Modeler

We use the OPRPMer to build the model of the operator performance and the resource provision, with which we can predict the operator performance based on the given resource provision.

The correlation between operator performance and resource provision is commonly complex and nonlinear. The linear regression model can not capture the latent features of the correlation well, resulting in bad prediction. Besides, in our scenario, the correlation samples set CS_O is commonly small. Using the single nonlinear regression model, like the SVR [13], leads to overfitting easily. The ensemble learning model can improve the robustness of prediction by integrating many weak classifiers, which is more suitable for small sample learning. We adopt the random forest regression (RFR) model in the OPRPMer to capture the nonlinear correlation between the operator performance and the resource provision. According to the experiments, compared to the boosting models, such as the Adaboost [14], GBDT [15] and XGBoost [16], the RFR model performs better. Because the bootstrap strategy adopted by RFR model can avoid overfitting effectively when the sample set is small.

We take the correlation sample set CS_O learned by the OOPSer as the input to build the model in the OPRPMer. When invoked by the QECer, the OPRPMer takes $r = (cpu, mem)$ as input to get the operator performance prediction p_o corresponding to the r.

2.4 QECer: Quantitative Elasticity Controller

In this section, we build the QECer, which can ensure the end-to-end latency with the minimum elastic-scaling cost. It contains two parts: the Cost Model and Quantitative & Cost-based Elastic Scaling Algorithm (QCESA).

Cost Model. We build a cost model to evaluate the total cost of all elastic-scaling actions for an operator from the current epoch S to the future epoch F. The total cost $W_o(\mathbf{Ins})$, the startup times $C^u_{o\cdot t}(Ins)$ and the shutdown times $C^d_{o\cdot t}(Ins)$ are defined as:

$$\begin{aligned} \min \quad & W_o(\mathbf{Ins}) = \min(\sum_{t_S}^{t_F} \sum_{r \in Res} |Ins^t_{o\cdot r}| p^r_o + \sum_{t_S}^{t_{F-1}} (p^u_o C^u_{o\cdot t}(Ins) + p^d_o C^d_{o\cdot t}(Ins))) \\ \text{s.t.} \quad & \sum_{r \in Res} |Ins^t_{o\cdot r}| Perf_{o\cdot r} \geq Workload_t, \quad \forall t \in [t_S, t_F] \end{aligned} \tag{1}$$

$$C^u_{o\cdot t}(\mathbf{Ins}) = \sum_{r \in Res} \max(0, Ins^{t+1}_{o\cdot r} - Ins^t_{o\cdot r}) \tag{2}$$

$$C^d_{o\cdot t}(\mathbf{Ins}) = \sum_{r \in Res} \max(0, Ins^t_{o\cdot r} - Ins^{t+1}_{o\cdot r}) \tag{3}$$

where p_o^r is the cost of system resources used by the single instance with resource r for the operator o. $|Ins_{o\cdot r}^t|$ is the instance number of operator o with resource r at time t. In addition, $Perf_{o\cdot r}$ denotes the performance of each operator and $\sum_{r\in Res}|Ins_{o\cdot r}^t|Perf_{o\cdot r}$ denotes the total performance of operator o at time t. $Workload_t$ is the workload at epoch t. In order to satisfy the end-to-end latency, we ensure that the performance of each operator is not less than the workload. In other word, $\sum_{r\in Res}|Ins_{o\cdot r}^t|Perf_{o\cdot r} \geq Workload_t$ at any time. And p_o^u is the startup-cost of a single o instance. p_o^d is the shutdown-cost of a single o instance.

QCESA. To solve this expression $min(W_o)$, we propose the Quantitative and Cost-based Elastic Scaling Algorithm (QCESA). We show it in Algorithm 2. The QCESA considers not only the cost of instance startup and shutdown, but also the correlation of operator performance and resource provision. We use the QCESA to balance these parts of the cost to guarantee a low cost.

At first, we use the QCESA to compute the max workload $workload_{max}$ during $t \in [t_C, t_F]$. Then use it to calculate all candidates at all time $t \in [t_C, t_F]$. Each candidate is a combination of instances with different resource provision and instance number, of which the total performance $Perf_{cand\cdot total}^t \in [Workload_t, Workload_{max}]$. At last, we use dynamic programming to calculate the minimal cost.

3 Experiments

3.1 Environment

Settings. Our experiments run on Kubernetes (K8S) cluster, which we use as the Resource Manager on the DataDock, including eight servers. The version of K8S is 1.14.1. There are two types of servers in the K8S cluster: two GPU servers and six CPU servers. Each GPU server comprises 36 cores Intel Xeon CPU E5-2697 v4 2.30 GHz, 256 GB memory, two NVIDIA GeForce GTX 1060ti cards, and 500 GB disks. Each CPU server comprises 36 cores Intel Xeon CPU E5-2697 v4 2.30 GHz, 256 GB memory, and 500 GB disks. We use the GPU servers to run the JobManager, conducting the training and evaluation. We adopt the CPU servers to run the Task Manager, in which the operator instance runs. Besides, we conduct the evaluation of the OPRPMer with sklearn 0.22.1 running on python 3.7.

Datasets. In Table 1, we show our datasets and the intermediate results of our model at different stages. Firstly, we present the We use the real online workload processed by the DataDock in a day as the original dataset (OriWL-1day). In OOPser, we use two sampling algorithms which are FSSA and DSA to sample OriWL-1day. F1, F2, F3, F4 and F5 denote different steps of FSSA. A, B, C and D represent that the original dataset is processed by these operators. CO1 denotes CPU operator and CMO1 denotes CPU-Memory operator. In OPRPMer, we use DSA-A/B/C/D as the train set and use F1-A/B/C/D as the test set. Then we obtain the output of the random forest regression (RFR),

Algorithm 2. QCESA

```
schedule( load, res )
 1: cand = loopCandidatesAtEachTime(load.max, load)
 2: if load.size == 1 then
 3:     plan.cost = res.start.cost
 4:     plan.setplan(res.start.index, res.start)
 5:     return plan
 6: end if
 7: if load.size == 2 then
 8:     if load[0] < load[1] then
 9:         plan.cost = res.end.cost + calcWarmup( res.start , res.end )
10:         plan.setplan(res.start.index, res.end)
11:         return plan
12:     else
13:         plan.cost = res.start.cost
14:         plan.setplan(res.start.index, res.start)
15:         return res.end
16:     end if
17: end if
18: plan.cost=max
19: for cur : load do
20:     for cand :cur.candidates do
21:         lplan = schedule( load.before(cur), res.with(cand))
22:         rplan = schedule( load.after(cur), res.with(cand))
23:         if plan.cost > lplan.cost+rplan.cost then
24:             plan.setplan( lplan,cur,rplan)
25:             plan.cost = lplan.cost+rplan.cost
26:         end if
27:     end for
28: end for
29: return plan
loopCandidatesAtEachTime(maxload, load)
 1: for  curload : load do
 2:     for cand : res  do
 3:         if  curload < cand.load < maxload  then
 4:             candidates.add(curload, cand)
 5:         end if
 6:     end for
 7: end for
 8: return candidates
```

DSA-RFR-A/B/C/D. In QECer, we should evaluate the system performance. From OPRMER, we use DSA-RFR-A/B/C/D as the input. From OMOPredictor, we adopt OP-PM-30 as the input, which represents the operator performance on Datadock online for 30 days. For the input of the BGElasor, we use OriWL-60days-FlowStat which represents the flow statistics for 60 days of data load on the DataDock online.

Table 1. The datasets description

Stage	IN or OUT		Datasets					
OOPSer	IN		OriWL-1day					
	OUT	CO1[A]	F1-A	F2-A	F3-A	F3-A	F5-A	DSA-A
		CO2[B]	F1-B	F2-B	F3-B	F3-B	F5-B	DSA-B
		CMO1[C]	F1-C	F2-C	F3-C	F3-C	F5-C	DSA-C
		CMO2[D]	F1-D	F2-D	F3-D	F3-D	F5-D	DSA-D
OPRPMer	IN	Train	DSA-A/B/C/D					
		Test	F1-A/B/C/D					
	OUT		DSA-RFR-A/B/C/D					
QECer	IN	From OPRMer	DSA-RFR-A/B/C/D					
		From OMOPredictor	OP-PM-30					
		From BGElasor	OriWL-60days-FlowStat					
	OUT		Resource Allocation Plan					

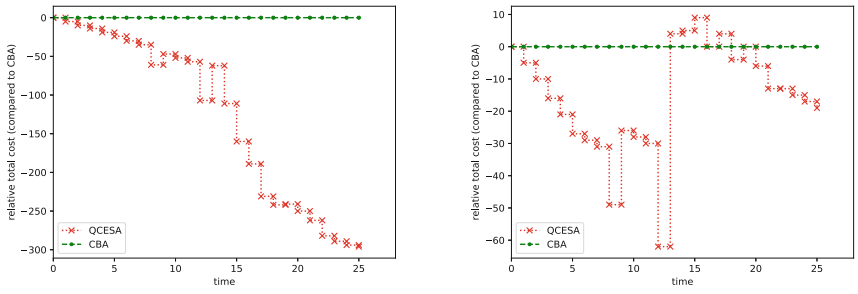
3.2 Performance Evaluation

In our algorithm, we guarantee the latency to reduce the total cost. Thus, we evaluate the Quantitative Elasticity Controller from two aspects: the total cost and the end-to-end latency guarantee. We use the Cost-Balance-Algorithm (CBA) [11] as the baseline algorithm. The CBA considers the running cost and the operation cost. Compared to the CBA, the QCESA takes the operator resource provision into account.

The performance of QCESA depends on the sampling of OOPSer and the predicted results of OPRPMer. For the OOPSer, we compare our method, DSA, with the Fixed-Step-Sampling-Algorithm (FSSA) to demonstrate that DSA is more accurate in the sampling stage to enhance scheduling accuracy and reduce the cost. For evaluating the OPRPMer, we compare the random forest regression model (RFR) with the following methods: Adaboost, GBDT and XGBoost, to demonstrate that RFR is more suitable for the current application scenarios. It can get more accurate prediction results and affect the overall performance of scheduling.

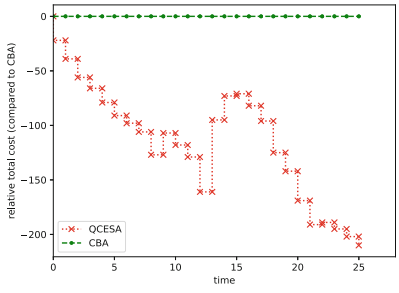
Total Cost. In this part, we take the CMO1 as an example to compare by using the total cost of elastic scaling. We use the workload prediction to generate the scaling plan for the CMO1 and calculate the total cost.

Moreover, to evaluate the effectiveness of the QCESA, we use four different resource provisions to test the CBA respectively. The four resource provision granularities are as follows: 1) $r_1 = (cpu = 0.6 * core, mem = 33.2\,\text{MB})$, 2) $r_2 = (cpu = 1.2 * core, mem = 33.3\,\text{MB})$, 3) $r_3 = (1.8 * core, mem = 33.5\,\text{MB})$, 4) $r_4 = (cpu = 2.4 * core, mem = 33.8\,\text{MB})$.

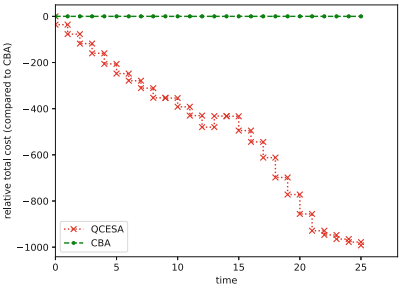


(a) QCESA vs. CBA with r_1

(b) QCESA vs. CBA with r_2



(c) QCESA vs. CBA with r_3



(d) QCESA vs. CBA with r_4

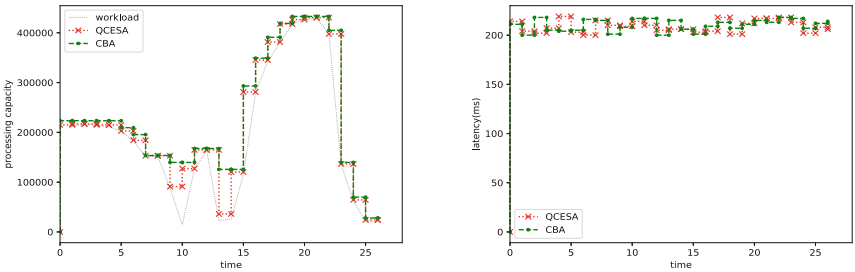
Fig. 2. Total cost

In Fig. 2, we can find that most of the time, the total cost of the QCESA is less than that of the CBA. Besides, as the system runs, the performance of the QCESA is becoming much higher than the CBA.

End-to-End Latency Guarantee. In this part, we focus on the end-to-end latency guarantee. We still take the CMO1 as an example to run on the DataDock and monitor the end-to-end latency.

In Fig. 3, we can see that both the QCESA and the CBA can guarantee that the performance of the operator is no less than the workload. And the end-to-end latency always stays stable and satisfies the requirement of the QoS. The reason is that both algorithms start the instances before the workload rises. Thus they can process the workload timely.

Impact of Sampling. To measure the impact of sampling, we compare the DSA with the Fixed-Step-Sampling-Algorithm (FSSA) in OOPSer. We run the FSSA and the DSA separately to collect the correlation samples of the four operators. For each operator, we run the DSA with the sampling step set to 1 and use the sampling result as the baseline. Besides, we also run the FSSA with the sampling step set to 2, 3, 4 and 5 as the contrast evaluation.



(a) workload and performance (b) end-to-end Latency

Fig. 3. QoS guarantee

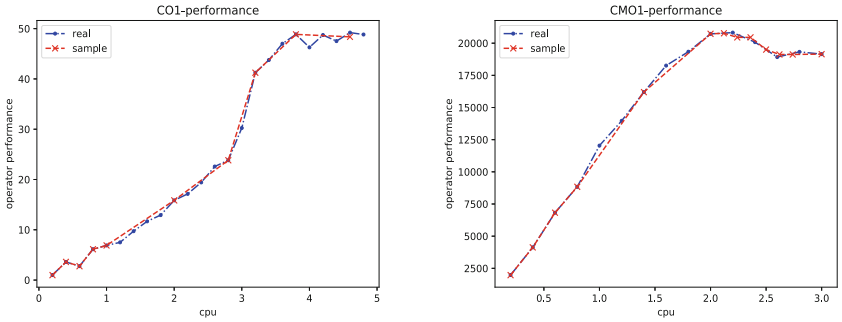
After we get all correlation samples of four operators with the FSSA and the DSA, we use the OPRPMer to build the OPRPMs. Then, we predict the operator performance based on the minimum sampling step using the OPRPMs. We use the Root Mean Square Errors (RMSE), the Mean Absolute Errors (MAE) and the Sampling Number (SN) to evaluate the effectiveness of the DSA. $RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{x}_i)^2}$, $MAE = \frac{1}{n}\sum_{i=1}^{n}|x_i - \hat{x}_i|$, where x_i is the operator performance of baseline, and $\hat{x}_i$ is the predicted operator performance.

Table 2. The performance of each sampling method

Algorithm	Operator type											
	COperator						CMOperator					
	CO1			CO2			CMO1			CMO2		
	RMSE	MAE	SN	RMSE	MAE	SN	RMSE	MAE	SN	RMSE	MAE	SN
$FSSA_2$	0.0047	0.0596	16	0.0031	0.0393	17	0.0058	0.0519	12	0.0199	0.1231	19
$FSSA_3$	0.0049	0.0455	11	0.0015	0.0305	12	0.0074	0.0613	9	0.0067	0.0563	13
$FSSA_4$	0.0066	0.0599	9	0.0033	0.0521	9	0.0247	0.1434	7	0.0233	0.0924	10
$FSSA_5$	0.0237	0.1131	7	0.0026	0.0423	8	0.0192	0.1157	6	0.0161	0.0938	9
DSA	**0.0046**	**0.0568**	**14**	**0.0022**	**0.0402**	**11**	**0.0056**	**0.0525**	**15**	**0.0112**	**0.0911**	**9**

As is shown in Table 2, we can observe that the performance of the FSSA is not stable. When the step of the FSSA is 2, the CO1 and the CMO1 get the best performance. But when the step of the FSSA is 3, the CO2 and the CMO2 get the best performance. And the DSA performs well for all the four operators. Its performance is close to or even reaches the best performance. Moreover, the DSA has fewer sampling numbers when reaching the same performance. It benefits from the dynamical sampling strategy.

We show the sampling result of the CO1 and the CMO2 in Fig. 4. We can see where the performance fluctuates obviously, the sampling step is close to the minimum sampling step. Instead, when the performance changes smoothly, the DSA only uses a few sampling points to capture the main characteristics of performance changes.



(a) sampling of CO1 (b) sampling of CMO1

Fig. 4. Sampling result

Comparison of Prediction Methods. To enhance the total performance, we should select the better prediction method for the OPRPMer. So we compare the random forest regression model with the following methods: Adaboost, GBDT and XGBoost, to demonstrate the effectiveness of the RFR model in this scenario.

We use the RMSE and the MAE to evaluate the performances of each model. There are several hyper-parameters in these approaches, we use the grid search and 10-folds cross-validation to select the key hyper-parameters. Besides, we normalize all the input to the range [0,1] using the Min-Max scaler. We repeat the experiment 10 times for each model to reduce the random experimental error and take the average of the whole test results as the final result.

As for the RFR, we set $bootstrap = True, criterion = 'mse', max\_features = 'auto', min\_samples\_leaf = 1, min\_samples\_split = 2, n\_estimators = 100$. As for the SVR, we set $kernel = 'rbf', gamma = 'scale', C = 1.0$. As for the Adaboost, we set $base\_estimator = None, learning\_rate = 1.0, loss = 'linear', n\_estimators = 50$. As for the GBDT, we set $n\_estimators = 100, criterion = 'friedman\_mse', max\_features = None, min\_samples\_leaf = 1$. As for the XGBoost, we set $booster = 'gbtree', learning\_rate = 0.1, max\_depth = 3, n\_estimators = 100$.

Table 3. The performance of each models

Model	Operator type							
	COperator				CMOperator			
	CO1		CO2		CMO1		CMO2	
	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE
SVR	0.0924	0.0833	0.1048	0.1040	0.0833	0.0819	0.0833	0.0819
Adaboost	0.0678	0.0514	0.0469	0.0390	0.0622	0.0599	0.0598	0.0575
GBDT	0.0687	0.0536	0.0550	0.0361	0.0821	0.0646	0.0706	0.0565
XGBoost	0.0722	0.0561	0.0522	0.0431	0.0986	0.0860	0.0986	0.0860
RFR	**0.0435**	**0.0372**	**0.0220**	**0.0197**	**0.0644**	**0.0533**	**0.0477**	**0.0399**

Table 3 shows that the effectiveness of the ensemble learning is significantly better than the SVR. Because in our scenario, the size of the correlation sample set is smaller, the advantage of the ensemble learning is more prominent.

Besides, there is not much difference between the Adaboost, GBDT, and XGBoost. However, compared to the above boosting models, the RFR performs better. Because the RFR model adopts the bootstrap strategy, it can effectively prevent overfitting when the size of the sample set is small.

4 Conclusion

In this paper, we present a quantitative elastic scaling framework, named QEScalor, to allocate resources for the operator instances quantitatively based on the actual performance requirements. It contains three key modules: the OOPSer, the OPRRMer and the QECer. Firstly, we use the OOPSer to learn the correlation samples of the operator performance and resource provision online. Then we use these samples as the input of the OPRPMer to build the operator performance and resource provision model (OPRPM) by using the random forest regression model. At last, we use the QECer to adjust the scaling plan according to the real workload fluctuation. The experimental results show that, compared with the state-of-the-art methods, the QEScalor is better on the real-world datasets. And we can address the problem which ignores the correlation between the operator performance and resource provision.

Acknowledgements. This work is supported by the National Key Research and Development Plan (2018YFC0825101).

References

1. Chandrasekaran, S., et al.: TelegraphCQ: continuous dataflow processing. In: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, p. 668 (2003)
2. Abadi, D.J., et al.: The design of the borealis stream processing engine. In: CIDR 2005, pp. 277–289 (2005)
3. Neumeyer, L., Robbins, B., Nair, A., Kesari, A.: S4: distributed stream computing platform. In: ICDMW 2010, pp. 170–177 (2010)
4. Carbone, P., Katsifodimos, A., Ewen, S., Markl, V., Haridi, S., Tzoumas, K.: Apache flink^{TM}: stream and batch processing in a single engine. IEEE Data Eng. Bull. **38**(4), 28–38 (2015)
5. Storm. http://storm.apache.org/
6. Zacheilas, N., Kalogeraki, V., Zygouras, N., Panagiotou, N., Gunopulos, D.: Elastic complex event processing exploiting prediction. In: 2015 IEEE International Conference on Big Data, Big Data 2015, pp. 213–222 (2015)
7. Hidalgo, N., Wladdimiro, D., Rosas, E.: Self-adaptive processing graph with operator fission for elastic stream processing. J. Syst. Softw. **127**, 205–216 (2017)
8. Wei, X., Li, L., Li, X., Wang, X., Gao, S., Li, H.: Pec: Proactive elastic collaborative resource scheduling in data stream processing. IEEE Trans. Parallel Distrib. Syst. **30**(7), 1628–1642 (2019)

9. Marangozova-Martin, V., Palma, N.D., El-Rheddane, A.: Multi-level elasticity for data stream processing. IEEE Trans. Parallel Distrib. Syst. **30**(10), 2326–2337 (2019)
10. Breiman, L.: Random forests. Mach. Learn. **45**(1), 5–32 (2001). https://doi.org/10.1023/A:1010933404324
11. Mu, W., Jin, Z., Wang, J., Zhu, W., Wang, W.: BGElasor: elastic-scaling framework for distributed streaming processing with deep neural network. In: Tang, X., Chen, Q., Bose, P., Zheng, W., Gaudiot, J.-L. (eds.) NPC 2019. LNCS, vol. 11783, pp. 120–131. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-30709-7_10
12. Mu, W., Jin, Z., Liu, F., Zhu, W., Wang, W.: OMOPredictor: an online multi-step operator performance prediction framework in distributed streaming processing (2019). unpublished thesis
13. Drucker, H., Burges, C.J.C., Kaufman, L., Smola, A.J., Vapnik, V.: Support vector regression machines. In: Mozer, M., Jordan, M.I., Petsche, T. (eds.) Advances in Neural Information Processing Systems, NIPS 1996, vol. 9, pp. 155–161. MIT Press (1996)
14. Freund, Y., Schapire, R.E.: A desicion-theoretic generalization of on-line learning and an application to boosting. In: Vitányi, P. (ed.) EuroCOLT 1995. LNCS, vol. 904, pp. 23–37. Springer, Heidelberg (1995). https://doi.org/10.1007/3-540-59119-2_166
15. Friedman, J.H.: Greedy function approximation: A gradient boosting machine. Ann. Stat. **29**(5), 1189–1232 (2001)
16. Chen, T., Guestrin, C.: XGBoost: a scalable tree boosting system. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 785–794 (2016)