OrientStream: A Framework for Dynamic Resource Allocation in Distributed Data Stream Management Systems

Chunkai Wang[1] Xiaofeng Meng[1] Qi Guo[2] Zujian Weng[1] Chen Yang[1]

[1]School of Information, Renmin University, Beijing, China
[2]Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China

[1]{wangchunkai,xfmeng,zjweng,yang_chen}@ruc.edu.cn [2]guoqi@ict.ac.cn

ABSTRACT

Distributed data stream management systems (DDSMS) are usually composed of upper layer relational query systems (RQS) and lower layer stream processing systems (SPS). When users submit new queries to RQS, a query planner needs to be converted into a directed acyclic graph (DAG) consisting of tasks which are running on SPS. Based on different query requests and data stream properties, SPS need to configure different deployments strategies. However, how to dynamically predict deployment configurations of SPS to ensure the processing throughput and low resource usage is a great challenge. This article presents OrientStream, a framework for dynamic resource allocation in DDSMS using incremental machine learning techniques. By introducing the data-level, query plan-level, operator-level and cluster-level's four-level feature extraction mechanism, we firstly use the different query workloads as training sets to predict the resource usage of DDSMS and then select the optimal resource configuration from candidate settings based on the current query requests and stream properties. Finally, we validate our approach on the open source SPS–Storm. Experiments show that OrientStream can reduce CPU usage of 8%-15% and memory usage of 38%-48% respectively.

CCS Concepts

•Information systems → Database query processing; Stream management;

Keywords

Stream Processing System, Relational Query System, Incremental Learning, Modeling and Prediction

CIKM '16, October 24–28, 2016, Indianapolis, IN, USA

DOI: http://dx.doi.org/10.1145/2983323.2983681

1. INTRODUCTION

Nowadays, many modern applications require large-scale continuous queries and analysis, such as analytic over microblogs in social networks, monitoring of high frequency trading in financial fields, real-time recommendation in electronic business and so on. Therefore, there have been many open source SPS, such as S4 [15] and Storm [20]. For enhancing the ease of use and processing capability of them, some RQS that contain query languages have been developed, such as Squall [1].

For completing the query requests given by users within the specified latency, DDSMS need to guarantee the high throughput and low latency of stream processing. This often requires users to set the relevant configurations in advance, such as the operator parallelism and the memory usage of processes. However, according to data stream properties (e.g., arrival rate and value distribution) and the difference of query tasks, the reasonable setting of configurations for ensuring both processing in real-time and low resource usage is a very challenging problem.

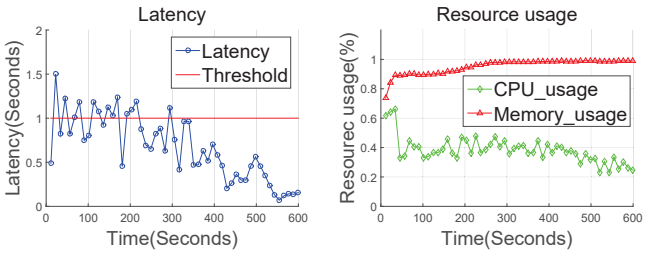
Motivating Example. Consider an application which analyzes the traffic GPS data collected from taxes and buses in real time. We take traffic-monitoring (TM) system as a query example, and Storm as a SPS example.

Traffic-monitoring needs to process and analyze the traffic GPS data collected from taxes and buses in real-time. The scenario is a typical application of streaming execution. We use GeoLife GPS Trajectories [21] for this workload that contains GPS trajectory data collected by 182 users in a period of over five years, and simulate a traffic-monitoring system. It contains a map matching processing logic which accepts trajectories data of a GPS-loggers or GPS-phones including altitude, latitude, and longitude, and determines the location of this object in terms of a road ID. The speed calculate processing logic accepts the road ID result from the matching processing logic and calculate the average speed according to the road ID.

The execution tasks in Storm can be expressed as a DAG (called *topology*), where nodes are data sources (called *spouts*) or processing logic (called *bolts*) which run in different processes (called *workers*), and edges indicate the flow of data between nodes. For real-time computation, the topology should be created at first. Then, before the deployment of the topology to the Storm cluster, the user should specify various topology parameters (e.g., the number of workers,

spout parallelism, and bolt parallelism, etc.) according to either experience or machine configurations. Finally, the Storm cluster receives continuous data stream and processes them in a real-time fashion.

However, the topology parameters are not aware of dynamic data stream. For example, as shown in Figure 1.a, due to the fixed parameters, some processing delays are beyond the user's threshold (e.g., 1 second) with the change of data stream rate. Meanwhile, it also leads to the higher cluster resources usage. As shown in Figure 1.b, the average usage of CPU is about 40%, and the average usage of memory (the user specifies the maximum memory usage of each worker) is more than 90%. As DDSMS are always deployed in the cloud environment, the problem of fixed configurations not only leads to high resource usage, reduces the user's return-on-investment (ROI), and even causes the system bottleneck.



(a) Latency (b) Resource Usage

Figure 1: Traffic-Monitoring Example

So, in this paper, we present the design of OrientStream that implements dynamic resource allocation for Storm. The contributions of our work are summarized as follows:

1. We construct initial training sets by using the different workloads for meeting the characteristics of RQS, divide queries in four different levels to predict CPU and memory usage, and use incremental learning technique to build prediction models in real time.

2. According to the related independent of different regression models, we give the algorithm of Ensemble Different kinds of Regressions (EDKR) to integrate different models for enhancing the prediction accuracy.

3. We introduce the outlier execution detection mechanism to monitor the anomalous executions. So as to further improve the prediction accuracy.

The rest of the paper is organized as follows. Section 2 surveys the related work. Then, there is the problem definition in section 3. And, in Section 4, we give the architecture of OrientStream. Next, we describe the prediction process in Section 5. Section 6 gives the results of our experiment evaluation. Finally, Section 7 concludes this paper.

2. RELATED WORK

To meet users' different query requests, DDSMS needs to build a complete operator library in RQS and dynamically set up the parallelism of processing nodes of SPS based on different query requests. There are roughly three lines of related work as below.

Dynamic Load Scheduling. Aeolus [19] is an optimizer to set up the parallel degree and the batch size of intra-node dataflow programs. It defined a cost model based on four measurements of processing tuple time, including shipping time, queuing time, pure processing time and actual processing time. According to the model, Aeolus can get the best configuration scheme for the parallel degree of operators and batch size. DRS [10] is a dynamic resource scheduler for cloud-based DDSMS. It designed an accurate performance model that can get the setting of optimal parallel degree of operators. Aniello et al. [7] proposed two scheduling algorithms by using topology-based static scheduling strategy and traffic-based dynamic scheduling strategy for reducing the latency of processing tuples and inter-node traffic between multiple topologies.

Machine Learning Techniques. Khoshkbarforoushha et al. [12] proposed a model based on the Mixture Density Network [8] for resource usage estimation of data stream processing workloads. This model can help users to make a decision about whether to register a new query or not. ALOJA [17] project presented an open, vendor neutral repository, featuring over Hadoop executions. It can predict query time and abnormal detection by using machine learning techniques to interpret Hadoop benchmark performance data and performance tuning.

Resource Estimation for RQS. Li et al. [14] set up two kinds of mechanisms of feature extractions: coarse-grained global features and fine-grained operator-specific features for different queries over Microsoft SQL Server. Akdere et al. [6] built three models for predicting query performance based on different query plans: query plan-level model, operator-level model and hybrid model for nested query.

OrientStream is orthogonal to the above works since we focus on constructing training sets for the data stream properties and the parameter configurations of different levels, predicting resource usage by online machine learning techniques, and dynamically tuning the related parameter configurations of DDSMS.

3. PROBLEM DEFINITION

Query requests always need to set different parameter configurations by user's experience. Whether the parameter configurations are reasonable directly affects the throughput and the latency of DDSMS, as well as the resource usage. With the change of data stream rate, we should dynamically adjust parameter configurations to meet user's needs of the throughput and latency thresholds. But, SPS are often not allowed to adjust parameters arbitrarily.

So, we need to predict the resource usage, the throughput and latency for any candidate configurations. Based on the predicted results, the optimal configuration is defined as the one using minimal CPU and memory resource while the throughput and the latency requirements are still guaranteed under the thresholds set by users. Thus, the problem can be modeled as a resource usage optimization problem as we discuss next.

Let $N = (n_1, n_2, n_3, ...)$ be the set of all nodes in a cluster. The CPU usage U_{CPU} and memory usage U_{Memory} of each node n_i can be defined as follows.

$$T(n_i) = \alpha * U_{CPU}(n_i) + \beta * U_{Memory}(n_i) \qquad (\alpha + \beta = 1) \quad (1)$$

where, α and β are the weight of the CPU and memory usage respectively, which set as 50% by default.

Then, for the entire cluster, the optimization problem is

$$\begin{aligned} &Minimize \sum_{n_i \in \mathcal{N}} T(n_i) \\ &s.t. \quad R(latency) < T(latency), \\ &and \quad R(throughput) > T(throughput). \end{aligned} \tag{2}$$

where, $R(latency)$ and $R(throughput)$ are the processing latency and throughput of query requests. $T(latency)$ and $T(throughput)$ are the thresholds set by users.

4. ORIENTSTREAM ARCHITECTURE

This section gives the overview of OrientStream and working principles of each step.

OrientStream Overview. Figure 2 illustrates the overall architecture of OrientStream, by extending the original Storm and Kafka [13] cluster. It is mainly divided into two parts: the left part is the hierarchical feature extractions, and the right part is the query monitor that is responsible for collecting the feature data and predicting the resource usage by incremental learning models.

In the left part, according to the characteristics of DDSMS and clusters, we divide feature extractions into four levels. There are data-level features of the data source, plan-level features of RQS, operator-level features of SPS and cluster-level features of the hardware platform (see section 5.1 for details). Each data source collects and transmits data stream using the Kafka message queue. In the right part, we analyze the real-time collected feature data and build the incremental learning models, and predict the resource usage of DDSMS using model-based methods.

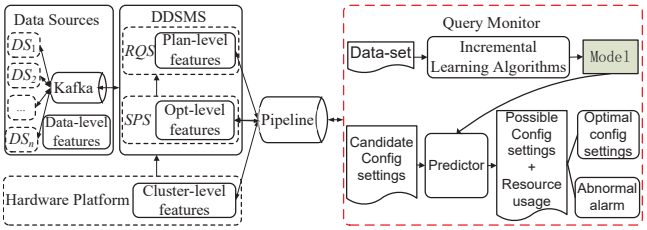


Figure 2: OrientStream architecture

Model Construction. During the incremental learning phrase, we collect the feature data dynamically to generate OrientStream data-set, use the data set to predict the accuracy of the each model and use the prediction results to re-train the model by incremental learning. According to the characteristics of the data-set, we choose different incremental learning algorithms to build models based on MOA [2] and Weka [3] toolkits.

Resource Prediction. *Query Monitor* is responsible for building incremental learning models and generating the *Predictor* by collecting the feature data. Then, We can use the *Predictor* to predict the candidate configuration settings, select the optimal configuration setting for using the least resource, and alarm the abnormal setting for using resource too much or the throughput/latency in excess of thresholds.

Dynamic Tuning. During model deployment, the constructed models are used for determining the optimal configuration setting for the current data rate, so as to minimize the resource usage while still preserving real-time property. Due to the limitations of the re-balance mechanism of Storm, we propose to deactivate the old topology and then switch to a new topology with the optimal configuration. To maintain the streaming behavior of the old topology, before deactivating it, the behavior data should be stored into Kafka. After deploying the new topology, the behavior data should be read from Kafka to recover the final state of the old topology. This process of switching to a new topology will last for about 1 minute. So, the application scenarios of OrientStream need to meet the following two requirements: (1) the monitor cycle of data stream is longer; (2) the data fluctuation is not frequent and far lower than the switching time.

5. MODELING RESOURCE PREDICTION

This section focuses on the implementation details, including features extraction, model building and integration, abnormal detection mechanism and applications.

5.1 Data-set Features

In this section, we will describe the features that use as the input OrientStream data-set to the learning model for predicting resource usage. For this purpose, we need to specify the level of input features for learning models. According to a series of experiments and specific domain knowledge, we choose features from the following four aspects:

Data-Level Features. We control the rate of input data stream as DATA_RATE by changing the spout parallelism. Also, the number of Kafka brokers and the number of topics' partitions are also taken into account. In addition, because the change of DATA_RATE could influence the rate of subsequent operators, we need to calculate the rate of relevant operators dynamically (e.g., map matching bolt rate and speed calculate bolt rate in the TM workload).

Query Plan-Level Features. The continuous query can generate DAG-style topology by using the query plan. The amount of data waiting to be processed and the window sliding step can be controlled by WINDOW_SIZE and SLIDING_STEP. Moreover, we count the number of input and output tuples in a window to predict the average selectivity, and count the number of different types of operators.

Operator-Level Features. We can monitor resource usage by adjusting the operator parallelism in different windows. If the same operator appears more than once, we take the average value as the criterion of feature extraction. Meanwhile, we need to involve the number of workers about Storm as the feature that can influence the processing efficiency of operators.

Cluster-Level Features. We defines cluster-level features including the number of worker nodes, CPU cores, memory capacity and network transmission rate.

5.2 Algorithms

According to analyzing the prediction results by using models to support incremental learning in MOA and Weka toolkits, we select four models with the highest prediction accuracy:

Bayes model: we use the Naive Bayes (NB) from the MOA toolkit. Bayes [11] algorithm updates internal counters with each new instance and uses the counter to assign a class in a probabilistic fashion to a new item in the stream.

Hoeffding trees model: we use the HoeffdingTree (HT), also from the MOA toolkit. A hoeffding tree or VFDT [9] is an incremental decision tree that is capable of learning from massive data stream. It exploits the fact that a small sample can often be enough to choose an optimal splitting attribute.

Online bagging model: we use the OzaBagging (Oza) [16] from the MOA toolkit and use hoeffding tree as the base learner. Oza and Russell proposed the online bagging algorithm that gives each instance a weight for re-sampling according to the Poisson distribution of λ equal to 1.

Nearest neighbors model: we use the IBk [5] from the Weka toolkit, through introducing updateable classifier method to achieve incremental learning characteristics.

5.3 EDKR Model

As we know, ensemble learning is one of the most effective methods to improve the accuracy of the regression algorithm, which is considered as one of the most effective learning ideas. Its contribution is to improve regression accuracy by aggregating multiple base models. However, it only ensembles the same regression model. As discussed in section 5.2, four models has no contact and the prediction is independent of each other. Therefore, we combine these four models and propose a regression model EDKR (Ensemble Different Kind of Regression).

In the process of prediction, firstly, EDKR lets each base regression model to make prediction; then, according to the relative absolute error (RAE) value of the each base regression model, EDKR calculates the voting weights respectively; finally, it determine the regression value of the sample with a weighted vote. Among them, RAE and the objective function of EDKR are defined as below:

Let N be the set of training samples in each regression model and n be the nth sample to be predicted. P_n is the prediction value of the regression model to the sample n. T_n is the true value of the sample n. AVG_N is the mean of N samples. Then,

$$RAE_n = \sum_{i=1}^{N} |P_{in} - T_i| \,/\, \sum_{j=1}^{N} |T_j - AVG_N| \qquad (3)$$

where, RAE_n is the RAE of the nth sample using specified base regression model.

So, based on the K base regression models, the objective function of EDKR to predict the sample n is,

$$F_n = \sum_{i=1}^{K} (1 - RAE_{in}) / (K - \sum_{j=1}^{K} RAE_{jn}) * P_{in} \qquad (4)$$

where, F_n is the RAE of the nth sample using EDKR model, and P_{in} is the prediction value of the nth sample using the ith base regression model.

5.4 Abnormal Detection Mechanism

Due to the abnormal execution of the cluster (e.g., the cluster load is not balanced, network transmission is not stable), or the observation does not fit into the model, it results in some abnormal training data. Based on the difference between the predicted value and the observed value (e.g., resource usage), we divide the training data into three types:

- *Normal.* Detected data can be predicted correctly by learning models.
- *Warning.* Detected data is mis-predicted, and more than a half of nearest neighbors to the data are mis-predicted too.
- *Outlier.* Detected data is mis-predicted, but more than a half of nearest neighbors to the data are predicted correctly.

We can improve the prediction accuracy by discarding abnormal training data and not getting it into the incremental learning model. Besides using this method to test new observations, we can re-train our models by subtracting observations marked as outlier. Algorithm 1 gives the detailed description of the detection procedure. For each detected data, firstly, we select the normal data based on the error threshold T_e (Algorithm 1 line 1-2). Then, for each abnormal test data, we compute the k_{NN} tuples that are nearest to the test data. Line 4-6 in Algorithm 2 depicts this process. Next, we count the number of warning or outlier of these k_{NN} tuples by the definition of warning or outlier category (Algorithm 1 line 7-13). Finally, we judge whether the test data is warning or outlier (Algorithm 1 line 14-18).

Algorithm 1 Abnormal detection algorithm.

Require:
Prediction Error Threshold T_e;
The number of tuples to nearest tag tuple k_{NN};
The regression file F contains predicted value and true value;
Ensure:
The type of detected tuple t;
1: **if** $(|t.PredictedValue - t.TrueValue| < T_e)$ **then**
2: t is *Normal*;
3: **else**
4: **for** $(i = 1; i < Number\_of\_TestDataSet; i++)$ **do**
5: Computing the k_{NN} tuples nearest to t;
6: **end for**
7: **for** $(j = 1; j < k_{NN}; j++)$ **do**
8: **if** $(|t_k.PredictedValue - t_k.TrueValue| > T_e)$ **then**
9: $N_{warning} + 1$;
10: **else**
11: $N_{outlier} + 1$;
12: **end if**
13: **end for**
14: **if** $(N_{warning} > N_{outlier})$ **then**
15: t is *Warning*;
16: **else**
17: t is *Outlier*;
18: **end if**
19: **end if**

6. EVALUATION

This section presents three workloads of each test scenario and gives the result of prediction of resource usage with different learning models. We take the prediction of CPU usage as an example to verify the effect of anomaly detection mechanism and verify the resource save situation through using the dynamic tuning mechanism.

The testbed is established on a cluster of ten nodes connected by a 1Gbit Ethernet switch. Five nodes are used to transmit data source through kafka. One node serves as the nimbus of Storm, and the remaining four nodes act as supervisor nodes. Each data source node and the nim-

bus node have a Intel E5-2620 2.00GHz four-core CPU and 4GB of DDR3 RAM. Each supervisor node has two Intel E5-2620 2.00GHz twelve-core CPU and 64G of DDR3 RAM. We implement comprehensive evaluations of our prototype on Storm-0.9.5 and Ubuntu-14.04.3.

In order to cover the different query characteristics, we choose three different workloads as below.

Traffic Monitoring (TM). This workload has been introduced in detail in Section 1.

Word Count (WC). Word count represents word frequency statistics of different sentences. It is composed of a bolt for splitting sentences into words and another one for counting the number of occurrences for each word in a hash-map. We use the word count data-set from HiBench [4]. There are about 3 million sentences and more than 30 million words for testing.

TPC-H (Q3). TPC-H is a decision support benchmark. The queries and the data populating the database have been chosen to have broad industry-wide relevance. In order to verify the query processing of multiple data streams, we choose Q3 as the workload that includes three data sources and use 15 million tuples for each data source for this workload. It contains three filter bolts to filter data source, two join bolts to make equi-join for the three data sources, a group bolt to group the result of join bolts, and a order bolt to order the result of the group bolt.

The sink bolt is used to merge the output results. For simplicity, we only use sink as the count statistics of received results. The topologies of three workloads are shown in Figure 3. Our workloads cover two aspects: (1) various topology structural complexity, from single chain topology of one data source to complex DAG topology of multiple data sources. (2) different runtime characteristics, such as memory intensive of TM workload, communication intensive of WC workload and compute intensive of Q3 workload.

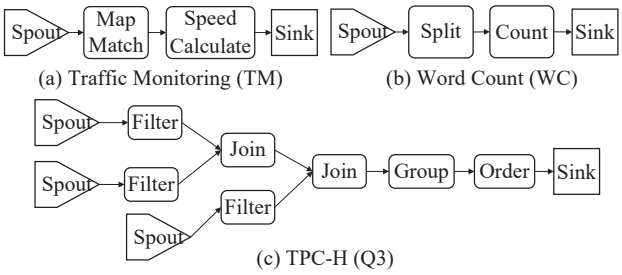


Figure 3: Topology of three different workloads

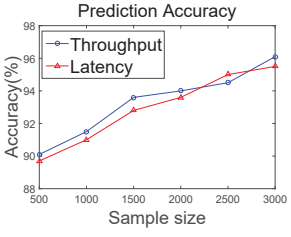


Figure 4: Accuracy

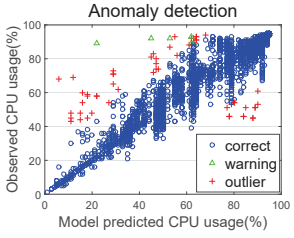


Figure 5: Anomaly Detection

6.1 Online Resource Prediction

In order to improve the prediction accuracy, we collect a training sample in every 3 minutes, and calculate the mean of CPU usage, memory usage, throughput and latency as the prediction targets. For each workload, we respectively collect 3000 samples by setting data rate and window sizes randomly.

According to $T(throughput)$ and $T(latency)$ set by users, we build two binary classification models and use J48 in Weka to classify. J48 is an open source java implementation of C4.5 [18] algorithm. The classification accuracy of the throughput and latency about three different workloads is shown in Figure 4.

The accuracy of predicting the throughput and latency is maintained at about 90% to 96% with increasing samples(from 500 to 3000). This provides a high security for the next step in predicting the resource usage.

For the regression prediction of resource usage, we use different learning models mentioned in section 5.2 and 5.3. Table 1 and 2 show the test results of the mean absolute error (MAE) and RAE per method based on different workloads. The experimental results show that:

(1) The average RAE of prediction memory usage is lower than that of prediction CPU usage, because of the fluctuation of memory usage is lower than that of CPU usage.

(2) EDKR is better than the best model in the four models. For prediction of CPU usage in different workloads, MAE can reduce 0.65-0.67, and RAE can reduce 3.5%-6.2%. For prediction of memory usage in different workloads, MAE can reduce 0.41-0.55 and RAE can reduce 2.55%-3.1%.

Models	TM		WC		Q3	
	MAE	RAE	MAE	RAE	MAE	RAE
NB	4.8633	0.2478	4.7251	0.2297	5.5328	0.2871
HT	5.3241	0.2687	5.1275	0.2503	5.5936	0.2971
Oza	4.7925	0.2365	4.5238	0.2156	5.1350	0.2665
IBk	5.6412	0.2825	5.5179	0.2655	5.8911	0.3059
EDKR	4.1325	0.2018	3.8729	0.1537	4.4655	0.2317

Table 1: MAE and RAE of CPU usage prediction

Models	TM		WC		Q3	
	MAE	RAE	MAE	RAE	MAE	RAE
NB	4.3926	0.2538	4.0158	0.2246	4.8593	0.2883
HT	4.8726	0.2955	4.4715	0.2643	5.1827	0.3390
Oza	4.6528	0.2511	0.3271	4.2819	5.0001	0.2871
IBk	2.7612	0.1285	2.5463	0.1157	2.8739	0.1305
EDKR	2.2109	0.0975	2.1035	0.0902	2.4639	0.1028

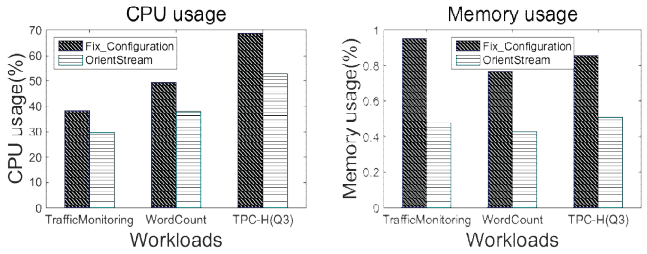
Table 2: MAE and RAE of memory usage prediction

6.2 Abnormal Filter Prediction

In this section, we take 3000 data samples of the CPU usage of TM workload as an example, use the EDKR as the prediction model, and set the prediction error threshold T_e to one standard deviation, set the number of tuples to nearest target to 5. As shown in Figure 5, there are 55 outlier records and 8 warning records in the data set. We subtract the outlier records in the data samples, and use to re-learn the prediction model. We predict MAE from 4.1325 to 3.8563, RAE from 0.2018 to 0.1765.

6.3 Dynamic Tuning

As shown in Figure 6.a and 6.b, by monitoring the whole running process of different workloads, we can find that OrientStream can save 8%-15% CPU usage and 38%-48% memory usage respectively. Among them, TM can save 8% of the CPU usage and 48% of the memory usage due to its memory intensive characteristic. WC save 12% of the CPU usage and 38% of the memory usage due to its communication intensive characteristic. Q3 can save 15% of the CPU usage and 45% of the memory usage due to its compute intensive characteristic. Note that we ignore the time overhead of switching to a new topology.



(a) CPU Usage (b) Memory Usage

Figure 6: Resource Usage

7. CONCLUSIONS AND FUTURE WORK

In this paper. we propose OrientStream to predict the resource usage of DDSMS by using incremental learning techniques, develop the learning tool that can be implemented data acquisition automatically based on the different workloads. Our approach uses MOA and Weka toolkits to build real-time prediction model by defining four-level features extraction mechanism. In addition, we introduce the EDKR algorithm and the outlier execution detection mechanism for improving the prediction accuracy, and verify the validity of our methods. In the future, we plan to build up the state operator for saving the state information at run time. In this case, we can achieve the arbitrary adjustment of different configurations without building a new topology.

8. ACKNOWLEDGMENTS

This research was partially supported by the grants from the Natural Science Foundation of China (No. 61532016, 61532010, 61379050, 91224008); the National Key R&D Program (No. 2016YFB1000602, 2016YFB1000603); Specialized Research Fund for the Doctoral Program of Higher Education (No. 20130004130001), and the Fundamental Research Funds for the Central Universities, the Research Funds of Renmin University (No. 11XNL010).

9. REFERENCES

[1] https://github.com/epfldata/squall/.
[2] http://moa.cs.waikato.ac.nz/.
[3] http://www.cs.waikato.ac.nz/ml/weka/.
[4] https://github.com/intel-hadoop/HiBench.
[5] D. W. Aha, D. Kibler, and M. K. Albert. Instance-based learning algorithms. *Machine Learning*, 6(1):37–66, 1991.
[6] M. Akdere, U. C?Etintemel, M. Riondato, E. Upfal, and S. B. Zdonik. Learning-based query performance modeling and prediction. In *Data Engineering (ICDE), 2012 IEEE 28th International Conference on*, pages 390–401, 2012.
[7] L. Aniello, R. Baldoni, and L. Querzoni. Adaptive online scheduling in storm. In *ACM International Conference on Distributed Event-Based Systems*, pages 207–218, 2013.
[8] C. Bishop. *Mixture Density Networks*. 1994.
[9] P. Domingos and G. Hulten. Mining high-speed data streams. In *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 71–80, 2000.
[10] T. Z. J. Fu, J. Ding, R. T. B. Ma, M. Winslett, Y. Yang, and Z. Zhang. Drs: Dynamic resource scheduling for real-time analytics over fast streams. *Computer Science*, 690(1):411–420, 2015.
[11] G. H. John and P. Langley. Estimating continuous distributions in bayesian classifiers. In *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, pages 338–345, 2013.
[12] A. Khoshkbarforoushha, R. Ranjan, R. Gaire, P. P. Jayaraman, J. Hosking, and E. Abbasnejad. Resource usage estimation of data stream processing workloads in datacenter clouds. *Computer Science*, 2015.
[13] J. Kreps, L. Corp, N. Narkhede, J. Rao, and L. Corp. Kafka: a distributed messaging system for log processing. netdbą́r11. *Proceedings of the Netdb*, 2011.
[14] J. Li, A. C. Nig, V. Narasayya, and S. Chaudhuri. Robust estimation of resource consumption for sql queries using statistical techniques. *Eprint Arxiv*, 5(11):1555–1566, 2012.
[15] L. Neumeyer, B. Robbins, A. Nair, and A. Kesari. S4: Distributed stream computing platform. In *Icdmw 2010, the IEEE International Conference on Data Mining Workshops, Sydney, Australia, 14 December*, pages 170–177, 2010.
[16] N. C. Oza and S. Russell. Experimental comparisons of online and batch versions of bagging and boosting. *Carcinogenesis*, 22(3):515–517(3), 2001.
[17] N. Poggi, D. Carrera, A. Call, and S. Mendoza. Aloja: A systematic study of hadoop deployment variables to enable automated characterization of cost-effectiveness. In *IEEE International Conference on Big Data*, pages 905–913, 2015.
[18] J. R. Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., 2014.
[19] M. J. Sax, M. Castellanos, Q. Chen, and M. Hsu. Aeolus: An optimizer for distributed intra-node-parallel streaming systems. In *IEEE International Conference on Data Engineering*, pages 1280–1283, 2013.
[20] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, and J. Donham. Storm@twitter. 2014.
[21] Y. Zheng, L. Zhang, X. Xie, and W. Y. Ma. Mining interesting locations and travel sequences from gps trajectories. In *International Conference on World Wide Web*, pages 791–800, 2009.