Supporting Cloud Computing with the Virtual Block Store System

Xiaoming Gao
Pervasive Technology Institute, Indiana University
gao4@cs.indiana.edu

Mike Lowe
University Information Technology Services, Indiana University
jomlowe@iupui.edu

Marlon Pierce
Pervasive Technology Institute, Indiana University
mpierce@cs.indiana.edu

Abstract

The fast development of cloud computing systems stimulate the needs for a standalone block storage system, which can provide persistent block storage services to the virtual machines maintained by the clouds. This paper presents the Virtual Block Store (VBS) System, a standalone block storage system built on the basis of LVM, iSCSI, and Xen hypervisor, which can provide basic block storage services such as volume creation and attachment. The concept and functional interface of VBS are based on the Amazon Elastic Block Store (EBS) service; moreover, VBS can be used independently with an existing volume server and Xen nodes, and can be easily extended to support other virtual machine managers, or integrated with various cloud computing systems. Preliminary experiments show that a VBS volume can provide I/O performance that is similar to an ATA over Ethernet virtual device, and comparable to a local logical volume.

1. Introduction

The technology of cloud computing has been developing very fast, with a number of implementation systems in both commercial and research areas, such as Amazon Elastic Compute Cloud (EC2) [1] service, Eucalyptus [2], and the Nimbus [3] project at the University of Chicago. Most of these systems provide implementation of Infrastructure as a Service (IaaS) based on existing virtual machine managers (VMMs), such as Xen [4], KVM [5], etc., and fulfill users' requirements for computing resources through allocation of virtual machines and construction of virtual clusters [6]. Research experiences have shown that cloud computing has successfully supported the computation requirements of many scientific computing applications [7] from various fields.

Besides requirements for computing power, applications have needs for storage services that could be used in combination with VM instances, especially in data intensive environments. Distributed file systems such as Hadoop [8] and Amazon S3[9] can provide reliable storage services in the form of large file operations, but these services require VMs to be clients of their file systems, and only address part of the applications' needs. In many cases users need raw block storage devices which they can use as if they were local disks inside the VMs, so that they can create their own file systems and databases. Many existing computing systems, such as Amazon EC2, can support a certain type of "instance storage" to their VM instances by creating disk image files locally on VMM nodes, but such instance storage services have two problems:

(1) The storage space is limited. Since disk image files are created locally on the VMM node, their sizes are constrained by the amount of physical resources available on that node. Meanwhile, VM instances running on the same VMM node have to compete for available storage space. Moreover, there is no central and flexible way to extend the storage space of the whole cloud environment except adding more storage devices to each VMM node in management;

(2) Data storage is not persistent. The lifetime of the storage devices is tightly coupled with the life time of the VM instances they are attached to; once the VM instances are terminated, their image files are deleted and so is the data stored on them. There is no way to back up the storage devices or keep them alive so that the data could be shared among multiple VM instances with different lifetimes.

To address these two problems, cloud users need special services which allow them to create persistent off-instance block storage devices, whose lifetime is independent of the VM instances they are attached to. Further, users should be able to extend their storage space as needed through the creation of more devices,

which should not be constrained by the resource limit of any single VMM node. There are already some implementations for such services, such as the Amazon Elastic Block Store (EBS) service [10], and Eucalyptus' implementation of the EBS interface based on the technology of ATA over Ethernet (AoE) [11]. However, these services are specially designed for, and therefore tightly coupled with, their cloud computing environments. As a result, it is hard to extend them to support other VMM platforms and could computing environments; e.g., it is hard to make Eucalyptus' EBS implementation work with a Nimbus cloud.

This paper presents the Virtual Block Store (VBS) System, a standalone block storage system developed by the Community Grids Lab of Indiana University. We have built a prototype of VBS based on LVM [12], iSCSI [13], and Xen hypervisor, which can provide basic block storage services such as volume and snapshot creation and deletion, and volume attachment to a running Xen DomU instance. The concept and functional interfaces of VBS are based on Amazon EBS; however, since VBS is designed to work independently and directly with volume servers and VMM nodes, it has the following advantages compared to Amazon EBS and Eucalyptus' EBS implementation:

(1) VBS is built on the basis of an open web service architecture, so it can be easily extended to support other types of volume servers and VMMs;

(2) VBS is not coupled with any specific cloud computing environment, so it can be flexibly integrated with any cloud computing systems. For example, we have successfully integrated it with Nimbus in a simple approach, as demonstrated in Section 6.

We would like to contribute our prototype of VBS as an open source framework which demonstrates a basic way to build EBS-like block storage systems, so that researchers can either use it as a start point to build their own storage systems with special purposes, or integrate it with their cloud computing systems to provide integrated block storage services.

The rest of this paper is organized as follows. Section 2 talks about related technologies. Section 3 presents the functional interfaces as well as typical use cases of VBS. Section 4 and 5 describe the web service architecture and implementation details of VBS. Section 6 demonstrates the integration of VBS with Nimbus. Section 7 discusses the results of our preliminary performance tests. Section 8 prospects our future work and concludes.

2. Related technologies

This section introduces related technologies used in VBS, including LVM [12], iSCSI [13], and Xen hypervisor [4]; and compares VBS with existing persistent off-instance block storage services, including Amazon EBS [10], and Eucalyptus' EBS implementation [2] based on AoE[11].

2.1. LVM

LVM is a Logical Volume Manager for the Linux operating system. Here a "logical volume" refers to a logically created disk drive or partition; it can be created from part of a single physical disk, or across multiple disks, and can be used just like a physical disk by upper-level applications. LVM is available on most Linux distributions, such as Debian, Red Hat, Fedora, Gentoo, openSUSE, Ubuntu, etc., providing a rich set of logical volume management functions, including creation and removal of logical volumes and their snapshots, volume group management, dynamic volume and volume group resize, logical volume stripping and mirroring, etc.

Our VBS prototype leverages LVM for volume and snapshot creation and deletion. For more information about LVM, please refer to [12].

2.2. The iSCSI protocol

The term "iSCSI" stands for "Internet Small Computer System Interface", which is an IP-based Storage Area Network (SAN) protocol. By carrying SCSI commands over IP networks, iSCSI enables remote access to block storage devices as if they were local SCSI devices. iSCSI uses TCP/IP to transfer data over network, and there are two types of roles involved in the transmission process: a "target" located on the server side and an "initiator" on the client, as shown in Figure 1. To enable a client's remote access of a disk on the server, the disk is first exported by iSCSI as a target on the server; then the client can use an iSCSI initiator utility to discover the targets on the server, and finally log in to the remote target with the discovered target name. After the log-in is completed, a virtual iSCSI disk is created on the client, which can be used just like a local SCSI disk. Once the client completes its work, it can log out from the target, and finally the target could be deleted on the server.

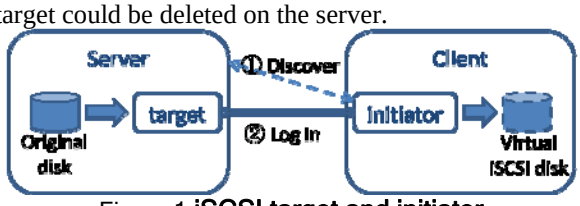

Figure 1 iSCSI target and initiator

iSCSI can support sharing of disks with two layers of multipath. To share an original disk on the server among multiple clients, the user can either have one

disk exported as multiple targets, or one target connected to multiple initiators. The iSCSI protocol takes care of necessary synchronization work on the block level; e.g., there cannot be two initiators writing to the same block section on a disk at the same time.

Our VBS prototype leverages iSCSI for remote access of logical volumes from VMM nodes. Further, iSCSI's support for multipath can be utilized for potential sharing of disk volumes among multiple VM instances.

2.3. Xen hypervisor

The Xen hypervisor is an open source virtual machine monitor for x86, x86_64, IA64, PowerPC, and other CPU architectures. It supports efficient concurrent execution of multiple guest VM instances, which could have a wide range of operating systems running on them, including Windows, Linux, Solaris, and various versions of the BSD operating systems. There are two types of "domains" running on a node managed by Xen hypervisor: Dom0 (or domain zero) and DomU. Dom0 is the first domain started by Xen on boot; it has direct access to the hardware, and is responsible for creating other domains (DomU) and coordinating their concurrent executions. A DomU is a domain created by Dom0; it is the actual VM instance where a guest OS is running. A DomU does not have direct access to the hardware by default; it must run a FrontendDriver which communicates with Dom0 to sent hardware requests.

Xen provides the virtual block device (VBD) technique, which allows the user to attach a block device or disk image file in Dom0 to a DomU VM instance, so that it can be used as a local block device in DomU. Our VBS prototype utilizes the VBD technique for attachment and detachment of virtual disk volumes to/from VM instances.

2.4. Amazon EBS

The Amazon.com Inc. provides the Amazon EBS service as a part of Amazon EC2 [1]. It supports a full set of persistent and off-instance disk volume storage operations that are suitable for use with its EC2 VM instances, including volume and snapshot creation and deletion, volume and snapshot description, and volume attachment and detachment to/from a VM instance. For more explanation about the web service interface of Amazon EBS, please refer to [10].

Little has been known about the design and implementation of Amazon EBS since they are kept as commercial secrets. The problem with Amazon EBS is that it is not a standalone system, but tightly coupled with Amazon EC2. It is impossible to attach a disk volume created with Amazon EBS to a Nimbus VM instance. In contrast, our prototype of VBS is designed as a standalone system which works directly with VMM nodes, and thus can be easily integrated with any cloud computing environment.

2.5. Eucalyptus' EBS implementation

The Eucalyptus cloud computing system has implemented exactly the same interface as Amazon EBS based on the AoE technology. However, their EBS implementation is also tightly coupled with its cloud management system, which means it is hard to integrate their implementation with other cloud computing systems. The use of AoE can help make the volume operations and data transmission more efficient, because AoE runs directly over Ethernet and incurs no TCP/IP overhead. Nonetheless, it also faces two limits compared with an iSCSI-based solution:

(1) The routability of a AoE based solution is limited inside an Ethernet environment, which means that the disk volume server and VMM nodes of Eucalyptus must be all located in the same LAN, or VLAN at least;

(2) Sharing of disk volumes is harder in an AoE environment. AoE provides mechanisms such as reserve and release command and config string to coordinate the concurrent access from different clients, but these are not real native target sharing mechanisms among multiple clients. For example, if one target is reserved by one specific client and that client goes down, there will be no normal way for another client to come over and resume the use of the target device. The only way to deal with this type of failure is for the administrator to force release the target so that it could be available to other clients.

3. VBS interface and use cases

To address the requirements of a standalone block storage service as mentioned in Section 1, VBS provides the following operation methods in its web service interface:

create-volume <size> <comment> <snapshot id>
delete-volume <volume id>
describe-volumes [<volume id> <volume id> …]
create-snapshot <volume id> <comment>
delete-snapshot <snapshot id>
describe-snapshot [<snapshot id> <snapshot id> …]
attach-volume <volume id> <VMM hostname> <VM id> <VM device>
detach-volume <volume id>

We designed the VBS interface based on the interface of Amazon EBS [10], with the following special differences:

(1) There is no concept of "availability zone" in our VBS prototype, so the "create-volume" and "create-snapshot" methods accept a "comment" parameter instead of "availabilityZone", which is used in Amazon EBS;

(2) In Amazon EBS, the "attach-volume" operation accepts an EC2 VM instance ID besides a volume ID and a VM device path, while in VBS the EC2 instance ID is replaced by a VM instance ID and the hostname of the VMM node where the VM is running, because VBS is independent of any specific cloud computing environment, and works directly with VMM nodes;

(3) The "detach-volume" operation accepts only one parameter: the ID of the volume to be detached. This is because one disk volume is only allowed to be attached to one VM instance in current VBS implementation. More parameter about the volume's attachment will be needed once volume sharing is supported in the future.

For more information about the web service interface of VBS, please refer to [14]. Using the operations of this interface, the user can apply VBS to two typical use cases, as shown in Figure 2 and 3. In Figure 2, users can create multiple logical volumes (LV) in VBS and attach them to one or more VM instances; moreover, they can extend their storage space by just creating new volumes as needed. The lifetime of VBS volumes is independent of the VM instances – they are maintained by VBS as long as "delete-volume" is not called on them. In Figure 3, the user can first create a snapshot of a volume which already contains some basic data or software environment, and then create new volumes based on the snapshot, and attach them to different VM instances, so that all VMs can have the same basic data and software environment, and start doing their own computations and generate different output results.

Figure 2 Use of VBS: extendable storage

Figure 3 Use of VBS: snapshots

4. VBS web service architecture

Figure 4 VBS web service architecture

To implement the volume operations presented in Section 3, we built a flexible web service architecture for VBS, as shown in Figure 4. There are two types of physical nodes, volume server and VMM nodes, and three types of web service modules, the VBS Service module, the Volume Delegate module, and the VMM Delegate module, in the architecture.

In our current design there could be only one volume server node, and multiple VMM nodes. All physical storage devices used by VBS are installed on the volume server node, and LVM [12] is used to manage these storage devices, as well as logical volumes and volume groups. iSCSI [13] is used for remote access of logical volumes from VMM nodes, and the "ietadm" utility is used by the volume server for managing iSCSI targets created from logical volumes. On VMM nodes, the Xen hypervisor [4] is used to manage VM instances, the "iscsiadm" utility is used for controlling iSCSI initiators, and the VBD technique is used for attaching/detaching a virtual iSCSI device to/from a DomU VM instance.

The Volume Delegate module is a web service running on the volume server, responsible for executing LVM commands for volume and snapshot creation and deletion, and iSCSI commands for target creation and deletion based on logical volumes.

The VMM Delegate module is a web service running on a VMM node, responsible for executing iSCSI commands for log-in and log-out from initiators to targets, and Xen VBD commands for attaching and detaching iSCSI virtual devices to/from VM instances.

The VBS Service module is the frontend web service which accepts requests from the VBS clients, and completes their VBS operations by coordinating the actions of the Volume Delegate and the VMM Delegate. Details about the coordination are covered in Section 5.

This web service based architecture is very open and flexible, and can be easily extended to support other types of volume servers and VMMs. The main difference with other volume servers and VMMs is that they use different sets of commands to complete logical volume management and VM instance management. Therefore, one way to support them is to build new Volume Delegate and VMM Delegate modules to wrap up the different sets of commands. As long as the new modules keep the same interfaces, they can still work with the other VBS modules seamlessly.

Another solution for supporting different volume servers and VMMs is a technique called "command line extraction", which is currently used by the VBS prototype. The current implementations of Volume Delegate and VMM Delegate use the Apache Ant technology [15] to wrap up and execute the LVM and Xen commands, as shown in the upper part of Figure 5. The source codes for creating and executing the ant project are actually constructing an executable command line from a specific "command line pattern" by replacing a set of parameters in the pattern. If we can extract these command line patterns out from the source codes, and save them in a property file, as shown in Figure 5, then it will be possible to support different commands from other volume servers and VMMs by just modifying the patterns in the property file, without touching and rebuilding the web service codes. This solution can work as long as the new command line pattern accepts the same set of parameters, which is the normal case for most VBS operations. For example, the command line pattern in Xen to attach a block device in Dom0 to a DomU instance is "xm block-attach <VM ID> phy:<VMM dev> <VM dev>"; if we want to make the VMM Delegate work with the proxmox [16] VMM platform which manages KVM VM instances, all we need to do is just changing this command line pattern to "qm set <VM ID> -<VM dev> <VMM dev>".

Figure 5 Command line extraction

5. VBS implementation

This section explains the implementation details of VBS, including workflows and mechanisms for maintaining system consistency.

5.1. Workflows

Figure 6 demonstrates the workflows of main VBS operations. Due to space limitations the workflows of "describe-volumes" and "describe-snapshots" are not included; in brief, these two operations just look up the metadata of the given volume or snapshot IDs, and return the corresponding information. Here we just explain the workflow actions of "create-volume" and "attach-volume" as an example, and the actions of the rest operations could be inferred from Figure 6 in a similar way.

Figure 6 VBS workflows

To handle a client's request for the "create-volume" operation, VBS Service invokes Volume Delegate, which then executes the "lvcreate" command to create a new logical volume of the size given by the VBS client. After the new logical volume is created, Volume Delegate checks if the new volume is required to have the same data as a specific snapshot. If such a snapshot's ID is specified, Volume Delegate will first return the new volume's information, along with a "pending" status, to VBS Service, and then start a new thread to copy the content from the specified snapshot to the new volume with the Linux "dd" command, and finally return an "available" status, which denotes that the new volume is ready for attachment with a VM instance, to VBS Service after the copy is completed. If no snapshot ID is specified, Volume Delegate will just return the new volume's information, along with an

"available" status, to VBS Service. Note that the asynchronous copy operation is necessary for quick response, since it's a time consuming job.

For the "attach-volume" operation, VBS Service first invokes Volume Delegate to export an iSCSI [13] target based on the given volume ID by executing the "ietadm new" command. After the new iSCSI target is created, VBS Service will invoke the VMM Delegate running on the VMM node specified by the client, which then carries out the following operations:

(1) Execute the "iscsiadm –discover" command to discover all targets on the volume server;

(2) Execute the "iscsiadm –login" command to login to the newly created target. This will result in a virtual iSCSI device created on the VMM node;

(3) Execute the "xm block-attach" command to attach the virtual iSCSI device to the VM instance with the given VM ID.

5.2. Consistency mechanisms

The VBS system need to maintain two levels of consistency: metadata level consistency and system level consistency.

We use a small on-disk HSQLDB [17] database to keep the metadata in our VBS prototype. The database contains three major tables: the "volumes" table, the "snapshots" table, and the "attachments" table, and applies various database techniques to maintain the metadata consistency, such as dependencies and transactions. New volume IDs and snapshot IDs are generated from the hash codes of UUIDs, and duplicated hashing results are eliminated with the help of one additional database table, the "ids" table, which keeps track of all IDs in use.

To maintain the system level consistency, we add a "roll-back" mechanism to the implementations of VBS operations which involve multiple action steps. For example, during the execution of the four steps of "attach-volume", if any step fails, the execution of the previous steps will all be recovered, so that the whole system can roll-back to a previously consistent status.

6. Integration with Nimbus

We have successfully integrated VBS with the Nimbus [3] project, and would like to use it as an example to demonstrate the integration strategy of VBS with a cloud computing environment.

Nimbus is a cloud computing platform developed by University of Chicago. It provides services such as leasing of VM instances and creation of virtual clusters by manipulating Xen hypervisor [4] nodes. Nimbus supports both a WRSF [18] web service interface and an Amazon EC2-compatible web service interface, so that the user can also use an Amazon EC2 [1] client to access a Nimbus cloud. However, the support for Amazon EC2 interface is incomplete, and there is no EBS-like implementation in Nimbus yet.

To complete the integration, we need to consider the difference between the integrated version and the standalone version of VBS. In the integrated version, the requirements for most VBS operations are similar; the difference lies in the operation of "attach-volume": the integrated version of "attach-volume" should accept a Nimbus instance ID as the destination of the attachment, instead of the specific Xen DomU ID and the hostname of the corresponding VMM node. Since VBS requires the VM ID and VMM hostname to complete the attachment on the right node, we need a mechanism in the integrated version to find out the corresponding Xen DomU ID and VMM hostname associated with a given Nimbus instance ID. To implement this mechanism, we introduce an auxiliary module, the VBS_Nimbus Service, to the VBS architecture, as shown in Figure 7.

Figure 7 Integration with Nimbus

In the new architecture, the VBS_Nimbus Service module is the frontend web service accepting a client's request. The client provides a volume ID, a Nimbus instance ID, and a VM device path when calling the "attach-volume" operation. Upon receiving the client's request, the VBS_Nimbus Service will first make a resource property query to the workspace web service of Nimbus to ask for the Xen DomU ID and the corresponding VMM hostname associated with the given Nimbus instance ID. After the result is returned, the VBS_Nimbus Service will be able to invoke the original VBS Service to complete the operation.

Since the original Nimbus workspace service does return the information we need, we modified the implementation of the "Manager" interface of Nimbus RM API, so that the workspace service will append the Xen DomU ID and VMM hostname to the network properties of a Nimbus instance when answering a

resource property query. It takes only one line of Java source code to complete the necessary modification.

Similarly, we can integrate VBS with other cloud computing systems by introducing auxiliary modules and corresponding query mechanisms.

7. Preliminary performance test

To test the performance of VBS, we set up a single VM and single volume test bed in a private 1Gb Ethernet LAN of Indiana University (IU). We started one volume server and one VMM node in the test bed. The volume server is configured with 4 Intel Xeon 2.8G CPUs and 512MB of memory, and is running Red Hat Enterprise Linux (RHEL) 5.3 and LVM 2.0. The VMM node is configured with 2 AMD Opteron 2.52G CPUs and 1.5GB of memory, and is running RHEL 5.3, Xen [4] 2.0, and LVM [12] 2.0. One DomU is started on the VMM node, with 1 AMD Opteron 2.52G CPU, 512MB of memory, and a CentOS 5.2 disk image with 2GB of disk space. For comparison purposes, we tested the performance of 4 different types of virtual disks on this test bed, with Bonnie++ 1.03e [19]. The virtual disk types tested are listed below:

VBS-LVM: a 5GB VBS volume is created and attached to the VM instance. An ext2 file system is created on it.

AoE-LVM: a 5GB logical volume is on the volume server and exported as an AoE [11] target; the target is discovered on the VMM node as a virtual AoE device, which is then attached to the VM instance. An ext2 file system is created on the volume.

Local-LVM: a 5GB logical volume is created locally on the VMM node with LVM, and attached to the VM instance. An ext2 file system is created on it.

Local-Image: the CentOS 5.2 disk image file is also tested with Bonnie++. It has an ext3 file system and 1.6GB available space.

We also created a Eucalyptus VM instance and an EBS volume on the Eucalyptus test bed of IU, and tested its performance with Bonnie++ for comparison with VBS. The Eucalyptus test bed has a dedicated 1Gb Ethernet LAN. The volume server is configured with 2 Intel Xeon 5150 2.66GHz dual-core CPUs and 12GB of memory. It is running on RHEL 5.2, and uses 2 300GB 15K SAS disks in a RAID1 configuration for volume management, instead of LVM. The VMM node is configured with 2 Intel Xeon L5420 2.50GHz quad-core CPUs and 32 GB of memory, and is running RHEL 5.2 and Xen hypervisor 3.0. The VM instance has 4 Intel Xeon L5420 2.5G/6144KB CPUs and 2GB of memory, and is running RHEL 5.0. We define the virtual disk type in this test as:

Euca-AoE: a 5GB EBS volume is created on the Eucalyptus test bed and attached to a Eucalyptus VM instance. An ext2 file system is created on it.

According to the default configuration of Bonnie++, a 4GB file was created for testing the Euca-AoE disk, and a 512MB file was used for the other disk types. Each type of disk was tested 10 times, and the average throughput performances of 5 types of I/O operations of all disks are given in Table 1. The data unit is KiloBytes/second. Due to space limit we can just list a brief definition for each I/O operation here, and for detailed information please refer to [19].

Per-Ch Write: the file is written using putc().

Block Write: the file is created using write(2).

Rewrite: each BUFSIZ of the file is read with read(2), dirtied, and rewritten with write(2), requiring an lseek(2).

Per-Ch Read: The file is read using getc().

Block Read: The file is read using read(2).

Table 1 **Throughput test results**

Disk Type	Per-Ch Write	Block Write	Rewrite	Per-Ch Read	Block Read
VBS-LVM	27454.9	29646.3	14326.6	25133.8	27725.4
AoE-LVM	19391.8	20628.6	17698	31758.1	42239.7
Local-LVM	60246.2	82553.7	27427.9	59315.6	57572.6
Local-Image	59134.7	426567.6	79927.6	60800.7	457278.1
Euca-AoE	52805	95706	36832.3	47086.6	78989.5

There are several notable observations that we can get from Table 1:

(1) The performance difference between different disk types is large for block operations, and relatively small for Per-Ch operations. This is because in Per-Ch operations there is only one character transferred for each call, and most time is spent on CPU overhead.

(2) VBS-LVM is faster in write and slower in read than AoE-LVM, and their overall performance is close. This indicates that iSCSI is better optimized for write operations, and could be as efficient as an AoE solution under the same environment configurations. Moreover, the performance of VBS-LVM can reach to as high as 35%-55% of Local-LVM, as further demonstrated in Figure 8.

(3) The overall performance of VBS-LVM is about 30%-50% percent of Euca-AoE. Since the performance of VBS-LVM is close to AoE-LVM, this implies that the difference with Euca-AoE comes from the different environment configuration. The better hardware, especially the use of SAS disks in RAID1 instead of LVM, and the dedicated Ethernet, help provide a high throughput for EBS in the Eucalyptus test bed.

(4) The performance of Euca-AoE is close to Local-Image for Per-Ch operations, but not comparable for block operations, because the 1Gb Ethernet puts a hard limit on the transmission speed of Euca-AoE. It should be noticed that the Block Read and Block Write throughput of Euca-AoE can get close to the Ethernet

wire speed, which is consistent with the testing results of iSCSI and AoE from VMWare and Coraid [20, 21].

(5) The throughput of Euca-AoE for block operations is even faster than Local-LVM, which implies that VBS-LVM also has the potential to exceed Local-LVM in a properly configured environment.

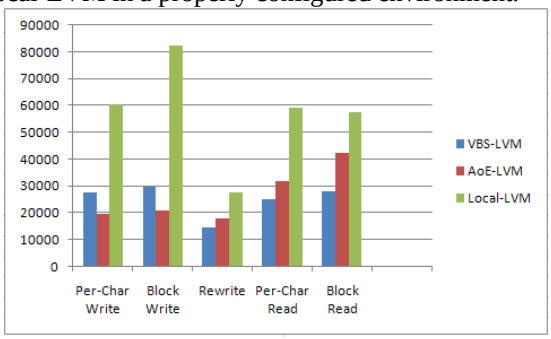

Figure 8 **Throughput comparison**

Figure 9 presents the comparison of all disk types on the operations of sequential file creation, random file creation, and random file deletion. The data unit of the vertical axis is files/second. We can see that the performances of all disk types are similar, because these operations involve little data transmission.

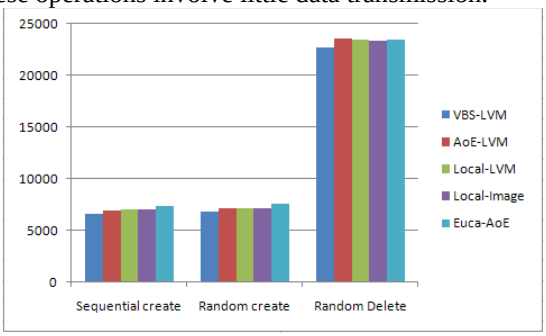

Figure 9 **File creation and deletion comparison**

8. Conclusion and future work

This paper presents the VBS system, an open source standalone block storage system that can provide persistent and off-instance block storage services to VM instances maintained in cloud computing environments. Compared with existing services, such as Amazon EBS [10] and Eucalyptus' EBS [2] implementation, VBS has a similar web interface and supports similar logical volume operations such as volume creation and attachment; moreover, based on a flexible web service architecture, VBS can be easily extended to support different types of volume servers and VMMs, or integrated with various cloud computing systems such as Nimbus. Our preliminary performance tests show that VBS can provide throughput that is similar to an AoE-based solution under the same test bed configuration, and has the potential to exceed the performance of a local LVM [12] volume with better hardware support. There are several directions that we will continue to work on in the future:

First, sharing of VBS volumes among multiple VM instances is a potential requirement of many applications. By leveraging the multipath support in iSCSI, read-only sharing of volumes can be achieved with little modification. Furthermore, read-and-write sharing could be implemented by applying the technology of shared disk file systems.

Second, user management is not implemented yet in VBS. In the short run, a "user" table can be added to the VBS database for recording the ownership of volumes and snapshots. In the long run, a more complete user control mechanism is necessary to handle issues such as access control and space quota.

Third, the single volume server architecture of VBS is neither reliable nor scalable. We will consider ways to extend the single volume server to a distributed network of multiple servers to improve the reliability and scalability of VBS.

References

[1] Amazon EC2 service, http://aws.amazon.com/ec2/.
[2] Eucalyptus, http://open.eucalyptus.com/.
[3] The Nimbus project, http://workspace.globus.org/.
[4] The Xen hypervisor, http://www.xen.org/.
[5] KVM, http://www.linux-kvm.org/page/Main_Page.
[6] K. Keahey, T. Freeman, "Contextualization: Providing One-Click Virtual Clusters", *Proceedings of 2008 Fourth IEEE International Conference on eScience*, Indianapolis, IN, December 2008, pp. 301-308.
[7] K. Keahey, T. Freeman, et al., "Science Clouds: Early Experiences in Cloud Computing for Scientific Applications", *Proceedings of Cloud Computing and Its Applications 2008 (CCA-08)*, Chicago, IL. October 2008.
[8] The Apache Hadoop project, http://hadoop.apache.org/.
[9] Amazon S3 service, http://aws.amazon.com/s3/.
[10] Amazon EBS service, http://aws.amazon.com/ebs/.
[11] S. Hopkins, B. Coile, "The ATA over Ethernet Protocol Specification", *Technical Report*, The Brantley Coile Company, Inc., February 2009.
[12] LVM, http://tldp.org/HOWTO/LVM-HOWTO/.
[13] The iSCSI protocol, http://tools.ietf.org/html/rfc3720.
[14] VBS web service interface definition, http://cglc.uits.iu.edu:8080/axis2/services/VbsService?wsdl.
[15] The Apache Ant project, http://ant.apache.org/.
[16] Proxmox VE, http://pve.proxmox.com/wiki/Main_Page.
[17] The HSQLDB database engine, http://hsqldb.org/.
[18] OASIS Web Services Resource Framework (WSRF) TC, http://docs.oasis-open.org/wsrf/wsrf-ws_resource-1.2-spec-os.pdf.
[19] Bonnie++, http://www.coker.com.au/bonnie++/
[20] "Comparison of Storage Protocol Performance", VMwaer Inc., Jan 23, 2008.
[21] "AoE Performance Comparison", Coraid Inc., 2009.