Dynamic Block-level Cache Management for Cloud Computing Systems

Dulcardo Arteaga (PhD. Student), Douglas Otstott (MS. Student), Ming Zhao (Professor)
Florida International University, {darte003, dotst001, ming}@cs.fiu.edu

ABSTRACT

Block-level distributed storage systems (e.g., SAN, iSCSI) are commonly used in the emerging cloud computing systems to provide virtual machine (VM) storage. They allow fast VM migration across different hosts and improved VM availability leveraging typical fault-tolerance measures (e.g., RAID) available in such storage systems. However, as the size of cloud systems and the number of hosted VMs rapidly grow, the scalability of shared block-level storage systems becomes a serious issue. This paper proposes to address this issue by using client-side storage to implement block-level caching and exploit the data locality available in VM data accesses. By leveraging the capacity of fast storage devices such as SSD available on the VM hosts, this approach has the potential to substantially improve the performance of VMs and the load on the shared storage system. This approach is implemented upon dm-cache, a generic block-level caching utility. Our current prototype supports cache sharing across different VMs in order to maximize cache utilization.

1. INTRODUCTION

System virtualization is the key enabling technology of the emerging cloud computing systems. It enables flexible server consolidation and allows applications to be conveniently deployed along with their required execution environment through virtual machines (VM). Applications hosted on VMs can be relocated across different physical servers to improve performance, reduce resource usage, achieve better isolation, and save power consumption. To enable fast VM migration and improve data availability, cloud systems commonly employ storage area networks (SAN) or IP-based SAN (e.g., iSCSI [3], NBD [5]) to store VM images for a set of VM hosts. Such a shared storage system allows VMs to be quickly migrated across different hosts. It also provides improved availability of VMs by leveraging the commonly deployed fault-tolerance measures such as RAID.

However, as the size of cloud systems and the number of hosted VMs rapidly grow, the scalability of shared block-level storage systems becomes a serious issue. The performance of VMs can be hurt substantially when the storage system is overloaded. Meanwhile, the VMs can also adversely impact each other's performance when they compete for the shared storage system. This paper proposes to address the above scalability issue by leveraging the client-side storage to implement block-level caching and exploit data locality. The feasibility of this approach is supported by the continuing deceasing cost of storage and the availability of new, fast storage devices (e.g., SSD) _on the VM hosts. With local cache storage, each VM can improve the I/O performance by doing I/O locally to the cache device, whereas the load on the storage system can be reduced. This proposed approach is implemented upon dm-cache [6], a generic block-level caching utility. Our current prototype supports different VMs on the same host to share the cache device in order to maximally utilize the available cache capacity.

2. DESIGN

A shared block-level cache enables multiple co-hosted VMs to use the same cache device without causing data corruption. It can maximize the cache utilization as any VM can use any part of the available cache.

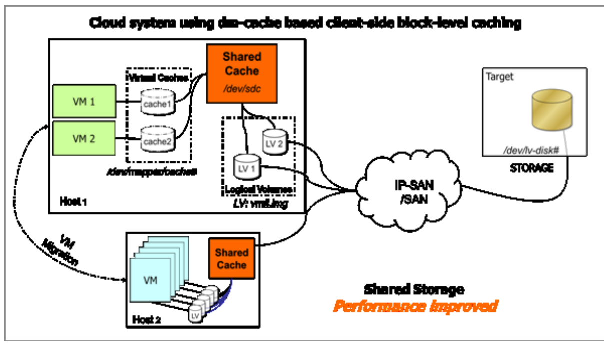

Figure 1. Architecture

Figure 1 illustrates an example of the architecture of our proposed approach. In this example we have two VMs each with its own virtual disk (vm1.img and vm2.img) which is stored directly on a logical volume (LV) remotely accessed through iSCSI/SAN. The local storage device (/dev/sdc) available on the VM host is used to provide block-level caching for the VM images. In order for the VMs to share the cache device, we create a virtual cache for each VM's virtual disk, which is only another level of mapping between the virtual disk and cache device. All virtual caches are in the end mapped to the same physical cache device. The per-VM virtual cache helps the shared-cache differentiate the IOs issued by different VMs.

The use of virtual caches helps the shared cache tag the blocks belong to different VMs, which is also necessary to handle VM migration. When a VM migrates across hosts, its data stored in the cache needs to be flushed. Tagged blocks allow the shared-cache to flush only the data of the migrating VM but not the entire cache.

3. IMPLEMENTATION

Our proposed approach is implemented upon dm-cache [6] a generic block-level disk cache for storage networking. It is built upon the Linux device mapper [2], a generic block device virtualization infrastructure. It can be transparently plugged into to the physical servers to provide cache for VM disks, which use LVs, connected to remote storage using storage area network (SAN) or iSCSI [3]. Current implementation of dm-cache can be modified to support cache sharing through multiple VMs.

In order to implement the shared-cache feature upon dm-cache, we need to map the I/Os issued to different VM images to the same shared cache device. To address this need, we will leverage device-mapper's [2] capabilities of mapping block-level I/Os between different block devices. It works by processing data passed in from a virtual block device, that it itself provides, and then passing the resultant data on to another block device. Using device-mapper implementation multiple devices can be mapped to a single one, leaving all the management to the driver, which has to deal with multiple I/Os arriving from different devices.

The core data structure of the generic block layer is a descriptor of an ongoing I/O block device operation called "bio". Each bio essentially includes an identifier for a disk storage area, the initial sector number and the number of sectors included in the storage area and the one or more segments describing the memory areas involved in the I/O operation. Dm-cache uses a structure called a "cacheblock" to express data relevant to a source block to cache block mapping. A cacheblock includes the address of the block in the source device, the address of the block in the cache device and the status of the block in the cache (valid, invalid, reserved, dirty or writeback). Dm-cache takes the bio structure field bi_sector and bi_bdev as inputs for mapping a block from source device to cache device. The first one identifies the sector number and second identifies the device. With these two parameters we can guarantee unique mappings for all sectors from different VM LVs.

Finally, in order to do mapping from source device to cache device, dm-cache uses radix tree to store the cache-blocks and an LRU linked list for block replacement. These algorithms allow fast cache look-ups and replacements and insertions as well as full cache utilization.

4. EVALUATION

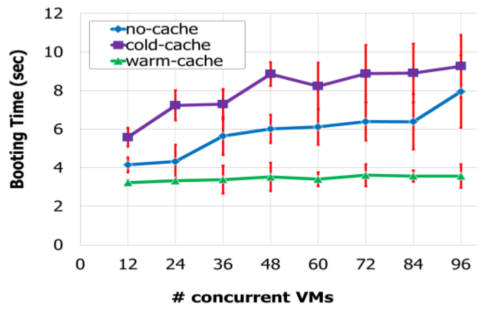

Figure 2. Concurrent Botting

Figure 2 shows the average boot time of multiple VMs at the same time, the number of VMs goes from 12 to 96 placed on 8 different physical hosts, all of them sharing the same network storage device. As we can see when the cache is warm we get the best response time that is up to 45% of performance improvement, but there is still room to improve when the cache is cold.

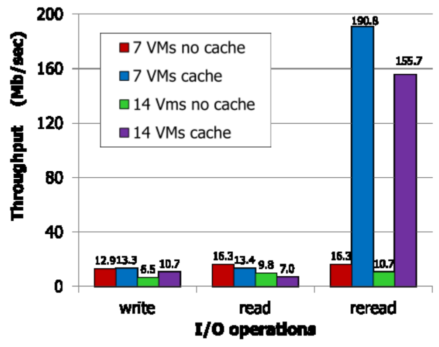

Figure 3. IOzone

Figure 3 shows the result of IOzone benchmark, we compare the throughput on 7 and 14 VMs running the benchmark concurrently, and all the VMs are sharing the same network storage device. As we can appreciate we get the higher throughput for reread operation since the IOs are hitting the cache, for the other operations we do not have a significant difference in throughput when hitting or not the cache.

5. CONCLUSIONS AND FUTURE WORK

We are evaluating our prototype implementation using typical benchmarks such as IOzone [4] and IOmetter [7] in a representative cloud setting using iSCSI-based storage system. In our future work we will consider more intelligent cache management algorithms that allocate the shared cache capacity among the VMs in a way that guarantees each VM's while maintaining good cache utilization. We will study how to tailor such algorithms according to the characteristics of SSDs. In addition, we will consider cooperative caching which allows multiple VM to share their caches, leading to better cache performance and utilization.

6. REFERENCES

[1] ATA over Ethernet, AoE protocol specification URL: http://support.coraid.com/documents/AoEr11.pdf

[2] Device-mapper URL: http://sources.redhat.com/dm/

[3] Internet Small Computer Systems Interface (iSCSI) RFC 3720 URL: http://tools.ietf.org/html/rfc3720

[4] IOzone File System benchmark URL: http://www.iozone.org

[5] Network Block Devic URL: http://nbd.sourceforge.net/

[6] Eric Van Hensbergen and Ming Zhao. Dynamic Policy Disk Caching for Storage Networking. URL: http://visa.cs.fiu.edu/ming/dmcache

[7] IOmetter benchmark URL: http://www.iometer.org