VirtCache: Managing Virtual Disk Performance Variation In Distributed File Systems For The Cloud

Rajesh Vellore Arumugam *, Quanqing Xu*, Haixiang Shi* and Qingchao Cai† Yonggang Wen†

*Data Storage Institute (DCT Division), Singapore
Email: {Rajesh_VA,Xu_Quanqing,SHI_Haixiang}@dsi.a-star.edu.sg

†School of Computer Engineering
Nanyang Technological University, Singapore
Email: {QCCai,YGWEN}@ntu.edu.sg

***Abstract*—As Applications are moved from physical servers to virtual machines sharing storage resources, they experience large variation in I/O latencies. While maintaining average performance in such virtualized environments is important to conform to service level agreements (SLA), cloud users also expect their applications to have minimum variation in tail end latencies like 90th percentile latency for predictable performance. This becomes a challenging problem as the deviation in the application's 90th percentile I/O latency from average latency under storage resource sharing (VM consolidation) can be very high. We show through experiments under VM consolidation that during peak loads this latency variation from average can be as much as 5 times compared to when the application has exclusive access to the storage devices. This variation in performance exists for both Hard drives (HDD) and Solid state drives (SSD). To minimize this large latency variation, we propose a dynamic I/O redirection and caching mechanism called VirtCache. VirtCache can pro-actively detect storage device contention at the storage server and temporarily redirect the peaking virtual disk workload to a dynamically instantiated distributed read-write cache. We have implemented our system in GlusterFS, a commonly used distributed file system deployed as a backing store in the cloud. Our system can achieve from 50% to 83% reduction in the 90th percentile latency deviation from average compared to previous work as we move from low load conditions to peak non uniform consolidated VM workloads. With our VirtCache system, Cloud providers can guarantee predictable performance for the cloud users as if their application has exclusive access to the storage resources.**

***Index Terms*—Storage, Distributed file system, Service level objective, Distributed Cache, Virtualization.**

I. Introduction

Virtualization being the main enabler for the cloud, abstracts physical resources like CPU, memory, network and storage to the cloud users so that they are pooled, shared and optimally utilized. While pooling and sharing of resources is one of the major driving model of the cloud, it also introduces several challenges. One of the major challenge deals with conforming to varied service level objectives (SLO) of different applications sharing the cloud resources. Specifically SLO in the context of VMs sharing storage resources (VM consolidation) is more challenging compared to other resources like CPU and memory [1]. This is because there is a high degree of performance variation like latencies in VM consolidation compared to when the VM workloads were run in isolation. There are two main reasons for this variation in latencies. The first reason being the highly unpredictable random characteristics of VM disk I/O workloads [2] leading to heavy seeks. The second reason is the high variability in I/O intensity from interfering workloads competing for the I/O queues at the storage devices. In addition to the IOPS (I/O per second) expected by the application, Cloud users also desire to have minimum variation in application latencies like 90th percentile or tail end latencies. Complying with this becomes challenging under VM consolidation, since storage is usually provisioned only for meeting non peak workloads from each VM. Under increasing I/O activity or peak workloads from any of the VM, the system may not be able to meet the 90th percentile latency even if it can minimize the deviation in the average latency. It has been shown that reducing performance variability like tail end latencies is more critical than just providing average performance guarantees. In this paper we are concerned with reducing the variation of the 90th percentile I/O latency from average latency during peak loads from any of the VM under a consolidated scenario. Reducing tail end latencies will have significant impact on the cloud user perceived application response times.

Several techniques for managing performance in shared storage and virtualized environments have been studied and explored before. These techniques include performance isolation [3], proportional sharing of storage resources [1], caching [4], proportional allocation of SSD resources [5], VM migration [6] and dynamic instantiation of host side virtual cache appliances (VCA) [7]. Even though SSDs are being extensively used as a cache in some of these work for their high random I/O performance, they too have high variability in performance especially for writes. Many of these papers deal with meeting SLO (Service level objectives) goals like IOPS (I/O per second) and average I/O latencies under VM consolidation. But little work has been done to address the problem of high variance in performance (like the 90th percentile latency variation from average) as applications are moved from non-consolidated environments to virtualized

environments especially during periods of peak loads.

To address the problem of large I/O latency variation, we propose a dynamic I/O redirection and caching mechanism called VirtCache. VirtCache can pro-actively detect storage device contention at the storage server and temporarily redirect the peaking virtual disk workload to a dynamically instantiated distributed read-write cache. We have implemented this system on top of the GlusterFS file system [8] which is widely used as a backing store in OpenStack [9]. We show that by dynamically redirecting workloads from contended storage devices to server caches across the cluster we could reduce the variance in 90th percentile latency under peak workloads. While the caches are always there to be utilized across the server cluster, the peaking I/O load is not uniformly distributed. We address this by redirecting the concentrated load from virtual disk files on specific storage servers to other server's less utilized caches. The following are our main contributions in this paper:

1) A distributed storage side caching mechanism called VirtCache to minimize I/O latency variation during peak VM workloads
2) Mechanism to Pro-actively detect storage device contention, I/O tracking and redirection to the distributed cache
3) Implementation of VirtCache in a popular distributed file system (GlusterFS) and detailed evaluation showing the benefits.

The rest of the paper is organized as follows. In section II we briefly describe the motivation behind our work that explains the source of the latency variation. This is followed by the architecture of our system explained in section III. In section IV we describe our detail experimental results and validation of our assumptions on the impact of I/O queueing on the performance variation. In section V we describe the related work on performance isolation and I/O offloading in virtual environments. Finally we conclude in section VI with areas of further improvements in our implementation and future research work.

II. Motivation

The major cause of storage latency variance in virtualized environments is due to the I/O interference among the workloads sharing the storage system. This interference effect has been shown to exist both for HDD as well as SSDs [10]. For non-sequential workloads in HDDs, the latency contribution comes from both the seeks and during heavy loads due to I/O request queuing at the device [11]. For SSD, the variance in the latency mainly occurs when writes interferes with reads. Under write intensive periods of an I/O workload, this variance in latency is dominated by the program-erase cycles and garbage collection (GC) activity. For example when a program/erase cycle is running in the SSD, subsequent read I/Os had to wait till it completes [12]. To understand the effects of queueing on performance variation we experimented with virtual machines with consolidated I/O workloads on GlusterFS virtual disks as back end storage. Due to space constraints we summarize our results as follows. Under consolidated workloads (we experimented with consolidation of multiple server I/O workloads generated with FIO [13] and filebench [14]), the deviation of the 90th percentile latency from non-consolidated case increased from 4 to 15 times for the HDD case. For SSD this variation was around 2 to 5 times from the non-consolidated case. We also measured the I/O queue sizes at the Gluster servers. We found that the large deviation in the 90th percentile latency was attributed to the high variation in the I/O queue sizes at the storage servers during consolidated peak loads. To address this problem we have developed VirtCache. We show through our experiments in section IV, that VirtCache can maintain the variation in the 90th percentile latency at peak loads almost same as that for the non peak scenario.

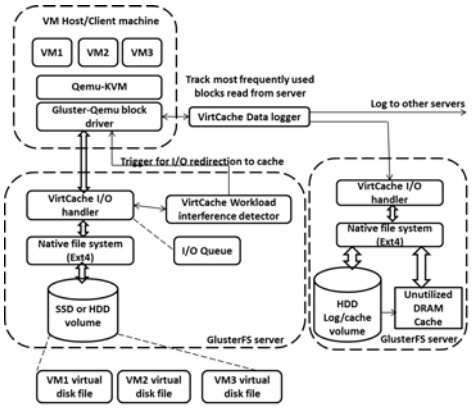


Fig. 1: GlusterFS architecture with VirtCache components

III. VirtCache Architecture

The modified GlusterFS stack with VirtCache is shown in the Fig. 1. We modified the IO-threads translator in GlusterFS to incorporate our caching mechanisms. The VirtCache system consists of the VirtCache I/O handler, Workload interference detector and data logger. We also modified the Qemu-Gluster block driver to incorporate the I/O redirection and dynamic allocation of memory space in the cluster. The following sections describe in detail each of the various components in VirtCache.

A. Workload Interference detection

The interference across VM workload can be considered as contention for the device I/O queues. The papers on storage performance models in [15] and [11] use queuing theory based models to predict latency and throughput based on the outstanding I/O at the device. The work done in [15] analyses contention of competing VM workloads on the same storage device. It models the VM consolidated performance as a function of arrival rates and response times when the workloads were run in isolation. The paper on storage performance management in [11] uses Little's law to model the storage latency as a function of outstanding I/O at the storage device. But these models require that characteristics of the device and

workload to be profiled before the models can be used to predict performance like latencies and throughput. In our case we need a model that can be used at run time to pro-actively detect interference and contention among VM workloads. We borrow the product form queueing model studied in [16]. We use this to predict the latency expected from the device by simulating arrivals of I/O requests from the VM based on the observed arrival rate on the storage server. The arrival rate to compute the latency from the model is set at around 10% higher than the observed to detect the contention earlier. The response time or latency R_i for the ith VM at the storage device is given by:

$$R_i = d_i/(1 - \sum[\lambda_i/n] \times d_l)$$

where d_i is the device response time for the ith VM, λ_i is the arrival rate for the ith VM. n is the number of parallel I/O that can be issued to the storage device. For both the RAID5 array and SSD we set this to be 16. We observed that for the SSD we used the latency increased after we reach a queue depth of around. 16. This is because even though the number of outstanding I/O can be increased beyond this value (to get higher throughput or IOPS) the latency for individual I/O still increases. Therefore we keep this value at 16. The device latency d_l is the average latency without any queuing or outstanding I/O at the device. The values d_i and d_l can be measured during non-peak workloads. The 10% higher arrival rate provides enough headroom for the I/O redirection to be initiated before the actual performance deviation occurs. In the beginning, VirtCache obtains the average latency and 90th percentile latency for each VM from the current observed value during low I/O loads. Low I/O loads are identified as the period when the average I/O queue depth is below a configured threshold (we chose 4).

VirtCache constantly monitors the observed and predicted values of response times for every time window for each VM over small time intervals. From these measurements it obtains the average latency and the 90th percentile latency from both the observed and the values obtained through the model. When there is a deviation in the difference between the average latency and the 90th percentile latency from the low load case, it triggers the I/O redirection. Since the control simulation is done at a higher arrival rate than the observed rate, the performance variation is detected in advance to provide a small headroom. Once VirtCache detects the performance variation from low load case, it selects a VM for I/O offloading based on two parameters. The first parameter is the maximum latency deviation from the low load case. The second parameter is the cacheability requirement. VirtCache chooses the VM which has the maximum latency deviation and the maximum cache hit for a given cache size. For VMs with same cache hits the VM with higher latency deviation is chosen. For this, a hit rate curve is maintained by the VirtCache I/O handler. We do not explore on how to obtain the hit rate curve here as this is beyond the scope of this paper. Once the load in the system goes back to normal, the I/O is redirected back from the cache to the primary VM disks. For this we monitor the I/O rate at the block driver on the host machine. The pseudo code of the I/O detection of performance variation is shown in algorithm 1

Algorithm 1: Algorithm to detect 90th percentile latency violation.

```
1  foreach ith VM do
2      foreach timePeriod do
3          get λ_i^o;
4          λ_i^s = λ_i^o + 0.1 × λ_i^o;
5          R_i = d_i / Σ[λ_i^s/n] × d_l;
6      end
7      Get R_i^m from all R_i collected in timePeriod ;
8      if (R_i^m − R_i^l) > threshold;
9      Trigger VM selection and I/O redirection;
10 end
```

Here λ_i^o and λ_i^s are the observed and simulated values of the I/O arrival rates for the ith VM workload. The values R_i^m and R_i^l represent the 90th percentile latencies during the current observation period and the low load period respectively. The value $threshold$ is set at 5% of R_i^l. Lines 2 to 7 obtains the 90th percentile latency R_i^m for each ith VM workload through the simulation for each $timePeriod$. Line 8 checks if this exceeds the value $threshold$. If the threshold exceeds then the VM selection and IO redirection is triggered in line 9. The next section explains how VirtCache tracks block regions on the VM disk files for caching.

B. VirtCache I/O Handler

The VirtCache I/O handler module is a modified version of the IO-threads translator in GlusterFS. Gluster uses the name "translator" since in addition to normal file I/O it also implements other operations on files (like stat, getattr etc). The VirtCache I/O handler intercepts all read/write I/O to the back end storage device. It can identify the different VM disk files based on the global file id. The VirtCache I/O handler also tracks and provides block level I/O statistics for the VM disk files to the Gluster client. Note that since all translators in Gluster work at the file level, the VirtCache I/O handler provides a translation of file offsets to block identifiers in the logical VM disk file. The Gluster client uses the block level statistics collected by the VirtCache I/O handler to log frequently accessed data. This is done as follows. For every configured time window (typically 10 minutes), the handler records the blocks that were accessed from the client in a LFU (Least frequently used) cache. The handler only stores the block identifiers and not the actual data. At the end of each configured time interval, the handler constructs a Bloom filter [17] of the top N frequently accessed items in the LFU (Least frequently used) cache that can fit inside the given cache size ($16GB$). The Bloom filter provides a compact and efficient mechanism for recording the I/O block access. Instead of passing the entire LFU cache meta-data, the I/O handler

passes the Bloom filter to the Gluster client (host machine) over another extended attribute set on the VM disk file. This extended attribute is periodically read by the client. This period is same as used by the server to refresh the Bloom filter. Equipped with the last N block access record at the storage server the client performs the data logging on the log volume. This is done by the data logger on the client whose function is explained in the next section.

C. Data Logger

We modified the Qemu-KVM block driver to implement the data logging. The data logger uses the Bloom filter provided by the VirtCache I/O handler through a extended attribute on the VM disk file. Whenever the block driver sees a block id in the response from the server matching a id in the Bloom filter, it writes this block on the log volume available in the cluster. Since this set represents the frequently used blocks, it is likely that this will be accessed when the I/O to the primary volume where the VM disk file resides is offloaded to the other servers containing the dynamic cache. When the I/O to the VM disk is offloaded, this set of blocks can be read back from the log volume to warm up the distributed caches quickly. Since this set is refreshed periodically by the VirtCache I/O handler, it also represents the recently used frequently accessed blocks.

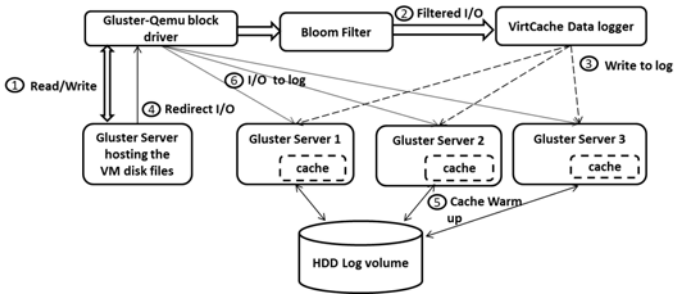


Fig. 2: High level control flow between components in VirtCache. The bloom filter is used to filter I/O that needs caching. Control flow numbered 1 to 3 represent activity before the I/O redirection. Control flow numbered from 4 to 6 represent activity during I/O redirection to the distributed cache

We assume that the size of the file system cache inside the VM itself is not sufficient to hold this entire set of blocks. Our assumption is valid since this is the reason there was an I/O to the server in the first place. By only logging the most recently and frequently used data set we can reduce the space required on the log volume to store this data and as well as reduce the I/O to the caching servers. During the I/O redirection period, this logged data is read from the servers if there is a cache hit to the Bloom filter. For false positives (which are rare)the I/O is retried on the primary server having the VM disk image. The schematic in Fig. 2 shows the overall control flow between the different components on the client and server side during normal I/O and I/O redirection. Note that the I/O offloading is for both read and write. The offloaded writes are persisted on the distributed log volume and later moved to the primary VM disk file when the normal load returns in the system.

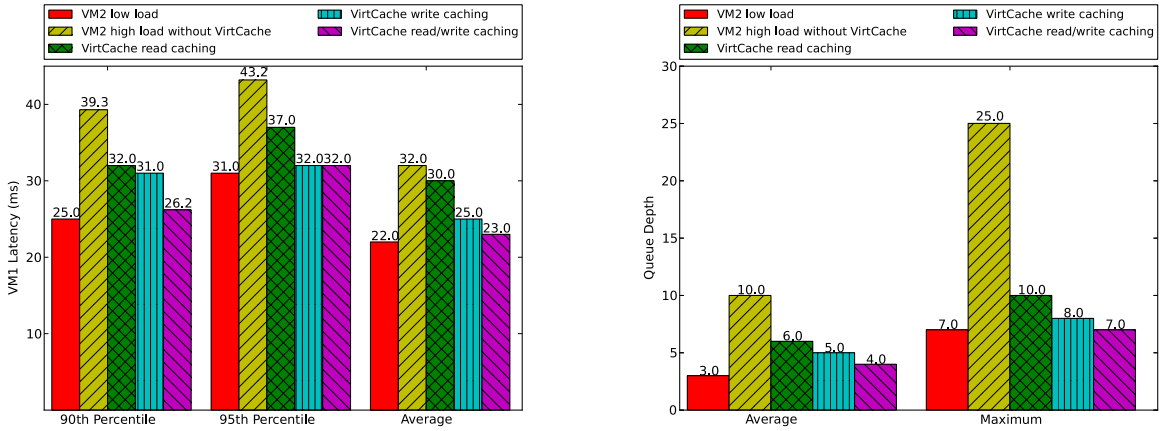
IV. Evaluation Results

The Evaluation Gluster storage servers used was a Xeon Server with 4GB RAM configured with RAID5 with 4 SATA drives (500GB capacity) for HDD storage. We used two real workload traces. The first one was a Microsoft exchange trace collected in our Data center with Loadgen software. We emulated an environment with 1000 user mail boxes with 100MB each with an average of around 500 emails sent/received per day from each user. The other workload was the TPCC trace downloaded from SNIA trace repository [18]. The TPCC trace were publicly available trace captured at Microsoft Research on their enterprise servers. The Exchange workload was around 30% read and 70% write, whereas the TPCC workload was around 66% read and 33% write. Both have a cache hit rate of 60% and 33% respectively for a 8GB cache. The VM2 workload was used to simulate a low load and high load scenario. At low load the VM2 fio workload was kept at 200 IOPS and for high load was kept at 400 IOPS (close to maximum IOPS for 4 drives). Similarly for SSD, we used 1000 IOPS and 2000 IOPS as low load and high load cases respectively. The distributed cache size in all our experiments were fixed at 16GB. The size of the log volume was also set to this value for each VM.

The Legend in all the graphs indicates scenarios with and without VirtCache (low load and high load). As shown in the Legend, VirtCache Read caching refers to caching and I/O redirection only for reads, VirtCache write caching refers to I/O redirection only for writes similar to the Everest system [19], VirtCache read/write caching refers to I/O redirection and caching for both reads and writes. The vertical axis refers to the VM1(VM2) latency/queue depth against the different scenarios indicated in the Legend.

A. Exchange Workload on HDD

The set of Fig. 3 show the 90th percentile, 95th percentile and Average latencies for the cases of VM2 low load, VM2 high load and VM1 with Exchange workload without VirtCache, consolidation of VM1 and VM2 high load with read only caching, write caching and read/write caching. We also include the 95th percentile latency in the results. Since in some of the cases VirtCache can also reduce the variation in the 95th percentile from the average. From the graph we can see a huge variation in the 90th percentile latency as we move from the low load scenario to high load one. The performance variation is around 80% in the high load case (VM1 and VM2) compared to that under low load. With read caching alone VirtCache can reduce this variation to below the low load case (2ms compared to 3ms). But the average latency reduction is still small compared to the case without VirtCache. This is because this particular workload is more write intensive and could not provide significant improvement with a read cache alone. The write only case can absorb most of the writes in the distributed cache and therefore provides a



(a) VM1 (Exchange workload) disk latency as VM2 (FIO) workload is varied from low load to peak load)

(b) Queue depth at the storage server as VM2 (FIO) workload is varied

Fig. 3: VirtCache caching for Exchange workload on HDD. VM1 runs replayed trace of an Exchange workload. VM2 runs FIO with IOPS varied from 200 (low load) to 400 (peak load).

significant reduction in average latency. But the 90th and 95th percentile latencies are still large compared to the average. The read/write cache can almost bring the average latency to the low load case. It can also bring the 90th percentile latency within 7% of the low load case. This is because most of the reads and writes are offloaded from the VM disk file to the distributed cache servers. A fraction of the load with cache misses still has to be served from the VM disk file. Since VM2 generates a significant rate of I/O, the average latency of VM1 is slightly higher than that of the low load case. VirtCache can also bring the 95th percentile latency to levels same as the low load scenario. We can also see that the read/write caching can achieve around 50% reduction in variation of the 90th percentile latency from average compared to the write offloading/caching alone. Compared to the case without VirtCache during peak workload the latency variation is reduced to around 56%.

B. TPCC workload on HDD

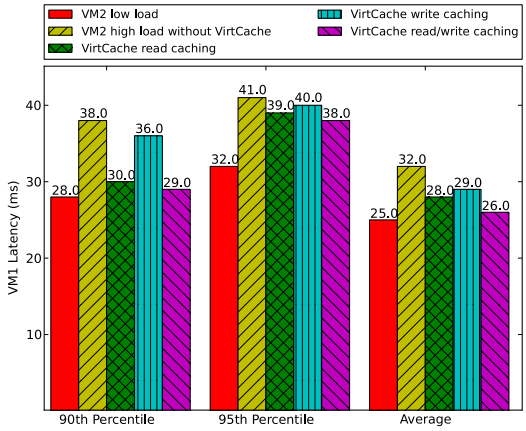
For the TPCC workload, the performance variation for the peak conditions without VirtCache increases by twice the amount in the low load case. We can also see from Fig. 4, that the maximum I/O queue size increased to around 15 that is twice to that during low loads. Since this workload has more reads than writes we get a significant improvement in both the average latency and the 90th percentile latency variation. The 90th percentile latency variation from average in the high load scenario is in fact lesser than the low load case. With just the VirtCache write caching both the average latency and the 90th percentile latency variation from the average is higher than the low load case. With VirtCache read/write caching, we can bring down the average latency levels close to the low load case. The variation in the 90th percentile latency is the same as the low load case with VirtCache. The variation in this case is around 50% smaller compared to that without VirtCache during peak loads. The 95th percentile latency of this workload is higher than for that of Exchange workload. This is due to the lower cache hit rate of this workload compared to the Exchange workload.

C. Exchange workload on SSD

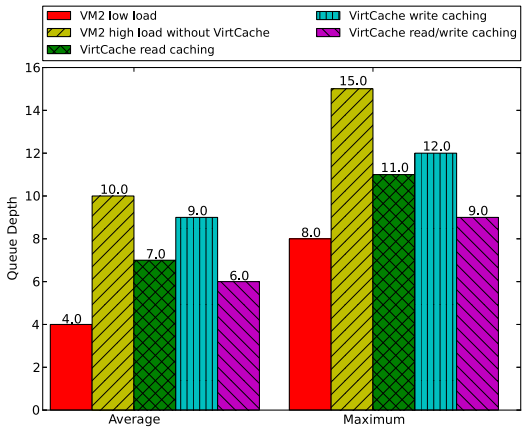
We also tested with Exchange workload on VM disk files on the SSD and the results are shown in Fig. 5. We could not test for the TPCC case since the block access pattern for this workload spans more than 400GB which could not be tested on a single SSD drive. For the exchange workload, even for SSD we can still see significant improvement in both the average latency and the 90th percentile latency variation compared to the average. Under VirtCache read/write caching both the 90th and 95th percentile latency were close to the low load case since we can get a very high hit rate for this workload. The variation of the 90th percentile latency was also close to that of the low load case. The VirtCache read/write caching for even SSD volumes can do better by around 65% compared to the write offloading alone.

D. Consolidation of Exchange workload with Webserver workload on HDD

We also experimented with consolidation of the Exchange workload (VM1) with a FIO webserver workload (VM2) with different IOPS rate.The low load case was simulated with 200 IOPS as before and the high load was replayed with 400 IOPS. VirtCache chooses the Webserver workload to be offloaded since it has a higher cacheability with only 4GB required for the cache to get around 60% hit rate. As can be seen from the graph in Fig. 6b for Exchange workload the 90th percentile latency variation from average is around 3.5 times at high load compared to the low load case. This variation is reduced to less than the low load case with VirtCache read/write caching. We can also see that the VirtCache read/write caching can

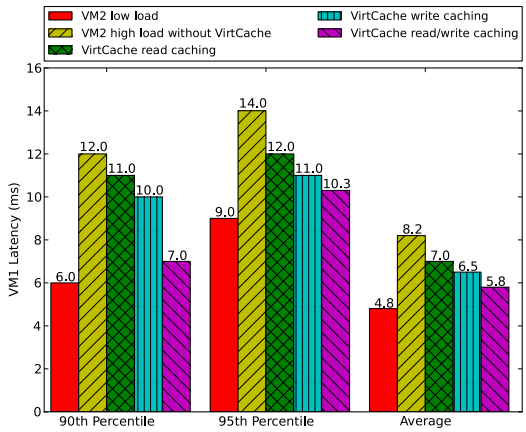


(a) VM1 (TPCC workload) disk latency as VM2 (FIO) workload is varied from low load to peak load

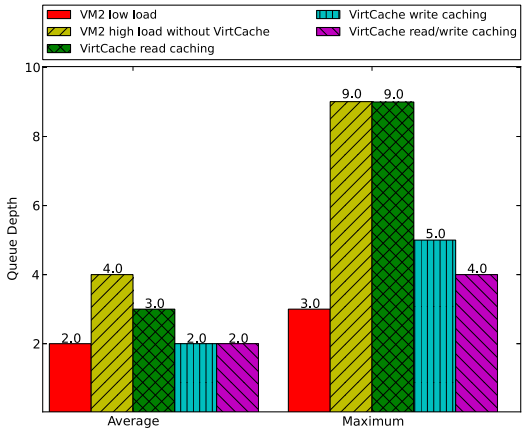


(b) Queue depth at the storage server as VM2 (FIO) workload is varied

Fig. 4: VirtCache caching for TPCC workload on HDD. VM1 runs replayed trace of TPCC workload. VM2 runs FIO with IOPS varied from 200 (low load) to 400 (peak load).



(a) VM1 (Exchange) disk latency as VM2 (FIO) workload is varied from low load to peak load



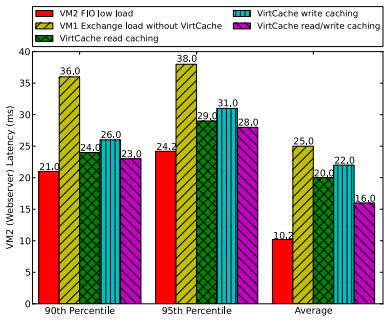
(b) Queue depth at the storage server as VM2 (FIO) workload is varied

Fig. 5: VirtCache caching for Exchange workload on SSD. VM1 runs replayed trace of an Exchange workload. VM2 runs FIO with IOPS varied from 1000 (low load) to 2000 (peak load).

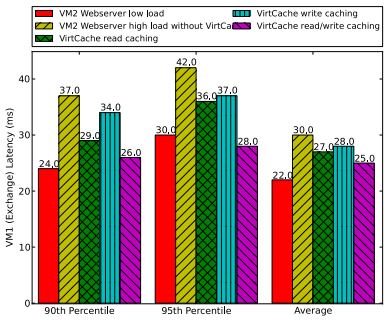
reduce the 90th percentile variation by around $1/6$th (from 6ms to 1ms) compared to write offloading alone (around 83% reduction). We also compare the effect of the caching for the Webserver workload. For this we compare the latencies from our previous experiment with a Webserver VM consolidated with an FIO lower load. In the Fig. 6a this is shown as "VM2 FIO low load". The "VM1 Exchange load without VirtCache" shows the latency values for the Webserver workload in consolidation with Exchange (VM1). The 90th and 95th percentile latency values are as high as that of Webserver since the queue depths are common for both workloads at the device (see Fig. 6c). Due to the redirection of the I/O to the cache we can see a large reduction in the average latency under VirtCache read/write caching. The 90th percentile latency variation (from average) is also reduced to below that of the low load case.

E. Effect of number of VMs

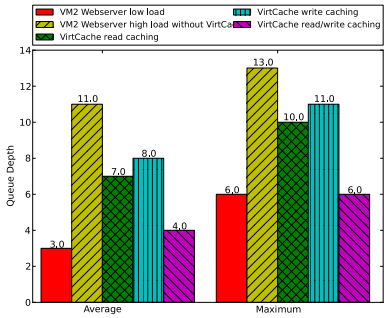
As another experimental scenario to test the scalability of VirtCache, we varied the number of VM instances from two servers (hosts) and measured the latency variation as the IOPS is increased from each VM. We replayed a synthetic workload from these VMs and measured the deviation of the 90th percentile latency of the workloads from the non-consolidated case. The Fig. 7 shows the deviation in percentage from the non-consolidated case as the number of VMs are increased. As a reasonable value for caching, we generated around 30 - 60 percentage hit rate in the synthetic traces. The virtual disk files for the workloads were created on a single Gluster server with SSD. The number of Gluster servers were fixed at four each



(a) VM2 (Webserver) disk latency in consolidation with Exchange (VM1) with VM2 chosen for caching



(b) VM1 (Exchange) disk latency in consolidation with Webserver (VM2) with VM2 chosen for caching



(c) Queue depth for VM1 (Exchange) and VM2 (Webserver) consolidation

Fig. 6: VirtCache caching for VM1 (Exchange) and VM2 (Webserver) on HDD. The Exchange and Webserver workloads are run on VM1 and VM2 respectively. VirtCache chooses the Web server workload to be offloaded to the distributed cache.

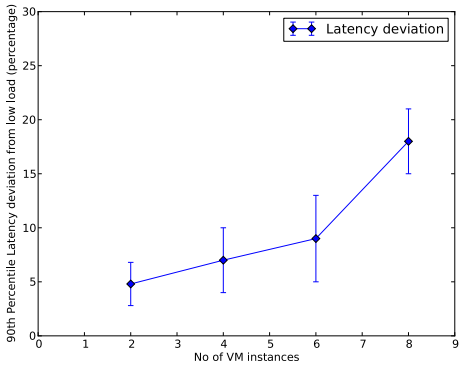


Fig. 7: Latency deviation from non-consolidated case as the number of VM instances are increased. The number of Gluster storage node is fixed at 4. Each VM workload exhibits a cache hit rate of 30 to 60 percentage from a synthetic workload replayed with FIO with IOPS of 300

with 4GB RAM (same configuration as used in the previous experiments). As shown in the figure as the number VMs are increased from 2 to 8 the deviation increases from 5 to 20 percent from the low load case. Some of the workloads I/O were moved to the less utilized server's caches. This reduced the contention on the virtual disk file hosting server, thereby reducing the I/O queuing at the storage device.

V. Related Work

Interference among workloads can be reduced by isolation techniques like proportional allocation and sharing of storage resources as described in [1]. To achieve fair sharing of the storage resource a mechanism to control the I/O queues at the storage system was proposed in this paper. But this technique can only provide proportional sharing of the available IOPS among workloads. It also does not guarantee absolute latency like the 90th percentile latencies. Moreover, throttling I/O as described in the paper can only reduce the I/O latency at the cost of sacrificing the IOPS proportionally across the VMs. Performance isolation techniques like Argon [3] are only applicable to disk based systems and work only for large sequential workloads whereas VM workloads are highly non-sequential. The inspiration for our work comes from the Everest system described in the paper [19]. That paper describes a mechanism of offloading writes from contended storage volumes to less loaded storage volumes during peak workloads. In addition to the Everest system not being considered for virtual disk workloads, our work diverges in the following areas. We consider offloading of reads as well as writes by logging frequently accessed regions of the virtual disk to a distributed log volume. With write offloading alone we could not reduce the 90th percentile latency variability at peak loads to same value as during non-peak periods. this could not be achieved. There are several other works which use replication to redirect I/Os to servers holding replicas when the primary server is heavily loaded. But this scheme can only be used for small data objects like normal files. Keeping entire replicas of VM disk files is not feasible as this leads to more storage space utilization. It is also not effective to keep multiple replicas when these are needed only during peak loads. VirtCache therefore uses the approach of keeping copies of only the current hot block regions on the VM disk for handling peak workloads.

The work done in [10] is similar to our goals of achieving performance predictability in Virtualized environments. The authors have implemented a Non work conserving time slice based scheduler to provide strict isolation for the different VM workloads. Though this system was effective in keeping the average performance of the consolidated VMs close to that during isolation, the authors have not explored on reducing the variation in tail end performance (like the 90th percentile latency). As far as we know, the work described in [7] is the closest to our idea. It describes a dynamic instantiation of a Virtual cache appliance (VCA) during peak workloads on the *host* machine. Some of the spare memory capacity

on a compute node running the VMs is used as a read cache. I/O from the VM which interferes with other workloads is temporarily redirected to this cache. We differ from this approach since we dynamically instantiate caches on the entire *storage side* cluster (not on the host) where we have more flexibility and capacity. We do not have the restriction of capacity or controllability of the *storage side* caches as assumed in that paper since that work was targeted for hosts that use an enterprise storage system. This is because storage servers on cloud file systems (like Gluster) are commodity machines with enough DRAM/memory resources that can be shared. It should also be noted that the VCA instantiation mechanism described in the paper did not explore the use of I/O redirection to improve performance variability like 90th percentile latencies.

VI. Conclusion and Future Work

From our motivating experiments in section II, we observed that storage sharing by multiple VMs introduce large performance variations during peak loads. This problem not only exists for HDDs but also present in SSD as well. We have shown through our experiments that by temporarily redirecting I/O to a distributed cache during peak loads we can reduce the variation in performance of VM workloads to those during normal loads. In some of the cases we could in fact achieve average and 90th percentile latencies very close to that of the off-peak periods. Compared to write offloading alone we can achieve around 50% to 80% improvement in reducing the performance variation. One of the problems in our approach could be that when there is a high I/O activity on all the servers in the cluster. This could lead to contention for DRAM space from competing offloaded workloads. In such scenarios we can prioritize workloads which are critical or can tolerate minimum variation in performance. We can partition the available cache between competing workloads based on their priorities. We consider this as our Future work.

Acknowledgment

This research was supported by A*STAR Thematic strategic research program (TSRP) grant no. 1121720013

References

[1] A. Gulati, I. Ahmad, and C. A. Waldspurger, "Parda: proportional allocation of resources for distributed storage access," in *Proccedings of the 7th conference on File and storage technologies*, ser. FAST '09. Berkeley, CA, USA: USENIX Association, 2009, pp. 85–98. [Online]. Available: http://dl.acm.org/citation.cfm?id=1525908.1525915

[2] V. Tarasov, D. Hildebrand, G. Kuenning, and E. Zadok, "Virtual machineworkloads: The case for new benchmarks for nas," in *Proccedings of the 11th conference on File and storage technologies*, ser. FAST '13. Berkeley, CA, USA: USENIX Association, 2013, pp. 307–320.

[3] M. Wachs, M. Abd-El-Malek, E. Thereska, and G. R. Ganger, "Argon: performance insulation for shared storage servers," in *Proceedings of the 5th USENIX conference on File and Storage Technologies*, ser. FAST '07. Berkeley, CA, USA: USENIX Association, 2007, pp. 5–5. [Online]. Available: http://dl.acm.org/citation.cfm?id=1267903.1267908

[4] P. Goyal, D. Modha, R. Tewari, and D. Jadav, "Cachecow: Providing qos for storage system caches," in *In SIGMETRICS*, 2003, pp. 306–307.

[5] P. Sehgal, K. Voruganti, and R. Sundaram, "Slo-aware hybrid store," in *Mass Storage Systems and Technologies (MSST), 2012 IEEE 28th Symposium on*, april 2012, pp. 1 –6.

[6] A. Gulati, C. Kumar, I. Ahmad, and K. Kumar, "Basil: automated io load balancing across storage devices," in *Proceedings of the 8th USENIX conference on File and storage technologies*, ser. FAST'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 13–13. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855511.1855524

[7] L. N. Bairavasundaram, G. Soundararajan, V. Mathur, K. Voruganti, and K. Srinivasan, "Responding rapidly to service level violations using virtual appliances," *SIGOPS Oper. Syst. Rev.*, vol. 46, no. 3, pp. 32–40, Dec. 2012. [Online]. Available: http://doi.acm.org/10.1145/2421648.2421654

[8] GlusterFS. Glusterfs file system. [Online]. Available: http://www.gluster.org/

[9] OpenStack. Openstack: The open source cloud operating system. [Online]. Available: http://www.openstack.org/

[10] C. Li, I. Goiri, A. Bhattacharjee, R. Bianchini, and T. Nguyen, "Quantifying and improving i/o predictability in virtualized systems," in *Quality of Service (IWQoS), 2013 IEEE/ACM 21st International Symposium on*, 2013, pp. 1–6.

[11] A. Gulati, G. S. R. Shanmuganathan, I. Ahmad, C. A. Waldspurger, and M. Uysal, "Pesto: online storage performance management in virtualized datacenters," in *2nd ACM Symposium on Cloud Computing*, 2011, http://www.odysci.com/article/1010113016091773. [Online]. Available: http://portal.acm.org/citation.cfm?id=2038916.2038935

[12] G. Wu and X. He, "Reducing ssd read latency via nand flash program and erase suspension," in *Proceedings of the 10th USENIX conference on File and Storage Technologies*, ser. FAST'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 10–10. [Online]. Available: http://dl.acm.org/citation.cfm?id=2208461.2208471

[13] FIO. Fio benchmarking utility. [Online]. Available: http://freecode.com/projects/fio/

[14] Filebench. Filebench benchmarking utility. [Online]. Available: http://sourceforge.net/projects/filebench/

[15] S. Kraft, G. Casale, D. Krishnamurthy, D. Greer, and P. Kilpatrick, "Performance models of storage contention in cloud environments," *Software and Systems Modeling*, pp. 1–24, 2012. [Online]. Available: http://dx.doi.org/10.1007/s10270-012-0227-2

[16] M. Bennani and D. Menasce, "Resource allocation for autonomic data centers using analytic performance models," in *Autonomic Computing, 2005. ICAC 2005. Proceedings. Second International Conference on*, 2005, pp. 229–240.

[17] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, Jul. 1970. [Online]. Available: http://doi.acm.org/10.1145/362686.362692

[18] IOTTA. Storage networking industry association's input/output traces, tools, and analysis technical work group. [Online]. Available: http://www.test.org/doe/

[19] D. Narayanan, A. Donnelly, E. Thereska, S. Elnikety, and A. Rowstron, "Everest: scaling down peak loads through i/o off-loading," in *Proceedings of the 8th USENIX conference on Operating systems design and implementation*, ser. OSDI'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 15–28. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855741.1855743