Research on Big Data Management and Storage Based on Linux Container

Jing Yang

Hanjiang Normal University, Shiyan 442000, China

8041556@sohu.com

Keywords: Big data, Linux Container, Distributed System.

Abstract. These years, a new generation of virtualization technology, aka Linux container，is becoming more and more popular. Linux container is a lightweight virtualization technology which providing resource isolation with little performance overhead. This paper designs and implements a solution for Big Data management and storage based on Linux container. The main work and contributions of this paper are as follows:

1. Researched the next generation distributed resource scheduling technology. Normally, to integrating into existing distributed schedulers, distributed applications need to write a specific accessing module. We designed a new 2-layer distributed resource scheduler architecture which reduced the coupling between scheduler and distributed applications.
2. Researched the combination of Linux container with Big Data system. We brought Linux container into the Big Data system to replace traditional virtualization technology, thus improving the resource utilization of the underlying hardware.
3. Designed and implemented a prototype system and tested its flexibility, application performance overhead and scheduling overhead.

Introduction

With the development of Big Data technologies, distributed computing and storage systems become more and more mature and diverse. Nowadays, a specific distributed system sometimes cannot handle a whole computing application. What's more, a single cluster may serve different users who have different demands. As a result, a cluster providing diverse processing and storage service is needed urgently by the industry. These years, a new generation of virtualization technology, aka Linux container，is becoming more and more popular.

Storage systems are quite important in clusters. Traditional high performance computing applications uses file system for storing the data. Largely used distributed file system in the high performance computing is Lustre. Scientific computing usually generate high volume of unstructured data or they have their own formats of result records. Thus applications can manage their data stored on distributed file systems efficiently. However, with the current development of applications running on top of clusters, different applications will need different storage systems for their special requirements. For example, the MPI programs can run aside with MapReduce programs on the same cluster, the former need the file systems like Lustre or Ceph [1] and the later will need a file system like HDFS [2] [3]. And some graph processing application need some form of graph storage system different from any of the previous storage mechanism. Needless to say, the database systems [4] [5] will be quite important for some large scale data processing applications . Various kinds of database systems might needed like HBase or MongoDB. Such phenomenon needs the re-thinking of storage side in the computing cluster environment. In one word, individual application might need a specific storage system. This will become even more prevalence with the more and more largescale data processing applications running on clusters deployed in data centers. Based on the high performance computing application characteristics, traditional distributed schedulers usually do not consider storage systems. Storage systems were considered as static and deployed together when the cluster was setup i.e. hardware getting ready and operating system installed. The storage systems were

considered as part of the infrastructure not part of the scheduling procedure. This was the situation both for high performance computing and large scale data processing applications if they were considered separately. For example, LSF provides the schedulers for high performance computing clusters for years. And the scheduler of MapReduce in Hadoop will only concern about how to schedule MapReduce jobs. Recent development of YARN scheduler can embrace different kinds of applications such as using a single scheduler to schedule both MPI jobs and MapReduce jobs. However, this is not enough because the storage system still does not involve with the scheduling. These two types of applications need the distributed file systems installed. Considering that there might be more applications running on top of the same physical cluster, installing all possible storage systems forehand is not feasible. It needs systematic way to achieve the goal of scheduling storage system for every specific application. Our solution is to integrate the storage system together with the distributed scheduler.

In this paper, to avoid the overhead brought by virtual machines while still providing enough flexibility of supporting different applications, we use the operating system virtualization to box the application environment instead of using virtual machines. There are many incarnation of operating system virtualization such as Linux containers, BSD Jails or Solaris Zones. Flex uses containers under Linux. As the operating system level virtualization brings much lower overhead than virtual machines, it is now becoming more and more prevalent in recent deployment. For example, PlanetLab is the typical application of Linux container for providing the multi-user access of a global platform. Using container instead of virtual machines can support more users.

Background

Distributed Scheduler. Our work is tightly related to the distributed scheduler. For running applications on a cluster, there needs a scheduler for managing and coordinating the computing resources and all the activities in the cluster. Usually the scheduler will get job descriptions submitted by users. And also, the cluster states, including the detail state of every node in the cluster, will be reported to the scheduler. Based on such information, the scheduler can then decide which job might be the next job for execution. The cluster administrator can control the scheduler by providing the parameters for task queues such as the job priority. Typically, the storage system information will not be used for scheduling as the jobs running in the cluster will always consider the storage system as available and can be used directly. Thus, the applications running in the cluster will use a fixed configuration of storage system. This is usually not a problem for applications with relative same characteristics like traditional HPC programs using MPI. However, with the diverse applications with huge different workloads running in the same cluster, the assumption is not valid anymore. Applications might have very different requirements for storage applications. No unified storage mechanism can fit all the storage features needed by those applications. That is why we need a new generation of distributed scheduler for better supporting heterogeneous application types running on clusters.

Virtual machines are widely used in cloud computing platforms, this phenomenon begins to influence the distributed scheduler. The scheduler can run inside virtual machines as they did in traditional clusters. Also, the scheduler can now direct virtual machines. The main focus will be start and shutdown the machines at appropriate time. Although the storage system can be installed inside virtual machines and provide each application with its own configuration of a storage system, the performance and management overhead make this mechanism hardly satisfactory as discussed before. Flex system is heavily rely on the operating system level virtualization . Operating system level virtualization is different from the virtual machines. Fig.1 shows the comparison between virtual machines and operating system level virtualization. Virtual machines are usually to virtualize hardware including the instructions, registers, memory as well as IO devices. The purpose of virtual machine is to run operating system inside it without any modification to the OS binary. As a result, such heavy virtualization can bring unavoidable overhead. The industry is now trying hard to improve the virtual machine performance. In addition, as discussed before, because of the strong boundary between the environment inside and outside of the virtual machine, the stored data cannot be easily

shared among related applications. Operating system level virtualization does not mean to provide the full hardware compatible virtualized layer. Unmodified operating system binaries can not run in such environment. Instead, operating system virtualization only runs a single kernel with separate namespaces for different runtime environments. Because only a single kernel gets executed on the hardware platform, there will be no overhead for simulating hardware instructions or doing binary translation to executing an instruction. The operating system level virtualization is considered to have very little overhead or no overhead. The kernel provides namespaces like different virtual machines. Each namespace can be considered as a separate independent environment for running applications. Each namespace is thus called as a container, which can separate applications.

As containers are independent environment, all the needed configurations can be packed into containers. This can provide the flexibility for various applications. Containers can use the local file system to provide another fake root file system for running a new environment for applications. Thus, to share the data among applications is considered easy. Flex system uses the containers to achieve the goal of storage scheduling. Different applications can run inside different containers and access the customized storage system running together with the application. After the application finish the corresponding work, the data generated can be saved on the fake file system. The files can be accessed from the outside. This is a quite convenient way for sharing data among different applications. There will be no need for transferring data from or to a virtual machine.

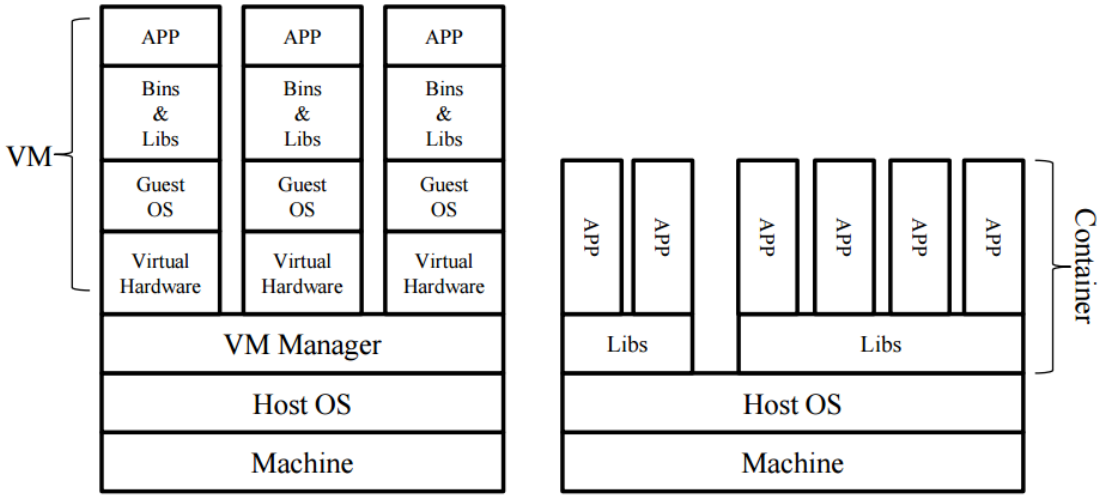

Fig. 1 Virtual Machines vs. Linux Container

System Architecture

Architecture. As with the traditional design of distributed scheduler, Flex also contains a management node and several (tens to thousands) slave nodes. Fig.2 gives the architecture of Flex system. A master process will run in the management node. Worker processes are running on slave nodes. The single master design might bring the performance and fault tolerant issue. However, as the master will not involve any large processing work such as doing computation or data storage. The main job of master is doing management and there will not be too much workload for the master process. Thus, the master node will not be a performance bottleneck. For the fault tolerant issue, currently Flex rely on some other mechanism to provide master fault tolerant. For example, one can store the persistent information in the high fault tolerant component like ZooKeeper and rely on this facility to assign a single master. Similar mechanism can be found in the system like BigTable. This architecture is commonly used in the distributed scheduler running in the current data center cluster. And many infrastructure and applications uses this architecture to simplify the design of a distributed system. Another reason to embrace this architecture here is because Flex is in fact directly extending the traditional distributed scheduler. Making a small change will make the mechanism more acceptable to the users.

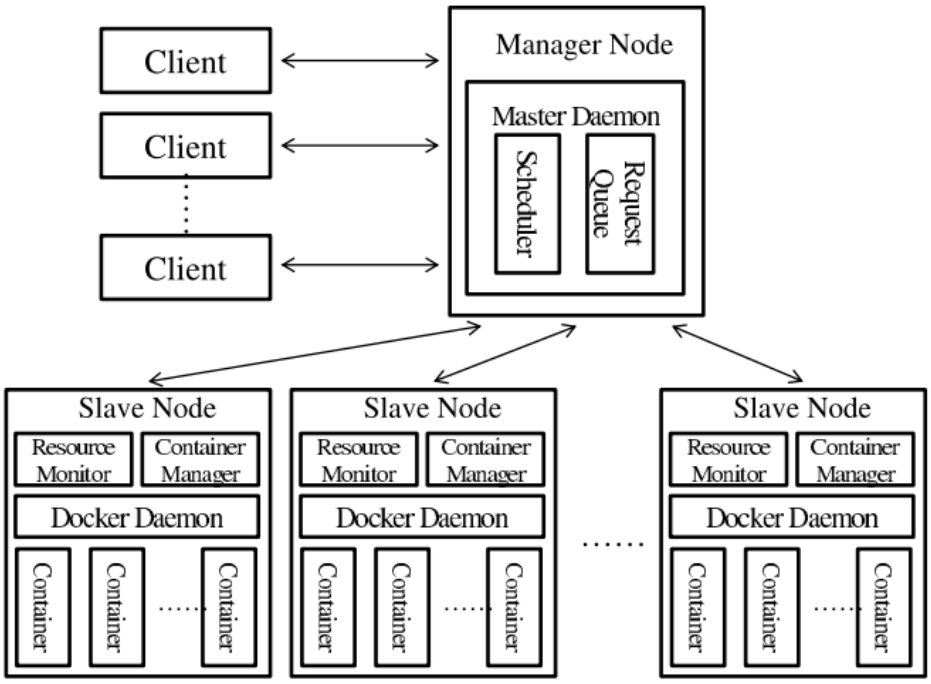

Fig. 2 Flex Architecture for Storage Scheduling

System Monitoring Framework. Before any scheduling decision can be made, the master program must get enough information for the status of the current running cluster. The distributed monitoring framework is also built on top of the master and slaves. The slaves will extract the local status information from the system interfaces including the CPU, memory, network and storage information. And also the status for running containers in a specific slave will also be reported to the master.

Evaluation

Experimental Setup. We did our evaluation on a small cluster containing 32 nodes. All these machines were connected to a switch with 1000Mbps bandwidth. Each machine was equipped with 4 cores CPU at 1.8GHz, 16G DDR3 memory and one 500GB SSD drive. Linux 64bits CentOS 6.4 with upgraded kernel version 2.6.32 was installed in all machines. For testing the effectiveness of Flex, we did three types of evaluation. The flexibility test was used to test the compatibility of Flex. Various kinds of applications were deployed and got executed on the Flex platform to see whether the programs could run properly. And also this test showed the convenience to deploy applications with containers. The application performance overhead tested the application performance overhead brought by the Flex platform. This test could prove the viability of the platform in practical situations. Because the storage systems sometimes run continuously in the cluster environment, any technology applied should be concerned about its influence of performance overhead. Last test was about the overhead of scheduling. The purpose of this test was to see the latency brought by the extra steps of storage scheduling. This overhead will be critical for some small jobs, which might last very short time. The latency of scheduler should be considered.

Results and Discussion. We have evaluated some typical applications for high performance computing as well as large-scale data processing applications. For example, MPI programs were used as the typical applications of high performance computing. Various Hadoop workloads were used as example programs of large scale data processing. And also we had done several storage workloads test for getting the compatibility results for storage scheduling.

Table 1 Application

Application	Test Program
MPICH2	Parallel Pi Calculation
Hadoop 2.6.0	Built-in Word Count Program and HDFS benchmark tool
Spark 1.2.1	Logistic Regression Program and Word Count Program
Redis 2.8.19	Built-in redis-benchmark Program
MongoDB 2.6.9	Benchmark tool Mongo-perf
MooseFS 2.0.60	Benchmark tool Filebench

Table 1 shows the applications in the flexibility test. It can be seen that all these applications can run in the Flex platform. The Linux container can perfectly support running applications inside. For the distributed storage systems or most distributed systems, the components are usually run in the application level. This means that Flex can support all these applications. However, for some applications that have kernel components, Flex cannot provide the virtualized environment for such applications. This does not mean Flex is not able to run such applications. In fact, containers can support such applications. The reason of Flex does not want to run such application is that the application might escape from the control of the platform.

Conclusion

In this paper, we have discussed various aspects of using operating system level virtualization for extending the current distributed scheduler. We focus on the storage scheduling of the system as for the emerging large scale data processing applications need specialized storage systems. A prototype system called Flex was built. Flex can build customized environment for running distributed jobs. Before running an application, the needed storage system will be firstly deployed. Using containers can provide flexible deployment of application as well as the needed storage system. Flex can avoid two problems in previous platforms. The applications can use specific storage environments instead of sharing a common one as in traditional high performance computing cluster. And also data can be easily shared among applications that are not easily achievable through virtual machines used in cloud computing platform. In conclusion, the extension made by Flex for the distributed scheduler can achieve high flexibility for running application and sharing data with very low overhead. Experimental results show that the extensions can be applied in practical situation.

Acknowledgements

This work was supported by Hubei Provincial Department of Education Science and Technology Research Project “Research on Key Technologies and Application of Big Data’s Reliable Storage in Distributed Environment” (Project No. B2015440).

References

[1] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. Long, and C. Maltzahn, “Ceph: A scalable, high-performance distributed file system,” in Proceedings of the 7th symposium on Operating systems design and implementation. USENIX Association, 2006, pp. 307–320.

[2] S. Ghemawat, H. Gobioff, and S.-T. Leung, “The google file system,” in ACM SIGOPS operating systems review, vol. 37, no. 5. ACM, 2003, pp. 29–43.

[3] D. Borthakur, “Hdfs architecture guide,” Hadoop Apache Project, p. 53, 2008.

[4] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, “Bigtable: A distributed storage system for structured data,” ACM Transactions on Computer Systems (TOCS), vol. 26, no. 2, p. 4, 2008.

[5] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, “Dynamo: amazon’s highly available key-value store,” in ACM SIGOPS Operating Systems Review, vol. 41, no. 6. ACM, 2007, pp. 205–220.