Performance Analysis of Storm in a Real-World Big Data Stream Computing Environment

Hongbin Yan[1], Dawei Sun[1(✉)], Shang Gao[2], and Zhangbing Zhou[1]

[1] School of Information Engineering, China University of Geosciences, Beijing 100083, People's Republic of China
{yanhongbin, sundaweicn}@cugb.edu.cn, zhangbing.zhou@gmail.com

[2] School of Information Technology, Deakin University, Geelong, Victoria 3216, Australia
shang.gao@deakin.edu.au

Abstract. As an important distributed real-time computation system, Storm has been widely used in a number of applications such as online machine learning, continuous computation, distributed RPC, and more. Storm is designed to process massive data streams in real time. However, there have been few studies conducted to evaluate the performance characteristics clusters in Storm. In this paper, we analyze the performance of a Storm cluster mainly from two aspects, hardware configuration and parallelism setting. Key factors that affect the throughput and latency of the Storm cluster are identified, and the performance of Storm's fault-tolerant mechanism is evaluated, which help users use the computation system more efficiently.

Keywords: Storm · Performance analysis · Stream computing
Big data computing · Big data

1 Introduction

1.1 Background

With the rapid development of the Internet, the amount of data generated by various industries is growing exponentially. It is expected that the amount of data generated worldwide will exceed 80 ZB by 2020, which shows that we have entered the era of big data [1]. Big data contains a lot of useful information; therefore, data analysis and calculation becomes more and more important [2]. In recent years, there have been some data processing systems developed such as Hadoop and Spark which made it possible for us to handle much more data, However, they are not real-time systems, and cannot handle data in real time. Large-scale real-time data processing has become a business need, and the appearance of Storm fills the gap.

As one of the most popular distributed data processing systems, Hadoop has been successfully applied to various industries. It uses a simple programming model for distributed processing of large datasets [3, 4]. While Storm is more like a real-time

I. Romdhani et al. (Eds.): CollaborateCom 2017, LNICST 252, pp. 624–634, 2018.
https://doi.org/10.1007/978-3-030-00916-8_57

Hadoop computing system. It has its own master node and work node that are connected by external resources and submitted with application code to complete calculation tasks like Hadoop. As stated in paper [5, 6], in terms of data processing, especially for batch computing and streaming computing [7], their data processing methods are very different. Hadoop stores the data first, then processes the static data. Hard disk, as the intermediate media of data exchange, needs to read and write data during processing [8]. However, the data of Storm is always stored in memory. The data is read through the network into memory directly. In terms of computing speed, Storm is much faster than Hadoop because of this feature. Storm is therefore being applied in various industries more and more widely.

1.2 Purposes

We have been developing real-time big data processing applications using Storm for years, and understand how important it is to improve the processing efficiency. After investigating all kind of features and mechanisms of Storm, we identify the key factors which affect system throughput capacity and latency. In this paper, we examine the performance of a Storm cluster from two aspects, hardware configuration and parallelism setting, and analyze the influencing factors for throughput capacity and latency. These factors will help users design and implement more efficient applications based on Storm platform.

1.3 Paper Organization

The rest of this paper is organized as follows. Section 2 introduces the components, features and working mechanism of Storm system. Section 3 describes the experimental environment, experimental procedure and experimental results. Section 4 analyzes the factors affecting Storm throughput and latency. Finally, this paper is concluded in Sect. 5.

2 Storm System

2.1 Cluster Composition

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 can be used with any programming language. Storm is mainly used for data stream processing and real-time search [9, 10].

The various components of Storm are shown in Fig. 1:

Nimbus: The master node in a Storm cluster, responsible for sending code in the cluster, assigning tasks, and monitoring the entire cluster state.
Supervisor: The work node in a Storm cluster, responsible for accepting nimbus assigned tasks, starting and stopping their own management of the worker process.
Worker: Worker is a process that runs specific processing component logic.
Executor: Executor is a concrete physical thread in the Worker process.

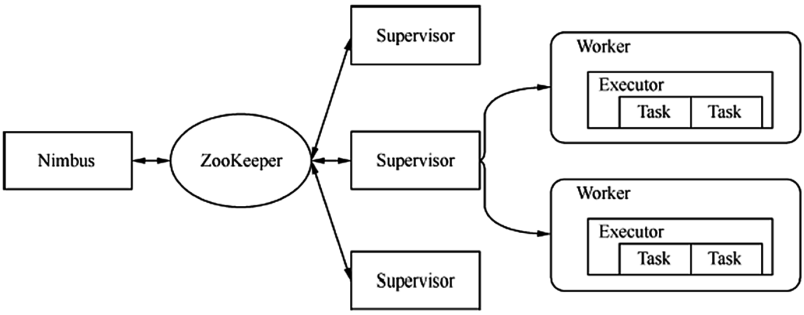

Fig. 1. Structure of a Storm cluster

Task: Task is the work that each component does.
Zookeeper: 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.
Topology: Topology is a real-time application running in Storm, similar to a job in Hadoop, but it will run until it is explicitly killed [11].

As shown in Fig. 2. There are two components in a topology, Spout and Bolt. Spout is a component that generates the source data stream in the topology. Normally a spout pulls data from the outside and then sends it to the Bolt component for its consumption. Bolt is a component that accepts the data sent from Spout and processes it. It can perform filtering, merging, writing database and other operations.

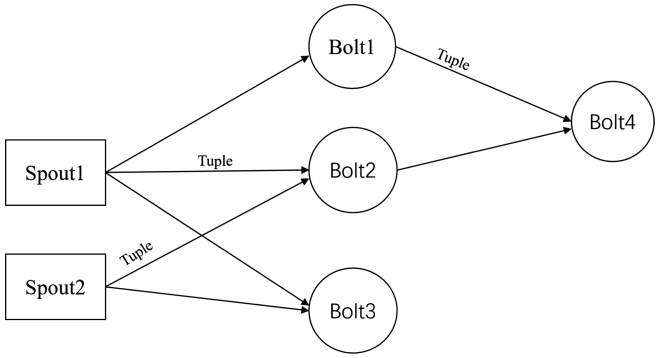

Fig. 2. Structure of a topology in Storm

Tuple is a basic unit of message passing between Spout and Bolt, and the constant Tuple will form a Stream. These data streams can be grouped between two components according to mission requirements [12]. The most common grouping methods are the following four:

- Shuffle Grouping: Randomly distributed tuple to the bolt and each bolt at the same level accepts the same number of Tuples.
- Fields Grouping: Grouped based on the value of the specified one or more fields.
- All Grouping: Copy all tuples to all bolt tasks and use them carefully.
- Global Grouping: Route all tuples to a single task, Storm select the task of receiving data by the latest task ID.

2.2 Work Mechanism

When a client submits a task topology to the master node Nimbus, Nimbus first establishes a local directory based on the configuration information of the topology, instructing zookeeper to assign the task to each work node, finally 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. Now 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. 3.

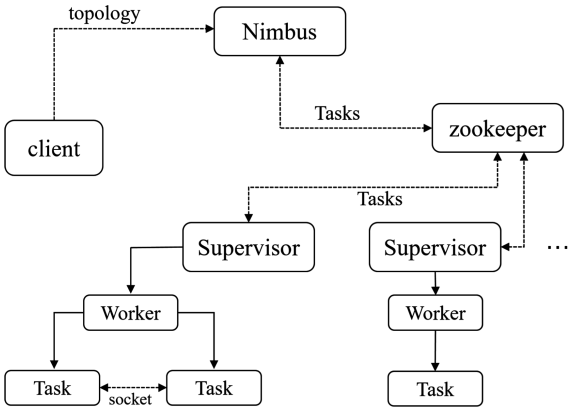

Fig. 3. Workflow of a Storm cluster

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

3 Experiment

3.1 Experimental Environment

In this experiment, Storm parallelism and fault tolerance are tested using the following hardware configuration: intel Core i5-2400CPU @ 3.10 GHz × 4, memory 4G, and

the operating system is 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 hardware configurations are used: the first group is a Dell desktop computer, with processor intel Core i5-2400CPU @ 3.10 GHz × 4, memory 4G, and 64-bit operating system ubuntu 16.04 LTS; second group is a DELL laptop, with processor Intel (R) Core (TM) i5-2430 M CPU @ 2.4 GHz × 4, memory 8G, and 64-bit ubuntu 16.04 LTS; the third group is a HP laptop, with processor Intel (R) Core (TM) i7-7700HQ CPU @ 2.80 GHz 2.81 GHz, 8G memory, and 64-bit ubuntu 16.04 LTS.

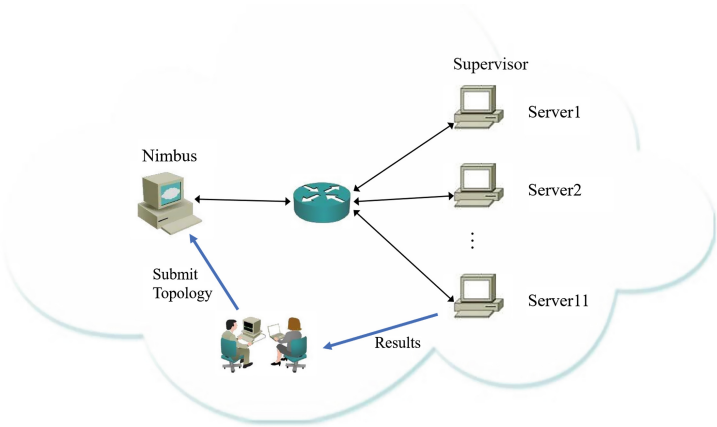

Fig. 4. Experimental environment.

3.2 Experimental Topology

The experimental network topology is shown in Fig. 4. The task topology used in this experiment 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 topology structure is shown in Fig. 5.

3.3 Experimental Process

(1) Parallelism Test

In Storm settings, the degree of parallelism is generally addressed in three areas: a topology specifies how many worker processes run in parallel; a worker process specifies how many executor threads run in parallel; and an executor thread specifies how many tasks run in parallel. In parallelism tests, the first two areas are mainly considered, with each executor assigned one task by default.

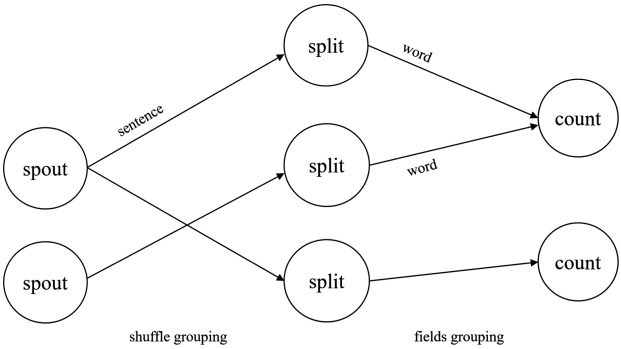

Fig. 5. Structure of the wordcount topology

Firstly, the number of threads assigned to three components spout, split, and count is (5, 8, 12) unchanged, and the number of Workers is set to 1, 3, 6, 12, and 24 respectively.

Secondly, the number of workers for each topology is not changed, and the number of threads increases in turn. The first test sets the number of Worker 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). The second test sets the number of Worker to 12, and the number of threads is the same as the first.

Finally, the number of processes set by the topologies is the same, the total number of threads of all components is unchanged, but the number of threads of each component is adjusted. The number of Worker 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), respectively.

(2) Hardware Performance Test

The number of Workers set by the topology and the number of threads for each component are unchanged. The configuration of the three machines is shown in Fig. 6.

PC	CPU	Memory	operating system
DELL Desktop	Intel Core i5-2400 CPU @ 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

Fig. 6. Hardware configuration

(3) Robustness Test

We test Storm on fault handling and analyze its impact on throughput and latency. Two special scenarios are designed. In the first scenario, we shut down two of the nodes when the cluster is running up to 20 min. In the second scenario, we shut down one node when the cluster is running up to 20 min. After the two tests, we compare them with the normal scenario.

3.4 Experimental Results

In the parallelism test, when the number of threads keeps constant and the number of Workers in the topology increases, the throughput of the system and the latency of processing data are essentially unchanged. The average throughput in an hour is about 366 Tuple/s. 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 Worker increases, the processing latency does not change much as shown in Fig. 7. It basically remains between 3 and 4 ms.

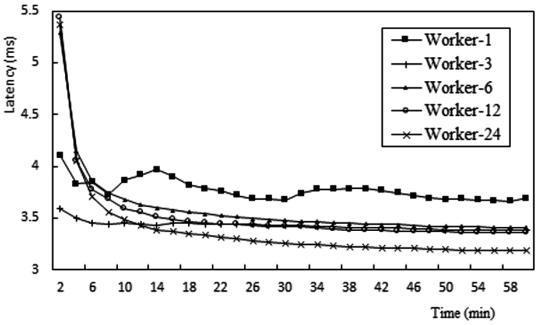

Fig. 7. Latency test results of worker

When the number of Worker in the topology is 3, as the number of threads assigned to each component increases, the average throughput over an hour also increase. Also, 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. In terms of system latency, as shown in Fig. 8, if the number of Worker in the topology is 3, when the number of threads assigned to each component increase, 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. The same result applies when the number of Worker is set to 12, when the delay achieves minimum, the number of threads assigned to each component is (100, 60, 240).

Given the number of processes in topologies keeps the same, the total number of threads of all components unchanged, if the number of threads in the three components is set to (5, 8, 12), (5, 10, 10), (5, 12, 8), (3, 10, 12) (7, 10, 8), (7, 8, 10), (9, 7, 9), (12, 5, 8) respectively, the average throughput and latency during the one-hour testing

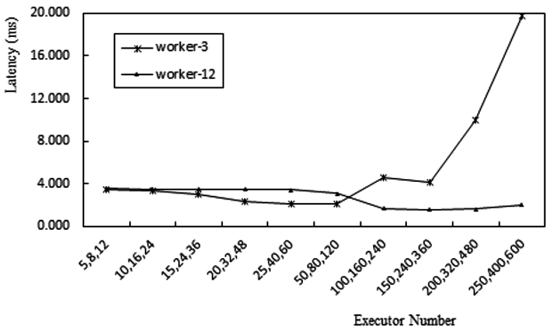

Fig. 8. Latency test results of executor

period are shown in Fig. 9. It is observed that the system throughput is only related to the number of threads assigned to the delivery component spout; 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. In terms of system latency, 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.

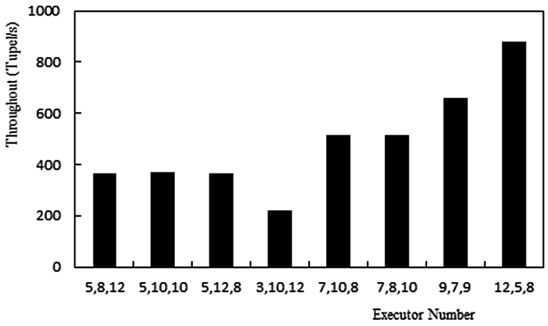

Fig. 9. Throughout test results of thread allocation

In the hardware performance test, when using three different performance computers to run the same topology, the amount of data processed within one hour is basically the same. The average throughput is 366 Tuple/s. But in terms of data processing latency, the DELL laptop with poorer performance has obviously higher latency than the other two as shown in Fig. 10. The average delay of DELL notebook with lowest CPU performance is much higher than the other two during the one-hour testing period. CPU performance has a greater impact on data processing latency than memory.

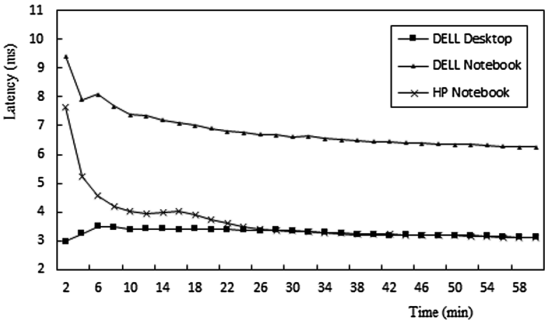

Fig. 10. Results of hardware performance test

In the robustness test, as shown in Fig. 11. In the first test, when two nodes are shut down after 20 min, 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 10 min later. In the second test, we shut down a node after 20 min, the outcome observed is almost the same as the first. There is slight reduction in the number of data processed within one hour in the second test. The average throughput is 361,351,350 Tuple/s. But in terms of system latency, task redistribution does not affect it.

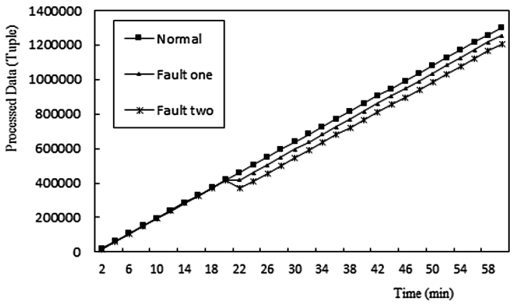

Fig. 11. Results of robustness test

4 Result Analysis

After analyzing the test results, we can conclude that:

- Increasing the number of Worker 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 Worker number 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.

- Storm handles faulty nodes quickly. Task redistribution has little impact on the data processing speed and does not increase the data processing latency.

5 Conclusion

In this paper, we test the Storm cluster from three aspects: the parallelism, hardware performance and fault handling. The results observed show that Storm has strong scalability and robustness. For users who use Storm to develop real-time big data processing applications and want to improve the speed of processing, it is recommended that: besides of increasing the number of Worker or threads that assigned to components, the number of the Worker and the threads of individual components should also be set appropriately. To improve the degree of parallelism, both of the CPU performance and system memory should be considered. What's more, each node should leave out memory to accommodate redistributed tasks from failed nodes.

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.

Acknowledgment. This work is supported by the National Natural Science Foundation of China under Grant No. 61602428; the Fundamental Research Funds for the Central Universities under Grant No. 2652015338.

References

1. Václav, S., Jana, N., Fatos, X., Leonard, B.: Geometrical and topological approaches to Big Data. Future Gener. Comput. Syst. **67**, 286–296 (2017)
2. Chen, D.Q., et al.: Real-time or near real-time persisting daily healthcare data into HDFS and ElasticSearch Index inside a Big Data platform. IEEE Trans. Ind. Inform. **13**(2), 595–606 (2017)
3. Mavridis, L., Karatza, H.: Performance evaluation of cloud-based log file analysis with Apache Hadoop and Apache Spark. J. Syst. Softw. **125**, 133–151 (2017)
4. Lv, Z.H., Song, H.B., Basanta-Val, P., Steed, A., Jo, M.: Next-generation Big Data analytics: state of the art, challenges, and future research topics. IEEE Trans. Ind. Inform. **13**(4), 1891–1899 (2017)
5. Zhang, J., Li, C.L., Zhu, L.Y., Liu, Y.P.: The real-time scheduling strategy based on traffic and load balancing in storm. In: Proceedings of the 18th International Conference on High Performance Computing and Communications, pp. 372–379. IEEE Press (2016)
6. Xu, J.F., Miao, D.Q., Zhang, Y.J., Zhang, Z.F.: A three-way decisions model with probabilistic rough sets for stream computing. Int. J. Approx. Reason. **88**, 1–22 (2017)
7. Zhang, W.S., Xu, L., Li, Z.W., Lu, Q.H., Liu, Y.: A deep-intelligence framework for online video processing. IEEE Softw. **33**(2), 44–51 (2016)

8. Rahman, M.W., Islam, N.S., Lu, X.Y., Panda, D.K.: A comprehensive study of MapReduce over lustre for intermediate data placement and shuffle strategies on HPC clusters. IEEE Trans. Parallel Distrib. Syst. **28**(3), 633–646 (2017)
9. Karunaratne, P., Karunasekera, S., Harwood, A.: Distributed stream clustering using micro-clusters on Apache Storm. J. Parallel Distrib. Comput. **108**, 74–84 (2017)
10. Cardellini, V., Nardelli, M., Luzi, D.: Elastic stateful stream processing in storm. In: Proceedings of the 14th International Conference on High Performance Computing & Simulation, pp. 583–590. IEEE Press (2016)
11. Shieh, C.K., Huang, S.W., Sun, L.D., Tsai, M.F., Chilamkurti, N.: A topology-based scaling mechanism for Apache Storm. Int. J. Netw. Manag. **27**(3), 1–12 (2017)
12. Li, C.L., Zhang, J., Luo, Y.L.: Real-time scheduling based on optimized topology and communication traffic in distributed real-time computation platform of storm. J. Netw. Comput. Appl. **87**, 100–115 (2017)