Cgroup++: Enhancing I/O Resource Management of Linux Cgroup on NUMA Systems with NVMe SSDs

Junghi Min, Sungyong Ahn, Kwanghyun La, Wooseok Chang
SW R&D Center, Device Solutions, Samsung Electronics Co., Ltd.
Seoul, Korea
{junghi.min, sungyong.ahn, nala.la, wooseok_chang}@samsung.com

Jihong Kim
Dept. of CSE, Seoul National University,
Seoul, Korea
jihong@davinci.snu.ac.kr

ABSTRACT

For container-based virtualization such as Linux container (LXC), efficient and proportional resource sharing is an important design requirement. However, existing container resource management techniques do not adequately meet this requirement on modern server machines, especially NUMA machines with NVMe SSDs. In this paper, we propose an efficient proportional-share Linux Cgroup, called Cgroup++, for container-based virtualization. Unlike Cgroup, Cgroup++ takes into account of the storage asymmetry of modern NUMA machines in managing storage I/O requests. By exploiting the storage asymmetry in scheduling CPU cores for a given Cgroup instance, Cgroup++ improves the I/O performance of the Cgroup instance. Cgroup++ also supports proportional I/O sharing among multiple Cgroup instances using a weight-based throttling scheme in the I/O throttling layer for the NVMe SSDs.

Categories and Subject Descriptors

D.4.2 [**Storage Management**]: Allocation/deallocation strategies

General Terms

Management, Performance

Keywords

Linux container, Cgroup, proportional sharing, NUMA scheduling, storage asymmetry

1. INTRODUCTION

As container-based virtualization gets more widely employed on modern server systems such as NUMA machines with NVMe SSDs, however, managing host system resources fairly as well as efficiently among multiple containers becomes more difficult because of various asymmetric configurations of NUMA machines. Figure. 1 shows an overall organization of a storage-asymmetric NUMA machine, the Dell R920 machine, which was used for our experimental evaluations. As shown in Figure. 1, the Dell R920 is a storage-asymmetric NUMA machine because an NVMe SSD is directly attached to node 0 only. A remote NVMe SSD access from node 1 takes about 24% more time than a local storage access from node 0.

We assume that LXC is used for container-based virtualization in this paper. In LXC, which allows multiple Linux instances (called as containers) to run on a single host, each container is custom-configured by Linux Control group (Cgroup). Since multiple containers with different service requirements can compete for shared resources, proportional resource sharing is an important requirement for container-based virtualization. Furthermore, proportional I/O sharing becomes more important with recent high I/O bandwidth NVMe SSDs. However, the current Cgroup does not adequately handle the storage asymmetry problem on our target NUMA machine and does not support the proportional I/O sharing for NVMe SSDs.

In this paper, we propose an enhanced version of Cgroup, called Cgroup++, which overcomes the current limitations of Cgroup on storage asymmetric NUMA machines. Cgroup++ is different from the existing Cgroup in two aspects. First, Cgroup++ improves the CPU core allocation scheme of Cgroup by explicitly considering storage asymmetry of our target NUMA machines so that proportional storage sharing and high I/O bandwidth can be simultaneously achieved. Second, Cgroup++ supports proportional I/O sharing based on a weight-based throttling scheme to NVMe SSDs. In order for each container to share a proper portion of the total I/O bandwidth, our proposed scheme allocates a specific credit periodically to each Cgroup instance according to its I/O bandwidth requirement defined by an I/O weight and throttles I/O service for the Cgroup instance once the allocated credit for a period is completed consumed. Our experimental results show that Cgroup++ can improve the I/O bandwidth by up to 19% over Cgroup while proportionally distributing the requested I/O bandwidth share of each container.

2. LIMITATIONS OF LINUX CGROUP ON STORAGE-ASYMMETRIC MACHINES

The Cpuset subsystem of Cgroup is used to make a bundle of CPU cores, memory nodes, and tasks. For a given Cgroup instance, the current Cpuset bundling scheme, which is storage asymmetry-oblivious, simply reserves the required number of CPU cores. Although this simple scheme works reasonably well with CPU cores in UMA (Uniform Memory Access) systems with identical storage access times, it doesn't work well with a storage-asymmetric NUMA machine as shown in Figure. 2. In order for a task allocation scheme to work properly on storage-asymmetric NUMA machines, it must be able to avoid costly remote storage accesses when necessary. However, the default Linux task scheduler has no concept of storage asymmetry, thus making it very difficult to prevent costly remote accesses from occurring for

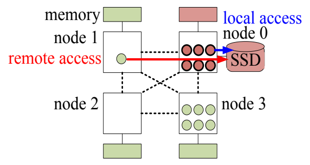


Figure 1. An example of storage-asymmetric systems.

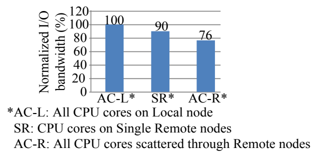


Figure 2. An impact of CPU core allocation schemes.

Middleware Posters and Demos '15, December 07-11 2015, Vancouver, BC, Canada
ACM 978-1-4503-3729-8/15/12.
http://dx.doi.org/10.1145/2830894.2830901

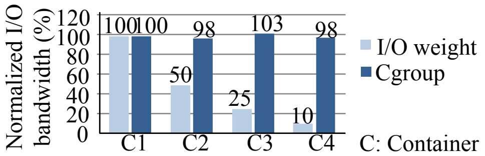


Figure 3. Disproportional I/O resource sharing of Cgroup.

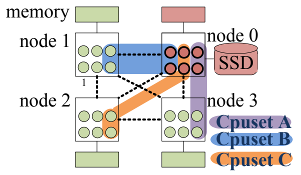
a given task or a given container.

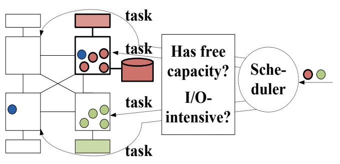
Another limitation of Linux Cgroup is related to NVMe interface, new high-performance storage interface for PCI Express (PCIe) based SSDs which provides the maximum 64K hardware queues. In order to support NVMe interface, the Linux kernel (since v3.19) has been modified as well to implement a multi-queue block layer over the existing single-queue block layer [4]. However, because Cgroup supported proportional I/O sharing inside the CFQ I/O scheduler [1] for the single-queue block layer only, no proportional I/O sharing is natively supported for NVMe SSDs in Linux. Figure. 3 shows that how current Cgroup handles proportional I/O sharing for NVMe SSDs among four containers. In this preliminary experiment, we set the I/O weights of four containers to 1000, 500, 250, and 100, respectively. However, as shown in Figure. 3, the current Cgroup does not satisfy the requested I/O proportionality for the containers.

3. PROPORTIONAL STORAGE SHARING SCHEME FOR NVME SSDs

Since CPU cores on the local node can access NVMe SSDs faster than CPU cores on the remote nodes, if CPU cores on the local node were assigned to a single container, it becomes difficult to support I/O proportionality in an efficient fashion. Therefore, when we create a container, we distribute CPU cores on the local nodes to different containers according to the requested number of CPU cores for each container. Figures 4(a) and 4(b) depict an example of distributing local node CPU cores for a container and the task assignment phase of the proposed task scheduler, respectively.

As explained in Sec. 2, since Linux Cgroup does not support proportional I/O sharing for NVMe SSDs, we added a weight-based throttling scheme to the Linux Cgroup I/O throttling layer. In order to support proportional storage sharing, we assign credits to cgroups based on their I/O weight and throttle I/O requests of the cgroup having no available credit until next replenish period. The credits of a cgroup indicate the total number of sectors that all the tasks in the cgroup can request within the specific period. The credit calculation and allocation mechanism is designed based in CFQ I/O scheduler. Because all cgroups in the system share the *TotalCredit* (the sum of credits assigned to all cgroup instances) according to their I/O weight, proportionality can be achieved. Moreover, work conserving is achieved by adjusting *TotalCredit* to storage bandwidth periodically. I/O service rate of each cgroup





(a) An example of distributing CPU cores of the local node (b) An example of CPU allocation under the proposed task scheduler

Figure 4. CPU allocation and task scheduling of Cgroup++.

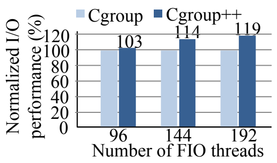


Figure 5. A comparison of I/O performance.

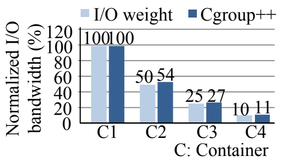


Figure 6. Proportional I/O sharing of Cgroup++.

is monitored and *TotalCredit* is recalculated to prevent the spare bandwidth of storage devices.

4. EXPERIMENTAL RESULTS

The proposed Cgroup++ was implemented in Linux kernel v4.0.4 and evaluated on a Dell R920 machine. The Dell R920 machine consists of four NUMA nodes, and each NUMA node has 24 x86 CPU cores. We configured the Dell R920 machine to have four Samsung XS1715 NVMe SSDs and all four SSDs are connected to the PCIe slots of a single node only. Our test environment setup reflects a typical NUMA machine [2]. FIO benchmark program is used as task applications.

We varied the number of FIO threads from 96 to 192 so that we can compare the efficiency of our scheduler over different I/O intensiveness. Each FIO thread performs 4-KB random reads for 10 minutes. As shown in Figure. 5, the proposed task scheduler of Cgroup++ increases the I/O bandwidth by 14% and 19% over Cgroup when the container used 144 FIO threads and 192 FIO threads, respectively.

In order to evaluate the effect of proportional I/O sharing of Cgroup++, Figure. 6 compares normalized I/O bandwidth results of four containers which has same configuration as Sec. 2. The result shows that weight-based throttling scheme of Cgroup++ allocates the portion of I/O bandwidth very accurately following the required proportional shares of four containers. Furthermore, we can observe that there is little wasted I/O bandwidth to support proportional sharing. It indicates Cgroup++ achieves work conserving property as well as proportional I/O sharing.

5. CONCLUSIONS

We have presented an improved version of Cgroup, called Cgroup++, which supports efficient proportional storage sharing for container-based virtualization on modern NUMA machines with NVMe SSDs. The key insight behind our proposed Cgroup++ is to make storage asymmetry of modern NUMA machines explicit, so that this storage asymmetry is taken account of managing host I/O resources for containers..

6. REFERENCES

[1] Axboe, J., Linux block IO - present and future, in Proc. of Ottawa Linux Symp, 2004.

[2] Intel Motherboard Hardware v2.0. http://www.opencompute.org/assets/download/Open-Compute-Project-Intel-Motherboard-v2.0.pdf.

[3] Lepers, B., Quéma, V., and Fedorova, A., Thread and Memory Placement on NUMA Systems: Asymmetry Matters, in Proc. of the USENIX Conference on USENIX Annual Technical Conference, 2012.

[4] Bjørling, M., Axboe, J., Nellans, D., and Bonnet, P., Linux block IO: introducing multi-queue SSD access on multi-core systems, in Proc. of the 6th International Systems and Storage Conference, 2013