DROAllocator: A Dynamic Resource-Aware Operator Allocation Framework in Distributed Streaming Processing

Fan Liu, Zongze Jin, Weimin Mu, Weilin Zhu, Yun Zhang, Weiping Wang

DROAllocator: A Dynamic Resource-aware Operator Allocation Framework in Distributed Streaming Processing

Fan Liu[1,2], Zongze Jin[1] *, Weimin Mu[1], Weilin Zhu[1], Yun Zhang[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, jinzongze, muweimin, zhuweilin, zhangyun, wangweiping}@iie.ac.cn

Abstract. With the rapid development of Internet services and the Internet of Things(IoT), many studies focus on operator allocation to enhance the DSPAs'(data stream processing applications) performance and resource utilization. However, the existing approaches ignore the dynamic changes of the node resources to allocate the operator instances to guarantee the performance, which increasing the number of migration leads to the waste of resources and the instability. To address these issues, we propose a framework named DROAllocator to select the appropriate nodes to allocate the operator instances. By capturing the change tendency of the node resources and the operator performance, our allocation mechanism decreases the number of migration to enhance the performance. The experimental results show the DROAllocator not only decrease the number of migrations to allocate the operator instances to ensure the end-to-end throughput and the latency, but also enhance the resource utilization.

Keywords: data stream processing · operator allocation · resource change · deep learning.

1 Introduction

With the increasing development of Internet services and the Internet of Things(IoT), a huge amount of data has been generated from the social networks, the electronic commence, the urban intelligent transportation, and so on. The data is usually in the form of continuous streams and it should be processed by the distributed stream processing systems(DSPSs), such as Flink [1], to mine the values.

Many studies have focused on the DSPSs [2]. Especially, in the multi-user and multi-task concurrent competition environment, to adapt the fluctuating and

* Corresponding author

abrupt data stream load by provisioning appropriate resources, many researchers [3–5] adopt the operator instantiation and the operator allocation to enhance the efficiency of the DSPSs. The operator instantiation determines the number of operator instances and the resource requirements for each operator instance to process the varying data in real-time. The operator allocation refers to the results of the operator instantiation to allocate the operator instances to the appropriate nodes.

The existing approaches make operator allocation decisions based on different current states of resources, including CPU, memory, networks, disk, load, and so on. Nardelli et al. [6] considers the network delays to allocate the operator instances and proposes several heuristics to achieve the best trade-off between the resolution time and the quality of the computed allocation solution. Li et al. [7] refers to the state of the nodes and the workload to decide where the operator instances are allocated and adopts a novel approach using the deep reinforcement learning to minimize the end-to-end latency. Wang et al. [8] proposes a network-aware and partition-based resource management scheme, which leverages the inter-operator traffic, the network condition and the resource capacity to arrange the operator instances, to enhance the performance of the DSPSs. Pietzuch et al. [9] uses the current state of the stream, the network, and the node resources to allocate the operator instances to improve the network utilization and reduce the stream latency.

However, the above approaches only consider the current state and ignore the change of the resources to allocate the operator instances to the nodes. Concretely, the nodes are used by multiple tasks and users, so the resources of the nodes are constantly changing. The resource changes of the nodes affect the available resources of the containers [10]. Accordingly, the performance of the operator instances in the containers is affected. It will cause the failure of the allocation scheme and requires operator migration to ensure the DSPAs' performance shortly, which is costly and unstable.

For example, the traditional approaches collect the resource metrics of the nodes at present, then analyze the appropriate allocation node for different goals, such as minimizing the end-to-end latency. Finally, the operator instances are allocated to the selected nodes by containers, such as Kubernetes [11]. With the DSPAs running, due to the change of node resources, it can not provide enough resources for the execution of the operator instances. Thus, we have to migrate the operator instances to other available nodes to ensure the system stability and low cost of adaptation.

In this paper, we propose a Dynamic Resource-aware Operator Allocation Framework (DROAllocator) to allocate the operator instances to the appropriate nodes. We consider the change tendency of the node resources and the performance of operator instances from the past to the future to select the appropriate allocation nodes and reduce unnecessary migration. The contributions of our work are summarized as follows:

– We first consider the change of the resources of nodes to address the problem which ignores the change. We present the DROAllocator which contains

three core parts: the Node Resource Predictor, the Node Clusterer, and the Resource Aware Scheduler.
- To predict the resources in the next time, we adopt BiGRU to capture the feature of the change tendency of the node resources in the Node Resource Predictor.
- To measure the relationship between the resources of the node clusters and the performance of the operator instances, we first aggregate the nodes with similar resource state and change pattern into a cluster and leverage the Pearson correlation coefficient in the Node Clusterer.
- In the Resource Aware Scheduler, we propose a greedy-based merge algorithm to find the appropriate nodes for the operator instances and use a cost-based reallocation instance selection algorithm to find the best combination of the operator instances to adapt the dynamically fluctuating data input rate.
- Finally, we run the DROAllocator on DataDock, which is our data stream processing system. The experimental results demonstrate that our DROAllocator not only decrease the number of migration while ensuring the DSPAs' performance, but also enhances the resource utilization.

The rest of this paper is organized as follows. Section 2 describes the design of our DROAllocator. The experimental results are shown in Section 3. Finally, we conclude our work in Section 4.

2 Framework

2.1 Overview

In this paper, we consider the change tendency of the node resources and the performance of operator instances from the past to the future to select the appropriate allocation nodes and reduce unnecessary migration. We design the DROAllocator to allocate the operator instances to the appropriate nodes. As shown in Fig. 1, the DROAllocator contains three core modules: the Node Resource Predictor(NRP), the Node Clusterer(NC) and the Resource Aware Scheduler(RAS).

At the runtime, the Metric Collector collects the resources of each node, the performance metrics of each operator instance, and the data input rate in real-time, then stores them in MetricDatabase. The NRP analyzes the large collected data of node resources to get the change tendency of node resources. Then the NC aggregates the nodes into multiple clusters at different times based on the prediction results of the NRP. Besides, we refer to the result of the QEScalor [12], which analyzes the requirements of resources for the operator instances quantitatively based on the data input rate. Finally, the RAS allocates the operator instances to the most appropriate nodes and perform the instance reallocation, to satisfy the resource requirements of the operator instances with least migrations. As shown in Fig. 1, we use the green part to present the state before the operator instance allocation. And we use the blue part to present the

operator instance allocation at time t_0. At last, we use the red part to illustrate the allocation is still valid in the future t_f.

2.2 Node Resources Predictor

In the multi-user and multi-tasking competition environment, the available resources of a node change dynamically over time. Compared with the existing methods, many studies [13, 14] demonstrate the Recurrent Neural Network (RNN), which models the time series data and the process prediction tasks, can show better performance. The BiGRU networks not only consider the dependence of time series from the past to the future, but also the dependence from the future to the past, which are more efficient than the BiLSTM in the training process [15]. We use the BiGRU to learn the changing patterns of node resources and make multi-step prediction of the node resources.

The input of our NRP is $x_t = (nr_i^{t-h+1}, nr_i^{t-h+2}, ..., nr_i^t)$, which presents the sequence of the resources of node i over the past h time period. We consider the CPU utilization, the memory consumption, and the system load average to represent the node resource state, which are significant resource factors affecting the operator performance [16]. So the resources of node i at time t can be expressed as $nr_i^t = (cpu_i^t, mem_i^t, load_i^t)$. And the output is $y_t = (nr_i^{t+1}, nr_i^{t+2}, ..., nr_i^{t+f})$, which denotes the sequence of the node resources in the future f time period.

During the training process, the NRP continuously reads the sequences of the node resources over the past h time period and in the future f time period from the MetricDatabase to build the NRP, which are normalized to the range [0,1] with the Min-Max scaler [15]. During the predicting process, we get the multi-step prediction of the node resources in the future f time period.

2.3 Node Clusterer

The performance of an operator instance is related to the resource changes of the running node, as shown in Fig. 2. The performance of the compute-intensive operator instance improves with the decrease of CPU utilization. The impact of nodes with similar resources on the performance of operator instances is considered to be similar. In order to find the nodes that satisfy the operator instance in the future, we aggregate the physical nodes into multiple clusters based on the resources of the nodes, which dynamically change over time. We use the K-means clustering algorithm [17] to aggregate the nodes at every moment in the future f time period.

Our NC takes the resource state of the nodes at the current and the future moments as the input, which are multi-step prediction results of the NRP. The input is expressed as $D_{nr} = (D_{nr}^{t+1}, D_{nr}^{t+2}, ..., D_{nr}^{t+f})$, where $D_{nr}^{t+j} = (nr_1^{t+j}, nr_2^{t+j}, ...)$. Our NC aggregates the nodes into multiple clusters at the current and future moments, and we use $D_{cls} = (D_{cls}^{t+1}, D_{cls}^{t+2}, ..., D_{cls}^{t+f})$ as the output, where D_{cls}^{t+j} denotes the clusters at time $t + j$.

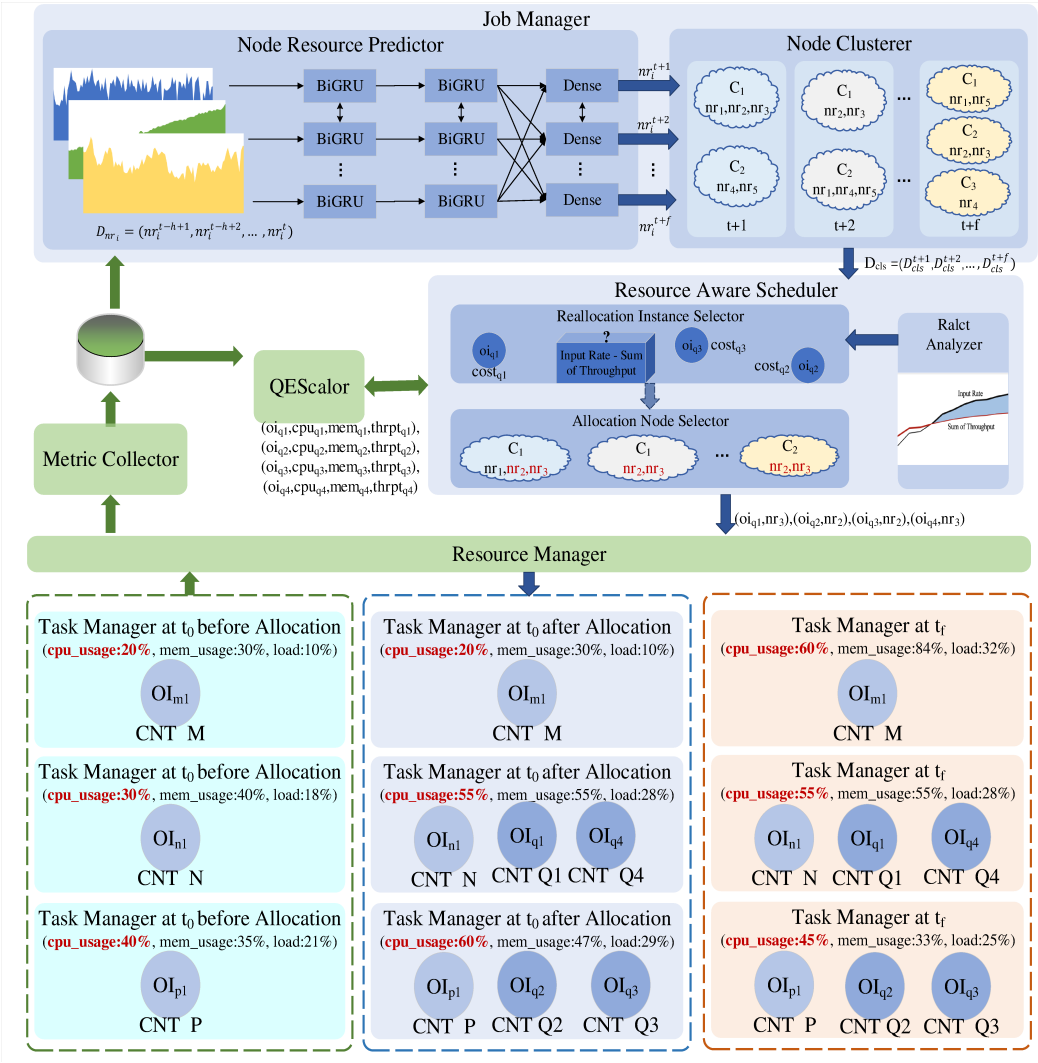
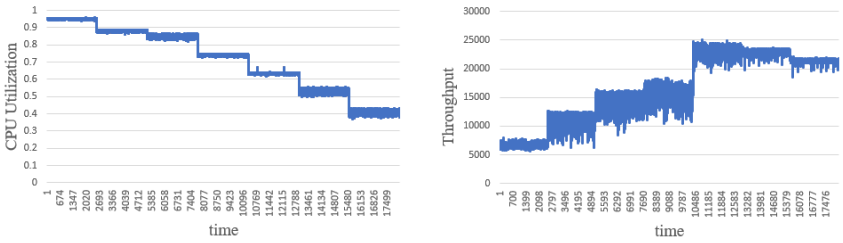


Fig. 1. DROAllocator architecture.



(a) the CPU utilization (b) the throughput

Fig. 2. Correlation Analysis between the CPU Resource and the Performance of the Compute-intensive Operator Instance.

2.4 Resource Aware Scheduler

We use the Resource Aware Scheduler(RAS) to analyze the most appropriate nodes to allocate the operator instances. It contains three parts: the Allocation Node Selector, the Reallocation Analyzer, and the Reallocation Instance Selector.

In the RAS, the node cluster results of our NC, $D_{cls} = (D_{cls}^{t+1}, D_{cls}^{t+2}, ..., D_{cls}^{t+f})$, are used as the input. Besides, we use the resource requirements and the theoretical maximum performance of the operator instances $(oi_{qk}, cpu_{qk}, mem_{qk}, thrpt_{qk})$ from the QEScalor [12] as the input. The allocation relationship between the operator instances and the nodes (oi_{qk}, TM_i) is the output of our RAS.

Allocation Node Selector. We use the Allocation Node Selector to find the appropriate nodes to allocate the operator instances. The operator instances run on the nodes with less migration and higher performance.

Specifically, based on the clustering results of the NC, we get multiple clusters at every moment in the future f time period. We use the Pearson correlation coefficient [18] to measure the correlation between the resources of node clusters and the performance of operator instances. The clusters with the lower CPU utilization, memory consumption, and system load are available clusters to allocate the operator instances. Then we present a greedy-based merge algorithm, named GMA, to get the allocation nodes at different times with less migration. Our GMA finds the most suitable allocation node by calculating the intersection of available clusters at each time, and the operator instance will not migrate or migrate rarely when running on the node.

Reallocation Analyzer. We use the Reallocation Analyzer to measure whether some instances of an operator should be reallocated to other nodes, when the overall processing performance of the instances of an operator is less than the data input rate. Specifically, we monitor the actual performance of all instances of the operator at time t from the MetricDatabase, and calculate the sum of the actual performance of all instances, which is represented as ϑ_{actual}^{t}. We get

Algorithm 1 GMA.

```
exit_intsct = true
while exit_intsct do
  exit_intsct = false
  for k ∈ [t+1, t+f) do
    intsct = intersection(D_cls^k, D_cls^{k+1})
    if !intsct.isempty() and (intsct != D_cls^k or intsct != D_cls^{k+1}) then
      D_cls^k = intsct, D_cls^{k+1} = intsct, exit_intsct = true
    end if
  end for
end while
```

the theoretical maximum performance of all instances of the operator from the QEScalor, and calculate the sum of the theoretical performance of all instances, which is represented as ϑ_{peak}. We get the data input rate at time t from the MetricDatabase, which is represented as γ^t.

Then, we compare ϑ^t_{actual} with ϑ_{peak} and compare ϑ_{peak} with γ^t. If $\vartheta^t_{actual} < \vartheta_{peak}$ and $\vartheta_{peak} \geq \gamma^t$, we get that the operator instances can not process the data with the input rate because some instances do not achieve their theoretical maximum performance. Thus if we reallocate some or all of these instances to other more appropriate nodes, the operator instances may process the data with the input rate. In this case, we calculate the difference between the data input rate and the sum of actual performance, which is denoted as $\gamma^t - \vartheta_{actual}$, and is the input of the Reallocation Instance Selector.

While in other cases, such as $\vartheta^t_{actual} = \vartheta_{peak}$ or $\vartheta_{peak} < \gamma^t$, it demonstrates that the operator instantiation of the QEScalor is unreasonable and our RAS is unable to process these cases. So we feed these cases back to the QEScalor to adapt the fluctuating data input rate.

Reallocation Instance Selector. When the Reallocation Analyzer gets that it's necessary to reallocate and migrate some instances, we use the Reallocation Instance Selector to find the best combination of the operator instances that fail to achieve the maximum performance. Then these instances are reallocated by the Allocation Node Selector to the nodes with more resources to get better performance. We build a cost model to evaluate the total cost of the reallocation of the instances. The cost is defined as:

$$\min \quad C(n) = \sum_{k=1}^{n} c^k_{rsc} + c_{on} + c_{off} + c_{rrt}, \quad \text{s.t.} \quad \sum_{k=1}^{n} \Delta p_k \geq \Delta W \tag{1}$$

where $C(n)$ is the total cost, n is the number of selected instances, c^k_{rsc} is the cost of system resources used by the kth selected instance, c_{on} is the startup cost, c_{off} is the shutdown cost, and c_{rrt} is the re-routing cost. In the constraint, we use Δp_k, which represents the performance improvement, to indicate the

difference between the optimal performance and the actual performance of the kth operator instance. We use ΔW to denote the difference between the data input rate and the sum of the actual performance of all operator instances.

Then we propose a cost-based reallocation instance selection algorithm, named CRISA, to find the best combination of the operator instances that fail to achieve the theoretical optimal performance. We use the CRISA to pick the best combination, which satisfies the constraint and the least cost. Our CRISA is divided into 3 steps. Firstly, we compare the theoretical optimal performance of the operator instances with the actual runtime performance to obtain the set of candidate reallocation operator instances. Secondly, we rank the candidate set in descending order according to the theoretically improved performance of the candidate reallocation operator instances. At last, we iteratively select k operator instances from the candidate set with the minimum reallocation cost.

Algorithm 2 CRISA.

for $k \in [0, o_q.instance\_num)$ **do**
  if $oi_{qk}.actual\_thprt < oi_{qk}.peak\_thprt$ **then**
    $oi_{qk}.incrs\_thprt = oi_{qk}.peak\_thprt - oi_{qk}.actual\_thprt$
    $candidates$.append(oi_{qk})
  end if
end for
sort_in_desc($candidates.incrs\_thrpt$)
for $k \in [1, candidates.num]$ **do**
  for $j \in [0, candidates.num)$ **do**
    $k\_candidates$ = select_oi($candidates[j, candidates.num), k$)
    if sum($k\_candidates.incrs\_thrpt$) $\geqslant \Delta W$ and $C(k\_candidates) < min_C$ **then**
      $mgrt\_ois = k\_candidates$, $min_C = C(k\_candidates)$
    else
      break
    end if
  end for
end for

3 Experiments

3.1 Datasets and Settings

We collect the resources of the physical machine nodes, the performance metrics of the operator instances, and the data input rate in experiments, as shown in Table 1.

Specifically, The node resource dataset is divided into a training set and a testing set. The training set includes data from the first 20 days, which are used to train the prediction models. And the testing set contains data from the next 10 days, which are used to evaluate the effectiveness of the models. Besides, to

construct the operator performance dataset, we run the compute-intensive operator instances (the keyword extraction operator) which consumes a lot of CPU resources on every node. The node resource dataset and the operator performance dataset are used to analyze the correlation between the cluster resources and the operator performance. Furthermore, we get the operator instantiation data, including the resource requirements and the theoretical maximum performance of every operator instance, from the QEScalor to construct the operator instantiation dataset.

We conduct experiments in a cluster environment which consists of eleven machines. These machines are all configured with Intel Xeon CPU E5-2620 v3 2.30 GHz 24 cores, 128 GB memory, and 599 GB disks. We use one machine to run JobManager and MetricDatabase to collect data in real-time. We use three machines with NVIDIA TESLA P4 GPUs to conduct training and testing of our prediction, clustering, and scheduling models. The remaining seven machines are used to run the operator instances which are deployed by Kubernetes.

In the node resource prediction analysis, we compare our NRP with the ARIMA, the SVR, the GRU, and the BiLSTM. Specifically, we all use ReLU as the activation function, adam as the optimizer, MSE [15] as the loss, and we set the size of hidden layer to 128 for the LSTM, the GRU, the BiLSTM and our NRP. We use rbf as the kernel, scale as the gamma, and C is set to 100.0 for the SVR. In the operator instance allocation, we collect the migration number and calculate the sum of throughput of 22 operator instances running on the 7 nodes for 24 hours. We do our experiments repeatedly ten times.

Table 1. The Datasets in the experiments

Dataset name	Metrics	Collection frequency	Collection duration
Node Resource Dataset	CPU utilization, memory consumption and system load average	1 Minute	30 Days
Operator Performance Dataset	Throughput(Number of data processed per minute)	1 Minute	30 Days
Input Rate Dataset	Input rate(Number of data received per minute)	1 Minute	30 Days
Operator Instantiation Dataset	CPU_req, mem_req, thrpt_peak	-	30 Days

3.2 Metrics

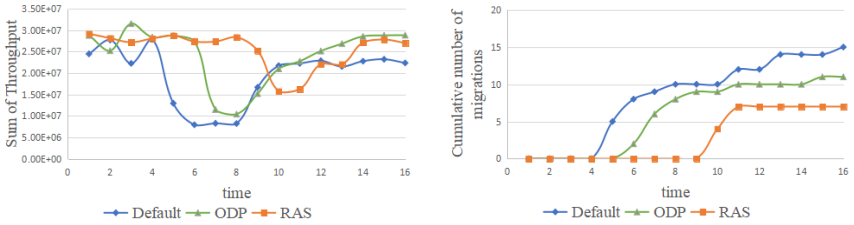
We use the Root Mean Square Error(RMSE) and the Mean Absolute Error(MAE) metrics [15] to evaluate the performance of the node resource prediction models. And we use the overall throughput and the cumulative number of migrations of the operator instances to evaluate the performance of the operator instance allocation.

3.3 Results

Operator Instance Allocation Analysis. To evaluate the performance of the operator instance allocation, we compare our RAS with the system default allocation scheme and the RES-ODP [6]. We use the overall throughput and

the cumulative number of migrations of the operator instances to measure the operator instance allocation.

As shown in Fig. 3, the overall throughput of our RAS is higher than the default scheme. And there is a bit difference between our RAS with the RES-ODP. But the throughput of our RAS is more stable and the change is small, compared with other methods. More importantly, our RAS is significantly lower than other methods for the cumulative number of migrations.



(a) the sum of end-to-end throughput (b) the cumulative number of migrations

Fig. 3. Operator instance allocation analysis.

The system default allocation scheme which leverages round-robin only considers whether the node is available the current time. The RES-ODP, which uses the linear relaxation of the ILP formulation of ODP, considers the current network delay when allocating the operator instances. When the resources of the nodes change over time and the allocation nodes are no longer appropriate to run some instances, these methods should reallocate and migrate some or all instances to other nodes. Thus our RAS outperforms the default and the RES-ODP allocation scheme, which takes the change of the node resources into account and tries to allocate the instances to the nodes with the longest runtime and the least migration.

Node Resource Prediction Analysis. The experimental results of the five-step forward prediction are shown in Table 2. The recurrent neural networks, such as the LSTM, the GRU, the BiLSTM, and the NRP, perform significantly better than the traditional methods for the node resource prediction, such as the ARIMA and the SVR. The recurrent neural networks capture more latent information from the massive of the collected data and learn the changing trend of the node resources more accurately. Besides, our NRP outperforms the LSTM and GRU, because the NRP analyzes data of node resources in two directions to get more latent features in the time dimension. Furthermore, our NRP shows faster training and higher prediction accuracy than BiLSTM.

Correlation Analysis between the Resources of Node clusters and the Performance of the Operator Instances. We calculate the Pearson correla-

Table 2. The five-step ahead prediction performance of node resources for different models

Model type	ARIMA	SVR	GRU	LSTM	BiLSTM	NRP
RMSE	0.0517±0.0003	0.0449±0.0004	0.0243±0.0002	0.0289±0.0002	0.0282±0.0003	0.0234±0.0002
MAE	0.0397±0.0002	0.0341±0.0003	0.0145±0.0003	0.0156±0.0002	0.0136±0.0002	0.0116±0.0003

tion coefficient between the node resources and the performance of the compute-intensive operator instances. We get -0.8946, -0.4554, -0.7214 for the correlation coefficient between the CPU utilization, memory consumption, system load with the performance of the compute-intensive operator instances respectively.

The results show the resources of the node clusters and the operator performance are negatively correlated. The operator performance is higher in the node clusters with the lower CPU utilization, memory consumption, and system load.

4 Conclusion

In this paper, we propose an operator allocation framework, named DROAllocator, to address the problem which ignores the dynamic changes of the node resources to allocate the operator instances to guarantee the performance. It contains three core modules: the Node Resource Predictor(NRP), the Node Clusterer(NC) and the Resource Aware Scheduler(RAS). We use the NRP to get the precise change tendency prediction of the node resources by the BiGRU. Then we refer to the change tendency of the node resources and analyze the relationship between the resources of the node clusters and the performance of the operator instances. Finally, the RAS allocates the operator instances to the most appropriate nodes and reallocate some operator instances when the overall performance of the operator instances is less than the data input rate. Experiments demonstrate our framework not only improve the DSPAs' performance, but also enhance the resource utilization.

References

1. P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, "Apache flink^{TM}: Stream and batch processing in a single engine," *IEEE Data Eng. Bull.*, vol. 38, no. 4, pp. 28–38, 2015.
2. L. Neumeyer, B. Robbins, A. Nair, and A. Kesari, "S4: distributed stream computing platform," in *ICDMW 2010, The 10th IEEE International Conference on Data Mining Workshops, Sydney, Australia, 13 December 2010*, pp. 170–177, 2010.
3. 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*, pp. 705–722, 2019.
4. F. Lombardi, L. Aniello, S. Bonomi, and L. Querzoni, "Elastic symbiotic scaling of operators and resources in stream processing systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 3, pp. 572–585, 2018.

5. V. Cardellini, V. Grassi, F. L. Presti, and M. Nardelli, "Distributed qos-aware scheduling in storm," in *Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems, DEBS '15, Oslo, Norway, June 29 - July 3, 2015*, pp. 344–347, 2015.
6. M. Nardelli, V. Cardellini, V. Grassi, and F. L. Presti, "Efficient operator placement for distributed data stream processing applications," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 8, pp. 1753–1767, 2019.
7. T. Li, Z. Xu, J. Tang, and Y. Wang, "Model-free control for distributed stream data processing using deep reinforcement learning," *Proc. VLDB Endow.*, vol. 11, no. 6, pp. 705–718, 2018.
8. Y. Wang, Z. Tari, X. Huang, and A. Y. Zomaya, "A network-aware and partition-based resource management scheme for data stream processing," in *Proceedings of the 48th International Conference on Parallel Processing, ICPP 2019, Kyoto, Japan, August 05-08, 2019*, pp. 20:1–20:10, 2019.
9. P. R. Pietzuch, J. Ledlie, J. Shneidman, M. Roussopoulos, M. Welsh, and M. I. Seltzer, "Network-aware operator placement for stream-processing systems," in *Proceedings of the 22nd International Conference on Data Engineering, ICDE 2006, 3-8 April 2006, Atlanta, GA, USA*, p. 49, 2006.
10. W. Lloyd, S. Pallickara, O. David, J. Lyon, M. Arabi, and K. Rojas, "Performance implications of multi-tier application deployments on infrastructure-as-a-service clouds: Towards performance modeling," *Future Generation Comp. Syst.*, vol. 29, no. 5, pp. 1254–1264, 2013.
11. D. Bernstein, "Containers and cloud: From LXC to docker to kubernetes," *IEEE Cloud Comput.*, vol. 1, no. 3, pp. 81–84, 2014.
12. 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*, pp. 147–160, 2020.
13. D. Liao, J. Xu, G. Li, W. Huang, W. Liu, and J. Li, "Popularity prediction on online articles with deep fusion of temporal process and content features," in *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pp. 200–207, 2019.
14. 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.
15. 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.
16. J. Zhu, R. Yang, C. Hu, T. Wo, S. Xue, J. Ouyang, and J. Xu, "Perphon: a ml-based agent for workload co-location via performance prediction and resource inference," in *Proceedings of the ACM Symposium on Cloud Computing, SoCC 2019, Santa Cruz, CA, USA, November 20-23, 2019*, p. 478, 2019.
17. H. Yu, G. Wen, J. Gan, W. Zheng, and C. Lei, "Self-paced learning for K-means clustering algorithm," *Pattern Recognit. Lett.*, vol. 132, pp. 69–75, 2020.
18. Z. Ahmed and S. Kumar, "Pearson's correlation coefficient in the theory of networks: A comment," *CoRR*, vol. abs/1803.06937, 2018.