Trace Analysis for Block-level Caching in Cloud Computing Systems

Dulcardo Arteaga, Ming Zhao
{darte003, ming}@cs.fiu.edu
Florida International University

Pim Van Riezen, Lennard Zwart
{pi,lennard}@cloudvps.com
Cloud VPS

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, for fast VM migration and improved availability. 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. Client-side solid-state-based caching has the potential to improve the performance of cloud VM storage by employing solid-state drives (SSDs) available on the client-side of the storage system to exploit the locality inherent in VM IOs. However, there are several key questions to effectively use SSD caches in clouds. First, because of the limited capacity and high cost of SSDs, it is important to determine the proper size and configuration of the caches. Second, because of the diversity of cloud workloads, it is also critical to properly allocate the limited SSD cache capacity among concurrently hosted VMs. This project provides answers to these questions by studying hundreds of GBs of and months long block IO traces collected from real-world private (FIU) and public (CloudVPS) cloud systems.

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) to store VM images for a set of VM hosts. 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. In order to overcome this issue we use *dm-cache* [1], a block-level caching that enables client-side storage and exploit data locality.

The feasibility of using *dm-cache* 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.

However, there are several key questions that need to be answered in order to make effective use of SSD caches in cloud storage systems. First, *how to size the SSD caches?* Given the capacity and cost constraints of SSDs, there need to be enough locality in VM IOs in order to make SSD-based caching cost effective. Otherwise, cloud may not be a good target for SSD caching. Second, *how to configure the cache polices?* Policies on cache replacement and prefetching are important to the cache performance; the policy of how to handle writes in the cache has implications on not only write performance but also data durability. Third, *how to share the shared cache capacity among concurrent VMs?* A single SSD cache may be shared by up to a hundred VMs running on the same host, while the VM workloads vary in terms of locality, read/write ratio, sequentiality, and burstiness. The lack of understanding of all these characteristics may lead to a cache sharing policy that unfairly treat the competing workloads and underutilize the cache resources. This project try to provide answers to the above questions by studying hundreds of GBs of block IO traces collected from the Florida International University private cloud and the Cloud VPS public cloud systems.

The rest of this work is organized as follow: Section 2 describe the trace analysis. Section 3 present the working set size analysis. Section 4 shows the study on proportional cache allocation. Finally we present our future work.

2 Block-IO Trace Analysis

We studied traces from two different locations, one collected at *cloudVPS* (`http://www.cloudvps.com/`) and the other collected at our servers here at *FIU* (web, moodle, and file servers). The traces are divided into sets, and summarized in Tables 1 and 2.

For every set of traces we conducted a thorough analysis to get the following data:

- The amount of read and write operations
- The amount of hits, using different cache policies (write-back, write-through, write-allocate)
- The amount of IO operations per second (IOPS)

Server Name	Time (days)	Size (GB)
webserver	79	18.2
moodle	54	150
bear	64	62

Table 1: FIU traces

Host Name	Number of VMs	Size (GB)
rhine	90	7
seine	18	3.3
heinen	62	4.5

Table 2: CloudVPS traces

- The amount of sequential reads and writes

Figure 1 shows a summary of the analysis for FIU webserver traces. This analysis determines if the traces is read or write intensive, and the hit ratio using different policies (write-back, write-through, write-allocate). The analysis shows that FIU-webserver is write intensive and the best cache policy to use is write-back.

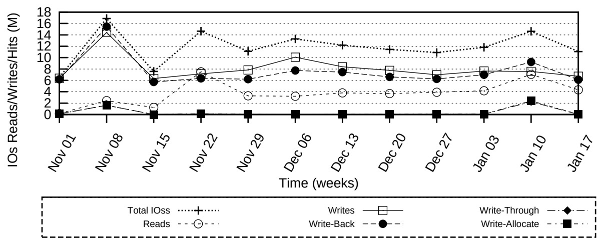


Figure 1: FIU webserver trace analysis

3 Working Set Size Analysis

For the Working Set Size (WSS) analysis we performed evaluation using different windows size "delta", which can be based on the number of accesses or based on time elapsed. The analysis includes:

- WSS analysis with delta as "number of accesses"and "number of seconds"
- Predict next WSS using double exponential smoothing mechanism
- Evaluate the accuracy of each predicted WSS

This analysis will give an understanding of VM's cache needs, and help us determine how to better collocate multiple VMs when all of them are sharing the same cache device. Figure 2 shows the WSS calculated using a *delta* specified based on the number of accesses, with a *delta* of 1024. Here we can appreciate certain stability of WSS throughout the entire three month of the trace and an evident change of WSS occurred in the middle of the trace.

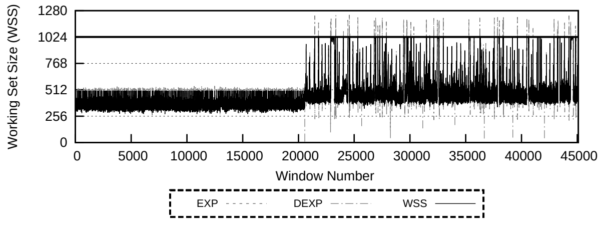
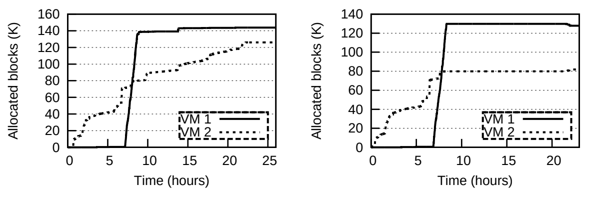


Figure 2: WSS-analysis using (delta accesses: 1024)



(a) Static *(b)* Dynamic

Figure 3: Cache Allocation Policies

4 Proportional Cache Allocation

With the understanding of VMs' cache needs, we further investigate how to enforce the cache allocation among the VMs sharing the same SSD cache. Cache allocation can be enforced statically for competing VMs' workloads; however, due the dynamism inherent in the workloads, the WSSes change over time, so the allocation may also need to be updated dynamically.

4.1 Static Allocation

This approach enforces logical partitioning at replacement time: if the cache is not full, a VM can use empty slots in the cache beyond its allocated share; when it is full, a VM that has not used up its share takes its space back by replacing a block from another VM that has borrowed its share. Figure 3a shows the results of replay day long traces using static allocation.

4.2 Dynamic Allocation

We enhanced *dm-cache* to analyze workloads online. This *wss-anlyzer* keeps track the information (block address and time) of received IOs, and calculates WSSes for each workload online. Based on the observed WSSes, it also predict the WSSes of the workloads for the next window. Using the predicted WSSes of all the workloads, *dm-cache* reallocates the cache among the competing VMs and enforce the allocation at replacement time. Figure 3b shows the trace replay results for dynamic allocation.

5 Conclusions and Future Work

The analysis performed on the traces definitively gives us a better understanding of block-level cache for cloud computing systems on its requirements and implications, but also raise more questions. For example, how to utilize the sequentiality in the workloads? How to combine access-based and time-based WSS analysis? These questions will be addressed in our future work.

References

[1] Dynamic block-level storage caching for cloud computing systems. `http://visa.cs.fiu.edu/dm-cache`.