Docker Container Networking Based Apache Storm and Flink Benchmark Test

Tao Liu
Dept. of Computer Technology
China University of Mining and Technology, Beijing
Beijing, China
liutao4_3@163.com

Zhihong Yang
Dept. of Computer Technology
China University of Geosciences, Beijing
Beijing, China
zhihongyang@cugb.edu.cn

Yuzhong Sun
Dept. of Computer Science
Institute of Computing Technology, Chinese Academy of Sciences
Beijing, China
yuzhongsun@ict.ac.cn

***Abstract*—Many distributed stream computing engines have emerged to handle big data, and they can be deployed in cloud environments consisting of native networks or container networks. Most of the benchmark research on stream computing engines are carried out under the native network, and the research on the impact on container network on stream computing engines is currently inadequate. However, the use of container network will inevitably lead to performance degradation, which is the disadvantage of all virtual networks. In this work, we build Apache Storm and Apache Flink, which are Streaming Computation Engines in container network and native network environments and conduct performance measurements through experiments processing textual data to verify how much performance decreases in container network. Experiments show that the throughput in a container network environment is 1%-5% lower and CPU utilization is 11%-18% lower than in a local network environment.**

Keywords—Container Network, Apache Storm, Apache Flink, Streaming Computation Engines, Benchmark Test.

I. INTRODUCTION

In recent years, emerging information technologies and application methods such as the Internet of Things, mobile Internet, social platforms, and advertising are rapidly developing and evolving, resulting in a rapid increase in data volume and pushing human society into the era of big data. In the context of big data, the scale of streaming data continues to increase. In order to process the ever-increasing streaming data, a series of streaming computing engines have emerged. There are several representative stream computing engines including Apache Spark [1], Apache Storm [2] and Apache Flink [3]. These technologies are widely used in companies such as Yahoo, Twitter, and Alibaba. In particular, Apache Flink has gradually become the most popular big data processing technology due to its simple code, integrated flow and batch features, and rich ecology.

Distributed systems such as streaming computing engines are scalable and can be configured on multiple servers. However, as the number of servers in use grows it becomes increasingly difficult to configure distributed systems. To solve this problem, distributed environments can be easily built using Docker [4], and the key to containers communicating with each other is container network [5]. Since virtualization systems like Docker suffer from performance degradation, some developers configure distributed systems directly in their local environment instead of using Docker.

However, researchers have considered only two cases, native network and single container [6, 7], and ignored container network. in this paper, we build a system for processing textual data onto Apache Storm and Apache Flink, which are streaming data processing systems, and examine the extent of performance degradation through benchmarking experiments in both the native network and container network environments.

II. RELATED WORKS

A. Docker

As shown in Figure 1, specific examples of KVM and Docker are used to demonstrate the differences between traditional and emerging virtualization systems. Unlike traditional KVM virtualization systems that require a KVM-based hypervisor and guest OS installed on top of the Host OS to achieve complete hardware isolation, Docker containers only need to drive the Docker engine on top of the Host OS to build an isolated execution environment. In other words, Docker virtualization system does not need to install Guest OS separately on top of Docker engine, but only needs to install some code libraries and programs to share hardware resources with the host and create an isolated execution environment. So, developers use Docker to package application software and the system tools, system dependencies and runtime tools required by the application software in a read-only hierarchical image, and deploy the application of the image.

In the last decade, virtualization systems have been widely used in large data centers, embedded systems and personal computers. A comparative study of multiple virtualization systems is being conducted to validate the performance benefits of Docker by describing the differences between traditional and emerging virtualization systems as described above [8, 9, 10].

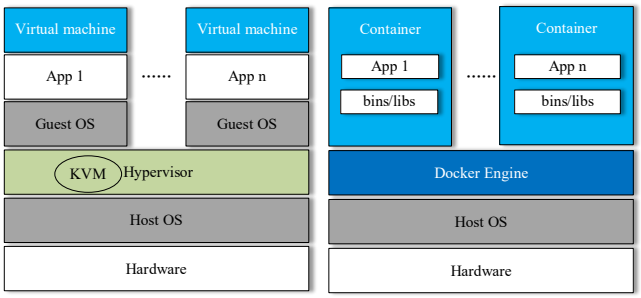

Fig. 1. KVM and Docker Architecture

B. Container Network

The key to building a container cluster is container network, and the use of container network is to solve the problem of communication between containers. At present, Docker official website and open-source community have proposed many solutions to solve the container in the case of container isolation from mutual communication. Among them, Swarm Overlay is Docker's native solution for container communication, which has the advantages of simple

configuration and its potential application value is very large [5].

As shown in Figure 2, this is the network structure of Overlay. Overlay needs to add K-V storage (Redis, Consul, Ectd, etc.) as the storage system for data such as IPs and ports, and then get the network configuration information from this storage system to ensure that the container clusters are on the same network segment.

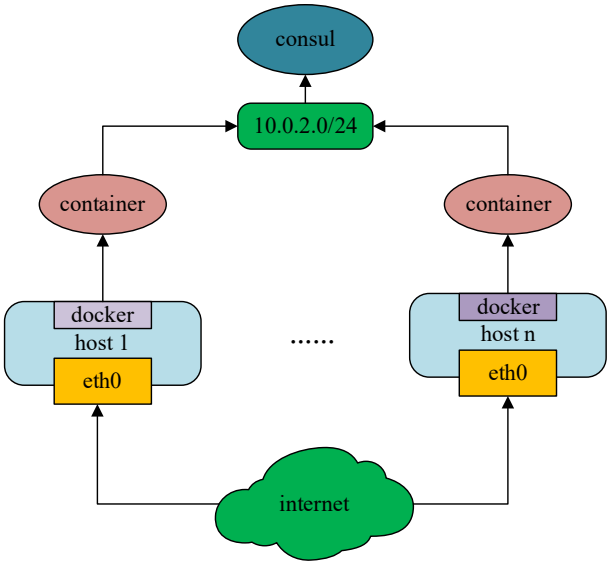

Fig. 2. Overlay Network Architecture

C. *Apache Storm and Apache Flink*

Apache Storm and Apache Flink are both open sources, distributed and memory-based stream computing engines. As shown in Figure 3, they both uses directed acyclic graph (DAG) as the computational model, which consists of three phases: pulling data, transforming data, and storing computational results. Apache Storm is programmed in a DAG-oriented structure while Apache Flink is programmed in a data-oriented way.

The computational model of Apache Storm consists of Spout and Bolt, where Spout reads a stream of data from the outside, which consists of a continuous stream of Tuple, Bolt receives the stream and processes it, and finally delivers the result to the storage system or to the next module of the business. As mentioned above, the series of tasks performed by Spout, Bolt and Data Stream are defined as Topology, which obviously uses DAG to represent the structure and logic of Data Stream processing. Spout, Bolt and Data Flow forwarding need to be defined by the developer, so Storm is programmed for DAG structure.

Apache Flink is a stream computing engine that uses DAG as the computational model and has the same computational process as Storm, but unlike Storm, Flink abstracts many operations and provides them to the user in the form of an API. The user doesn't feel like he or she is building a DAG, and is more focused on each step of the data flow. Specifically, Storm requires developers to define each step of the operation themselves, whereas Apache Flink directly uses functions such as map, flatmap and keyby to complete data processing tasks. So Flink is a data-oriented way of programming.

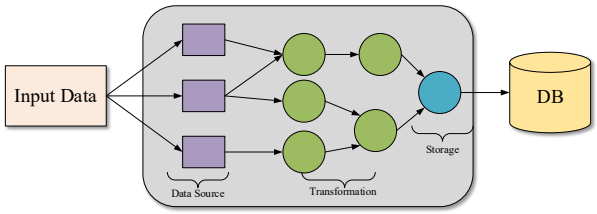

Fig. 3. Storm and Flink calculation model

III. Benchmark Design

Two benchmark tests were implemented based on the structural features of the Storm and Flink computational models.

As shown in Figure 4, the first benchmark test is a CPU-intensive linear computational model consisting of four vertices. The tasks represented by each vertex are as follows:

1) The task numbered 1 pulls text data from Kafka.

2) The task number 2 cuts the string into words.

3) The task numbered 3 counts the number of each word.

4) The task numbered 4 puts the calculation results into Redis.

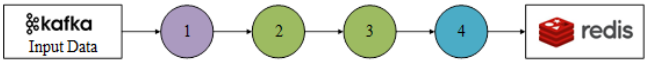

Fig. 4. Linear calculation model

As shown in Figure 5, the first one is a Network-intensive diamond-shaped computing model consisting of five vertices. Each vertex represents a task as follows:

1) The task numbered 1 pulls text data from Kafka.

2) The tasks numbered 2, 3 and 4 splice the string with a question mark character.

3) The task numbered 5 receives the data and passes it directly to the next node without doing any processing.

4) The task numbered 6 puts the calculation results into Redis.

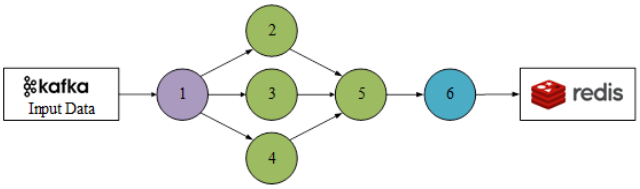

Fig. 5. Diamond calculation model

IV. Experiments And Results

In this section performance test are performed on the Kafka-connected distributed data stream processing engine. First, we deploy Apache Storm and Apache Flink directly on three servers, then deploy these two stream computing engines on the container network, and finally run the two benchmark tests designed above separately.

A. *Experimental environment*

As shown in Table 1, the experimental environment consists of five servers, two of which deploy Kafka message middleware and Redis database, and the remaining three servers deploy stream computing engines in native network and container network, respectively.

TABLE I. EXPERIMENT ENVIRONMENT

Server number	Configuration Information	
	Software	Hardware
Node-1	Redis: 6.2.3 Docker: 20.10.5 OS: Centos7	CPU: Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00GHz × 2 HDD: 7.6TB RAM: 46GB
Node-2	Apache Kafka: 2.5.1 Docker: 20.10.5 OS: Centos7	CPU: Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00GHz × 2 HDD: 7.4TB RAM: 64GB
Node-3	Apache Storm: 2.2.0 Apache Flink: 1.12 Docker: 20.10.5 OS: Centos7	CPU: Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00GHz × 2 HDD: 7.3TB RAM: 46GB
Node-4		
Node-5		

B. *Throughput and CPU Utilization*

First, input about 40GB of textual data, and each processing block is about 1.31MB, then the benchmark test is run for 15 minutes to store the calculation results in the database, test the CPU usage during the run, and finally count the throughput.

As shown in Figure 6, the CPU-intensive benchmark test has lower throughput in the container network environment than in the native network environment, but the reduction is insignificant. The results of this experiment also show that the throughput of Flink is about 40% higher than that of Storm. As shown in Figure 7, The average CPU resource utilization of Storm with three service servers is about 50% higher than that of Flink and the CPU utilization degradation in the container network environment is low. In summary, Flink performs better than Storm in this experiment.

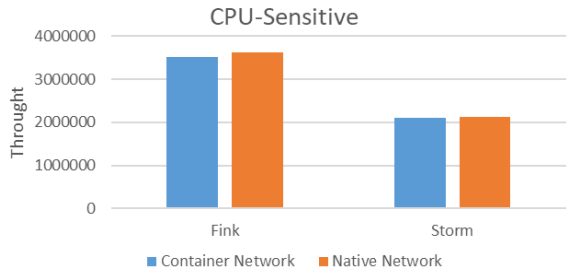

Fig. 6. CPU-intensive calculation model

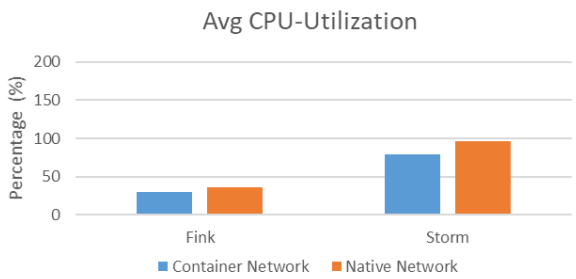

Fig. 7. Linear model CPU usage

As shown in Figure 8 below, the Network-intensive benchmark has lower throughput in the container network environment than in the native network environment, but the difference is extremely small. However, the throughput of Flink is tens of times higher than Storm, and the reason is related to their scheduling algorithm. As shown in Figure 9 below, the CPU resource utilization in the container network environment is lower than the throughput in the native network environment, and the CPU resource utilization of Storm is higher than that of Flink. In conclusion, the impact on container network on Storm and Flink is minimal, and the main performance loss is caused by the containers themselves.

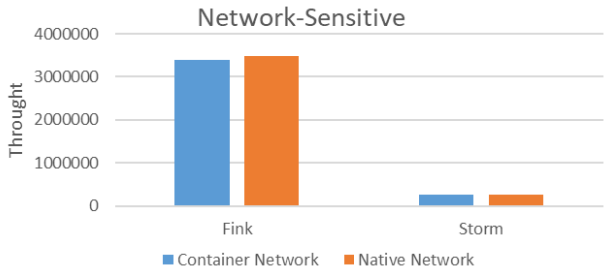

Fig. 8. Network-intensive calculation model

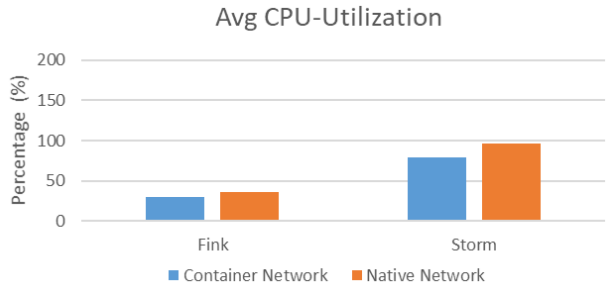

Fig. 9. Diamond model CPU usage

V. CONCLUSION

In this paper, we build a container network, deploy Apache Flink and Apache Storm on top of this platform, and use two benchmarks to measure their performance in container network and native network environments, respectively. The two benchmarks represent CPU-intensive and network-intensive, respectively, and the experiments show that Flink outperforms Storm by 1%-5% lower throughput and 11%-18% lower CPU usage in the container network environment compared to the native network environment. in addition, the experimental results show that the performance loss caused by the containers themselves is considered much larger than that of the container network. In future research, we will build different types of container networks and explore the adaptability of stream computing engines to various container networks.

ACKNOWLEDGMENT

This work was partially supported by the Institute of Computing Technology, Chinese Academy of Sciences and supported by the National Key Research Program of the Networked Operating System for Cloud Computing (No. 2016YFB1000505).

REFERENCES

[1] M. Zaharia, et al., "Spark: Cluster computing with working sets," In Proc. of the 2nd USENIX Conference on Hot Topics in Cloud Computing, p. 10, June. 2010.

[2] A. Toshniwal, et al., "Storm@ Twitter," In Proc. SIGMOD of the International Conference on Management of Data, ACM, pp. 147- 156, June. 2014.

[3] P. Carbone, et al., "Apache flink: Stream and batch processing in a single engine," Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, Vol. 36, No. 4, pp. 28-38, 2015.

[4] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," Linux Journal, Vol. 2014, No. 239, 2014.

[5] Zeng H, et al., "Measurement and evaluation for docker container networking," In Proc. of 9th International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, IEEE, pp. 105- 108, Oct. 2017.

[6] Chintapalli S, et al., "Benchmarking streaming computation engines: Storm, flink and spark streaming," In Proc. of IEEE 30th IEEE International Parallel and Distributed Processing Symposium Workshops, IEEE, pp. 1789-1792, May. 2016.

[7] Bang J, Choi M J, "Docker environment based Apache Storm and Spark Benchmark Test," In Proc. of 21th Asia-Pacific Network Operations and Management Symposium, IEEE, pp. 322-325, Sept. 2020.

[8] Mbongue J M, Kwadjo D T, Bobda C, "Performance Exploration of Virtualization Systems," arXiv preprint arXiv:2103.07092, 2021.

[9] C. Yong, L. Ga-Won, and H. Eui-Nam, "Proposal, “Proposal of Container- Based HPC Structures and Performance Ana lysis, " Journal of Information Processing Systems, Vol. 14, No. 6, pp. 1398-1404, Dec. 2018.

[10] M. T. Chung, N. Quang-Hung, M. Nguyen and N. Thoai, "Using Docker in high performance computing applications, " In Proc. of IEEE 6th International Conference on Communications and Electronics, IEEE, pp. 52-57, Jul. 2016.

