A Data Stream Prediction Strategy for Elastic Stream Computing Systems

Hanchu Zhang[1], Dawei Sun[1], Atul Sajjanhar[2], and Rajkumar Buyya[3]

[1] School of Information Engineering, China University of Geosciences, Beijing, 100083, P.R. China
zhanghanchu@cugb.edu.cn, sundaweicn@cugb.edu.cn
[2]School of Information Technology, Deakin University, Victoria 3216, Australia

atul.sajjanhar@deakin.edu.au
[3] Cloud Computing and Distributed Systems (CLOUDS) Laboratory, School of Computing and Information Systems, The University of Melbourne, Australia
rbuyya@unimelb.edu.au

Abstract. In a distributed stream processing system, elastic resource provisioning/scheduling is the main factor that affects system performance and limits system applications. However, in the data stream computing platform, resource allocation is often suboptimal due to the large fluctuations of the data stream rate, which creates a performance bottleneck for the cluster. In this paper, we propose a data stream prediction strategy (Dp-Stream) for elastic computing system to mitigate the resource allocation issue. First, we establish a back propagation (BP) neural network prediction model based on genetic simulated annealing algorithm to predict the trend of the data stream rate in the next time window of the cluster; second, according to the time latency, the estimation model adjusts the resources allocated to the critical operations of the critical path in the Directed Acyclic Graph (DAG) and finally, the resource communication cost is optimized. We evaluate the prediction accuracy and system latency of the proposed scheduling strategy in Storm. The experimental results prove the feasibility and effectiveness of the proposed strategy.

Keywords: Data stream prediction, Resource scheduling, Stream computing, Back propagation neural network, Storm.

1 Introduction

1.1 Background and Motivation

With the continuous development of science and technology, society has entered the era of big data, which is driven by a series of smart applications, smart devices, and smart services, including social networks, smart phones, and intelligent transportation [1]. In these scenarios, the amount of data is showing a trend of rapid growth, so it can be seen that the demand for real-time data stream processing services is also increasing. There is no doubt that the analysis and processing of real-time data has a very broad application prospect, and it is also a huge challenge. Therefore, a large number of distributed stream processing real-time computing platforms represented by Storm [2] and Spark Streaming [3] are derived to address the problem of timeliness of data, so that these data can be processed quickly within the constraints of time.

In a distributed stream processing system, the transmission rate of data stream usually has high volatility, for example, data stream surges during e-commerce promotional activities. On the other hand, data streams are reduced during the emergency incidents of smart power system. In this case, if the stream processing system can adjust resources according to the data demand when the data stream fluctuates the resource utilization of the system will be improved, so the research on elastic resource scheduling is of great significance [4]. Although Storm can meet the basic needs of data stream processing, it has many shortcomings in supporting elastic resource scheduling. The ideal elastic resource scheduling should accommodate the data stream traffic increase, and adjust the system to increase the resource allocation in real time [5]; and when the data stream traffic drops, the system should reclaim part of resources, in a timely manner. At the same time, if the system's resources are adjusted when the changed data stream arrives in the system, it will greatly increase the system's response time. Therefore, if resources are adjusted in advance before the data stream rate changes, a lot of system response time utilized for resource adjustments upon data arrival can be salvaged, thereby reducing the latency of the system and improving the performance of the system. As a result, we can predict the rate of the data stream to deploy resources in advance.

1.2 Contributions

Based on the above background, we propose a data stream prediction strategy (Dp-Stream) for elastic stream computing system. We mitigate the problem by making the following contributions:

(1) We develop a BP neural network model based on genetic simulated annealing algorithm to predict the change in trend of the data stream rate in the next time window of the cluster;

(2) We propose a resource scheduling strategy from a systematic perspective, according to the latency estimation model, resources are adjusted for the critical operations of the critical path in the DAG to obtain a lower system latency and higher resource utilization.

1.3 Paper Organization

The organization of the rest of the paper is as follows: Section 2 introduces the related work. In section 3, the application model, prediction model and latency estimation model are introduced. Section 4 focuses on Dp-Stream, including system architecture, data stream prediction algorithm and resource scheduling algorithm. Section 5 introduces the experimental environment, parameter settings and performance evaluation of Dp-Stream. Finally, Section 7 presents conclusions and future work.

2 Related Work

In stream processing systems, due to the volatility of data streams, insufficient system resources will cause system performance degradation. How to allocate resources effectively and reasonably is the main challenge faced by streaming computing platforms. At present, some researchers use predictive methods to optimize the allocation of system resources to improve system performance.

In [12], the authors used the model predictive control (MPC) method to design an elastic data stream processing strategy for multi-core systems, and proposed a tree structure to describe the search space, and used the branch and bound method to determine Optimal resource allocation, reducing MPC runtime overhead and optimizing throughput and latency performance.

[13] proposed the use of autoregressive integrated moving average (ARIMA) model to predict the input traffic changes in the working node, using the resource cost model to track and limit the node's CPU usage level within the acceptable range of strategies.

In [15], the authors used an integrated regression model to predict CPU and memory usage, and used incremental learning techniques to build the prediction model in real time. At the same time, according to the relative independence of different regression models, the weighted integral algorithm of the regression model is given, and the abnormal value detection mechanism is introduced to monitor the abnormal execution to improve the accuracy of prediction.

Table 1 Comparison of our work and other related work

Parameter	**Related Work**				**Dp-Stream**
	[13]	**[15]**	**[16]**	**[17]**	
Prediction Modelling	✓	✓	×	×	✓
Performance Modelling	✓	✓	✓	✓	✓
Elastic Strategy	×	×	×	✓	✓
Resource Saving	✓	✓	×	✓	✓

At the same time, there are many researchers working on the scheduling strategy in distributed stream processing system. [16] proposed a dynamic resource scheduling strategy DRS based on cloud data stream management system. It analyzes each node in the task topology through the Erlang formula to obtain its data processing latency, and then uses the Jackson queuing network to aggregate the processing latency of the entire topology on a weighted average.

In [17], the authors proposed an adaptive online solution for scheduling and resource implementation on a stream processing framework. It can determine the number of resources required by each instance in a timely manner to handle unexpected load peaks without causing congestion, and wasteful allocation of resources. A resource cost-aware layout algorithm is proposed, which can minimize the number of affected worker threads.

In summary, the above solutions provide valuable insights for the elastic scheduling strategy of distributed stream processing systems, including prediction methods, resource scheduling, and resource allocation. However, the current methods still have some limitations, so it is necessary to develop novel methods to adapt to the era of big data. The comparison between our work and other closely related works is given in Table 1.

3 Problem Statement

In this section, we introduce the application DAG model, prediction model and latency estimation model in big data stream computing environments.

3.1 Application Model

Data stream processing systems model the logic of the real-time application as a DAG, represented by $DAG = (V(G), E(G))$,where $V(G) = \{v_1, v_2, \cdots$ is a set of n vertices, and each vertex represents a Spout or Bolt component. Spout is responsible for reading data from the data source and sending tuples (Tuple) to the Topology. Tuples are units of data processed by Storm. Bolt encapsulate processing logic, realize the specific processing of data, each processing a tuple. $E(G) = \{e_{1,2}, e_{1,3}, \cdots$ is a finite set of directed edges, the weight associated with a vertex or an edge respectively represents its computation cost or communication cost. A vertex in each component is an instance, and the number of vertices in a component is the number of instances of the component, which is also called the parallelism of the component. For each DAG, first use the graph-based depth fist traversal algorithm to find the largest path from v_1 to v_n . This maximum path is called the critical path of the topological graph, all nodes on the critical path are called Critical Operations (CO).

3.2 Prediction Model

We propose a BP neural network based on the genetic simulated annealing algorithm to predict the input rate of data stream r_{t+1} of the application in the future time window $t+1$ based on the historical input rate $R_{in}=(r_1,r_2,r_3,...,r_t)$.In the BP neural network, the historical input rate of the application $R_{in}=(r_1,r_2,r_3,...,r_t)$ is used as the input data of the neural network, $W=(\omega_1,\omega_2,\omega_3,...,\omega_n)$ is the weight corresponding to the input data to the hidden layer, $H=(h_1,h_2,h_3,...,h_n)$ is the output value of the hidden layer, $Y=(y_1,y_2,y_3,...,y_k)$ is the output value of the output layer, and $T=(t_1,t_2,t_3,...,t_k)$ is the target value of the output layer. The output result of the hidden layer can be calculated by (1).

$$h_n = f\left(\sum_{i=1}^{t}\omega_{ni}\cdot r_i\right), \tag{1}$$

where we use $f=\dfrac{1}{1+e^{-kx}}$ as the transfer function of the BP neural network. And the output of the output layer can be expressed by (2).

$$y_k = f\left(\sum_{n=1}^{N}\lambda_{nk}\cdot h_n\right), \tag{2}$$

where, λ_{nk} is the weight between the hidden layer and the output layer. Therefore, the error between the target output and the actual output can be obtained by (3),

$$b=\frac{1}{2}(T-Y)^2=\frac{1}{2}\left[\sum_{k=1}^{K}t_k - f\left(\sum_{n=1}^{N}\lambda_n\cdot h_n\right)\right]^2. \tag{3}$$

Genetic algorithm is a kind of global search algorithm, which has strong global search ability, but it is prone to premature convergence, which leads to the problem of local optimal solution. In the process of searching for the optimal solution, the simulated annealing algorithm uses a certain probability to jump out of the local optimal solution to avoid the shortcomings of falling into the local optimal solution. In the forecasting process, the genetic simulated annealing algorithm is used to replace the back propagation process of the BP neural network. First, get the initial population from the initial solution $P(k)$ of the BP neural network according to the genetic algorithm, and then perform crossover and mutation operations on the initial population to obtain new individuals x'. At this time, the energy value of the system $E'=b'$ is obtained according to the simulated annealing algorithm. Then the energy difference can be expressed as $\Delta E = E'-E = b'-b$. Metropolis' probability acceptance criteria is used to determine whether to accept or reject the new state. The probability of the system accepting the state x' can be calculated by (4).

$$P = \begin{cases} 1 & , if\ \Delta E < 0, \\ \dfrac{1}{z(t_k)} \exp\left(-\dfrac{\Delta E}{K t_k}\right) & , otherwise, \end{cases} \tag{4}$$

where, K is Boltzmann's constant, t_k is the current temperature. If the probability P is an arbitrary random number in the interval $[0,1]$, the new state x is accepted, otherwise the current state x is retained. In the iterative process of the algorithm, the solid molecules continue to move to a place with relatively low energy, and the state probability distribution of the energy E of the solid in a certain state tends to the Gibbs distribution can be calculated by (5),

$$z = \frac{1}{\sum_i \exp\left(-\dfrac{\Delta E_i}{t_k}\right)}. \tag{5}$$

The genetic simulated annealing algorithm makes the initial point of the iteration continue to iterate from an initial solution, and correct the weights and error thresholds of the BP network to gradually find the optimal solution. Finally, the output of the output layer is used as the input rate of the application in the time window $t+1$ to be predicted.

3.3 Latency Estimation Model

In data stream processing system, for data tuples, the flag of processing completion is that it and all the intermediate tuples it produces are processed by its corresponding compute nodes. T is used to represent the time experienced from the data stream input tuple to the application to the completion of its processing by the last computing node. T_i is used to represent the time required for the v_i node to process the tuple. T_i including tuple processing time Tp_i, the time Ts_i required to transmit tuples from node v_i to v_j.

The tuple processing time is related to the processing rate of the task, therefore, the faster the processing rate, the shorter the processing time. Hence, it is inversely proportional to the average processing rate, that can be calculated by (6).

$$Tp_i = \frac{r_i \cdot \Delta t}{\sum_{k=1}^{k} p_{ik}}, \tag{6}$$

where p_{ik} is the tuple processing rate under the assumption that the instances of all nodes are homogeneous. It can be seen that when the processing rate of node v_i is larger, Tp_i is smaller. Therefore, when the current processing rate cannot achieve the lower processing time, we can increase the number of node instances (that is, increase the node parallelism) to reduce the processing time.

Then the time required to transmit tuples from node v_i to v_j is related to the network bandwidth, and if the instances of two nodes are on the same Worker node, the transmission time between them can be ignored. Therefore, Ts_i can be calculated by (7),

$$Ts_i = \begin{cases} \dfrac{r_i \cdot \Delta t}{B_{ij}} & ,if\ i, j\ in\ different\ Worker\ node. \\ 0 & ,otherwise. \end{cases} \tag{7}$$

The value T_i is the weight of the sum of the value of all calculated values, that can be expressed by (8).

$$T_i = w_i \cdot (Tp_i + Ts_i), \tag{8}$$

where, w_i is the weight of node v_i on the path.

In stream processing applications, the required time between data stream on the path is equal to the sum of the latency of the nodes with the longest required time on the path, that is, the sum of the latency of all nodes that belong to the critical operation on the critical path, so each input element in the application the response time T of the group can be expressed by (9),

$$T = \sum_i T_i \quad ,v_i \in CO. \tag{9}$$

Therefore, the optimization goal of this paper can be defined as: in the case of meeting the predicted data input rate, find the appropriate number of instances (executor) for each critical operation node of the critical path on the DAG, so as to minimize the latency and meet the requirements of each Worker node resource capacity to improve resource utilization.

4 Dp-Stream Overview

Based on the relevant model discussed in the Section 3, we propose a scheduling strategy, namely, Dp-Stream. Dp-Stream is a resource scheduling strategy based on the prediction of data stream rate. For data stream rates that fluctuate drastically, it can adjust system resources according to the stream rate before the changing data stream arrives to ensure low latency. In order to provide an overview of Dp-Stream, this section focuses on Dp-Stream, including data stream rate prediction algorithms, resource scheduling methods based on predicted stream rates and its system architecture.

4.1 System Architecture

The basic structure of Storm includes Nimbus, Zookeeper and several Worker nodes. Nimbus is responsible for receiving the DAG submitted by the user and deploying it to each Worker node. Each Worker node runs a Supervisor to monitor the tasks assigned by Nimbus to the Worker node. Zookeeper is used to save the working status of Nimbus and Supervisor.

In order to implement Dp-Stream, the basic structure of Storm needs to be improved. The improved cluster mainly includes an input rate monitoring module, a prediction module, and a resource scheduling module. The input rate monitoring module is used to monitor the current stream rate of the system data stream and save the record in the database. The prediction module applies historical input rate from the database to perform forecasting according to the algorithm proposed in Section 4.2. The resource scheduling module uses the IScheduler interface to implement the scheduling strategy proposed in Section 4.3. The Dp-Stream deployment architecture is shown in Figure 1.

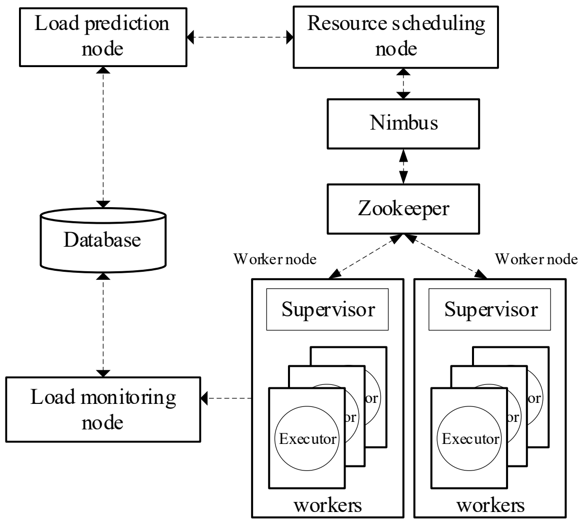

Fig. 1. Dp-Stream deployment architecture

4.2 Data Stream Prediction Algorithm

In order to realize a timely response of stream processing applications to rate changes, it is necessary to predict the data stream rate of the next time window through the prediction model, so as to allocate appropriate resources to the application in advance.

The genetic simulated annealing algorithm uses the annealing algorithm to find the optimal solution by adding the local optimal solution as a potential candidate, and then adjusting the entire search range with new candidates to improve the accuracy of the algorithm. BP neural network is prone to converge at the local optimal solution, so this paper proposes a BP neural network based on genetic simulated annealing algorithm as a prediction model, which is described in Algorithm1.

Algorithm 1: Prediction algorithm of BP network based on genetic simulated annealing algorithm

```
1.  Input: Population size $S$ , iteration number $N$ , crossover probability $P_c$ ,
    mutation probability $P_m$ , simulated annealing initial temperature $t_m$ , tem-
    perature decay function parameter $\alpha$
2.  Output: Data stream rate $r_{t+1}$ in the t+1 time window
3.  Initialize the BP network and encode the weight and threshold, and initial-
    ize the population $P(k)$ then
4.  Use fitness function to evaluate current individuals $E = b$
5.  for n<N do
6.    if fitness value satisfies the termination condition then
7.      return
8.    else
9.      Select and copy the population then
10.     Perform crossover and mutation operations on the population
11.       then do Simulated annealing operation to obtain a new fitness value
    $E' = b'$ at temperature $t_k$
12.         if $\Delta E > 0$
13.           Accept the new solution
14.         else Calculated $P$ by (4)
15.           if $P \in [0,1]$
16.             accept the new solution
17.           else keep the original solution
18.         then gradually lower the temperature, $t_{k+1} = \alpha t_k$ , $n = n+1$
19.  return
```

In Algorithm 1, the initial solution of the algorithm is obtained by the BP neural network, and the initial population of the genetic algorithm is obtained (Step 3). Then a fitness function is used to evaluate the fitness value of the candidates in the population. If the fitness value meets the termination condition of the algorithm, the result will be output, otherwise the population will be cross-mutated (Step 4 to 10). At temperature t_k, calculate the fitness value of the individuals in the population after genetic manipulation and compare it with their respective parental fitness values, and decide whether to accept the individuals after genetic manipulation according to Metropolis acceptance criteria (Step 11 to 19). Then the population is cooled down.

4.3 Resource Scheduling

After predicting the rate of data stream, Dp-Stream will perform resource adjustment and scheduling operations based on the data stream rate in the next time window obtained by the prediction algorithm. Firstly, execute resource adjustment of the critical operation nodes on the critical path in the DAG, increase or decrease the number of instances of the component to match the changes in the coming data stream rate, so as to ensure a lower system latency. Then place the changed instance in the appropriate Worker node according to the remaining resource capacity on the Worker node to ensure the resource utilization of the system. The scheduling algorithm of the Dp-Stream is described in Algorithm 2.

Algorithm 2: Resource scheduling algorithm

```
1.  **Input**: critical operation set CO, threshold $\omega$ and minimum value $\lambda$ of critical operation $T_i$, set of Worker nodes $W_j$
2.  **Output**: calculated time for critical operations $T_i$, the change number of instances of each critical operation node $n_i$, the placement of executor instances on each Worker node $W_j'$
3.  for each $v_i \in CO$ do
4.    calculate $T_i$ by (8)
5.    if $T_i > \omega$ then
6.      while $T_i > \omega$ do
7.        increase the number of instances $N_i = N_i + 1$, calculate $T_i$
8.    else if $T_i < \lambda$ then
```

```
9.      while $T_i < \lambda$ do
10.         reduce the number of instances $N_i = N_i - 1$, calculate $T_i$
11. for each $W_j$ do
12.   calculate the remaining resource capacity $R_j$
13. for each $n_i$ do
14.   if $n_i > 0$ then
15.     if the remaining capacity $R_j$ is more than the resource requirement of
        the instance $r_k$ and $R_j' = R_j - r_k$ is the smallest then
16.         place an instance into, $R_j = R_j'$
17.   else if $n_i < 0$ then
18.     if $W_j$ containing the instance and $R_j' = R_j + r_k$ is the biggest hen
19.         release the instance form $W_j'$, $R_j = R_j'$
20. return $T_i, W_j'$
```

In Algorithm 2, we adjust the resources of the critical operation nodes on the critical path in the DAG. First, calculate the latency of each critical operation node (Step 4). When the latency of the critical operation exceeds the threshold ω, add an Executor instance to the node until the latency is less than the threshold (Step 5 to 7). Similarly, when the latency of the critical operation is less than the minimum value λ, one Executor instance is reduced for the node until the latency is greater than the minimum value (Step 8 to 10). Then calculate the remaining resource capacity of each Worker node (steps 11-12). When an executor instance is placed, select a Worker node that meet the resource requirements of the instance and has the smallest remaining resource capacity after placing the instance to increase resource utilization (Step 13 to 16). Similarly, when releasing an executor instance, select the Worker node that contains the instance and the remaining resource capacity is the largest after the instance is released, so as to generate more resources to prepare for the subsequent placement of the instance (step 17-19).

5 Performance Evaluation

In this section, we first discuss the experimental environment and parameter settings, which is followed by analysis of performance evaluation results.

5.1 Experimental Environment and Parameter Setup

The proposed Dp-Stream system is developed based on Storm 2.1.0, and installed on top of CentOS 6.8. The cluster consists of 7 machines, with one designated machine serving as the master node, running Storm Nimbus, two designated as Zookeeper node, and the rest 4 machines working as Supervisor nodes. The Nimbus node is equipped with 12 Intel(R) Xeon(R) CPU X5650 @ 2.67 GHz 6-core processors and 12GB of memory. Zookeeper nodes and Supervisor nodes are virtual machines, equipped with 2 Intel(R) Xeon(R) CPU X5650 @ 2.67 GHz 2-core processors and 4GB of RAM. The software configuration of Dp-Stream platform is shown in Table 2.

Table 2 Software configuration of Dp-Stream

Software	**Version**
OS	CentOS 6.8 64bit
Storm	apache-storm-2.1.0
JDK	Jdk1.8 64bit
Zookeeper	zookeeper-3.4.14
Python	python 2.7.2
Maven	Maven 3.6.2
MySQL	Mysql 5.1.73

WordCount is an application used to count the frequency of words in English text. It consists of a Spout and two Bolt. Top-N is an application for counting trending topics, including one Spout and three Bolt. Here, we use WordCount and Top-N as the topology for our experiments.

5.2 Prediction model evaluation

In order to evaluate the accuracy of the prediction model for predicting the data stream rate, the experiment in this section simulates the phenomenon of violent rate fluctuations in the online business. The data stream rate is set as a random rate, and the arrival rate range of the random rate is set to [1000,3000] tuple/s. First, collect the actual simulation value of 200s as the training data of the neural network model. In order to describe the accuracy of the prediction algorithm more intuitively, we continuously collect 20s predicted and true values of the data stream, we recorded the predicted values and actual values of 5s, 10s, 15s, and 20s, and calculated the mean absolute error

(MAE), root mean squared error (RMSE) and mean relative error (MRE). The results are summarized in Table 3.

Table 3 Comparison of predicted value and actual value at different time and evaluation of prediction method

Time		Predicted Value		Actual Value	
5		2101		2056	
10		2511		2421	
15		2114		2201	
20		1864		1799	
MAE	61.92	**RMSE**	66.76	**MRE**	0.031

In Table 3, both MAE and RMSE are around 60, and MRE<0.05. Therefore, it can be concluded that our prediction method has a good accuracy and can meet the requirements of Dp-Stream for the accuracy of the prediction algorithm.

5.3 Resource Scheduling Strategy Evaluation

We apply the simulated data stream to the Dp-Stream and Storm's default scheduler, and observe the changes of system latency. The experiments evaluate the system latency under Dp-Stream and Storm default scheduler.

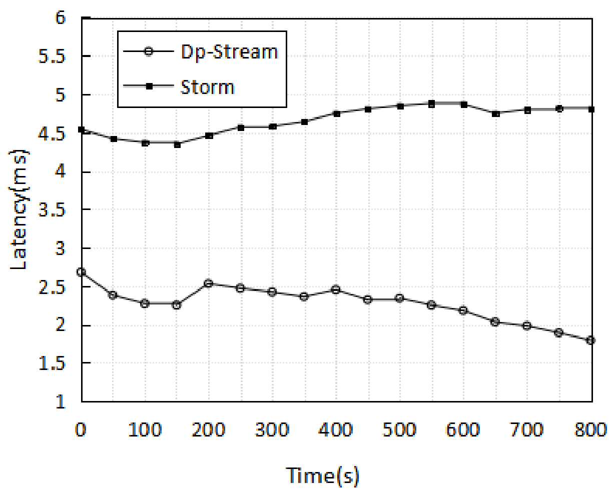

Fig. 2. System latency comparison when running WordCount

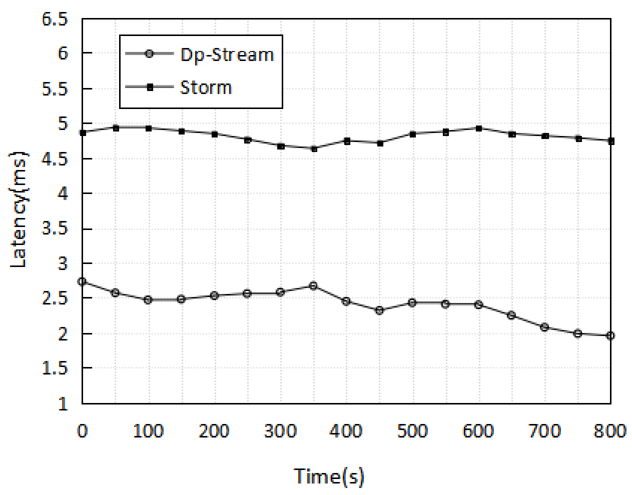

Fig. 3. System latency comparison when running Top-N

In Figure 2 and 3, the horizontal axis represents the experiment time (s), and the vertical axis represents the waiting time (ms) of the Dp-Stream and Storm default scheduler. Figure 2 records the system latency comparison of Dp-Stream and Storm default scheduler when running WordCount application. Due to the constant fluctuation of the input rate of the data stream, during the 800s observation period, the average system latency of Dp-Stream is about 2.27ms, and the average latency of Storm default scheduler is about 4.66ms. Figure 3 records the system latency comparison of Dp-Stream and Storm's default scheduler when running Top-N application, and the average system latency of Dp-Stream is about 2.40ms, and the average latency of Storm default scheduler is about 4.84ms. It can be seen from these two figures that compared with the default scheduler of Storm the latency of Dp-Stream is much lower than the default scheduler. The Dp-Stream has been adjusted in advance to deal with rate changes, so the latency has only a small fluctuation and the overall trend is decreasing. However, the latency of the default scheduler is higher than Dp-Stream and the overall trend is increasing. This shows that our Dp-Stream is more effective.

6 Conclusions and Future Work

In a distributed stream processing system, in the face of continuous changes in the data stream rate of the data stream, if there is no proper resource scheduling strategy, the latency of the application will increase significantly. In this paper, we propose a data stream prediction strategy for elastic stream computing system (Dp-Stream), with the goal of allocating system resources in advance to ensure low system latency before changing stream arrive. Experimental results show that the prediction model proposed in this paper is more accurate, and the proposed scheduling algorithm reduces system latency compared with Storm's default scheduler.

In the future, we will focus on improving the prediction algorithm to improve the accuracy of prediction, and considering heterogeneous situation of Worker nodes in the resource scheduling strategy to make the scheduling strategy better.

Acknowledgements. This work is supported by the National Natural Science Foundation of China under Grant No. 61972364 and the Fundamental Research Funds for the Central Universities under Grant No. 2652021001. This work is also supported by Melbourne-Chindia Cloud Computing (MC3) Research Network.

References

1. F. Zhu, Y. Lv, Y. Chen, X. Wang, G. Xiong, F. Y. Wang: Parallel Transportation Systems: Toward IoT-Enabled Smart Urban Traffic Control and Management. In: IEEE Transactions on Intelligent Transportation Systems, vol. 21, no. 10, pp. 4063-4071 (2020).
2. A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, J. Donham, N. Bhagat, S. Mittal, D. Ryaboy: Storm@twitter. In: 2014 ACM SIGMOD International Conference on Management of Data, SIGMOD 2014, ACM Press, pp. 147-156 (2014).
3. S. Chintapalli, D. Dagit, B. Evans, R. Farivar, T. Graves, M. Holderbaugh, Z. Liu, K. Nusbaum, K. Patil, B. J. Peng, P. Poulosky: Benchmarking Streaming Conputation Engines: Storm, Flink and Spak Streaming, In: 2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), pp. 1789-1792 (2016).
4. X. Liu, R. Buyya: Resource Management and Scheduling in Distributed Stream Processing Systems: A Taxonomy, Review, and Future Directions. In: ACM Computing Surveys, vol. 53, no.3, pp.1-41 (2020).
5. J. Zhang, C. Li, L. Zhu, Y. Liu: The Real-Time Scheduling Strategy Based on Traffic and Load Balancing in Storm. In: 2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems, Sydney, NSW, Australia, pp. 372-379 (2016).
6. W. Duan, L. Zhou: Task Scheduling Optimization Based on Firefly Algorithm in Storm. In: 2020 IEEE 10th International Conference on Electronics Information and Emergency Communication (ICEIEC), pp. 150-154 (2020).
7. T. Heinze, V. Pappalardo, Z. Jerzak, C. Fetzer: Auto-scaling techniques for elastic data stream processing. In: 2014 IEEE 30th International Conference on Data Engineering Workshops, Chicago, IL, USA, pp. 296-302 (2014).
8. N. Hidalgo, D. Wladdimiro, E. Rosasl: Self-adaptive processing graph with operator fission for elastic stream processing. The Journal of Systems & Software 127 (2017).

9. Cardellini V, Nardelli M, Luzi D: Elastic Stateful stream processing in storm. In: International Conference on High Performance Computing & Simulation, pp. 583-590 (2016).

10. Chakraborty R, Majumdar S: A priority-based resource scheduling technique for multitenant storm clusters. In: 2016 International Symposium on Performance Evaluation of Computer and Telecommunication Systems, pp. 1-6 (2016).

11. Y. Zhou, Y. Liu, C. Zhang, X. Peng, X. Oin: TOSS: A Topology-based Scheduler for Storm C1usters. In: 2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), New Orleans, LA, USA, pp. 587-596 (2020).

12. T. D. Matteis, G. Mencagli: Proactive elasticity and energy awareness in data stream processing. The Journal of Systems & Software, vol. 127, pp. 302-319 (2017).

13. M. R. Hoseiny Farahabady, H. R. Dehghani Samani, Y. Wang, A. Y. Zomaya, Z. Tari: A QoS-aware controller for Apache Storm. In: 2016 IEEE 15th International Symposium on Network Computing and Applications (NCA), Cambridge, MA, USA, pp. 334-342 (2016).

14. Hoseiny Farahabady M R, Zomaya A Y, Tari Z: QoS- and Contention- Aware Resource Provisioning in a Stream Processing Engine. In: 2017 IEEE International Conference on Cluster Computing (CLUSTER), pp. 137-146 (2017).

15. Wang C, Meng X, Guo Q, Weng Z, Yang C: OrientStream: A Framework for Dynamic Resource Allocation in Distributed Data Stream Management Systems. In: 25th ACM International on Conference on Information and Knowledge Management, ACM Press, pp. 2281-2286 (2016).

16. T. Z. J. Fu, J. Ding, R. T. B. Ma, M. Winslett, Y. Yang, Z. Zhang: DRS: Dynamic Resource Scheduling for Real-Time Analytics over Fast Streams. In: 2015 IEEE 35th International Conference on Distributed Computing Systems, Columbus, OH, USA, pp. 411-420 (2015).

17. S. Liu, J. Weng, J. H. Wang, C. An, Y. Zhou, J. Wang: An Adaptive Online Scheme for Scheduling and Resource Enforcement in Storm. In: IEEE/ACM Transactions on Networking, vol. 27, no. 4, pp. 1373-1386 (2019).

18. W. Wang, C. Zhang, X. Chen, Z. Li, H. Ding, X. Wen: An On-the-Fly Scheduling Strategy for Distributed Stream Processing Platform. In: 2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Melbourne, VIC, Australia, pp. 773-780 (2018).

19. X. Liu, R. Buyya: D-Storm: Dynamic Resource-Efficient Scheduling of Stream Processing Applications. In: 2017 IEEE 23rd International Conference on Parallel and Distributed Systems, pp. 485-492 (2017).

20. T. De Matteis, G. Mencagli: Elastic Scaling for Distributed Latency-Sensitive Data Stream Operators. In: 2017 25th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP), St. Petersburg, Russia, pp. 61-68 (2017).