Empirical Study of Application Runtime Performance using On-demand Streaming Virtual Disks in the Cloud

Han Chen, Minkyong Kim, Zhe Zhang, Hui Lei
IBM T.J. Watson Research Center
1101 Kitchawan Rd, Rt 134, P.O. Box 218, Yorktown Heights, NY 10598
{chenhan, minkyong, zhezhang, hlei}@us.ibm.com

ABSTRACT

As enterprises migrate more and more mission critical workloads to the cloud, the performance of a cloud computing system becomes increasingly important. The traditional method of pre-copying virtual machine images to hypervisors before VMs are booted results in long provisioning delays. On-demand streaming of virtual disks is used to speed up the provisioning but it may result in application runtime performance penalty. This paper seeks to quantify the runtime performance when using on-demand streaming through an empirical study. We have studied three representative application workloads: I/O micro-benchmarks, transactional application and Big Data. Two streaming protocols, NFS and iSCSI, are used in conjunction with two Copy-on-Write schemes, qcow2 from QEMU and dm-snapshot from Linux Device Mapper. We have also investigated the impact of page caching at hypervisor level on application runtime performance. The results show that I/O micro-benchmarks and transactional workload perform equally well with on-demand streaming as with pre-copied local virtual disks, while Big Data workload such as Hadoop sees a performance degradation up to 16%. We hope this study can provide insights into the performance aspect of various streaming technologies and offer guidelines to cloud operators and end users in implementing them.

Categories and Subject Descriptors

D.4.2 [**Operating Systems**]: Storage Management—*Storage Hierarchies*; C.2.4 [**Computer Communication Networks**]: Distributed Systems—*Distributed Applications*

General Terms

Performance, Experimentation

Keywords

Cloud Computing, Virtualization, Virtual Machine, Virtual Disk, Caching

1. INTRODUCTION

Middleware 2012 Industry Track December 3–7, 2012, Montreal, Quebec, Canada

As cloud computing (and virtualization in general) continues to gain wider adoption and more and more critical workloads are migrated to the cloud, the performance of a cloud has become an increasingly important aspect of consideration. In a typical cloud computing system, virtual machine (VM) states are represented as virtual disk files (also called images) that are stored in a storage subsystem. When a new VM is requested, a target hypervisor is chosen to host the guest VM, the corresponding virtual disk file is copied from the storage system to the hypervisor's local file system, and then the virtual machine is booted up from the virtual disk. In this typical scenario, the VM provisioning time consists of two main components: the virtual disk copying time and the OS boot time. Because the sizes of virtual disk files can be in the order of gigabytes, in a large cloud computing system, the image copying process can take up to minutes and results in slow provisioning.

Various methods and techniques have been proposed in the literature to speed up the image deployment process. One of the simplest methods is to eliminate the copying altogether, but instead to stream the virtual disks on-demand from the storage server to the hypervisors. This can be achieved using a number of protocols, such as Network File System (NFS) sharing and iSCSI. To allow multiple guest VMs to share a common base virtual disk file, Copy-on-Write (CoW) techniques are often implemented in conjunction with streaming virtual disk. A relatively small file (or disk volume) is allocated on the hypervisor, which stores any modified content pages of the underlying read-only streaming virtual disk file. On-demand streaming eliminates the time consumed to copy image during VM provisioning. In a well designed system, guest VMs can be provisioned at a rate of hundreds to thousands per hour, and thus enabling the rapid deployment of bursty workloads.

Compared with a pre-copied virtual disk file residing on the hypervisor's local filesystem, CoW with on-demand streaming introduces additional runtime overheads, which include the additional level of indirection in locating a virtual disk block, the inevitable fragmentation of the CoW file or volume, and additional network access overhead. This can deter cloud operators from implementing on-demand streaming virtual disks for the fear that the application runtime performance may become unacceptable to the end users. This paper seeks to quantify the application runtime performance when using on-demand streaming virtual disks as compared to pre-copied virtual disks, with the hope that the observations obtained here can provide insights into the performance characteristics of on-demand streaming and offer some guidelines and recommendations to potential users of this technology.

To this end, we have conducted a performance study using three representative application workloads—I/O micro-benchmarks, transactional workload, and Big Data workload—on a cloud computing test environment using on-demand streaming technology. Two net-

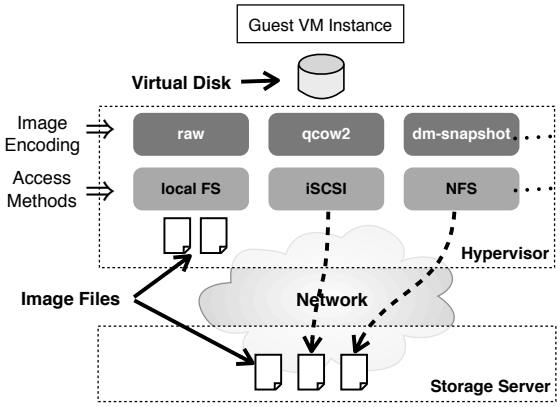


Figure 1: Different configurations of virtual disks

work protocols, NFS and iSCSI, along with two CoW technologies, qcow2 from QEMU and dm-snapshot from Linux Device Mapper, were evaluated against the baseline configuration of pre-copied virtual disk files. We also studied the impact of file system page caching at the hypervisor level on application runtime performance.

The rest of the paper is organized as follows. Section 2 provides additional background information and related work. Section 3 describes the methodology of our performance study. Section 4 presents the results of the study. Section 5 offers concluding remarks along with directions for future work.

2. BACKGROUND AND RELATED WORK

Infrastructure as a service (IaaS) cloud encapsulates user applications into virtual machines (*VMs*). The VMs are distributed on a large number of compute nodes to share the physical infrastructure. Virtualization enables many features such as consolidation for improving resource efficiency, live migration for easier maintenance, and so forth. The hard disk drive of a virtual machine (i.e., *virtual disk*) is typically emulated with a regular file on the hypervisor host (i.e., *VM image file*). I/O requests received at virtual disks are translated by the virtualization driver to regular file I/O requests to the image files.

A typical IaaS cloud, such as Amazon Elastic Compute Cloud (EC2), has thousands of VM images [5]. Therefore, it is impossible to store all image files on every hypervisor host. To facilitate VM image management, a shared storage system with a unified name space, which we denote as "VM image repository", is typically adopted to expose VM images to all hypervisor hosts. One commonly used architecture is to set up the shared storage system on a separate cluster from the hypervisor hosts, and connect the storage and hypervisor clusters via a storage area network. Another emerging scheme is to form a distributed storage system by aggregating the locally attached disks of hypervisor hosts [6]. In either scenario, when a VM is to be started on a hypervisor host, the majority of its image data is likely to be located remotely. Therefore, an interesting problem is how to configure the virtual disk for each VM to access. Figure 1 illustrates different dimensions of this configuration space.

In order to create a new VM instance in an IaaS cloud, a VM image needs to be available at its hypervisor host. As illustrated in Figure 1, one straightforward solution is to *pre-copy* the entire image to the compute nodes before a new VM is started. If an instance uses an image that the target hypervisor does not have, it may take a long time to start up that instance. A typical VM image file contains multiple gigabytes or even tens of gigabytes of data, which leads to severe delays in a heavily loaded cloud environment. Subsequent instances that use the same image on that host can start up faster as the image is locally available.

An alternative method to address this issue is to transfer the image data in an *on-demand streaming* fashion, where the parts of an image are copied as needed from the shared storage system to hypervisor hosts. This scheme is used by cloud operating environments such as IBM SmartCloud Provisioning (SCP) [8]. This *on-demand streaming* approach avoids expensive network transfers of entire image files and accelerates VM start up time. However, VM runtime performance can be degraded, as blocks of data may need to be fetched from the remote storage during runtime. In contrast, all requests to the virtual disk are served locally with the *pre-copy* approach.

In both the pre-copy and the on-demand streaming schemes, VM images can be stored in different formats. The most straightforward option is to use the *raw* format, where I/O requests to the virtual disk are served via a simple block-to-block address mapping. In order to support multiple VMs running on the same base image, copy-on-write techniques have been widely used, where a local snapshot is created for each VM to store all modified data blocks. The underlying image files remain unchanged until new images are captured. As shown in Figure 1, there are different copy-on-write schemes, including QEMU qcow2 [2], dm-snapshot [1], FVD [10], VirtualBox VDI [4], VMware VMDK [3], and so forth. In some schemes, such as qcow2, a separate file is created to store all data blocks that have been modified by the provisioned VM. Other schemes, such as dm-snapshot, work at the device level, without going through the operating system's virtual file system (VFS) layer.

Another dimension of virtual disk access is the network protocol between the storage server and compute nodes. A given cloud operating environment may choose to support a subset of all available protocols. For example, OpenStack supports the iSCSI and NFS protocols.

Recently, there have been many efforts on benchmarking cloud. However, most efforts focus on measuring public clouds such as Amazon EC2. As these public clouds do not offer flexibility in system configuration to the end users, one can only measure performance of a cloud, treating it as a blackbox. He *et al.* [7] studied the performance of high performance computing (HPC) applications in a public cloud. Jackson *et al.* [9] also analyzed HPC applications in the Amazon EC2 cloud. Our paper is the first to analyze the impact of different VM image formats and virtual disk storage configurations on application runtime performance.

3. METHODOLOGY

To represent a typical mid-range enterprise-grade cloud computing/virtualization environment, we set up a dedicated server cluster as our experiment testbed. We then configure the hypervisors to use the various on-demand streaming virtual disks described in the previous section. Application workloads are executed on this testbed using these virtual disk configurations.

3.1 Experiment Environment

Our experiment testbed, as shown in Figure 2, consists of four (4) IBM x3650 M2 xSeries Intel servers. Each server has two Intel Xeon X5570 quad-core processors clocked at 2.93 GHz (16 simultaneous threads can be executed with Hyperthreading), 32 GB of main memory, five (5) 300GB hard disks configured in RAID-0 (striping) mode and four Gigabit Ethernet adapters (two are used for guest workload traffic and the other two are used for management plane traffic). Two servers are used as compute nodes. They

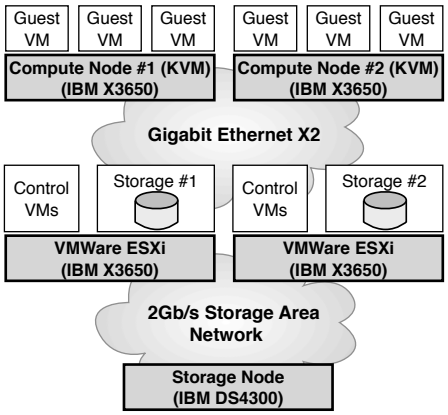


Figure 2: Hardware setup for the performance study

run RedHat Enterprise Linux v6.1 with KVM hypervisor. The other two are used for management and storage purposes and have VMWare ESXi hypervisor installed.

Image storage is provided by an IBM DS4300 storage server. The storage server is connected to the two VMWare servers via a Fiber Channel storage area network switch running at 2 Gb/s. Two storage VMs are created on the VMWare servers. Each is provided with a 1 TB volume on the SAN device. In turn, they expose the volumes to the KVM hypervisors using either iSCSI or NFS.

3.2 Streaming Virtual Disk Configurations

VM images used in our performance study are stored as image files on the SAN volumes that are mounted on the two storage nodes (#1 and #2). For NFS based streaming, we configure storage node #1 to export the directory that contains the VM image files using NFS. The two KVM hypervisors mount the NFS export and thus have access to the remote image files. For iSCSI based streaming, we export individual image files as iSCSI targets on storage node #2. We then create iSCSI initiators on the two KVM hypervisors. A Linux Device Mapper device is created as a result for each iSCSI initiator, which corresponds to a remote VM image file.

In the most common configuration, data created by a guest VM is not persisted across VM life spans. Such VMs are typically called *ephemeral*. For this type of VMs, Copy-on-Write (CoW) is implemented on top of shared, common streaming VM images. The qcow2 CoW format from QEMU is used for its ubiquity and wide adoption. The `qemu-img` utility is used to create the qcow2 format image file with the backing file being either an image file on the NFS mount or an iSCSI device file. For dm-snapshot based CoW, we use the Device Mapper `dmsetup` utility to create a snapshot volume based on a base device, which is either a loop device created from a remote NFS file or an iSCSI device directly.

Many cloud computing systems also support the notion of *persistent* VM, where guest-generated data is kept across VM life spans. In this case, the VM image file is first duplicated on the storage server, and then the streaming VM images are used directly to boot up the virtual machines.

To evaluate the effect of file system caching at the hypervisor[1], we explicitly specify the caching behavior in the libvirt virtual machine XML configuration file. By setting the cache attribute value of the driver element to "none", libvirt will open the image file (device) in `O_DIRECT` mode thus bypassing OS page caching. Setting it to "write-through" tells libvirt to open the file in `O_SYNC` mode, which enables file system page cache.

[1]referred to as "hypervisor caching" from now on for brevity

3.3 Application Workloads

We use three kinds of workloads to characterize the VM runtime performance under different conditions: I/O micro-benchmarks, transactional workload and Big Data workload.

I/O Micro-benchmarks. We use a benchmark tool called Micro-benchmark controller (MBC) to execute a suite of disk I/O micro-benchmarks. This is used to establish the baseline of the I/O performance of the virtual disks using different streaming configurations. The runtime performance of I/O intensive applications may approach the benchmark result. MBC executes two main benchmarks: DD and FFSB.

DD uses the Linux `dd` command to convert and copy a file, and measures the I/O performance under the following scenarios: sequential read, same file write, create, and rewrite. For sequential read, DD creates each file first in order to avoid caching effect and then measures the read performance. For same file write, multiple DD threads all write to the same file but to exclusive regions of that file by using different offsets. For create, DD truncates the file to zero to start from scratch and then measures file "create" performance. For rewrite, DD overwrites the existing file. For each operation, multiple threads are started simultaneously.

Flexible Filesystem Benchmark (FFSB) is a file system performance measurement tool. It is a multi-threaded application. Our set includes four different read/write patterns (large file create, sequential reads, random reads and random writes) and two different workload patterns (mail server and DB2 random access). The *large file create* operation creates 1 GB file with 4 KB block size. The *random read* operation reads 20 MB from each of 1024 files, 20 MB per file with 5 MB read size. The *random write* operation writes 20 MB to each of 1024 files with 5 MB write size. The *sequential read* operation reads 20 MB from each of 1024 files. *Mail server (ms)* and *DB2 random access* perform both read and write.

We have run MBC with varying numbers of threads (1, 4, 8 and 16) and block sizes (4 KB, 8 KB, 16 KB, 32 KB, 128 KB and 256 KB). Due to the space limitation, we have included only the results using 8 threads with the block size of 4 KB.

Transactional workload. Transactional applications represent one of the most commonly deployed enterprise applications. A typical web application uses a multi-tier architecture, which includes the Web (HTTP), application (e.g., JEE), persistence (database), and possibly integration tiers. We have chosen DayTrader, an IBM benchmark of three-tier transactional application simulating an online trading web application. It consists of IBM DB2 as persistence, IBM WebSphere Application Server (WAS) as application/web tier, and an JEE application.

The DayTrader application and middleware stack are deployed on a single large VM instance (2 vCPU, 4 GB memory, 10 GB virtual disk). Client HTTP workload is generated using Apache jmeter, which runs on a separate large VM instance (on a different hypervisor). A variable number of jmeter threads are used to vary the intensity of application workloads. Each thread issues 50,000 HTTP requests consecutively. Per DayTrader documentation, the application is warmed up with 10 threads and 2000 requests for each new test configuration so that we can measure the steady state performance.

The following runtime performance metrics are collected: throughput, average HTTP response time, and average CPU time per HTTP request. Due to space limit, we will only present the throughput numbers in the next section.

Big Data workload. Hadoop is a popular open source implementation of the MapReduce programming model. It is a data-intensive distributed data processing application. A cluster of nodes are used in the Hadoop experiment. The input data are distributed

among all nodes. Each node processes locally stored input data and writes output to local storage. The *sort* program is used, which is both read and write intensive. The input to sort is a randomly generated 4 GB dataset, stored in Hadoop Distributed File System (HDFS) with 3-way replication.

We have used 2 configurations: 8 and 16 Hadoop nodes in total. (In either case, equal number of nodes are started on each of the two compute hosts.) The performance metric for Hadoop is the job completion time, which is the elapsed time for the system to finish the sort task, assuming that data is already written to HDFS.

4. EXPERIMENT RESULTS

We have performed the aforementioned three application workloads in the test environment. The following subsections present the detailed results for each application workload type.

4.1 Ephemeral Volumes

The first set of experiments were performed with the on-demand streaming virtual disks configured in ephemeral mode.

I/O micro-benchmarks. Figures 3(a)-(d) show the results of MBC using an ephemeral storage. We used 8 threads with the block size of 4 KB. Each number represents the average value of 8 threads. The x-axis of DD shows four operations: sequential reads (seq-reads), same-file-writes (same-writes), creates (creates) and rewrites (rewrites). The x-axis of FFSB denotes eight operations: db2 reads (db2R), db2 writes (db2W), large file writes (largeW), mail server read (msR), mail server write (msW), random reads (randR), random writes (randW) and sequential reads (seqR).

Figures 3(a) and (c) show the throughput, using only the ephemeral storage on the VM instance, for DD and FFSB, respectively. Note that our base case is the raw format image on the local storage. If we compare the throughput of qcow2 to raw image, we find that the performance is similar. Using either iSCSI or NFS protocol did not make much difference for qcow2 images; we expect that this is because the overhead of using qcow2, which is in the user mode, becomes the bottleneck so that the overhead of NFS protocol compare to iSCSI did not make difference in the overall throughput.

Compared to the raw format, dm-snapshot performed much better in all cases, except *create* operation. On average across all the operations of DD and FFSB, throughput increased by a factor of 12.5× with iSCSI and 9.4× with NFS. One of the reasons that the raw format is much slower than dm-snapshot is because raw format requires file I/O while dm-snapshot uses block device directly.

In case of *create* operation, the raw format performed slightly better than dm-snapshot because *create* is meta-data operation and is not I/O intensive. In order to create a file, some blocks from the base image need to be read first and thus having the VM image on the local storage (*raw+local*) helps the *create* performance.

Since qcow2 performed similar to raw format, the performance improvement of dm-snapshot over qcow2 is similar to over raw format. This performance improvement comes from that fact that dm-snapshot runs in kernel mode, while qcow2 runs in user mode.

Figures 3(b) and (d) show the impact of enabling caching of VM's CoW file on the performance of DD and FFSB, respectively. In case of qcow2 images, caching has positive impact because caching reduces number of blocks that need to be fetched from the slow user-mode qcow2 file in the storage node. Thus, we observe a big benefit of caching for qcow2 cases. In case of dm-snapshot, caching has mostly negative impacts because reading blocks from dm-snapshot is fast enough and thus there would be little performance improvement. Also, enabling caching can incur competition between VM image blocks and application data for the limited hypervisor cache (32 GB).

Transactional Workload. We vary the number of jmeter threads from 2 to 10 to generate different levels of application workload to the DayTrader server. Figure 3(e) shows that the server has reached saturation point with about 10 jmeter threads. This indicates that our selection of jmeter thread numbers covers a board range of workload, from light to heavy.

For each of the four ephemeral on-demand streaming virtual disk configurations plus the baseline local raw disk format, we have performed the tests with hypervisor caching turned ON and OFF. Figure 3(f) shows the difference, expressed as percentage, between the application performance with hypervisor caching turned ON and that with caching OFF. We notice a small ($\leq 5\%$) but fairly consistent difference. Hypervisor caching helps application performance when dm-snapshot is used in conjunction with iSCSI, but hurts it when qcow2 with iSCSI or dm-snapshot with NFS is used. For the combinations of qcow2 with NFS and local raw disk the impact of caching is marginal at best.

The throughput numbers in Figure 3(e) represent the best caching setting for each configuration. Overall, we make the observation that, when appropriate caching policy is selected, the performance of a typical multi-tier transactional workload using on-demand streaming virtual disks is very close to that of using a pre-copied local raw virtual disk.

Big Data Workload. Figure 3(g) shows the execution time of the Hadoop *sort* program with 8 and 16 slave VMs. The best performance (lowest execution time) is achieved with the pre-copy policy, which places the entire VM image on the local file system before a VM starts. The performance gap between on-demand streaming and pre-copy is 6%-16%. The small degradation in runtime performance, together with the obvious advantage in provisioning delay, indicates that on-demand streaming virtual disks can be a viable option to support Big Data workloads in the cloud.

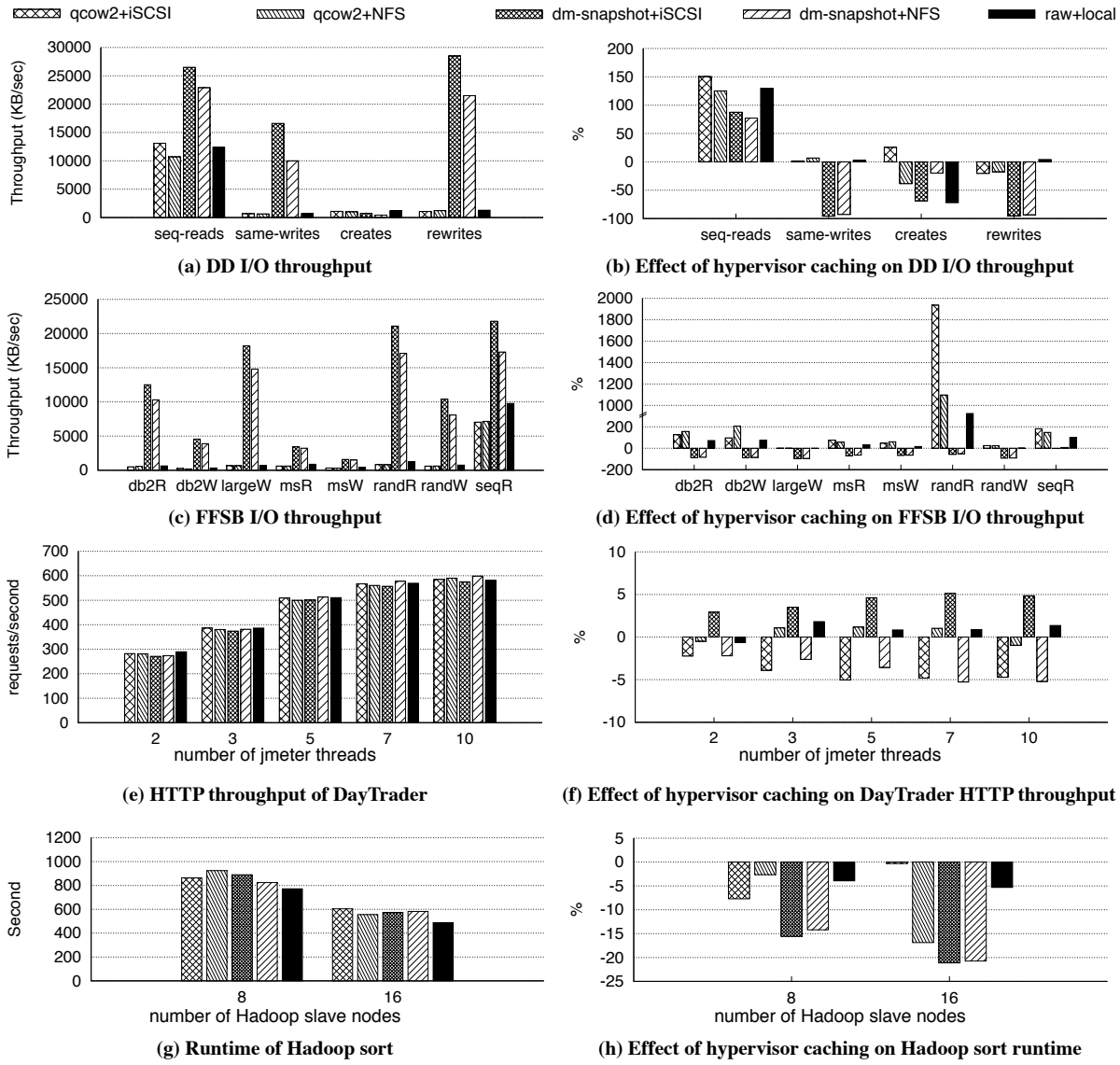
The effect of hypervisor caching is illustrated in Figure 3(h). In contrast to the other two workloads, Hadoop *sort* suffers performance degradation in all configurations when hypervisor caching is turned ON. This is because the *sort* program, similar to many other Big Data applications, has an I/O profile that is intensive in both read and write. Moreover, most read operations are for scanning the input and intermediate datasets, resulting in a low rate of data reuse. Keeping those data blocks in cache increases the memory pressure on the hypervisor host and reduces the effective memory size of each VM. As a consequence, within each VM, less memory can be used for write buffering, leading to longer I/O delays.

4.2 Persistent Volumes

The second set of experiments were performed with the on-demand streaming virtual disks configured in persistent mode.

I/O micro-benchmarks. Figures 4(a) and (b) show the throughput of DD and FFSB, respectively, using the persistent storage volume. The VM images are stored on the remote persistent storage in the raw format rather than copy-on-write format. These images are accessed either through iSCSI or NFS protocol. Our first observation is that hypervisor caching helps read performance, but does not make much different for write operations. As we have 8 threads running simultaneously, it is not surprising that caching helps read performance. The main reason that caching did not help write performance is because the hypervisor caching policy is *write-through* and thus enabling caching does not make any difference; *write-through* is the typical setting to prevent data loss in case of VM failures. (VM instances use *write-back* caching policy and thus writes are buffered in its own cache, although the size of its cache is limited.)

Our second observation is that the network protocols (iSCSI ver-



(a) DD I/O throughput

(b) Effect of hypervisor caching on DD I/O throughput

(c) FFSB I/O throughput

(d) Effect of hypervisor caching on FFSB I/O throughput

(e) HTTP throughput of DayTrader

(f) Effect of hypervisor caching on DayTrader HTTP throughput

(g) Runtime of Hadoop sort

(h) Effect of hypervisor caching on Hadoop sort runtime

Figure 3: Runtime performance of three different types of application workloads using ephemeral on-demand streaming virtual disks and the effect of hypervisor caching on runtime performance shown as percentage difference between performance with caching ON and that with caching OFF

sus NFS) did not make much difference for both read and write operations. This second observation is consistent with the ephemeral volume case where iSCSI did not make much difference for raw format (and qcow2); it provided performance benefit only for dm-snapshot.

Transactional Workload. The DayTrader performance test on persistent virtual disks was carried out in similar fashion as described in the previous subsection. The results are shown in Figure 4(c). We make the following observations.

First, with persistent virtual disk, DayTrader reaches approximately the same maximum throughput as with ephemeral virtual disk. This suggests that the workload is CPU bound but not I/O bound. Second, under light load, DayTrader actually has higher throughput with persistent virtual disk (compare the results for 2 threads in Figure 3(e) and Figure 4(c)). We believe the cause lies in the combination of the relatively low latency of storage area network and the overhead in copy-on-write. Third, the two streaming technologies, iSCSI and NFS, perform almost equally well. And finally, hypervisor caching seems to help iSCSI while hurt NFS, although the perceived difference in end performance is very small.

Big Data workload. As shown in Figure 4(d), when the entire virtual disk, including the base image and newly generated data blocks, is remotely located, Hadoop *sort* suffers severe performance degradation. This is because big data workloads are typically I/O bound and most I/O operations are on newly generated data blocks. With ephemeral virtual disks, regardless of different configurations, all newly generated data are stored in local filesystem of the hypervisor host. Another observation is that the NFS without caching configuration has much worse performance than the other 3 policies. This is because the NFS client passes the *no caching* policy to the NFS server. As a consequence, neither the hypervisor host nor the storage server will do prefetching on the Hadoop datasets. This generates a large number of small requests both over the network and on the hard disk.

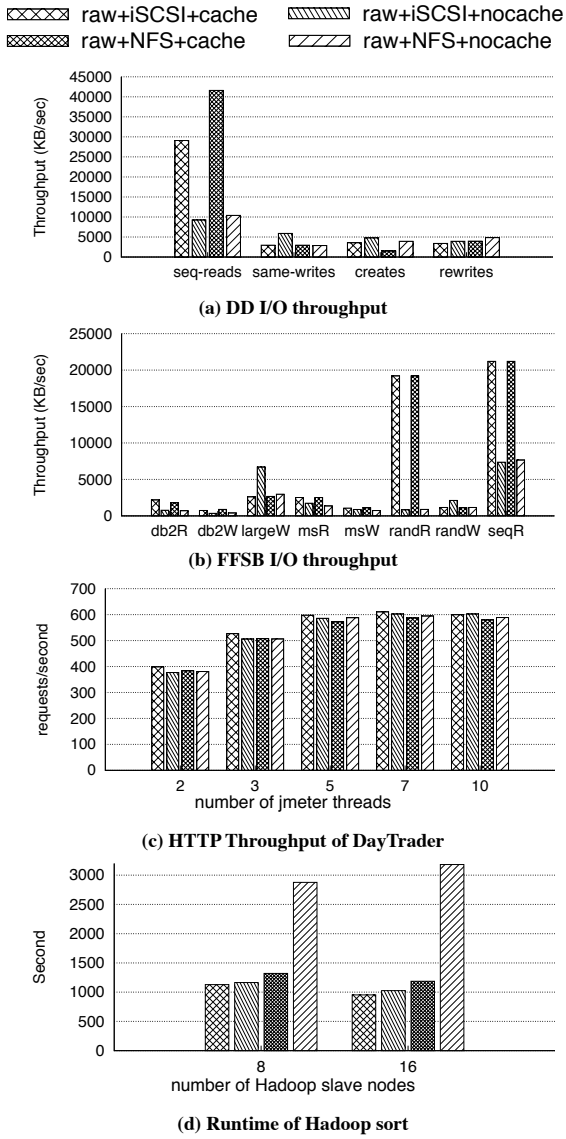


Figure 4: Runtime performance of three different application workloads using persistent on-demand streaming virtual disks and the effect of hypervisor caching on runtime performance

4.3 Summary

We summarize the results with the following observations.

1) MBC results suggest that dm-snapshot provides much better throughput than raw or qcow2 formats. With dm-snapshot, we would recommend to use iSCSI rather than NFS to access VM images and disable hypervisor cache.

2) DayTrader results suggest that on-demand streaming virtual disk configurations, used in either ephemeral or persistent setting, offer comparable runtime performance as pre-copied local raw virtual disks, and therefore they can be a practical alternative for traditional multi-tier transactional workloads, which are typical in enterprise computing environments.

3) For Big Data applications such as Hadoop, on-demand streaming virtual disks suffer from slight performance degradation.

In comparing the two main technology components of on-demand streaming virtual disk, we note that (1) NFS and iSCSI perform similarly well, and (2) dm-snapshot is a better option than qcow2.

5. CONCLUSION AND FUTURE WORK

This paper presents an empirical study of application runtime performance using on-demand streaming virtual disks in a cloud computing environment. We have selected one application for each of the three classes of commonly seen workloads—an I/O micro-benhchmark presenting I/O intensive workloads, DayTrader representing multi-tier tractional workloads and Hadoop sort representing Big Data analytics workloads. While this is by no means an exhaustive list, it offers evidence that on-demand streaming compares favorably to pre-copying of virtual disk with I/O micro-benchmarks and multi-tier transactional workloads. However, it does degrade performance by up to 16% for Big Data workloads such as Hadoop.

As future work, we plan to expand the scope of this study further. First, we will include additional benchmarks, for example, SPC-1, TPC-W for transactional processing, Mahout and Giraph for MapReduce-based machine learning and graph processing. The latter two are of particular interest as they gradually emerge as one of the most important workloads for the cloud. Second, we plan to perform an in-depth analysis of the root cause of the perceived runtime performance difference, by conducting tests with additional storage configurations, such as distributed replicated storage cluster, and studying the effect of existing background workloads. Finally, we hope to establish an analytical model for the runtime performance so that when a workload is deployed, an *a priori* analysis can be performed based on a number of factors such as workload's I/O profile, cloud's storage configuration, co-location with existing workloads, and a best virtual disk configuration selected.

Acknowledgments

The authors would like to thank Maria Ebling, Michael Engler, Ramesh Gopinath, Mike Head, Curtis Hrischuk, Alexei Karve, Andrew Kochut, Jonathan Munson, Stefan Pappe, Nathaniel Rockwell, Hidayatullah Shaikh, Harm Sluiman, Chunqiang Tang, and Andrew Trossman for their support and help for this study.

6. REFERENCES

[1] Device-mapper snapshot support. See `http://www.kernel.org/doc/Documentation/device-mapper/snapshot.txt`.

[2] The QCOW2 Image Format. See `http://people.gnome.org/~markmc/qcow-image-format.html`.

[3] Virtual machine disk format (VMDK). See `http://www.vmware.com/technical-resources/interfaces/vmdk.html`.

[4] Virtualbox vdi image storage. See `http://www.virtualbox.org/manual/ch05.html`.

[5] Amazon Web Services (AWS) Inc. Elastic Compute Cloud (EC2). See `http://aws.amazon.com`. VM image data retrieved from an author's AWS console on Aug 7, 2011.

[6] K. Gupta, R. Jain, I. Koltsidas, H. Pucha, P. Sarkar, M. Seaman, and D. Subhraveti. Gpfs-snc: An enterprise storage framework for virtual-machine clouds. *IBM Journal of Research and Development*, 55(6):2:1 –2:10, nov.-dec. 2011.

[7] Q. He, S. Zhou, B. Kobler, D. Duffy, and T. McGlynn. Case study for running hpc applications in public clouds. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, pages 395–401, New York, NY, USA, 2010. ACM.

[8] IBM. IBM smartcloud provisioning. See `http://www-142.ibm.com/software/products/us/en/smartcloud-provisioning/`.

[9] K. R. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf, H. J. Wasserman, and N. J. Wright. Performance analysis of high performance computing applications on the amazon web services cloud. In *Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science*, CLOUDCOM '10, pages 159–168, Washington, DC, USA, 2010. IEEE Computer Society.

[10] C. Tang. Fvd: a high-performance virtual machine image format for cloud. In *Proceedings of the 2011 USENIX conference on USENIX annual technical conference*, USENIXATC'11, pages 18–18, Berkeley, CA, USA, 2011. USENIX Association.