Performance Evaluation and Analysis of Multiple Scenarios of Big Data Stream Computing on Storm Platform

Dawei Sun[1], Hongbin Yan[1], Shang Gao[2] and Zhangbing Zhou[1]
1 School of Information Engineering, China University of Geosciences
Beijing, 100083, P.R. China
[e-mail: sundaweicn@cugb.edu.cn, yanhongbin@cugb.edu.cn, zhangbing.zhou@gmail.com]
2 School of Information Technology, Deakin University
Victoria 3216, Australia
[e-mail: shang.gao@deakin.edu.au]
*Corresponding author: Dawei Sun

Received October 25, 2017; accepted January 22, 2018; published July 31, 2018

Abstract

In big data era, fresh data grows rapidly every day. More than 30,000 gigabytes of data are created every second and the rate is accelerating. Many organizations rely heavily on real time streaming, while big data stream computing helps them spot opportunities and risks from real time big data. Storm, one of the most common online stream computing platforms, has been used for big data stream computing, with response time ranging from milliseconds to sub-seconds. The performance of Storm plays a crucial role in different application scenarios, however, few studies were conducted to evaluate the performance of Storm.

In this paper, we investigate the performance of Storm under different application scenarios. Our experimental results show that throughput and latency of Storm are greatly affected by the number of instances of each vertex in task topology, and the number of available resources in data center. The fault-tolerant mechanism of Storm works well in most big data stream computing environments. As a result, it is suggested that a dynamic topology, an elastic scheduling framework, and a memory based fault-tolerant mechanism are necessary for providing high throughput and low latency services on Storm platform.

Keywords: multiple scenarios, high throughput, low latency, stream computing, big data, Storm

This work is supported by the National Natural Science Foundation of China under Grant No.61602428, 61602106, and 61772479; and the Fundamental Research Funds for the Central Universities under Grant No. 2652015338.
We are grateful to thank the help of Mr Zhenhua Wang for his help about the experimentation implement and suggestions for improvement.
This paper is a substantially extended version of [10] presented at CollaborateCom 2017.

http://doi.org/10.3837/tiis.2018.07.002 **ISSN : 1976-7277**

1. Introduction

Big data is a term for datasets that are too large, too fast, too dispersed, and too unstructured. It is challenging for current hardware and software facilities to undertake their acquisition, access, analysis and/or application in reasonable amounts of time and space. Some popular features of big data are described by nVs, high Volume, high Velocity, high Variety, high Veracity, high Validity, high Value, and so on [1] [2]. There are many potential and highly useful values hidden in big data, and data has already drawn huge attention from researchers in sciences, and policy and decision makers in governments and enterprises. The rise of big data presents many opportunities and challenges [3] [4].

In big data era, more and more application rely heavily on real time processing of high volume, continuous data stream, such as social networks, telecommunications, emergency response, fraud detection, system monitoring, smart cities, and to name but a few. In a real time computing environment, data stream must be immediately processed to get prompt feedback. A big data stream computing system can be employed to process heterogeneous, real time, fluctuate over time, unbounded data stream in a distributed and scalable computing manner.

Storm [5] is one of the most popular open sourced big data stream computing systems, and has been widely used in many well-known companies and organizations [6] [7], such as Twitter, Alibaba, etc. Storm provides an on-the-fly computing paradigm, where the data is directly processed by running task topology in memory, without the need for storage on disk first. It keeps response time ranging from milliseconds to sub-seconds, and overcomes long response time problem (ranging from minutes to weeks) faced by big data batch computing systems, such as Hadoop, which provides a store-then-process computing paradigm [8] [9].

The performance of Storm plays a crucial role in many different application scenarios, however few studies were conducted to evaluate the corresponding performance of Storm [10] [11] [12]. We have been developing real-time big data processing applications on Storm for years, and deeply understand the importance of improving the processing efficiency. After investigating all kinds of features and mechanisms of Storm, we identify the key factors which mostly affect system throughput capacity and latency.

In this paper, we analyse the performance of Storm under different application scenarios. Our experimental results show that throughput and latency of Storm are greatly affected by the number of instances of each vertex in task topology, and the number of available resources in data center. The fault-tolerant mechanism of Storm works well in most big data stream computing environments. As a result, it is suggested that a dynamical topology, an elastic scheduling framework, and a memory based fault-tolerant mechanism are necessary for providing high throughput and low latency services on Storm platform.

1.1 Paper organization

The rest of this paper is organized as follows: In section 2, the background on computing paradigm of big data stream computing and description of application scenario on Storm platform are reviewed. Section 3 introduces the Storm architecture, Storm characteristic, and work mechanism of Storm. Section 4 focuses on the experimental environment and parameter settings. Section 5 provides performance evaluation and result analysis. Finally, conclusions and future work are given in section 6.

2. Background

In this section, background of big data stream computing is presented computing paradigm of big data stream computing, and description of application scenario on Storm platform, some releted work of data management in big data platform is also introduced.

2.1 Computing paradigm

In big data stream computing environments, stream computing is the model of straight through computing. It continuously integrates and computes data in motion to deliver real-time analytics, and enables users to detect in-sights in high velocity data stream which can only be detected and acted on at a moment's notice [13].

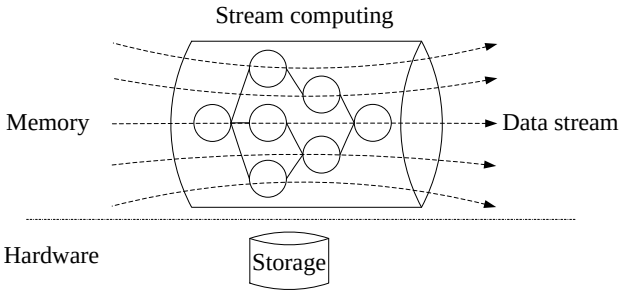

Fig. 1. Big data stream computing paradigm [13]

As shown in **Fig. 1**, an application is described by a directed graph. The continuous input data streams are computed in real time. The results are also updated in real time.

2.2 Description of application

On Storm platform, an application is described by a DAG (Directed Acyclic Graph), called a topology. There are two types of vertices in a DAG: spout and bolt. Spout vertex is a vertex of data sources, which sends data tuples (a data tuple is a key-value pair) to bolt vertex continuously, and a bolt vertex is a vertex to process data tuples in the way implemented by users. Both the instance number of spout and bolt vertex can be set by user to increase parallelism.

The DAG of an application can be further divided into two types: logic graph in function, and instance graph in runtime. As shown in **Fig. 2**, the Storm source code is the main part to achieve TOP_N computing function, which is provided by the user.

```
builder.setSpout("a", new TestWordSpout(), 2);
builder.setBolt("b", new RollingCountBolt(9, 3), 4).
    fieldsGrouping("a", new Fields("word"));
builder.setBolt("c", new IntermediateRankingsBolt(TOP_N), 3).
    fieldsGrouping("b", new Fields("obj"));
builder.setBolt("d", new TotalRankingsBolt(TOP_N), 1).
    globalGrouping("c");
```

Fig. 2. Source code of TOP_N in Storm

The corresponding logic graph is a linear pipeline (As shown in **Fig. 3**), where each vertex only has one upstream and downstream vertex, with *f* represents fieldsGrouping strategy of

data stream that transfers from upstream vertex to downstream vertex, and g represents globalGrouping strategy [5].

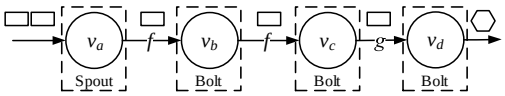

Fig. 3. Logical graph of TOP_N in Storm

The corresponding instance graph is a precedence constraint based directed acyclic graph (see **Fig. 4**), where vertex v_a is mapped into two parallel instances, vertex v_b is mapped into four parallel instances, and vertex v_c is mapped into three parallel instances. In Storm, the number of instances of each vertex is statically defined by user.

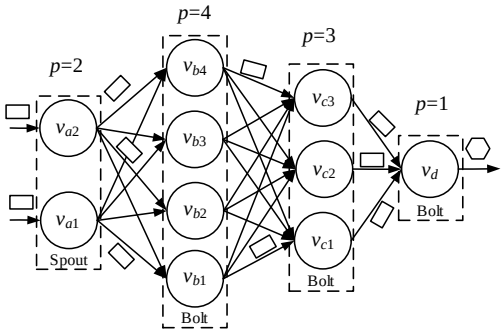

Fig. 4. Instance graph of TOP_N in Storm

2.3 Data management in big data platform

Resource management [14]-[17] is the key part in big data platform. Some orther work also focus on optimize resource management by smart algorithm, such as in [18]-[22]. Many work of data management have been done in big data platform. In this paper ,we focus on Storm platfrom, and try to give a comprehensive perfromance evaluation and analysis of multiple scenarios on Storm platform.

3. Storm Overview

In order to provide a bird's-eye view of the Storm platform, in this section, we discuss the overview of Storm, which includes Storm architecture, characteristics, and work mechanism.

3.1 Storm Architecture

Apache Storm is a free and open sourced distributed realtime computation system. The design of Storm makes it easy to process massive streams of data in real time and work with any programming language. Storm is mainly used for data stream processing and real-time search [5] [23] [24].

The architecture of Storm is shown in **Fig. 5**. The Storm's work is done by various types of components, with each component responsible for a simple task. Nimbus is the master node in a Storm cluster, responsible for sending code in the cluster, assigning tasks, and monitoring the entire cluster's state. Supervisor is the work node in a Storm cluster, responsible for accepting nimbus assigned tasks, starting and stopping the management of its own worker processes.Worker is a process that runs specific processing component logic. Executor is a concrete physical thread in one Worker process.Task is the work that each component does.

Zookeeper is an external resource that Storm relies heavily on, connecting the master node and the work node, coordinating the operation of the entire cluster.

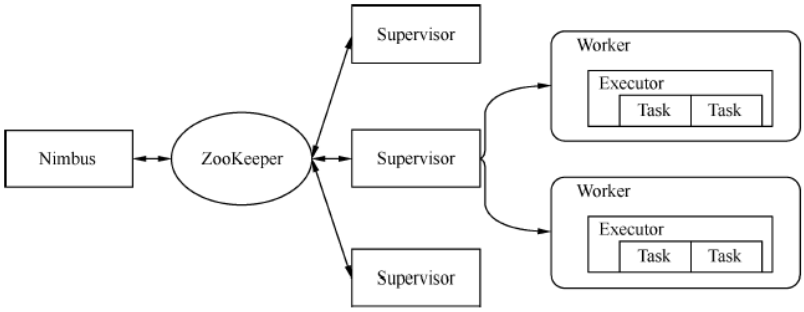

Fig. 5. Architecture of Storm

3.2 Storm Characteristics

Storm has the following characteristics [25] [26] [27]:

- Real-time: The Storm system is designed to ensure that messages can be processed quickly.
- High Fault Tolerance: Storm clusters are easy to manage, which automatically process processes and network anomalies. Moreover, Storm can guarantee that a processing logic runs forever, and if a process/task fails during processing, Storm will rearrange a new processing logic.
- Strong Stretch ability: To support the expansion of computing tasks, you only need to add a new machine to the cluster, then increase the degree of parallelism. The number of messages processed per second can even reach up to1 million.
- Language unrelated: The core part of Storm use Clojure language, however, its processing logic and message processing components can be defined by any language. For instance, the general utility is developed with python, and task topology is prepared with java.
- Wide application scenarios: Storm can be used to process messages and update databases (message flow processing), making continuous queries on high data volume and returning the results to clients (continuous calculation), and parallel processing for resource-intensive queries (distributed method calls).

3.2 Work Mechanism of Storm

When a client submits a task topology to master node Nimbus, Nimbus first establishes a local directory based on the configuration information of the topology, instructing zookeeper to assign the tasks to each work node, then starts the topology. Supervisors get assigned tasks from Zookeeper and start multiple worker processes, and establish the connection between tasks according to the configuration information of the topology [28] [29]. When the topology is running, the Storm system provides a UI monitoring interface in master node, through which the client can monitor the running status of the entire cluster in real time. The whole process is shown in the **Fig. 6**.

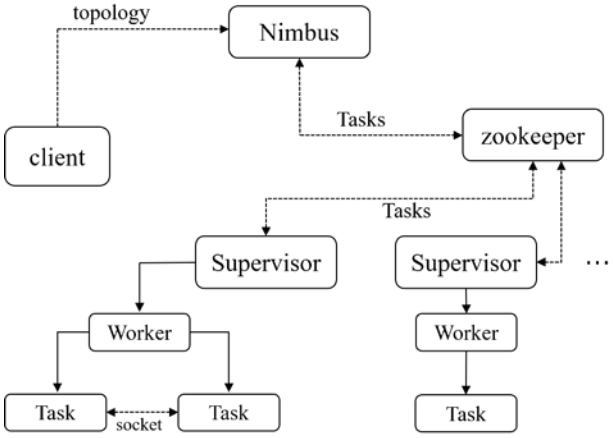

Fig. 6. Workflow of a Storm cluster

The configuration information of the topology and grouping methods might affect the system throughput and latency. We thoroughly examine the Storm work mechanisms, design and conduct series of experiments to investigate the potential factors that might affect the system processing efficiency.

4. Experiment and Parameter Setup

To evaluate the performance of the multiple scenarios of big data stream computing on Storm platfrom, experimental environment and parameter settings are discussed in this section.

4.1 Experimental Environment

In the series of experiments, Storm parallelism and fault tolerance are tested using the following hardware configuration: intel Core i5-2400CPU @ 3.10GHz × 4, memory 4G, with operating system 64-bit ubuntu 16.04 LTS. Twelve (12) machines are used to test parallelism, and ten (10) machines are used to test fault tolerance.

For hardware performance test, the following three groups of hardware configurations are used: the first group includes a Dell desktop computer, with processor intel Core i5-2400CPU @ 3.10GHz × 4, memory 4G, and 64-bit operating system ubuntu 16.04 LTS; the second group includes a DELL laptop, with processor Intel (R) Core (TM) i5-2430M CPU @ 2.4GHz × 4, memory 8G, and 64-bit ubuntu 16.04 LTS; the third group includes a HP laptop, with processor Intel (R) Core (TM) i7-7700HQ CPU @ 2.80GHz 2.81GHz, 8G memory, and 64-bit ubuntu 16.04 LTS. In terms of software versions, the tests use the Storm version 1.0.1, Zookeeper version of 3.3.6, in addition to JDK 1.8 and Python 2.7.2.

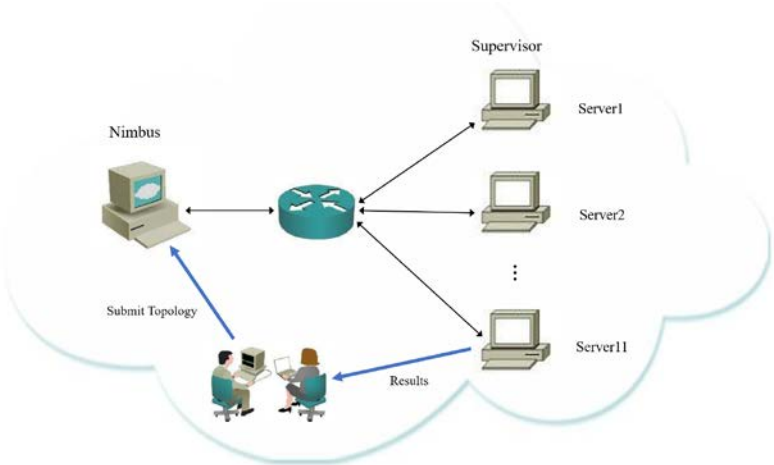

Fig. 7. Experiment network topology

4.2 Experimental Topology

The experimental network topology is shown in **Fig. 7**. The experiment uses two task topologies separately.

The first task topology, as shown in **Fig. 8**, is a typical word count program, which includes a Spout component (named spout) and two Bolt components (named split and count). Spout components are used to randomly launch English sentences. The split component receives the sentences sent from the spout component and divides it into words, and finally sends it to the count component for word counting.

The second task topology is TOP_N, as shown in **Fig. 9**, which contains a Spout component (named wordGenerator) and three Bolt components (named counter, intermediateRanker and finalRanker). Among them, wordGenerator is responsible for pushing all the words, the same topic pushed to the same counter (using Storm's fieldsGrouping to achieve). Counter receives the topic and saves the number of occurrences of the topic. For every one minute, the counter will push the number of occurrences of each topic to the intermediate processing node intermediateRanker (also using Storm's fieldsGrouping). The IntermediateRanker component saves a TOP_N list, receives the message sent and refreshes TOP_N according to the number of topics published to sort word; it also pushes the results to the final node finalRanker every two seconds. Finally, the finalRanker component summarizes each TOP_N received within two seconds and selects the final TOP_N.

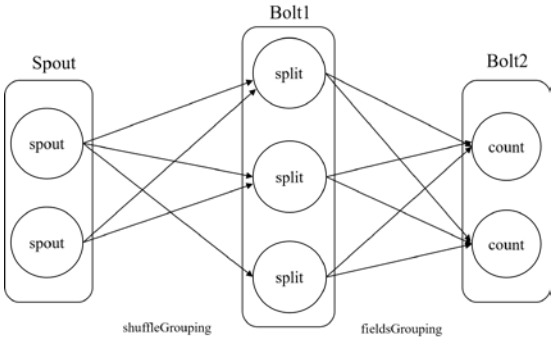

Fig. 8. Structure of the Wordcount topology

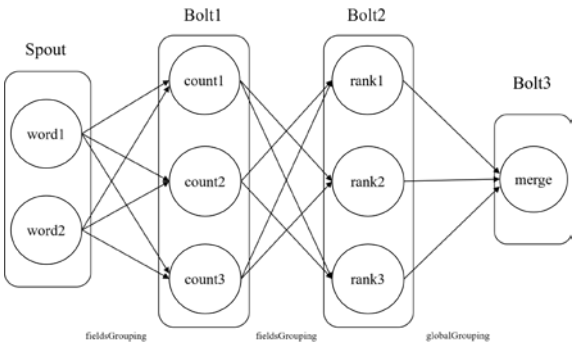

Fig. 9. Structure of the Top_N topology

4.3 Experimental Process

(1) Parallelism Test

In Storm, the degree of parallelism is generally addressed in three areas: topology which specifies how many Worker processes run in parallel; Worker process which specifies how many Executor threads run in parallel; and Executor thread which specifies how many Tasks run in parallel.

In parallelism tests, the first two areas are considered, with each Executor assigned with one Task by default. In order to eliminate the impact of hardware performance on the latency and throughput, 12 machines use the same hardware configuration. Since the default scheduling policy of Storm system is even distribution of Executors, only one Worker is running on each node. When the number of Workers in the topology is set to 24, each node runs two Workers. The testing time is one hour and the data is counted every two minutes.

Multiple scenarios are designed as follows:

Firstly, the number of threads assigned to the three components spout, split, and count is (5, 8, 12) and keeps unchanged in the WordCount topology. The number of threads assigned to the four components wordGenerator, counter, intermediateRanker and finalRanker is (5,4,1,4) in the Top_N topology. The number of Workers is set to 1, 3, 6, 12, and 24 respectively.

Secondly, for a given number of Workers in each topology, the number of threads increases in turn. In the WordCount Topology, when the first test sets the number of Workers to 3, the numbers of threads are (5,8,12), (10,16,24), (15,24,36), (20,34,48), (25,40,60), (50,80,120), (100,160,240), (150,240,360), (200,320,480), (250,400,600). When the second test sets the number of Workers to 12, the number of threads remains the same as the first. In the TOP_N Topology, the first test sets the number of Workers to 3 and the numbers of threads are (5,4,1,4), (10,8,2,8), (15,12,3,12), (20,16,4,16), (25,20,5,20), (50,40,10,40), (100,80,20,80), (150,120,30,120), (200,160,40,160), (250,200,50,200), respectively. The second test sets the number of Workers to 12, and the number of threads remains the same as the first.

Two sets of experiments test twenty sets of data for each topology and we then analyze the effect of changes made to the number of threads assigned to the three/four components on the throughput of the Storm system and the processing delay for a given a certain number of Workers.

Finally, the number of processes set in the topologies remains the same, the total number of threads of all components keeps unchanged, while the number of threads of each component is adjusted. When the number of Workers is set to 3, and the number of threads in the three components is (5, 8, 12), (5,10,10), (5,12,8), (3,10,12) (7,10,8), (7,8,10), (9,7,9), (12,5,8) in the WordCount topology. The number of threads in the four components is (2,5,2,5), (5,4,1,4), (5,2,1,6), (5,3,3,3) (5,6,1,2), (7,3,1,3), (9,2,1,2), respectively, in Top_N topology. The impact

of data transmission and data processing on system throughput and latency is analyzed by testing the different thread assignments of the components in the two topologies.

(2) Hardware Performance Test

In order to analyze the impact of hardware performance on Storm system operation, two scenario are designed as follows:

- the number of Workers set by the topologies and the number of threads for each component are unchanged;
- the number of Workers set by the topologies is 3 and the threads assigned to each component is unchanged.

The configuration of the three machines is shown in **Table 1** and these three machines used in both of the topologies.

Table 1. Hardware Configuration

PC	CPU	Memory	Operating System
DELL Desktop	Intel Core i5-2400CPU @ 3.10GHz × 4	4G	ubuntu 16.04 LTS
DELL Notebook	Intel (R) Core (TM) i5-2430M CPU @ 2.4GHz × 4	8G	ubuntu 16.04 LTS
HP Notebook	Intel (R) Core (TM) i7-7700HQ CPU @ 2.80GHz 2.81GHz	8G	ubuntu 16.04 LTS

(3) Robustness Test

We test Storm on fault handling and analyze its impact on throughput and latency. Three scenarios are designed. In the first scenario, we shut down one node when the cluster is running up to 20 minutes. In the second scenario, we shut down two of the nodes when the cluster is running up to 20 minutes. In the third scenario, we shut down three of the nodes when the cluster is running up to 20 minutes and shut down three of the nodes again when the cluster is running up to 40 minutes.

After the three tests, we compare the performance with that of normal scenario.

5. Performance Evaluation and Analysis

In this section, performance evaluation results are firstly discussed, followed by result analysis.

5.1 Experimental Results

(1) Parallelism Test

To test parallelism, we increase the number of Workers while keeping the number of threads constant in the topologies. In terms of throughput capacity, when the number of Workers in the topology is set to 1, 3, 6, 12, and 24 respectively, in the WordCount topology , the amount of data processed in an hour is about 13,10000 Tuples and throughput is about 366 Tuple/s as shown in **Fig. 10**; in the Top_N topology , the amount of data processed in an hour is about 2,00000 Tuples and throughput is about 55 Tuple/s as shown in **Fig. 11**. The throughput of the system is essentially unchanged. It can be seen, in the case of the same number of threads, increasing the number of Workers does not improve the Storm system throughput capacity significantly.

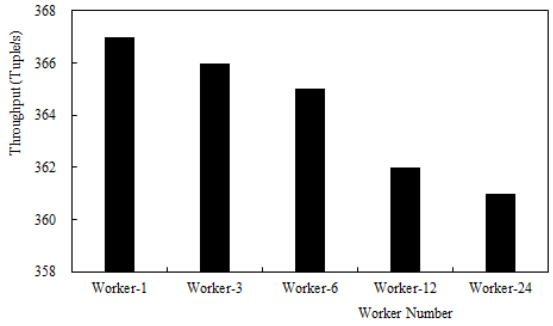

Fig. 10. Throughput of different Workers number in WordCount

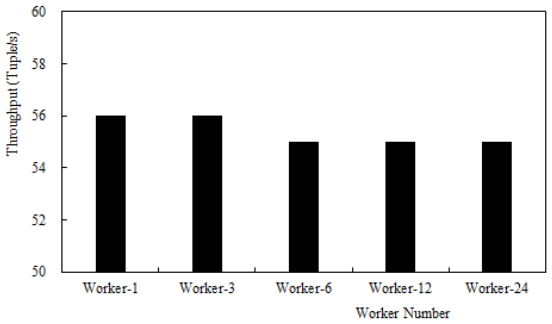

Fig. 11. Throughput of different Workers number in Top_N

In terms of processing latency, when the number of threads assigned to each component keeps constant and the system becomes stable, as the number of Workers increases, the processing latency does not change much. The delay is relatively large when the system starts to run, and then gradually reduced. In the WordCount topology, as shown in **Fig. 12**, when the Storm cluster is running steadily, the processing latency basically remains between 3 and 4 milliseconds and the average latency in one hour is 3.771, 3.426, 3.587, 3.527, 3.386 milliseconds respectively and about 95% of the delay comes from the process latency of the Bolt component that named split. When the number of Workers is set to 1, the delay in the one-hour test is very unstable and may be caused by the operation of other system programs during the test. In the Top_N topology, as shown in **Fig. 13**, the processing latency basically remains between 0.5 and 1 milliseconds and the average latency in one hour is 0.596, 0.625, 0.730, 0.949, 0.881 milliseconds respectively.

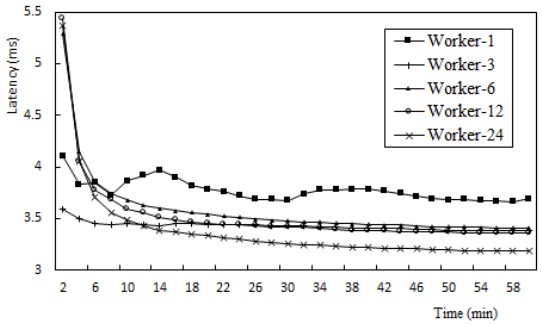

Fig. 12. Latency of different Workers number in WordCount

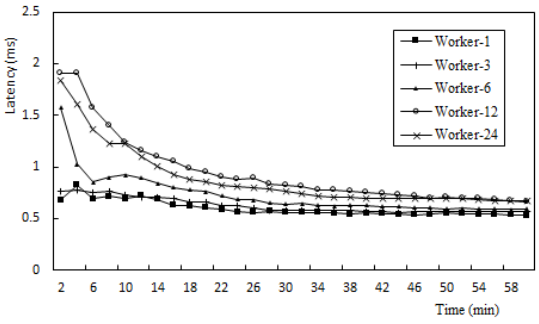

Fig. 13. Latency of different Workers number in Top_N

In terms of CPU utilization, during the one-hour test, the master node constantly assigns tasks to the Zookeeper, and the work node Supervisor also constantly receives the tasks from the Zookeeper, leading the CPU usage is very unstable in the two topologies: as the Worker number increases, the CPU usage decreases a little as shown in **Fig. 14** and **Fig. 15**; but when the number of Workers increases to 24, since there are only 12 work nodes, 24 Workers are evenly distributed to each node, and each work node runs two Workers, so that CPU utilization increases slightly by 12. As shown in **Fig. 16** and **Fig. 17**, the memory usage decreases a little when the number of threads allocated to each component is unchanged and the number of Workers is simply increased. Similarly, when the number of Workers is 24, since each node runs two Workers, the memory usage of the node increases. In the Storm system, usually a

Worker allocates 768M of memory and adds 64M to the logwriter process, so a Worker will consume about 832M of memory. But the Worker's memory space also depends on the amount of data flowing through the topology and the execution time of the code for each bolt unit. If the amount of data is large and the code execution time is long, we can increase the working memory of a single Worker.

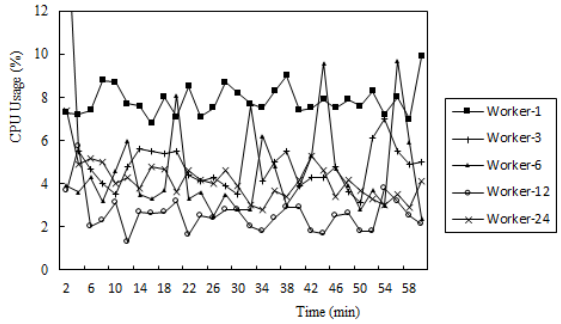

Fig. 14. CPU usage of different Workers number in WoedCount

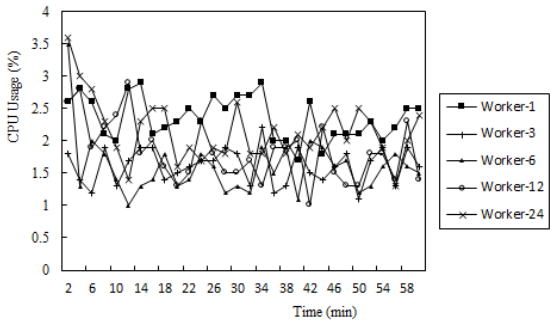

Fig. 15. CPU usage of different Workers number in Top_N

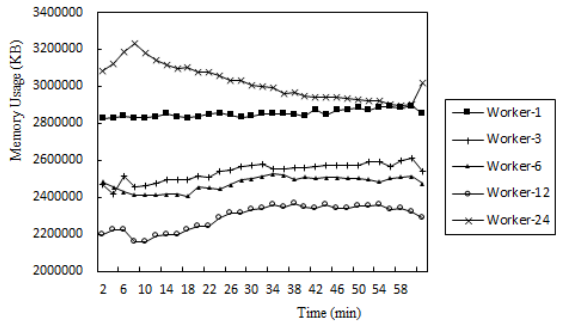

Fig. 16. Memory usage of different Workers number in WordCount

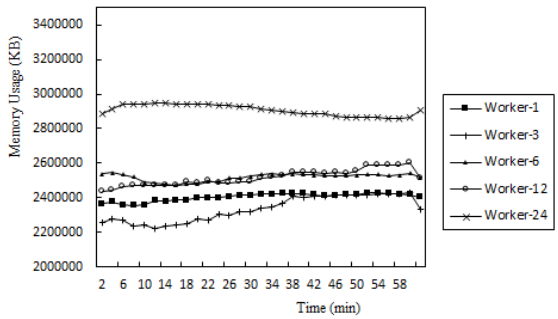

Fig. 17. Memory usage of different Workers number in Top_N

In the WordCount topology, when the number of Workers in the topology is 3, as the number of threads assigned to each component increases, the amount of data processed by the system and the average throughput over an hour also increase. As shown in **Fig. 18**, the average throughput within one hour is 366, 733, 1099, 1463, 1831, 3663, 7315, 10954, 14494 and 15828 Tuple/s respectively. If the number of Workers is set to 12, when the number of threads assigned to each component increases, the throughput also increases in a positive correlation and the average throughput within one hour is 362, 723, 1082, 1443, 1805, 3614, 7232, 10823, 14437 and 18025 Tuple/s respectively. The throughput is almost the same when Worker is set to 3; but when the thread is assigned to (250,400,600), the average throughput increases more than about 2,000 Tuple/s compared to it. In the Top_N topology, as shown in **Fig. 19**, when the Worker set to 3,the average throughtput is 56, 109, 163, 217, 272, 541, 1081, 1617, 2157, 2695 Tuples/s; when Worker set to 12, the average throughtput is 55, 108, 162, 215, 267, 531, 1065, 1600, 2127, 2659 Tuple/s respectively, also increases as the number of threads increases.

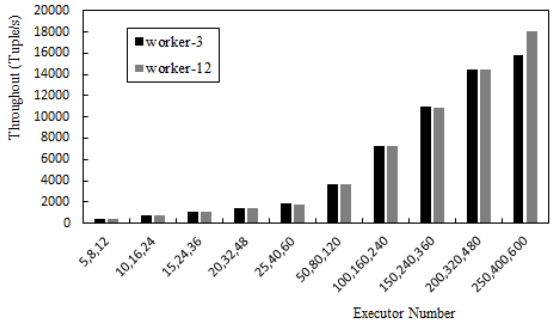

Fig. 18. Throughput of different Executor number in WordCount

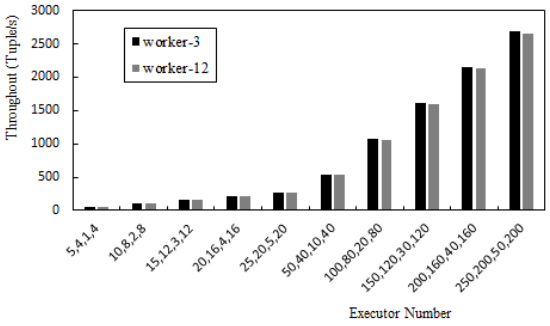

Fig. 19. Throughput of different Executor number in Top_N

In terms of system latency, as shown in **Fig. 20**, in the WordCount topology, if the number of Workers in the topology is 3, when the number of threads assigned to each component increases, the processing delay of the system decreases first but then increases. When the number of threads assigned to each component is (50,80,120), the processing delay of the system is minimized and then gradually increases. When the thread configuration exceeds (150,240,360), the delay grows faster and exceeded 20 milliseconds when the thread is configured to (250,400,600). The different result applies when the number of Workers is set to 12, when the number of threads assigned to the three components is less than (25, 40, 60), the average processing latency of the system within one hour is substantially the same, probably about 4 milliseconds. Between (25, 40, 60) and (150,240,360), the latency will gradually decrease to about 2ms, and when the thread configuration exceeds (150,240,360), the system average processing delay will slowly rise to 3 ms. Consider the reasons for this situation, when the number of threads allocated to three components increases, the CPU load is constantly increasing, although the total data processing capacity has greatly improved, a single data processing delay is increasing, processing efficiency is declining. In the Top_N topology, as shown in **Fig. 21**, with the number of threads increases, the system latency declines slightly. When the number of Workers set to 3, the processing delay of the system is minimized when the number of threads assigned to each component is (150,120,30,120) and then gradually increases. But when the number of Workers is set to 12, the data processing latency is declining continuously, because the fact that the minimum delay has exceeded (250, 200, 50, 200).

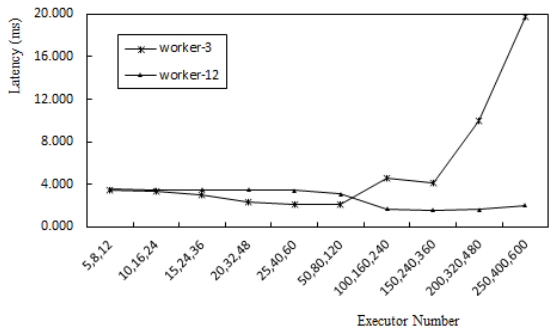

Fig. 20. Latency of different Executor number in WordCount

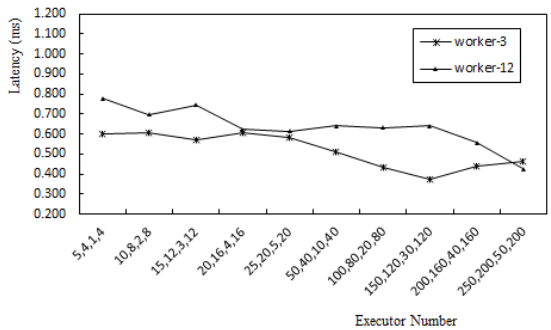

Fig. 21. Latency of different Executor number in Top_N

In terms of system utilization, as shown in **Fig. 22** and **Fig. 23**, with the increase of the number of threads, CPU usage gradually increases. But when the number of threads increases to a certain value, the CPU usage increases dramatically. In the WordCount topology, the threshold value is (25, 40, 60) when the number of Workers is 3; and the value is (100,160,240) when the number of Workers is 12. In the Top_N topology, the threshold value is (50,40,10,40) when the number of Workers is 3; and the value is (100,80,20,80) when the number of Workers is 12.

In terms of Memory usage, for the WordCount topology, as the number of threads increases, regardless of the number settings being the 3 or 12, the memory usage remains at around 2,500,000 as shown in **Fig. 24**. But when the number of Workers is 3, the memory usage will increase dramatically when the number of threads in the three components exceeds (200,320,480). In the Top_N topology, as shown in **Fig. 25**, with the increase of the number of threads, memory usage gradually increases. When the threads is (15,12,3,12), memory usage reaches a maximum value.

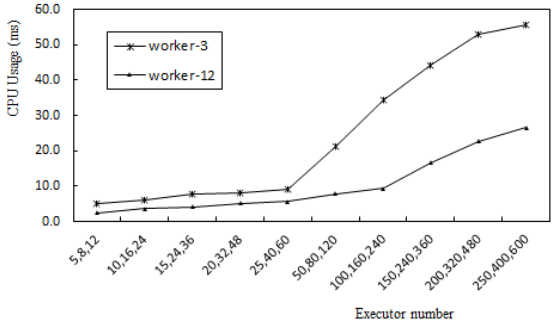

Fig. 22. CPU usage of different Executor number in WordCount

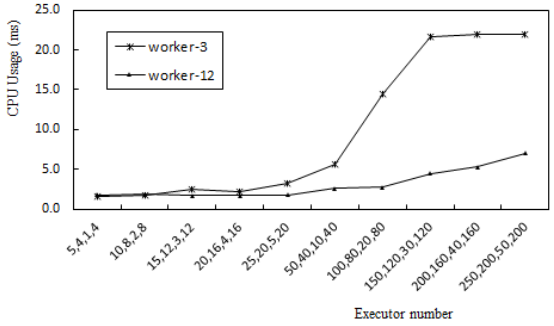

Fig. 23. CPU usage of different Executor number in Top_N

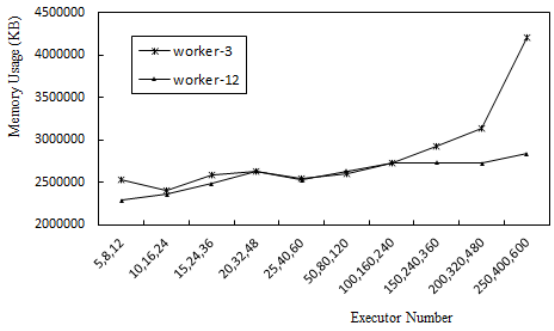

Fig. 24. Memory usage of different Executor number in WordCount

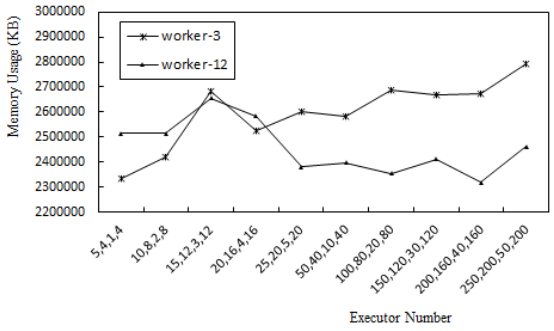

Fig. 25. Memory usage of different Executor number in Top_N

Given the number of Workers in topologies keeps the same and the total number of threads of all components unchanged, in the WordCount topology, if the number of threads in the three components is changed, the average throughput during the one-hour testing period is shown in **Fig. 26**. From the average throughput of the five configurations (5,8,12), (5,10,10), (5,12,8) and (7,10,8), (7,8,10), we find that when the number of threads assigned to the launching component stays constant, the average throughput in one hour remains the same regardless of how the two Bolt components are changed. When the number of threads assigned

to Spout component increases, the system throughput also increases; when the number of threads assigned to Spout component decreases, the system throughput also decreases. It is observed that the system throughput is only related to the number of threads assigned to the delivery component spout. In the Top_N topology, as shown in **Fig. 27**, similar to the WordCount, the system throughput is positively correlated with the number of threads that transmit components. When the number of launching component is unchanged, the average throughput is constant regardless of how the three Bolt components are changed.

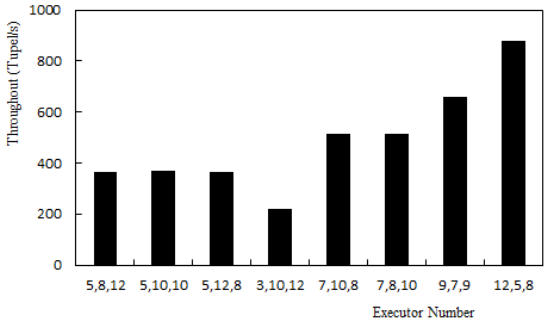

Fig. 26. Throughput of different thread assignments in WordCount

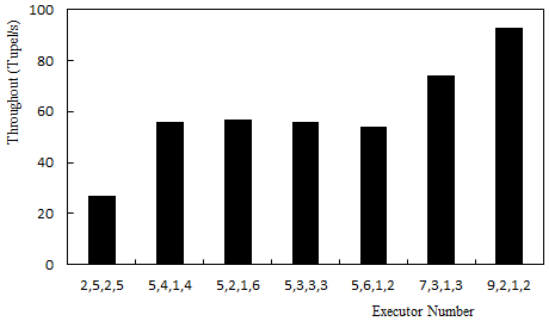

Fig. 27. Throughput of different thread assignments in Top_N

In terms of system latency, for the WordCount topology, when the number of threads assigned to the three components is (5,8,12), (5,12,8), (5,10,10), (9,7,9), (12,5,8) , the data processing delay is maintained at 3.5 milliseconds in one hour as shown in **Fig. 28**. But when the number of threads assigned to the three components is (7, 10, 8) and (3, 10, 12), the latency is very large when Storm cluster starts and then gradually reduces to 3.5 milliseconds when running to 40 minutes. Consider that some of the other system processes may consume the CPU at the beginning of the operation, resulting in lower data processing efficiency, and when these systems are finished, the data processing is back to normal. In the Top_N topology, as shown in **Fig. 29**, when the system is running steadily, the data processing delay is maintained at 0.8 milliseconds. Also, when the number of threads is (5,2,1,6) and (9,2,1,2), the latency is very large and then gradually reduced to normal, the reason is similar to WordCount. It can be concluded that when the number of Workers and the total number of threads are unchanged, no matter how the threads are allocated to the three components, the system latency is unchanged, although the delay is mainly caused by the Bolt component (named split), but adding threads to it does not decrease the delay at all.

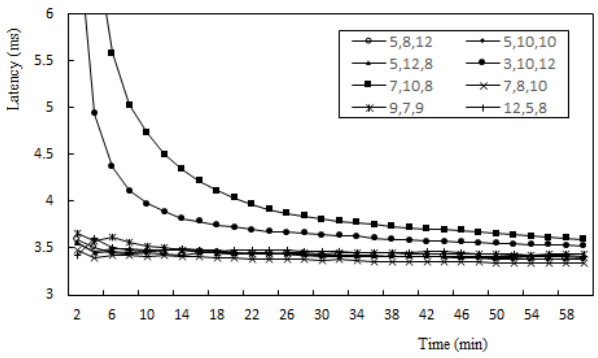

Fig. 28. Latency of different thread assignments in WordCount

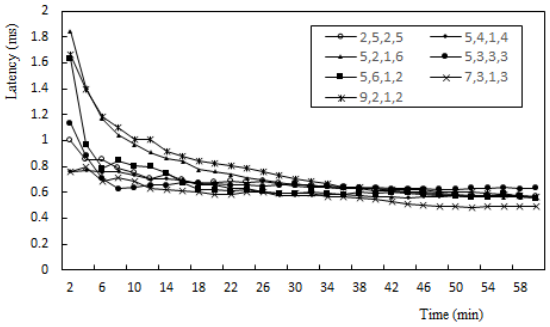

Fig. 29. Latency of different thread assignments in Top_N

In terms of system utilization, as shown in **Fig. 30** and **Fig. 31**, as the number of Workers and the total number of threads are unchanged, the average usage of the CPU during the one-hour testing period is virtually unchanged even if the thread assignment to the components is changed. When the Storm system runs the topology, the master node and the work node continuously allocate tasks and accept tasks, tracking and feedback operations on each Tuple, so CPU usage is unstable during topology processing. But the average utilization rate of CPUs is very small in one hour, which is 4.7%, 3.6%, 3.7%, 3.7%, 3.8%, 4.6%, 5.2%, 4.4% respectively in WordCount topology, basically maintains between 3% and 5%. And 1.5%, 1.6%, 1.5%, 1.5%,1.6%, 1.6% respectively in Top_N topology, maintains between 1% and 2%. As shown in **Fig. 32** and **Fig. 33**, because the size of the memory occupied by each Worker in the work node is fixed, the system's memory usage is essentially constant when the number of Workers settings in the topology is unchanged, even if the thread allocation for the three components is adjusted. There are several configurations of memory usage that are slightly increased during the experiment, possibly due to the impact of other system programs.

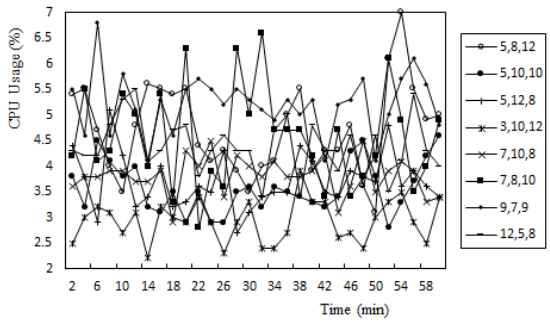

Fig. 30. CPU usage of different thread assignments in WordCount

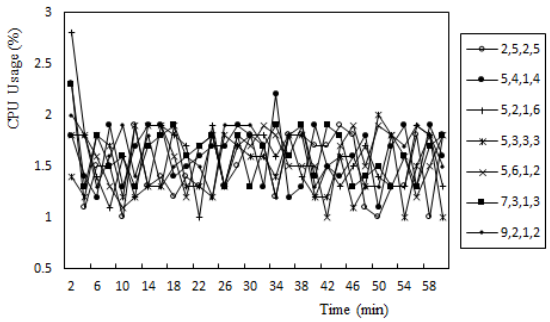

Fig. 31. CPU usage of different thread assignments in Top_N

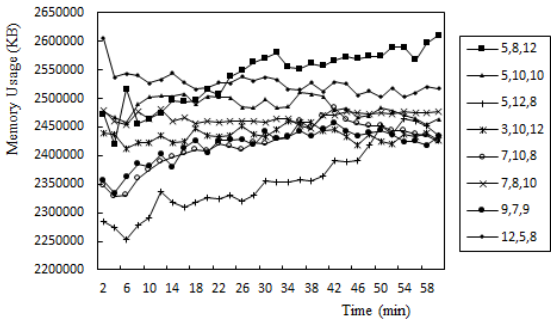

Fig. 32. Memory usage of different thread assignments in WordCount

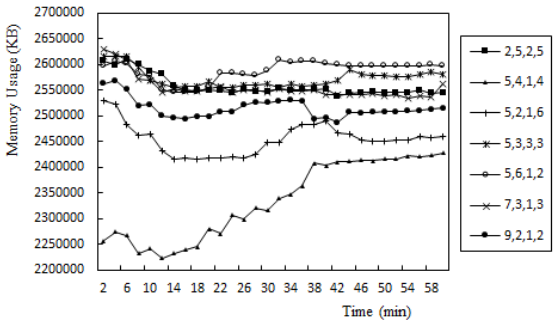

Fig. 33. Memory usage of different thread assignments in Top_N

(2) Hardware Performance Test

In the hardware performance test, as shown in **Fig. 34** and **Fig. 35**, when using three different performance computers to run the same topology, the amount of data processed within one hour is 1320700, 1314680, 1317100 respectively in the WordCount topology and the average throughput within one hour is 366, 365, 366 Tuple/s. In the Top_N topology, the amount of data processed is 200080, 198140, 198580 respectively and all average throughput

are 55 Tuple/s. Thus, when the amount of data processed by the system is not large, the hardware performance of each node will not affect the throughput of the Storm cluster.

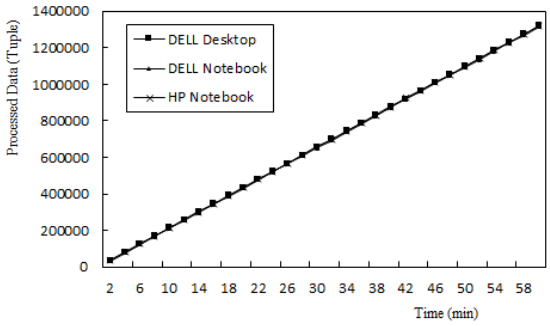

Fig. 34. Throughput of different hardware performance about WordCount

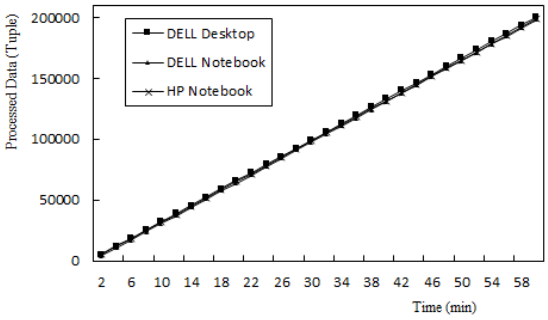

Fig. 35. Throughput of different hardware performance about Top_N

In terms of data processing latency, for the WordCount topology, as shown in **Fig. 36**, in an hour of testing, the data processing latency of DELL laptop (Intel (R) Core (TM) i5-2430M CPU @ 2.4GHz × 4, 8G) basically maintains between 7 and 8 milliseconds, HP notebook's (Intel (R) Core (TM) i7-7700HQ CPU @ 2.80GHz 2.81GHz, 8G) latency maintains at about 3.4 milliseconds, DELL desktop (Intel Core i5-2400CPU @ 3.10GHz × 4, 4G) maintains at 3.2 milliseconds. Also, in the Top_N topology, as shown in **Fig. 37**, the data processing latency of DELL laptop basically maintains at 2.75 milliseconds, HP notebook's latency maintains at about 0.68 milliseconds, DELL desktop maintains at 0.47 milliseconds. The average delay of DELL notebook with lowest CPU performance is much higher than the other two during the one-hour testing period. Although the memory of DELL desktop is only 4G, it's data processing delay is almost half of the DELL notebook and is lower than the HP notebook within an hour. So compared to memory, CPU performance has a greater impact on data processing latency.

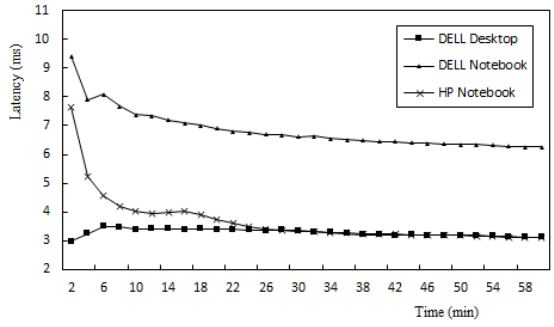

Fig. 36. Latency of different hardware performance about WordCount

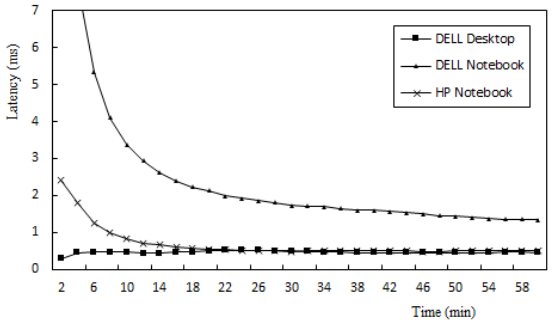

Fig. 37. Latency of different hardware performance about Top_N

(3) Robustness Test

As shown in **Fig. 38** and **Fig. 39**, in the first test, when one node is shut down after 20 minutes, the system reduces the speed of data processing, because the system has to redistribute the task from the down node. The data processing speed resumes to normal 2 minutes later. In the second test, we shut down two of nodes after 20 minutes, the outcome observed is almost the same as the first. The data processing rate becomes smaller in two

minutes, but lower than the first failure, because when two nodes do not perform the task, the master node needs to spend more system resources for task redistribution, so the data processing rate is lower than the one node failure. In the third test, when three of nodes are shut down after 20 minutes, data processing speed is declining, after two minutes of task redistribution, the speed of system processing data returns to normal. When running to 40 minutes, we shut down three nodes agin. Because each node in the Storm cluster has no extra memory, the system can not achieve the task redistribution, resulting in Storm system failure to properly process the data.

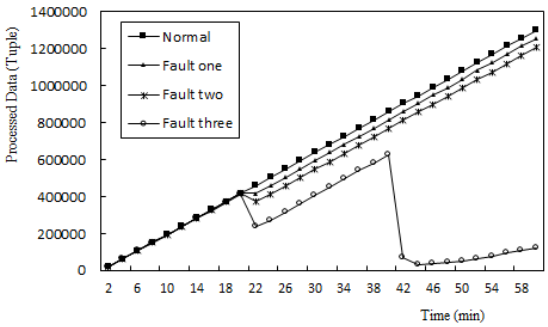

Fig. 38. Throught of robustness test about WordCount

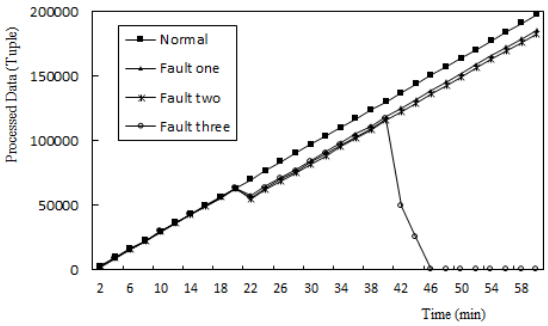

Fig. 39. Throught of robustness test about Top_N

In terms of system latency, as shown in **Fig. 40** and **Fig. 41**. At the beginning of the tests, closing one or two nodes, does not cause a sudden change to the system's calculation delay in the two minutes of task redistribution, but the delay gradually reduces as normal within one hour of test. When the system is running stable, they are basically maintained at about 3 milliseconds in WordCount topology and 1 milliseconds in Top_N topology. But in the third test, when the three nodes are closed for 40 minutes, data processing delays are also abnormal. The Storm system has a strong fault tolerance, when some nodes fail, and the redistribution of tasks does not affect the system's calculation delay. But each node should leave extra memory space when the system is running, so that the system can have enough space to reallocate the task when some nodes behave abnormal.

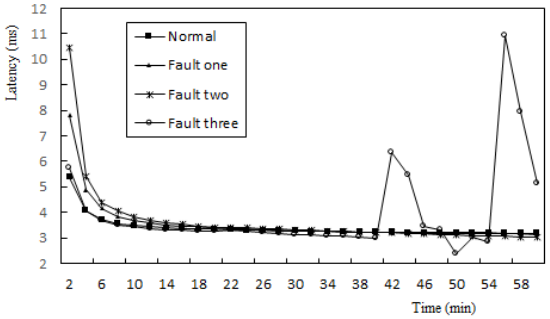

Fig. 40. Latency of robustness test about WordCount

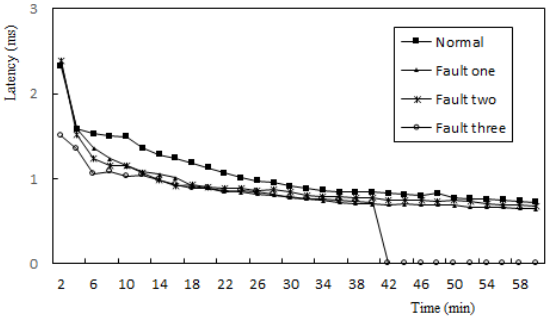

Fig. 41. Latency of robustness test about Top_N

5.2 Result Analysis

After analyzing the test results, we can conclude that:

- Increasing the number of Workers in a topology alone does not improve system throughput. The throughput of the system has positive correlation to the number of

threads assigned to each component and mainly related to the number of threads of components that emit data.

- The latency in data processing is related to the number of Workers and the number of threads that assigned to each component. However, the impact is limited. Larger numbers of Worker and threads do not lead to a significant low latency. Therefore, a reasonable value is to be set according to different computing tasks. The processing latency is also affected by hardware performance and mainly related to the CPU performance.
- Storm system handles faulty nodes quickly. Task redistribution has little impact on the data processing speed and does not increase the data processing latency. But each node should leave extra memory so that the system have enough space to reallocate the task when some nodes behave abnormal.

6. Conclusion

Big data stream computing helps organizations spot opportunities and risks from real time big data, and is being employed in many different application scenarios. Storm, one of the most common online stream computing platforms, is now used for computing big data stream, with response time ranging from milliseconds to sub-seconds. The performance of Storm plays a crucial role in many different application scenarios, however few studies were conducted to evaluate the performance of Storm.

In this paper, we investigate the performance of Storm under different application scenarios. Our experimental results show that throughput and latency of Storm is greatly affect by the number of instances of each vertex in task topology, and the number of available resources of data center.

The fault-tolerant mechanism of Storm works well in most big data stream computing environments. As a result, it is suggested that a dynamical task topology, an elastic scheduling framework, and a memory based fault-tolerant mechanism are necessory for providing high throughput and low latency services.

In future work, we are interested in investigating the impact on Storm performance caused by the topology and algorithm complexity. Moreover, we plan to examine the behavior of Storm on receiving data from the cloud and its impact on performance.

References

[1] C. L. P. Chen, C. Y. Zhang, "Data-intensive applications, challenges, techniques and technologies: A survey on Big Data," *Information Sciences*, vol. 275, pp. 314-347, Aug. 2014. Article (CrossRef Link)

[2] D. W. Sun and H. Tang, "Fast-FFA: a fast online scheduling approach for big data stream computing with future features-aware," *International Journal of Bio-Inspired Computation*, vol. 10(3), pp. 205-217, Sep. 2017. Article (CrossRef Link)

[3] A. Gani, A. Siddiqa, S. Shamshirband, F. Hanum, "A survey on indexing techniques for big data: taxonomy and performance evaluation," *Knowledge and Information Systems*, vol. 46(2, pp. 241-284), Feb. 2016. Article (CrossRef Link)

[4] M. D. Assuncao, R. N. Calheiros, S. Bianchi, M. A. S. Netto, R. Buyya, "Big Data computing and clouds: Trends and future directions," *Journal of Parallel and Distributed Computing*, vol. 79-80, pp. 3-15, May 2015. Article (CrossRef Link)

[5] 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 *Proc. of 2014 ACM*

SIGMOD International Conference on Management of Data, SIGMOD 2014, ACM Press, pp. 147-156, Jun. 2014. Article (CrossRef Link)
[6] C. Li, J. Zhang, Y. Luo, "Real-time scheduling based on optimized topology and communication traffic in distributed real-time computation platform of storm," *Journal of Network and Computer Applications*, vol. 87, pp. 100-115, Jun. 2017. Article (CrossRef Link)
[7] T. Li, J. Tang, J. Xu, "Performance modeling and predictive scheduling for distributed stream data processing," *IEEE Transactions on Big Data*, vol. 2(4), pp. 353-364, Dec. 2016. Article (CrossRef Link)
[8] Q. Cai, L. Ma, M. Gong and D. Tian, "A survey on network community detection based on evolutionary computation," *International Journal of Bio-Inspired Computation*, vol. 8(2), pp. 84-98, May 2016. Article (CrossRef Link)
[9] P. Novoa-Hernández, C. C. Corona and D. A. Pelta, "Self-adaptation in dynamic environments - a survey and open issues," *International Journal of Bio-Inspired Computation*, vol. 8(1), pp. 1-13, Feb. 2016. Article (CrossRef Link)
[10] H. B. Yan, D. W. Sun, S. Gao, and Z. B. Zhou, "Performance Analysis of Storm in a Real-World Big Data Stream Computing Environment," in *Proc. of 13th EAI International Conference on Collaborative Computing: Networking, Applications and Worksharing, CollaborateCom 2017*, Springer Press, in press, December 2017.
[11] A. Mozaffari, N. L. Azad, "Empirical investigation and analysis of the computational potentials of bio-inspired nonlinear model predictive controllers: success and challenges," *International Journal of Bio-Inspired Computation*, 9(1): 19-34, 2017. Article (CrossRef Link)
[12] A. Ouannas, A. T. Azar, S. Vaidyanathan, "On a simple approach for Q-S synchronisation of chaotic dynamical systems in continuous-time," *International Journal of Computing Science and Mathematics*, 8(1):20-27, 2017. Article (CrossRef Link)
[13] M. Behroozifar, "Computational method for one-dimensional heat equation subject to non-local conditions," *International Journal of Computing Science and Mathematics*, 8(2):157-165, 2017. Article (CrossRef Link)
[14] Z. Cui, Y. Cao, X. Cai, J. Cai, J. Chen, "Optimal LEACH protocol with modified bat algorithm for big data sensing systems in Internet of Things," *Journal of Parallel and Distributed Computing*, 2017. Article (CrossRef Link).
[15] X. Cai, H. Wang, Z. Cui, J. Cai, Y. Xue and L. Wang, "Bat Algorithm with Triangle-Flipping Strategy for Numerical Optimization," *International Journal of Machine Learning and Cybernetics*, 9(2):199-215, 2018. Article (CrossRef Link)
[16] M. Zhang, H. Wang, Z. Cui and J. Chen, "Hybrid Multi-Objective Cuckoo Search with Dynamical Local Search," *Memetic Computing*, 2017. Article (CrossRef Link).
[17] Z. Cui, B. Sun, G. Wang, Y. Xue, J. Chen, "A novel oriented cuckoo search algorithm to improve DV-Hop performance for cyber-physical systems," *Journal of Parallel and Distributed Computing*, 103:42-52, 2017. Article (CrossRef Link)
[18] R. Sivaraj, R. Devi Priya, "Bayesian-based parallel ant system for missing value estimation in large databases," *International Journal of Bio-Inspired Computation*, 9(2): 114-120, 2017. Article (CrossRef Link)
[19] X. Cai, X. Gao, Y. Xue, "Improved bat algorithm with optimal forage strategy and random disturbance strategy," *International Journal of Bio-inspired Computation*, 8(4):205-214, 2016. Article (CrossRef Link)
[20] P. Pongchairerks and V. Kachitvichyanukul "A two-level particle swarm optimisation algorithm for open-shop scheduling problem," *International Journal of Computing Science and Mathematics*, vol. 7(6), pp. 575-585, Dec. 2016. Article (CrossRef Link)
[21] G. Wang, X. Cai, Z. Cui, G. Min and J. Chen, "High Performance Computing for Cyber Physical Social Systems by Using Evolutionary Multi-Objective Optimization Algorithm," *IEEE Transactions on Emerging Topics in Computing*, 2017. Article (CrossRef Link).
[22] G. Yang and X. Zhang, "Task allocation algorithm for virtual design organisation in agile industrial design," *International Journal of Computing Science and Mathematics*, vol. 8(3), pp. 249-256, Jul. 2017. Article (CrossRef Link)

[23] Storm, Article (CrossRef Link).
[24] K. Kanoun, C. Tekin, D. Atienza, and M. Shaar, "Big-Data Streaming Applications Scheduling Based on Staged Multi-armed Bandits," *IEEE Transactions on Computers*, vol. 65(12), pp. 3591-3605, Dec. 2016. Article (CrossRef Link)
[25] V. Cardellini, M. Nardelli, and D. Luzi, "Elastic stateful stream processing in storm," in *Proc. of 2016 International Conference on High Performance Computing & Simulation*, HPCS 2016, IEEE Press, pp. 583-590, Jul. 2016. Article (CrossRef Link)
[26] Y. Peng, Y. Han, Y. Xiao, S. Ullah, "Heuristics for single machine scheduling problem with family setup times," *International Journal of Computing Science and Mathematics*, 8(2):166-174, 2017. Article (CrossRef Link)
[27] Y. Gu, and C. Q. Wu, "Performance analysis and optimization of distributed workflows in heterogeneous network environments," *IEEE Transactions on Computers*, vol. 65(4), pp. 1266-1282, Apr. 2016. Article (CrossRef Link)
[28] Z. Wu and S. Wang, "The structural topology optimisation based on parameterised level-set method in isogeometric analysis," *International Journal of Computing Science and Mathematics*, vol. 8(4), pp. 353-363, Aug. 2017. Article (CrossRef Link)
[29] B. Lohrmann, P. Janacik, O. Kao, "Elastic Stream Processing with Latency Guarantees," in *Proc. of 2015 IEEE 35th International Conference on Distributed Computing Systems*, ICDCS 2015, IEEE Press, pp. 399-410, Jul. 2015. Article (CrossRef Link)

Dawei Sun is currently an associate professor at the School of Information Engineering, China University of Geosciences, Beijing, P.R. China. He received his Ph.D. degree in computer science from Northeastern University, China in 2012, and finished the Postdoctoral position research at the department of computer science and technology of Tsinghua University, China in 2015. His current researches interests include big data computing, cloud computing, trust computing.

Hongbin Yan is currently pursuing a master degree at China University of Geosciences, Beijing. His current research interests include big data streaming computing.

Shang Gao received her Ph.D. degree in computer science from Northeastern University, Shenyang, China in 2000. She is currently a Lecturer at the School of Engineering and Information Technology, Deakin University, Geelong, Australia. Her current research interests include distributed collaboration, adaptive learning, and cloud computing.

Zhangbing Zhou is a professor at the School of Information Engineering, China University of Geosciences (Beijing), China, and serves as an adjunct professor at the Computer Science department, TELECOM SudParis, France. He received his PhD from the Digital Enterprise Research Institute (DERI), National University of Ireland, Galway (NUIG). His research interests include process-aware information system, service oriented computing, spatial and temporal database, and sensor network middleware.