Storm Scheduling Based on Non-cooperative Game

Haixu Ma1*, Feng Xin2, Guilei Wang3, Kaiwei Sun4, Mingyuan Xiong5
1-5School of Computer Science and Technology, Changchun University Of Science and Technology, Changchun, Jilin, 130022, China
*Corresponding author’s e-mail: 1761290514@qq.com

Abstract: With the continuous development of Internet technology, a large amount of data with rich value has been generated, and lagging data analysis will affect the timeliness of data, so real-time streaming data processing is becoming increasingly important. Storm is a pure streaming data processing framework, but it uses the polling scheduling algorithm by default. This algorithm ignores the network communication overhead between workers and cluster load balancing. Aiming at Storm's default scheduling problem, a non-cooperative game-based Storm scheduling algorithm (G-Storm) was proposed. Storm extracts the source data in real time through the component “Spout”, passes it to the logical processing component ”Bolt”, and finally loads it into the target warehouse. The experimental results show that the game scheduling algorithm proposed in this paper reduces the system processing delay by 28.6% compared with the default scheduling algorithm.

1. Introduction
With the continuous development of Internet technology, massive amounts of data with rich value have been generated, and lagging data analysis will affect the real-time role of data. For example, Facebook generates more than 10 petabytes of data per month, and the number of Weibo on Twitter is more than 100,000 per minute. The data continuously generated by practical activities gradually accumulated to the source database, and a large amount of data became precipitation data, which affected the real-time processing and analysis and mining of the source data, and thus could not provide a powerful information support for information decision makers. Therefore, massive amounts of data need to be processed in real time. Storm is a pure streaming data processing framework, but polling scheduling is used by default. There is room for improvement in network communication between nodes and cluster load balancing.

Literature[1] proposed two adaptive scheduling algorithms, offline and online, in the Storm environment. The online scheduling algorithm formulates scheduling strategies based on real-time monitoring of node loads and cluster communication loads, which makes up for the shortcomings of offline scheduling. For complex topologies, it is easy to fall into a local optimum. Xiong Anping and others introduced the concept of topological hot edges[2]. This algorithm migrates high-frequency hot-edge association task pairs to the same working node, but the algorithm only considers the task of optimizing high-frequency hot-edge communication within the topology. Lu Liang proposed a weight-based task scheduling algorithm[3], designed an edge weight gain model, and moved tasks to nodes with larger edge weight gain values. However, there is a problem that the load of one working node is excessively lower than that of the remaining nodes. Liu Su proposed a task scheduling strategy based on topology[4], which first moves the threads corresponding to the components with the largest topology to the nodes with sufficient CPU resources, but moving the tasks to a node as far as possible must affect
the load balance of the cluster. Wang Lin proposed to used the idea of ant colony algorithm to improve Storm scheduling\cite{5}, but the algorithm is prone to local optimal solutions.

2. Storm scheduling ideas
The Storm scheduling algorithm uses the polling algorithm by default. This algorithm has optimization space for node network communication and load balancing. The default scheduling simply distributes the instantiated executors evenly to each node, and there is a large amount of network communication overhead between the working nodes on the cluster. The main idea of scheduling is to minimize communication between nodes while taking into account cluster load balancing. The default scheduling simply distributes the instantiated executors evenly to each node, and there is a large amount of network communication overhead between the working nodes on the cluster. The main idea of scheduling is to minimize communication between nodes while taking into account cluster load balancing.

3. Related definitions
In Storm scheduling, try to maximize the internal traffic of the nodes (then the traffic between nodes is the smallest) while taking into account the load balance of the cluster, and introduce a revenue function (game function) to solve the optimal solution of the system scheduling. For the above model, the following definitions are made:

Defined $f$ as the weighted value of the total data transmission within the node, as shown in equation (1):

$$ f = \alpha F = \alpha \sum_{k=1}^{S} \sum_{i,j=1}^{S} S_{ij} r_{ij} $$

Where: $\alpha$ as the weight of internal traffic of the node, $F$ as the total amount of internal data transmission of the node, $S$ as the cluster slot set, $r_{ij}$ as the data flow between tasks, $S_{ij} r_{ij}$ as the internal data flow $r_{ij}$ of the first $k$ slot.

Define $\theta$ and $\eta$ as the standard deviation of CPU and memory load of the work node, as shown in formula (2) and (3):

$$ \theta = \sqrt{\frac{\sum_{k=1}^{N} (W_{mk} - \bar{W}_{mk})^2}{N}} \quad (2) $$

$$ \eta = \sqrt{\frac{\sum_{k=1}^{N} (M_{mk} - \bar{M}_{mk})^2}{N}} \quad (3) $$

Where: $\bar{W}_{mk}$ as average load of cluster CPU, $\bar{M}_{mk}$ as average load of cluster memory.

The problem of scheduling optimization can be transformed into the problem of the maximum value of the income function $u$ under the conditions of formula (4), (5) and (6).

$$ u = \frac{f}{g} = \frac{\alpha F}{\lambda \theta + \gamma \eta} \quad (4) $$

$$ 0 \leq f \leq \sum_{k=1}^{S} \sum_{i,j=1}^{S} S_{ij} r_{ij} \quad (5) $$

$$ g > 0 \quad (6) $$

Where: $g$ represents the weighted sum of cluster CPU and memory load standard deviation, and $\lambda$ and $\gamma$ represent the weights of $\theta$ and $\eta$ respectively. Generally $\lambda \geq \eta$, mainly considers the impact of CPU load on the system.
4. Game modeling
This paper constructs a non cooperative game model of storm topology control, in which the players in the game model or game participants are tasks in the scheduling system; Participant's strategy is that when they change their own strategies while others' strategies remain unchanged, the overall performance of the system improves, as shown in equation (7):

\[ \sigma = [\sigma_1, \sigma_2, \sigma_3, \ldots, \sigma_n] \] (7)

It is called a scheduling strategy vector. According to the definition of Nash equilibrium point, when the constructed storm task scheduling control model reaches Nash equilibrium through game, it can be considered that the system revenue has reached steady state. No task can improve itself and the overall utility of the system by only changing its scheduling strategy. Strategy set is a necessary and sufficient condition for Nash equilibrium point of the proposed multi-task resource allocation game model, which satisfies formula (8):

\[ u(\sigma_i^*, \sigma_j^*) \geq u(\sigma_i^*, \sigma_j^*) \] (8)

The game model proposed in this paper is based on the existence theorem of the Nash equilibrium II to find the existence of the Nash equilibrium. First, \( f \) is a non-empty bounded closed convex set on Euclidean space. With the continuous optimization of the game model, the load standard deviation gradually decreases and stabilizes. Therefore, \( g \) is a non-empty bounded closed concave on European space set. In summary, the return function \( u \) is a bounded closed convex set. There is a maximum return function \( u \) for topology task scheduling, so there is a Nash equilibrium point in the game model of the Storm cluster topology scheduling problem.

5. Algorithm Design
The premise of the implementation of this algorithm is to monitor the operating status of the cluster and consider load balancing while obtaining the maximum internal traffic of the nodes. The prime minister of this algorithm runs the default algorithm. After the operation is stable, it collects and stores the data flow between tasks in the node and the CPU load and memory load of the node. During the execution of the default algorithm, if the monitored CPU load duration interval exceeds the threshold, the Storm scheduling algorithm based on non-cooperative games is triggered. In Storm topology operation, if the CPU load of each node in the cluster continues to be uneven, that is, within a certain interval, the difference between the maximum and minimum CPU load of each working node in the cluster is greater than the threshold, then game scheduling is triggered again algorithm.

The game algorithm is based on collecting default scheduling information and then determining whether to trigger game scheduling. If the number of initially scheduled cluster node tasks is unbalanced, the initialization is repeated. Then iterate through all the tasks in each node, move to different nodes respectively, and record the optimal revenue value. The total number of traversal times \( M(50-100) \), if the optimal value has not changed, it is the optimal value. This scheduling algorithm starts from an adaptive perspective and uses an iterative approach to maximize the cluster's revenue, thereby deriving the optimal scheduling strategy. This process requires each task to traverse the slots, calculating the optimal return will consume a certain time.

6. Experimental results and analysis

6.1. Lab environment
The test environment is a Storm-1.2.2 cluster built by 13 PCs. Each computer is equipped with an Intel Core i7-9700K CPU @ 3.6 GHZ x8 processor and 8 GB of RAM. The machines are all installed with CentOS-7.0 64 system, interconnected by switched Ethernet 1Gbit in LAN. Three of them run Zookeeper cluster and Kafka cluster together, Nimbus, process UI and database Mysql run on one of them. The remaining 10 nodes run the Supervisor daemon.
6.2. Experimental program
Implement the Pluggable Scheduler using the pluggable custom task scheduler provided by the Storm framework, which is designed for developers. Use the system's built-in Lead Monitor to monitor cluster operation, and Load Monitor runs as a background process. The experimental parameters are ideal values determined after several experiments and fine-tuning. The experiment set up 10 worker nodes and 10 worker processes, that is, only one worker process is deployed on each worker node, which can effectively reduce the communication overhead between worker nodes. The game weights $\alpha$, $\lambda$, and $\gamma$ are 6, 4, and 2, respectively. When the node CPU load lasts for more than 60% for 80 seconds, the scheduling is triggered. Table 1 lists some experimental parameters.

| Parameter        | Value | Description       |
|------------------|-------|-------------------|
| Spout            | 16    |                   |
| Deal_Bolt        | 32    | Number of component instances |
| Load_Bolt        | 32    |                   |
| Worker           | 10    | Number of worker processes |
| Acer.executors   | 10    | Acer number       |
| Spout.Pending    | 50    | Spout cache length |

Table 1 G-Storm parameter configuration

In order to evaluate the performance of the proposed non-cooperative game algorithm G-Storm, this paper develops a data generator program to generate the source data, generating $1 \times 10^9$ tuples, each tuple contains 20 alphabetic fields. The processing flow proceeds as follows. The dataset is stored in a Kafka (topic=3, partition=12) message sequence database, and Storm reads the data from Kafka and loads it into a Mysql database after processing.

7. Experimental results
Using our designed data set, the non-cooperative game-based Storm scheduling algorithm G-Storm proposed in this paper is superior to the default polling scheduling algorithm adopted by Storm systems in terms of data processing latency. Experimental results show that the proposed improved algorithm reduces the system latency by about 28.6% compared to the default algorithm. As shown in Figure 1.

![Figure 1 Comparison of game scheduling and default scheduling](image)

8. conclusion
With the development of society, there is an urgent need to improve the performance of data real-time stream processing. Storm uses the polling scheduling algorithm by default. There is room for optimization in node network communication overhead and cluster load balancing. This paper proposes Storm scheduling for non-cooperative games. Experiments show that the improved Storm scheduling algorithm reduces system processing delay. Storm scheduling has not considered network bandwidth issues and the impact on data transmission hardware resources.

references:
[1] ANIELLO L, BALDONI R, QUERZONI L. Adaptive online scheduling in Storm[C]. Proceedings
of the 7th ACM International Conference on Distributed Event-Based Systems. New York: ACM, 2013: 207-218.

[2] Xiong Anping, Wang Xianwen, Zou Yang. Hot-Edge Scheduling Algorithm Based on Storm Topology [J]. Computer Engineering, 2017, 43 (1): 37-42.

[3] Lu Liang, Yu Jiong, Yun Chen. Weight-based task scheduling algorithm in Storm environment [J]. Journal of Computer Applications, 2018, 38 (3): 699-706.

[4] Liu Su, Yu Jiong, Lu Liang, Li Ziyang. Topology-based task scheduling strategy in Storm environment [J]. Journal of Computer Applications, 2018, 38 (12): 3481-3489.

[5] Wang Lin, Wang Jing. Storm task scheduling optimization based on improved ant colony algorithm [J]. Computer Measurement and Control, 2019, 27 (8): 236-240.

[6] Boyd S, Vandenberghe L. Convex Optimization[M]. Cambridge, UK: Cambridge University Press, 2004.