Supporting Cloud Computing with the Virtual Block Store System

Xiaoming Gao, Mike Lowe, Yu Ma, and Marlon Pierce
Indiana University
Bloomington, Indiana, USA
gao4@indiana.edu, jomlowe@iupui.edu, yuma@indiana.edu, mpierce@indiana.edu

***Abstract*—The fast development of cloud computing systems stimulates the needs for a standalone block storage system to provide persistent block storage services to virtual machines maintained by clouds. This paper presents the Virtual Block Store (VBS) System, a standalone block storage system built on the basis of Logical Volume Manager (LVM), Internet Small Computer System Interface (iSCSI), and Xen hypervisor. VBS can provide basic block storage services such as volume creation and attachment. The concept and functional interface of VBS are based on Amazon Elastic Block Store (EBS) service; moreover, VBS works independently with an existing LVM volume server and Xen nodes, and thus can be easily extended to support other types of volume servers and virtual machine managers, or integrated with various cloud computing systems. Preliminary I/O benchmark results are presented and analyzed, indicating that a VBS volume can provide throughput that is similar to an AT Attachment over Ethernet (AoE) virtual device.**

I. INTRODUCTION

Scientific cyberinfrastructures have been focusing on high performance computing and supercomputing, and treating data collections as byproducts. Demands for managing and storing vast datasets collected and generated by research communities have kept increasing, yet not been sufficiently satisfied at the infrastructure level. Cloud computing systems, such as Amazon Elastic Compute Cloud (EC2) [1], Eucalyptus [2] and Nimbus [3], have emerged as the next generation of cyberinfrastructure to provide computing resources in a dynamic way. Most of these systems implement Infrastructure as a Service (IaaS) based on existing virtual machine managers (VMMs), such as Xen [4], Kernel-based Virtual Machine (KVM) [5], etc., and fulfill users' requirements for computing resources through allocation of virtual machines (VMs) and construction of virtual clusters [6]. Research experiences have shown that cloud computing has successfully supported the computation requirements of many scientific applications [7] from various fields. Correspondingly, cloud related storage infrastructures can be a good option to address applications' requirements for data storage. One type of such storage infrastructures are distributed file systems such as Hadoop [8] and Amazon Simple Storage Service (S3) [9]. These systems can provide reliable storage services in the form of large file operations, but they require VMs to be clients of their file systems, and only address part of applications' needs. In many cases users need raw block storage devices which they can use as if they were local disks inside VMs, and create their own file systems and databases on them. Many existing cloud computing systems, such as Amazon EC2, can support a certain type of "instance storage" service to their VM instances by creating disk image files locally on VMM nodes, but such service have two problems:

First, the storage space is limited. Since disk image files are created locally on VMM nodes, their sizes are constrained by the amount of physical resources available on those nodes. 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;

Second, data storage is not persistent. The lifetime of the storage devices is tightly coupled with the VM instances they are attached to; once the VM instances are terminated, their image files are deleted. 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 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 Amazon Elastic Block Store (EBS) service [10], and Eucalyptus' implementation of the EBS interface based on the technology of AT Attachment 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 Logical Volume Manager (LVM) [12], Internet Small Computer System Interface (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:

First, VBS is built on the basis of an open Web services architecture, so it can be easily extended to support other types of volume servers and VMMs;

Second, VBS is not coupled with any specific cloud computing environment, so it can be flexibly integrated with any cloud computing system. For example, we have successfully integrated it with Nimbus in a simple approach, as demonstrated in Section VI.

We think of VBS as a right step towards a block storage infrastructure for cloud systems, which can not only supply flexible and persistent block storages to cloud users, but also provide an easy way for users to share and access their data collections through the use of logical volumes and snapshots, as explained in Section III. We would like to contribute our prototype of VBS as an open source framework demonstrating 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 specialized storage systems, or integrate it with their cloud computing systems to provide integrated block storage services.

The rest of this paper is organized as follows. Section II talks about related technologies. Section III presents the functional interfaces as well as typical use cases of VBS. Section IV and V describe the Web services architecture and implementation details of VBS. Section VI demonstrates the integration of VBS with Nimbus. Section VII discusses the results of our preliminary performance tests. Section VIII prospects our future work and concludes.

II. 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' [2] EBS implementation based on AoE [11].

A. 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 used just like one physical disk by upper-level applications. LVM is available on most Linux distributions, providing a rich set of logical volume management functions.

Our VBS prototype leverages LVM for volume and snapshot creation and deletion.

B. The iSCSI protocol

iSCSI 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 disks. 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 and an "initiator" on the client (Fig. 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, and then discovered by an iSCSI initiator utility on the client. After discovery, the iSCSI initiator can login to a discovered target name. Upon login, a virtual iSCSI disk is created on the client, which can be used just like a local SCSI disk.

iSCSI supports 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 at the block level, e.g., there cannot be two initiators writing to the same block section on a disk at the same time.

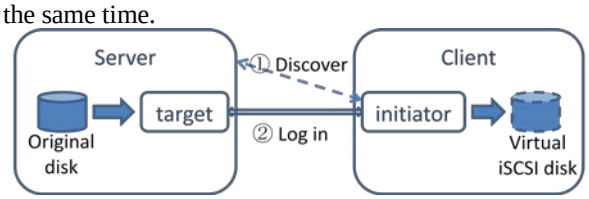

Figure 1. iSCSI target and initiator.

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

C. Xen hypervisor

The Xen hypervisor is an open source virtual machine monitor for multiple CPU architectures. It supports efficient concurrent execution of multiple guest VM instances running a wide range of operating systems. Two types of "domains" run on a node managed by Xen hypervisor: Dom0 (or domain zero) and DomU. Dom0 is the 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. 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, and must run a Frontend Driver to communicate with Dom0 for hardware requests.

Xen provides the virtual block device (VBD) technique, which allows users 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.

D. Amazon EBS

The Amazon.com Inc. provides 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 EC2 VM instances, including volume and snapshot creation and deletion, volume and snapshot description, and volume attachment and detachment to/from a VM instance.

Little has been known about the design and implementation of Amazon EBS since they are kept as commercial secrets. Amazon EBS is not a standalone system; it is tightly coupled with Amazon EC2. For example, 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 that works directly with VMM nodes, and thus can be easily integrated with any cloud computing environment.

E. Eucalyptus EBS implementation

Eucalyptus is an open source cloud computing system that has implemented exactly the same interface as Amazon EBS using the AoE technology. However, their EBS implementation is also tightly coupled with the cloud

management system. The use of AoE can help make volume operations and data transmission more efficient, because AoE runs directly over Ethernet and thus incurs no packaging overhead for IP and TCP. Nonetheless, it also faces two limits compared with an iSCSI-based solution:

First, the routability of an 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;

Second, sharing of disk volumes is harder in an AoE environment. AoE provides mechanisms including reserve/release command and config string to coordinate concurrent access from different clients, but these are not real native target sharing mechanisms. For example, if one target is reserved by a 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.

III. VBS INTERFACE AND USE CASES

To address the requirements of a standalone block storage service as mentioned in Section I, VBS provides the following operation methods in its Web services 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:

First, 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;

Second, in Amazon EBS the "attach-volume" method 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;

Third, 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 services interface of VBS, please refer to [14]. Using the operations of this interface, users can apply VBS to two typical use cases, as shown in Fig. 2 and 3. In Fig. 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 Fig. 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.

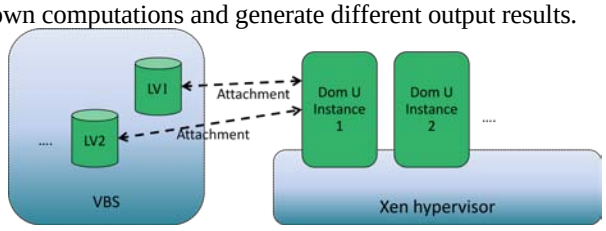

Figure 2. Use of VBS: extendable storage.

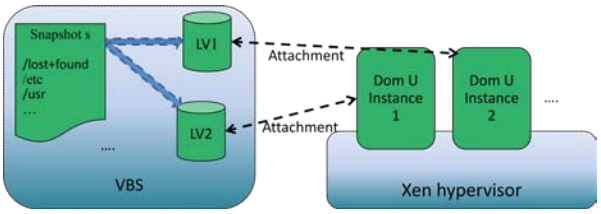

Figure 3. Use of VBS: snapshots.

IV. VBS WEB SERVICES ARCHITECTURE

To implement the volume operations presented in Section III, we built an open and flexible Web services architecture for VBS, as shown in Fig. 4. There are two types of physical nodes, volume server and VMM nodes, and three types of Web services 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. 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. We choose iSCSI to connect the volume server and VMM nodes for two reasons. First, iSCSI runs over TCP/IP and thus has better routability than AoE [11], which ensures that VBS can be applied in environments with Internet as well as LAN connections. Second, the multipath support of iSCSI provides a helpful utility to implement sharing of logical volumes among different VMs, which will be included in future versions of VBS. Users can utilize volume sharing to handle situations such as VM failures. For example, the user can safely have two VM instances, one as a working node and the other as a back-up node, attached to the same VBS volume; once the working node fails, the back-up node will be able to resume its work on that volume right away.

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.

Figure 4. VBS Web services architecture.

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 Volume Delegate and VMM Delegate. Details about the coordination are covered in Section V.

This Web services based architecture 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 apply the "adapter" design pattern [21] to different volume servers and VMMs, and build new Volume Delegate and VMM Delegate services to wrap up the different sets of commands. As long as the new services 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 we call "command line extraction", which is currently used by the VBS prototype. The Volume Delegate and VMM Delegate use Apache Ant technology [15] to wrap up and execute the LVM and Xen commands. These commands used to be hard coded in the program logics in early implementations of VBS, as demonstrated in the upper part of Fig. 5. The source codes shown here are actually first constructing an executable command line in the form of "xm block-attach <vmId> phy:<vmmDev> <vmDev> w" by replacing the bracketed parameters with dynamic values; and then executing the command line as an Ant ExecTask object. We call this special form of command line with bracketed parameters a "command line pattern".

Except for the hard coded implementation, another way for doing the same job is to extract these command line patterns out from source codes, save them in a property file, and then read them in during execution of the services, and dynamically create and execute the commands, as shown in the lower part of Fig. 5. In this way, we will be able to make the services execute different types of commands from different volume servers and VMM platforms by just modifying the patterns in the property file, without touching and rebuilding the Web services' codes. By reading in different command line patterns during runtime, the Volume Delegate and VMM Delegate services are actually running as "dynamic command adapters". 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 given above corresponds to a Xen command which attaches a block device in Dom0 to a DomU instance; if we want the VMM Delegate to complete the same job on a proxmox [16] VMM platform managing KVM instances, all we need to do is just changing this command line pattern to "qm set <vmId> -<vmDev> <vmmDev>".

Figure 5. Command line extraction.

V. VBS IMPLEMENTATION

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

A. *Workflows*

Fig. 6 demonstrates the workflows of main VBS operations. Due to space limitations the workflows of "describe-volumes" and "describe-snapshots" are not included; in short, 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 Fig. 6 in a similar way.

To handle a VBS 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 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. 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, because 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:

First, execute the “iscsiadm –discover” command to discover all targets on the volume server;

Second, 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;

Third, execute the “xm block-attach” command to attach the virtual iSCSI device to the VM instance with the given VM ID.

Figure 6. VBS workflows.

B. *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, such as dependencies and transactions, to maintain the metadata consistency. 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.

For VBS operations involving multiple execution steps, such as “attach-volume” and “detach-volume”, failure of any intermediate step will leave the whole system in an inconsistent status. To maintain the system level consistency, a “roll-back” mechanism is added to these multi-step VBS operations. 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.

VI. INTEGRATION WITH NIMBUS

Nimbus [3] is a cloud computing platform developed by University of Chicago. It adopts the concept of “virtual workspace” to manage VM instances, and provides services such as leasing of VMs and creation of virtual clusters [6] by manipulating Xen hypervisor [4] nodes. The construction of Nimbus components is shown in Fig. 7. At the backend, the workspace service implements the details of VM management, and is bridged to the frontend interfaces with the Resource Management API (RM API). At the frontend, Nimbus supports both a WRSF [18] Web services interface and an Amazon EC2 compatible Web services interface, so that users 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. Since Nimbus is an open source project under continuous development and updates, it provides an ideal platform for us to explore and test possible ways of integrating VBS with existing cloud systems. Our integration work with Nimbus has been successful, and we would like to use it as an example to demonstrate the strategy of integrating VBS with a cloud computing environment.

Figure 7. Nimbus components [20].

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 a specific Xen DomU ID and the hostname of the VMM node where the DomU is running. 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 Fig. 8.

The original Nimbus workspace service does not return the information we need. Therefore, 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.

Figure 8. Integration with Nimbus.

VII. 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, and their hardware configurations are given in Table I. The volume server is running Red Hat Enterprise Linux (RHEL) 5.3 and LVM [12] 2.0. The 4 147GB disks are grouped into two pairs, both in RAID1 configuration. All hard disks are managed by LVM, and new logical volumes are created on the RAID1s. The VMM node is running RHEL 5.3, Xen [4] 3.1.2 and LVM 2.0, and the 73GB hard disk is managed by LVM. One DomU is started on the VMM node, whose hardware configuration is also given in Table I.

TABLE I. HARDWARE CONFIGURATIONS OF VOLUME SERVER, VMM NODE, AND VM

	CPU	Memory	Disk
Volume server	4*Xeon 2.8G	512MB	4 * Seagate 147G 10K RPM SCSI (paired in RAID1)
VMM	2*Opteron 2.52G	1.5GB	1 Fujitsu 73G 10K RPM SCSI
VM	1 Operon 2.52G	256MB	4G disk image file (3.6G available)

For comparison purposes, we tested the performance of 4 different types of virtual disks on this test bed, as listed below:

VBS-LVM: a 5GB VBS volume is created and attached to the VM instance.

AoE-LVM: a 5GB logical volume is created 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.

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

Local-Image: the CentOS 5.2 disk image file is also tested.

An ext2 file system is created on all logical volumes, and an ext3 file system is created on the CentOS 5.2 disk image file. We used Bonnie++ [19] to complete our performance tests, and there were two reasons for this choice. First, the various file I/O operations tested by Bonnie++ make it possible for us to observe both the caching effects at different levels of the system and the actual transmission speed of each type of virtual disk. Second, Bonnie++ does file creation/deletion tests as well as read/write tests, so we can also observe and analyze the virtual disks' performance on file system metadata operations. We carried out the tests by mounting different disks to different directories in the VM, and running Bonnie++ under different directories. Due to the diversity of the virtual disk types, multiple levels of caching should be considered when choosing the file size used for tests. First, file system cache on the VM instance is utilized for access to files on all disks. Second, for access to VBS-LVM, caching is provided by the iSCSI target on the volume server. Third, for AoE-LVM, caching is provided by both the initiator on the VMM side, and the target on the volume server. Finally, Xen does not provide caching for physical devices attached to DomUs; but for disk image files used by VMs, the file system running in Dom0 will provide caching support. To avoid the impact from all these caching mechanisms, we ran Bonnie++ tests with three different file sizes: 512MB to double the memory size of the VM, 1GB to double the memory size of the volume server, and 3GB to double the memory size of the VMM node. Fig. 9-11 presents comparison of the transmission speed of VBS-LVM, AoE-LVM, and Local-LVM on each Bonnie++ operation in case of all file sizes. Due to space limit we 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). The default value of BUFSIZ is 8KB.

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

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

Figure 9. Performance comparison of VBS-LVM, AoE-LVM, and Local-LVM for file size of 512MB

Figure 10. Performance comparison of VBS-LVM, AoE-LVM, and Local-LVM for file size of 1GB

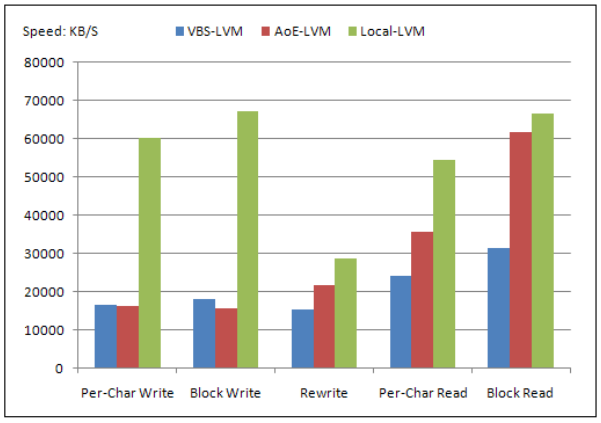

Figure 11. Performance comparison of VBS-LVM, AoE-LVM, and Local-LVM for file size of 3GB

There are several notable observations in Fig. 9-11. First, although the throughput of VBS-LVM is constantly less than Local-LVM, it is on the same magnitude. There are two reasons for the difference: (1) access to Local-LVM directly goes to the local disk on the VMM node, involving no overhead from the iSCSI protocol or network transmission; (2) according to our tests with the hdparm [22] utility on Linux, the hard disk on the VMM node can provide a faster I/O speed (about 83MB/s) than the RAID1s (about 65MB/s) on the volume server. Second, the throughput of VBS-LVM is close to AoE-LVM in general, indicating that VBS could be as efficient as an AoE based solution in the same environment. Specifically, it is constantly faster on write, and slower on read than AoE-LVM, implying that iSCSI is better optimized for write operations. Third, the difference on read operations between VBS-LVM and AoE-LVM gets larger when the file size goes up. This is because iSCSI only provides caching at the target side on the volume server, which has 512MB of memory. As a result, when file size goes over 512MB, AoE-LVM can benefit more from the cache at the initiator side, because the VMM node has 1.5GB of Memory.

Comparison between Local-LVM and Local-Image also uncovers some interesting trends, as shown in Fig. 12-14. The performance of Local-LVM is consistent, because in every test the only cache used is the file system cache inside the VM, which is constrained by the memory size of 256MB. In contrast, the throughput of Local-Image can reach to as high as 400-500MB/s when the file size is less than 1.5GB, but drops sharply to numbers similar to Local-LVM after that. This is because of the caching of the file system in Dom0. When the file size is less than the amount of memory most accesses to the disk image file can be handled through the cache; but after that such benefit is lost.

Fig. 15 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.

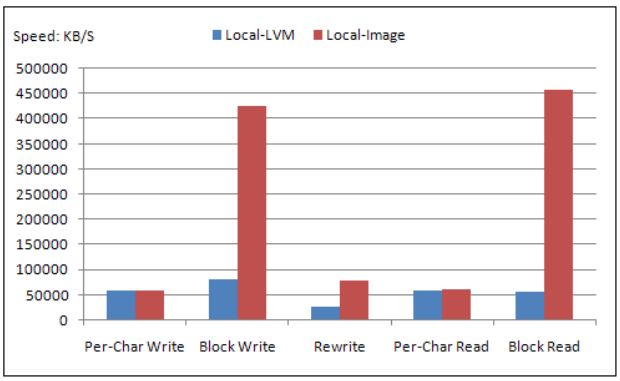

Figure 12. Performance comparison of Local-LVM and Local-Image for file size of 512MB.

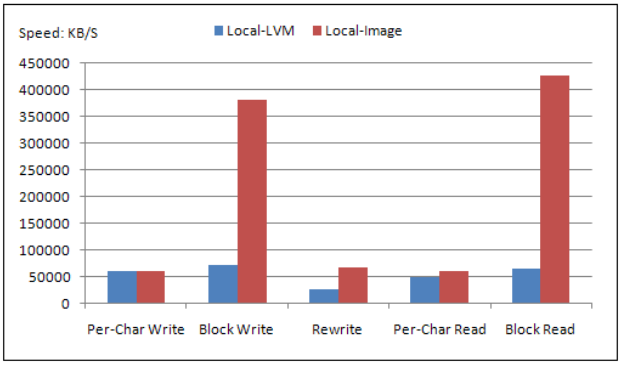

Figure 13. Performance comparison of Local-LVM and Local-Image for file size of 1GB.

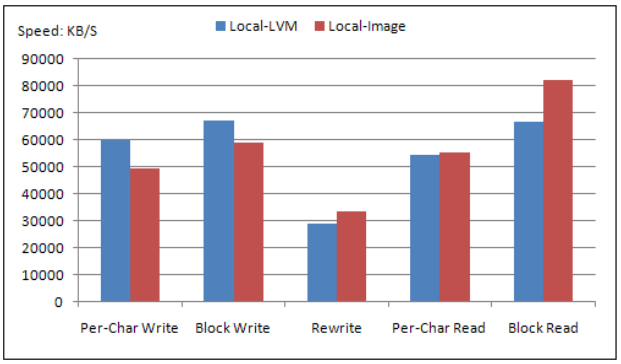

Figure 14. Performance comparison of Local-LVM and Local-Image for file size of 3GB.

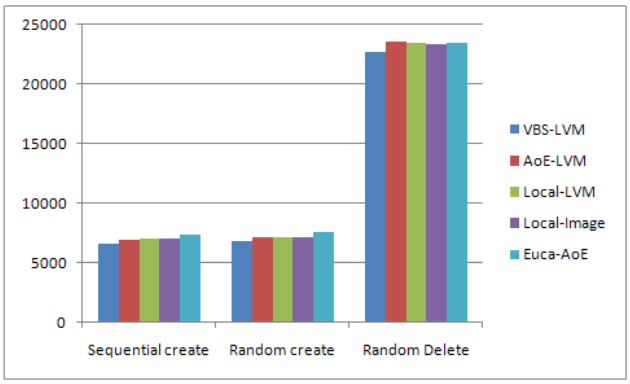

Figure 15. File creation and deletion comparison.

VIII. 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 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; moreover, based on a flexible Web services 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 environment configuration, and comparable to a local LVM [12] volume for accesses to files with modest sizes. 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, a user management component is needed in VBS to handle issues such as access control, space quota, and integration with cloud computing systems.

Third, no security mechanism has been applied in current VBS prototype yet. In order to protect users' data from various threats, two levels of security will need to be implemented: secure access to VBS Web services and secure data transmission over iSCSI.

Forth, 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.

Finally, current VBS implementation keeps only one copy for each logical volume. As a result, when the volume server and VM instances are deployed across wide area networks (WAN), it will be hard to guarantee the performance and availability of VBS volumes. To solve this problem, mechanisms such as volume duplication and synchronization could be considered in the future.

Acknowledgment

We would like to thank Prof. Geoffrey Fox for his guidance and comments about VBS use cases and other aspects; thank Kate Keahey, Jan-Philip Gehrcke, and the Nimbus group at University of Chicago for their help with the integration of VBS with Nimbus; thank Joe Rinkovsky for his help with the test bed configuration; thank Jun Wang for his help with the organization of test results. We also would like to thank all reviewers for their advice on the content and organization of this paper, and thank the program chairs of e-Science 2009, Paul Roe and David Wallom, for their help with our question about the reviews. Finally, many thanks to all of our colleagues at the Community Grids Lab of Indiana University.

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 services 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] Nimbus components, http://workspace.globus.org/vm/faq.html

[21] Adapter pattern, http://en.wikipedia.org/wiki/Adapter_pattern

[22] hdparm manual page, http://linux.die.net/man/8/hdparm