Advances in Computer Science Research, volume 50
4th International Conference on Electrical & Electronics Engineering and Computer Science (ICEEECS 2016)

Research on the Quality of Service of Docker Container based on the Linux Control Group

Weicheng Sun[1, a]

[1] School of Software Engineering, Shanghai Jiao Tong University, China 200240

[a]email: SunWeicheng0001@sjtu.edu.cn

Keywords: container; Quality of Service; Docker; Virtualization

Abstract. Container, such as Docker, is becoming more and more popular in recent years in cloud computing area. More and more people started using container as their choice to run services on the cloud. While containers are just like processes on the Linux, in fact, so it is quite difficult to do resource management like traditional virtual machine does. To meet different demands of cloud users in using cloud computing resources under the cloud computing environment, especially disk I/O resource, an effective disk I/O QoS (Quality of Service) for Docker or some other container-based cloud computing technique should be used. Thus, we designed dDiskQoS, a solution to do disk I/O QoS for container-based cloud computing technique, which can effectively manage and control disk I/O traffic. The key idea of dDiskQoS is to leverage the advantage of Linux cgroups technique and implement a multi-weighted algorithm to satisfy the disk I/O requirements from both time-sensitive containers and big data throughput containers. The evaluations show that the solution can meet the demands of different scenarios while accepting different parameters, to guarantee system efficiency. Additionally, there is very small overhead in loading our QoS solution. With dDiskQoS, operators can keep more containers running in one host, which will certainly save the cost and energy.

Introduction

Lightweight container-based virtualization has attracted increasing attention around the world in these years, especially Docker. Compared with traditional heavyweight virtualization technologies, like Xen or KVM, container-based virtualization is much more portable and easier to deploy for users. When using virtual machines, people are required to install a host OS by themselves and required libraries also, which cost a lot of time. Even operators can package a system with libraries installed into an ISO file, the installation time is still much longer than deploying a container. Saving time and easily deploying is the point that people prefer to choose container while deploying services on the cloud. Thus, while deploying micro-service, many people prefer Docker [1].

Because container-based virtualization is also used in cloud computing just like traditional heavyweight virtualization, and due to its high popularity, containers will also occupy huge amounts of machines in every data center to satisfy different kind of user demands. In this way, containers will meet a big challenge just like virtual machine does, that is, resources management. In the resources related with cloud computing, disk not just plays a role in data storing but also in system resources, so disk I/O resource is obviously quite important in all kind of resources. A high-performance, secure, flexible method to guarantee disk QoS can become a part in cloud computing fee, which can also be used to collect history data of cloud usage to make the whole system better. For users, they can enjoy more guaranteed I/O performance by choosing a strategy according to their requirements. These things can make cloud computing more fast and more stable.

Linux cgroups, which started by Engineers at Google in 2006 under the name “process containers”, and its name changed to “control cgroups” to avoid confusion caused by multiple meaning of “container” in the Linux kernel context. The technique is used to limit, account for, and isolate the resource usage (CPU, memory, disk I/O, network, etc.) of a collection of processes. In the paper, we will leverage the ability of Linux cgroups to do resource limiting on certain containers.

In this paper, we propose dDiskQoS, a complete solution to do disk I/O QoS for container-based cloud computing services. Since containers are just processes in face, Linux cgroups can also be applied. However, a direct way to assign cgroups configuration for every container doesn't make sense, because the situation may change all the time, for example a container may alternative its disk I/O rate due to its requirements. In this scenario, an unchangeable cgroups configuration cannot meet the goal of QoS. To do such a thing, we use a weighted schedule algorithm to make a balance between containers want short interactive time and containers want high data throughput.

Despite data on the local host, recent data centers usually use HDFS or something similar as their primary data store technique, which means the storing of data and the running of containers are separated. Under this condition, containers will not consider about where data is stored, it just concern about how to use the data. So, a complete solution should also improve the QoS of distributed file system like HDFS. In the future research, we will use the algorithm of dDiskQoS try to improve its performance.

Our evaluation shows that dDiskQoS substantially improves data center container data throughput and latency. We show that the latency of short-time containers has been shortened 20% in average and it has not obvious impact on the data throughput of containers with huge amounts of data I/O.

Contributions: In summary, dDiskQoS makes the following contributions: We (1) analyze the disk I/O under normal environment; (2) design a framework to do resource limiting; (3) propose an algorithm to meet different demands; (4) evaluate the resulting performance of several kind of applications on a testbed.

Related Works

The QoS of Disk I/O has been the subject of extensive research in many areas, including from application level, system level, and hardware level.

There are many prior researches attempted to improve the QoS of Disk I/O from different aspects, some in userspace and others in the kernel or a customized driver. Just like the paper [4], it does QoS on virtual disk while using traditional virtual machines and like the paper [5], which introduces a simple yet effective QoS classification built on top of the cloud environment due to an increasing number of business workloads are being migrated to the cloud. There is also some researcher use multi-level to do job scheduling like [8]. However, none of them can do QoS to a container level, for most of the papers focus on the performance of the whole system, which is not match our goal. Moreover, these tools are not very easy to deploy, some of them even need to do some modification in Linux kernel.

Proposed Approach

The control of QoS of Disk I/O for Docker container relies heavily on cgroups, both accounting and resources limiting. To do such a control of QoS, we should solve some problems, which is how to change the cgroups configuration of container dynamically and how to accounting precisely. Fortunately, cgroups has provided both feature, we just need to find a way how to use them. So the general approach can be described as below.

1. All the Docker container will firstly be assigned 1000 to its blkio-weight, which is the highest priority, to guarantee its I/O speed at its birth. The blkio-weight of Docker container can be set by Docker update –blkio-weight=$priotiry $container_id [2], or you can simply change the value in file/sys/fs/cgroup/blkio/docker/$container_id/blkio.weight [3]. For the Docker daemon will create a control group itself, it will not influence the other control group created by other before.
2. Then the dDiskQoS will keep an eye on the I/O for every Docker container and will accounting the I/O the container use. While the container use more than a threshold, the dDiskQoS will reduce its priority to the second level. When its usage beyond a threshold again, it will be reduced to a third level. So, the fourth level is similar.

3. To avoid the container in the lowest priority level be starved, the container in this level will be assigned with the highest level after some time to guarantee their QoS.

The idea can also be considered as a multi-level queue. The first queue has the highest priority, and second queue has a lower priority, the third queue has an even lower priority, and the fourth has the lowest priority. The capacity of each queue is increase, that is, the first queue has the smallest size and the fourth has the biggest. The multi-level queue has been widely used in many areas, like network scheduling [6] and system native block IO [7].

Implementation

The dDiskQoS can be divided into three parts, which is collection, calculation, proceeding.

Collection: This part is responsible for collecting information from Docker daemon and cgroups. For Docker, remote API can be invoked via HTTP request, which makes it easily integrated into dDiskQoS. The detail of HTTP requests can be seen in []. Since this feature, it is simple for dDiskQoS to get every container's PID, which is essential for the dDiskQoS to get information from cgroups, because all the information in cgroups are stored by PID. The luckiest thing is that one container usually has only one process running in it, so the dDiskQoS can easily keep track of every container by its PID.

Calculation: This part is used to consider which priority level a container should be put into. As described above, the whole system has four levels, so the calculation part will determine the priority of every container by its I/O usage. It will also consider when bring a container from the lowest level back to the highest level, which is also mentioned above.

Proceeding: This part is responsible for putting the decision calculation part made into effect. The step can be done via invoke Docker remote API or modify the cgroups configuration file directly. The second option can be used when the system is scaled to a distributed system, for the I/O operations may be separated with Docker daemon.

In our implementation, we use some specific value to be the threshold for every priority level, which is shown in Table 1.

Table 1. The threshold value for each priority level

Priority Level	**Threshold**
1	2 MB
2	8 MB
3	32 MB
4	128 MB

The initial value is quite easy, but it is enough to do improvements. In the future research, these values will be measured carefully to determine the optimal value for each priority level. Moreover, these values can be not stable, they can be dynamic, change all the time due the overload of I/O at that time.

Experiment and Performance Evaluation

All experiments will be run on two identical physical hosts whose configurations are shown in Table 2. We use Ubuntu 14.04 to be the experimental platform for its widely used over the world and its strong support for Docker.

For all experiments, we use an Ubuntu image, which can be download from Docker hub to be our base environment. All the containers will be assigned with different files with different size to read or write. We will compare with the original performance and the performance after the dDiskQoS applied to prove the correctness of our method.

Table 2. Experiment configuration

CPU	Intel i5-4590 (4 cores, 3.3 GHz)
Memory	8GB
NIC	Intel I217-LM 1Gbps
Drive	1TB HDD

Table 3. The time read operation takes in Experiment 1

Size	**Original**	**dDiskQoS**
1MB	3 ms	2 ms
10MB	33.1 ms	22.7 ms
100MB	2.85 s	2.25 s
1000MB	28.6 s	23.5 s

To make the simulation close to the real environment, all the experiments will use am extremely large file read operation as a background traffic. Based on the background traffic, we can see that how dDiskQoS effect the other I/O operations. For the I/O speed may be different in different due to many facts, so we repeated the tests specific times (e.g.: 50), and report the average value.

Table 3 presents the results in terms of reading files with different size with a background traffic always there, the original time and the time after applied dDiskQoS. The data in Table 3 shows that dDiskQoS can improve the time the read operation takes. The simple experiment shows the correctness of dDiskQoS, to look further at the impact of dDiskQoS on more realistic environment, we measured the performance of dDiskQoS with several read operations with different size simultaneously, and every operation may start at different time. The result of the experiment is shown in Table 4 and Table 5.

Table 4. File size and the time read operation started

File Size	**Start time**
2MB	2s
4MB	2s
16MB	3s
4MB	4s

Table 5. The time read operation takes in Experiment 1

Size	**Original**	**dDiskQoS**
2MB	6 ms	5 ms
64MB	1.9 s	1.7 s
16MB	50.2 ms	43.6 ms
4MB	12.4 ms	11.3 ms

From Table 5, the results show that even in a realistic environment, the dDiskQoS will still improve the reading performance. The data in Table 2 reveal that with the size of files getting larger, the difference between original and dDiskQoS will become small, but dDiskQoS takes a large effect on small size with a background traffic running. From this point of view, it can guarantee the performance of small size I/O operation even the system is busy with other large I/O operation, which is a big improvement.

Future research

The paper presents the method how to do QoS on a local Docker host with local storage. However, more and more network-based storage has been used in datacenter now, so it's a challenge how to do QoS in this situation. In our future research, we will try to apply the algorithm of dDiskQoS to these storage system, like HDFS, to improve the performance of I/O operation when the storage is separated from the local host.

As mentioned above, the value of each priority level may not be stable, they can be changed all the time due to the current overload of device. So, in our future research, these values should be measured carefully to determine if they are optimal values, if they are not good enough, we will try to find the optimal one. And we will try to find an algorithm to determine these values dynamically.

Summary

A generic mechanism based on cgroups to do QoS for containers was presented in this paper. dDiskQoS proves to be effective and efficient in guarantee I/O speed for small size I/O operations. The algorithm will make sure that the new operation will always get the highest priority at its beginning to reduce its latency if it is an interactive I/O operation. When the amount of I/O become large, dDiskQoS will reduce the priority of the operation to leave some space for the new operations that may occur at any time. The result is, almost all the I/O operations has been improved due to the average speed has been increased.

In current solution, dDiskQoS can only improve the performance on the local host with local storage, it cannot improve the performance when the storage separates from the local host. Thus, our future research will focus on this topic, that is, how to improve the I/O performance when Docker daemon runs on a host using a network-based storage. We will also try to integrate dDiskQoS into Docker to make it easier to be deployed, for the current version is a tool outside the Docker that should be deployed separately.

References

[1] Kratzke N. A lightweight virtualization cluster reference architecture derived from open source paas platforms[J]. Open J. Mob. Comput. Cloud Comput, 2014, 1: 17-30.

[2] Information on https://www.docker.com/

[3] Information on https://www.kernel.org/doc/Documentation/cgroup-v1/cgroups.txt

[4] Wu Y, Jia B, Qi Z. IO QoS: A New Disk I/O Scheduler Module with QoS Guarantee for Cloud Platform. InInformation Science and Engineering (ISISE), 2012 International Symposium on 2012 Dec 14 (pp. 441-444). IEEE.

[5] Silva M, Ryu KD, Da Silva D. VM performance isolation to support qos in cloud. InParallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International 2012 May 21 (pp. 1144-1151). IEEE.

[6] Azar Y, Richter Y. Management of multi-queue switches in QoS networks. Algorithmica. 2005 Sep 1;43(1-2):81-96.

[7] Bjørling M, Axboe J, Nellans D, Bonnet P. Linux block IO: introducing multi-queue SSD access on multi-core systems. InProceedings of the 6th international systems and storage conference 2013 Jun 30 (p. 22). ACM.

[8] Karthick AV, Ramaraj E, Subramanian RG. An efficient multi queue job scheduling for cloud computing. InComputing and Communication Technologies (WCCCT), 2014 World Congress on 2014 Feb 27 (pp. 164-166). IEEE.