Building a Distributed Block Storage System for Cloud Infrastructure

Marlon Pierce

Pervasive Technology

Institute,

Indiana University

mpierce@indiana.edu

Xiaoming Gao

Pervasive Technology

Institute,

Indiana University

gao4@indiana.edu

Mike Lowe

University Information

Technology Services,

Indiana University

jomlowe@iupui.edu

Yu Ma

Pervasive Technology Institute,

Indiana University

yuma@indiana.edu

Geoffrey Fox

Pervasive Technology Institute,

Indiana University

yuma@indiana.edu

Abstract

The fast development of cloud infrastructures stimulates the demands for cloud block storage systems. Example implementations include Amazon’s Elastic Block Store (EBS) service, Eucaliptus’ implementation of EBS interface, and the Virtual Block Store (VBS) system developed by the Community Grids Lab of Indiana University. Compared with other solutions, VBS is more flexible, and can be readily extended to support different types of Virtual Machine Managers and Cloud platforms. However, due to its single volume server architecture, VBS has the problem of single point failure and low scalability. This paper presents our latest improvements to VBS, including a new distributed architecture based on the Lustre file system, new workflows, better reliability and scalability, etc. Based on VBS, we propose a new type of distributed file systems, which in certain use cases can help solve some critical challenges faced by traditional cluster file systems, such as metadata maintenance, small file access, and performance degradation when the number of concurrent processes increases. Our performance tests show that the new VBS implementation can provide better throughput in both single-attachment and multi-attachment situations than our old solution.

# 1. Introduction

The area of cloud computing has been a popular topic in both industry and academia in recent years, resulting in products such as Amazon Elastic Compute Cloud (EC2) [1], Eucalyptus [2], Nimbus [3], OpenNebula [4], and OpenStack [5]. These systems typically implement Infrastructure as a Service (IaaS) in the form of Web services, and dynamically allocate computing resources to users in the form of virtual machines (VM). In this paper we call software implementations of these cloud computing systems "cloud platforms", and corresponding physical deployments "cloud infrastructures". The development of cloud infrastructures stimulates researchers' interests in cloud storage systems, including distributed file systems, such as Amazon Simple Storage Service (S3) [6] and Hadoop Distributed File System (HDFS) [7], and block storage systems, such as Amazon Elastic Block Store (EBS) [8], the EBS implementation in Eucalyptus, which we will call “Eucalyptus EBS” for short, and the Virtual Block Store (VBS) [9] system developed by the Community Grids Lab of Indiana University.

Our research in this paper focuses on block storage systems, or more specifically, on the VBS system that we have been working on. VBS implements similar interfaces to EBS, and provides persistent virtual block volumes to cloud users. Users can attach their volumes to VM instances created in cloud infrastructures, and then use them as if they were local disks installed on their VM's. Compared with the storage provided by VM instance images, virtual volumes have the advantages of persistency and extendibility. Volumes' life times are independent of VM instances, and thus can be repeatedly detached from terminated VM's and attached to new VM's. Users can create more and more volumes on their demand, not limited by the resources of the VM instance images or Virtual Machine Manager (VMM) nodes. VBS is designed to work directly with VMM's, with the goal of more flexibility, meaning that it can be readily extended to support various VMM platforms and cloud computing systems. However, due to its single volume server architecture, VBS has the problems of single point failure and low scalability. Additionally, the initial VBS prototype did not take security and access control into account. In order to solve these problems, we have built a new distributed architecture for VBS, based on the Lustre file system [10]. By utilizing Lustre’s distributed storage solution and fail-over mechanism, we were able to achieve simpler implementation, as well as higher reliability, scalability, and throughput performance for VBS. We call the system built on this new architecture “VBS-Lustre”. We also added a set of new features to VBS-Lustre, including secure access to services, volume ownership management, and read-only volume sharing. This paper will present the design and implementation of VBS-Lustre, compare it with VBS, and discuss its merits and shortcomings. Moreover, based on VBS-Lustre, we propose a type of distributed file system, which we believe can help solve some critical challenges faced by traditional cluster file systems, such as metadata management and small file access performance.

The rest of this paper is organized as follows. Section 2 gives a brief description of VBS and Lustre. Section 3 and 4 describe the distributed architecture of and implementation details of VBS-Lustre. Section 5 presents the performance test results of VBS-Lustre and compares them VBS. Section 6 proposes a new type of distributed file system based on VBS-Lustre. Section 7 concludes and prospects our future work.

# 2. Related technologies

## 2.1. VBS

VBS is a virtual block storage system that we built for providing block storage services to cloud platforms. Service operations include volume/snapshot creation and deletion, volume attachment and detachment, and volume/snapshot description. Fig. 1 shows a typical use case of VBS. After creating VM instances in a cloud infrastructure, users can create virtual block volumes in VBS, and attach them to their VM's. After the attachment is complete, they will be able to access the volumes inside their VM's, as if they were local disks. Moreover, users can create snapshots of their volumes, which are static "copies" of the volumes at a certain time point, and then create new volumes based on the snapshots, so that they will all have the same initial state and data as the snapshots. Users can then attach the new volumes to different VM's, launch different processes of computation and generate different results. Storage on volumes is off-instance and persistent, because volumes have different life times from VM instances, and will be maintained by VBS even after VM images are destroyed.

Figure 1. **Use of VBS: volumes and snapshots [9]**

Figure 2. **VBS web service architecture [9]**

VBS is designed to work directly with VMM's, and is not coupled with any specific cloud platform. Fig. 2 shows its Web service architecture. There are two types of nodes -- one volume server and one or more VMM nodes, and three types of Web services -- VBS Web service, Volume Delegate Web service, and VMM Delegate Web service, in the architecture. On the volume server, Logical Volume Manager (LVM) [11] is used to manage volumes. On VMM nodes, Xen [12] is used to manage VM instances, and the technique of Virtual Block Device (VBD) is used to attach a block device in Dom0 to DomU instances. The iSCSI [13] protocol is used for creating virtual iSCSI devices in Dom0 of the Xen nodes, the access to which is translated by iSCSI to remote access to the volumes created on the volume server. The Volume Delegate service is located on the volume server, responsible for completing LVM and iSCSI operations. A VMM Delegate service is deployed on each VMM node, responsible for completing iSCSI and Xen VBD operations. The VBS Web service sits in the front end and answers VBS clients' requests, and satisfies them by coordinating the operations of Volume Delegate service and VMM Delegate service.

This architecture is simple, and can be readily extended to support other types of VMM's and various cloud platforms [9]. However, the single volume server can result in problems of single point failure and low scalability. The failure of the volume server will take the whole system down and cause constant disk access errors on related VM instances. The bandwidth of the volume server is shared among all volume attachments; as a result, the throughput performance of the volumes could degrade fast as the number of attachments increases.

To solve these problems, we have built a new distributed architecture with better reliability and scalability for VBS, as will be discussed in Section 3.

## 2.2. Eucalyptus EBS

Eucalyptus is a private cloud platform that implements the same interfaces as Amazon EC2, S3, and EBS. Similar to VBS, Eucalyptus EBS is also built on a single volume server architecture, and the main difference is that Eucalyptus uses ATA over Ethernet [14] to enable remote access to volumes, which limits its usability within Ethernet networks. Therefore, it also suffers from the problems of single point failure and low scalability. For example, [15] reports EBS performance degradation in Eucalyptus in cases of multiple volume attachments, and [16] presents low performance results even in single attachment configurations. We guess Amazon EBS is built on a distributed architecture, but little is known about its actual design and implementation.

## 2.3. The Lustre file system

The Lustre file system is a well known open source cluster file system developed by Oracle. For simplicity, we also call it Lustre in this paper. Lustre has been deployed on many of the world’s largest and fastest high performance computing (HPC) clusters and supercomputers, such as the Jaguar supercomputer at Oak Ridge National Laboratory (ORNL), and Big Red at Indiana University.

Lustre uses a highly scalable distributed storage architecture, as shown in Fig. 3, and can support up to tens of thousands of client systems, petabytes (PB) of storage, and provide an aggregate I/O throughput of hundreds of gigabytes per second (GB/sec). There are four types of roles in this architecture: clients, metadata servers (MDS), Object Storage Servers (OSS), and Object Storage Targets (OST). MDS's manage the metadata of all files in the file system, and answer all clients’ namespace operation requests. OSS's are responsible for storing the actual data of files, and OST’s are physical storage devices connected to OSS's, such as disk arrays or storage area networks. A file system can have one or more MDS's and one or more OSS's, and each OSS can connect to one or more OST's. The networking layer of Luster can support various network connections, including Elan, Myrinet, InfiniBand, and TCP/IP.



Figure 3. **Lustre architecture [10]**

The following features of Lustre make it attractive for being used to build a distributed architecture for VBS:

(1) Distributed file storage. The Lustre file system uses an object-based storage model, and stores data in the form of objects on OST. File data is striped across objects on different OST's, and users can configure parameters such as stripe-size and stripe-count to achieve best performance. The capacity of a Lustre file system equals the sum of the capacities of OST's, and the aggregate available bandwidth equals the sum of the bandwidth offered by OSS's to clients. Users can extend storage capacity by dynamically adding more OSS's and OST's. Data striping balances work load among OSS's, leading to high I/O throughput, and excellent scalability as the number of client increases;

(2) High reliability mechanisms. As shown in Fig. 3, MDS's and OSS's can both be compared into failover pairs with shared storage, so that when one node in a pair fails, the other one will take over its work load until it recovers. OST's can be configured as RAID to deal with disk failures. These mechanisms can be leveraged to improve the reliability of VBS.

# 3. VBS-Lustre architecture

Leveraging the advantages of Lustre, we built the distributed architecture in Fig. 4 to solve the problems of VBS, and we call the new-architecture-based system "VBS-Lustre". In this architecture, a Lustre file system is used as the backend for storing all the volumes, and each volume or snapshot is implemented as a file in Lustre. We call the file corresponding to a volume or snapshot a “volume file” or a “snapshot file”. Therefore, all OSS's in Lustre are volume servers for VBS-Lustre, and there can be multiple Volume Delegate services deployed. However, Volume Delegate services don't have to be located on OSS's; they can be running on any Lustre client node. Every VMM node is configured as a Lustre client, and still has one VMM Delegate service running on it. The iSCSI protocol is no more used, since VMM nodes can directly access volumes through file system interfaces. We change the name of the frontend Web service in VBS-Lustre to "VBSLustre service", and this service can be deployed anywhere, as long as it can communicate with Volume Delegate services and VMM Delegate services. A database is used to manage volume metadata, including the mapping between volume ids and Lustre file paths, attachment information, etc. It is only accessed by the VBSLustre service. As in VBS, the VBSLustre service completes clients' volume operation requests by coordinating the actions of Volume Delegate services and VMM Delegate services. Details about how the coordination happens will be covered in Section 4.

Figure 4. **VBS-Lustre architecture**

Compared with the architecture of VBS, this architecture has the following advantages:

(1) Since volumes are implemented as files, volume data is striped across objects stored on different OST's. Therefore, the maximum volume size is not limited to the capacity of any single OST or OSS. Moreover, since Lustre is optimized for I/O access to large files, and volume sizes are usually on the level of tens or hundreds of gigabytes, we can get much better volume throughput than in VBS, as will be shown in Section 7;

(2) Accesses to volumes are now distributed across all OSS's, so the aggregate throughput is not limited to any single volume server, and the whole system is much more scalable than VBS;

(3) Leveraging Lustre's high availability mechanism, volume servers (i.e., OSS's) can be configured into failover pairs with shared storages, so that the failure of any single volume server does not have a significant impact on the whole system. Moreover, since volume storage is distributed across different OSS's, even the failure of a pair of volume servers is not necessarily a fatal problem for the whole system. To avoid single point failure of the VBSLustre service, multiple service instances can be deployed on different nodes, and they can share the same database. The reliability of the database can be guaranteed by utilizing mature database reliability technologies in industry.

# 4. VBS-Lustre implementation

This section presents the implementation details of VBS-Lustre, including workflows, security and access control, and read-only sharing of volumes.

## 4.1. Workflows

Figure 5. **VBS-Lustre workflows**

Workflows define the coordination between Web services in VBS-Lustre for competing clients' volume operation requests. The volume operations provided by VBS-Lustre are exactly the same as VBS, but due to the new architecture, the workflows of most operations are different, as shown in Fig. 5. Most workflows in VBS-Lustre are simpler, as explained in the following:

(1) Create-volume and describe-volume. After receiving a client’s request for creating a new volume of a given size, the VBSLustre service will first generate a new volume id and a path for the corresponding volume file, and then invoke a Volume Delegate service to create the new volume file. The Volume Delegate service will first check if there is enough space in the Lustre file system for the new file. If the answer is yes, the Volume Delegate service will first return a temporary success message, and then start a new thread to complete the creation of the new file. Upon receiving the success message, the VBSLustre service will create a new record of metadata for the new volume with a status of “pending”, and return this record to the client. If there is not enough space for the new volume, the Volume Delegate service will return a failure message to the VBSLustre service, which will then return a failure result to the client. When starting the new file creation thread, the Volume Delegate service checks if the new volume should be created based on a snapshot. If the path of a snapshot is given, the thread will execute the “cp” command to copy the snapshot file to the volume file path; otherwise, the thread will execute the “dd” command to fill the new volume file with zeroes until the file size reaches the requested volume size. After the thread finishes, the Volume Delegate service will invoke the VBSLustre service to update the status of the new volume. If the command succeeds, the status will be set to “available”; otherwise to “failure : cmd error”, and a detailed error message will be sent to VBSLustre service and logged. After the creation of a volume, the client can call the describe-volume operation on it, and the VBSLustre service will return related metadata.

(2) Create-snapshot and describe-snapshot. The workflows for snapshot creation and description are similar to those of volumes. The only difference is that the new file creation thread always executes the “cp” command to copy the volume file to the path of the new snapshot file.

(3) Attach-volume. An attach-volume request specifies which volume should be attached to which VM, and which VMM is managing the VM. Upon receiving a request, the VBSLustre service will invoke the corresponding VMM Delegate service to execute the “xm block-attach” command to attach the volume file as a block device on the requested VM. If the command succeeds, the VBSLustre service will add an attachment metadata record for the volume, and return the attachment information to the client; otherwise a failure message is returned. After a volume is attached, the response to a describe-volume operation on it will contain its attachment information.

(4) Detach-volume. The workflow of the detach-volume operation is similar to attach-volume. The only difference is that the command executed by the Volume Delegate service is “xm block-detach”.

(5) Delete-volume and delete-snapshot. The workflows of the delete-volumes and delete-snapshot operations are similar. Upon receiving a request, the VBSLustre service will invoke a Volume Delegate service to execute the “rm –f” command to delete the corresponding volume or snapshot file. If the command succeeds, the VBSLustre service will delete the metadata of the volume or snapshot and return success to the client; otherwise a failure message is returned.

## 4.2. Security and access control

In VBS-Lustre, Web service accesses are protected with https channels; users are authenticated through public key authentication, and are only authorized to take operations on volumes and snapshots they created.

Web services in VBS-Lustre are deployed with the Axis2 [17] technology, and public key authentication is implemented by applying the rampart module. New users are created by adding their certificate to the trusted certificate store of the VBS-Lustre service, and the subject names contained in the certificates are added as user ids. When the VBSLustre service is invoked by a client, it will first get the certificate of the client through the “Message Context” provided by Axis2, find the subject name as the user id, and then check if the volume or snapshot that the client is trying to operate on is created by the same user id. If not, an error message will be returned to the client.

## 4.3. Read-only volume sharing

Here by “read-only volume sharing”, we mean attaching a volume to multiple VM instances at the same time. This is not supported in either Amazon EBS or Eucalyptus EBS, but is actually a very useful feature in many cases, especially when the shared volume is large, and it takes a significant amount of time and space to duplicate it. For example, in the QuakeSim [18] project which the authors are also involved in, there are situations where we have a large set of GPS data and want to perform different types of analysis on it. In this case, we can deploy the processes for different analysis on different VM’s, which share a common volume containing the data set in read-only mode, and attach a separate volume in writable mode to each VM. After the attachment is done, we can start the processes on different VM’s at the same time, and direct their output to the writable volumes.

VBS-Lustre supports read-only volume sharing by adding an “attach-mode” parameter to the attach-volume operation, and adding this information to attachment metadata. When a client tries to attach an already attached volume to another VM, the VBSLustre service will check if the attach-modes of both the existing attachment(s) and the new operation are read-only, and will only allow the operation to continue if the check is passed. On the VMM node, the VMM Delegate service completes a read-only attachment by executing the “xm block-attach” command with an argument of “r”, instead of “w”. The distributed volume storage architecture of VBS-Lustre can provide good throughput to concurrent reads from multiple VM instances.

# 5. Preliminary performance test

To test the performance improvement of VBS-Lustre, we used the same machines as we did when testing the performance of VBS in [9]. The volume server used in VBS tests was also used as the MDS of Lustre in VBS-Lustre tests, which has 4 Intel Xeon 2.8G CPUs, 512MB of memory, 1 Lsi Logic 40GB Ultra320 SCSI hard disk, and 2 Seagate 147GB 10K RPM Ultra320 SCSI hard disks. It is running Red Hat Enterprise Linux (RHEL) 5.3 and all hard disks are managed by LVM 2.0. The VMM node used in VBS tests was also used as a VMM node in VBS-Lustre tests, which is configured with 2 AMD Opteron 2.52G CPUs, 1.5GB of memory, and one Fujitsu 73GB 10K RPM Ultra320 SCSI hard disk, and is running RHEL 5.3, Xen 3.1.2 and LVM 2.0. The hard disk is managed by LVM.

We added three more nodes with the same configurations as this VMM node to the VBS-Lustre testbed. One of them was used as another VMM node, and the other two were used as OSS's of Lustre. One DomU is started on each VMM node, with 1 AMD Opteron 2.52G CPU, 256MB of memory, and a CentOS 5.2 disk image with 4GB of disk space. Lustre 1.8 is installed on all nodes. The MDS uses a 20GB LVM logical volume to store Lustre namespace metadata. Each OSS is attached to 4 OST's, each of which is a 10GB LVM logical volume. Therefore, there are 8 OST's in total, providing an aggregate capacity of 80GB. A stripe size of 4MB is used in Lustre, and volume files are striped across all OST's.

We created two 5GB volumes in VBS and two 5GB volumes in VBS-Lustre. An ext2 file system is created on each volume, and we tested the performance of VBS and VBS-Lustre in both single-volume situations and two-volume situations. In single-volume situations, one VBS volume and one VBS-Lustre volume was tested respectively by being attached to a VM. In two-volume situations, two VBS volumes and two VBS-Lustre volumes were tested respectively by being attached to two VM's. Bonnie++ 1.03e [19] was used to complete the tests, and a file size of 4GB was used in each test to exceed the memory cache size at all possible layers, including on VM, on VMM, on the VBS volume server, and on Lustre OSS's. In each test, the testing process was repeated 10 times to avoid accidental interruptions. In two-volume situations, the testing processes on the two VM's were started at the same time.

Table 1 shows the throughput difference between VBS and VBS-Lustre. Numbers are average values of 10 test runs. The average values of two-volume tests are computed by dividing the average aggregate throughput by 2. As can be seen, there is not a big difference for the per-char operations, because they are CPU-intensive. But for the block write and read operations, VBS-Lustre out-performs VBS by 55%-100% in both single-volume and two-volume tests. Due to the single volume server architecture, the aggregate throughput of VBS cannot exceed the bandwidth of the volume server. On the contrary, the aggregate throughput of VBS-Lustre reaches more than 150MB/s in the two-volume test, although there is a 10%-15% performance degradation of the average numbers, due to variable degrees of competition for accessing the same OST’s between the two VM’s. Limited by resources in our test bed, we were not able to carry out tests in a larger scale, but it is obviously expectable that VBS will have more significant performance degradations once the number of attachment is over 3, and there will be a larger difference between VBS and VBS-Lustre.

Table 1. **VBS and VBS-Lustre throughput (KB/s)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test type | Per-char write | Block write | Per-char read | Block read |
| VBS  single-volume | 50930 | 54633 | 27802 | 32094 |
| VBS-Lustre single-volume | 54470 | 85628 | 41607 | 69162 |
| VBS  two-volume | 46594 | 50196 | 25789 | 31657 |
| VBS-Lustre  two-volume | 49656 | 75236 | 40625 | 59322 |

VBS-Lustre experiences more fluctuations of performance in the two-volume test, as shown in Fig. 8, but even the lowest throughput values are still above the average numbers of VBS.

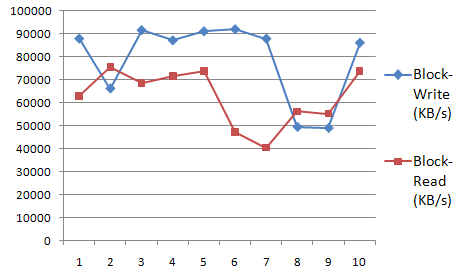


Figure 8. **VBS-Lustre throughput fluctuations (KB/s)**

Table 2 presents the metadata performance of VBS-Lustre in both single-volume and two-volume tests. The numbers are average values of 10 runs. In the two-volume test there is only a slight degradation, and the aggregate metadata operation throughput is almost twice as high as in the single-volume test.

Table 2. **VBS-Lustre metadata performance (files/s)**

|  |  |  |  |
| --- | --- | --- | --- |
| Test type | Sequential create | Random create | Random delete |
| single-volume | 6662 | 6758 | 23340 |
| two-volume VM1 | 6594 | 6692 | 22875 |
| double-volume VM2 | 6588 | 6737 | 23294 |
| double-volume Agg. | 13182 | 13428 | 46169 |

# 6. Proposing a new type of distributed file system based on VBS-Lustre

Traditional cluster file systems are facing many challenges, such as metadata maintenance, small file access, and performance degradation when the number of concurrent processes increases. For example, before Lustre 2.0, there is be only one active MDS in a Lustre file system, which could finish 3000-15000 metadata operations per second [10]. When the number of concurrent processes gets large, the MDS could become a performance bottleneck of the whole cluster. Lustre will support clustered MDS's from version 2.0, so that the workload of metadata operations is distributed among multiple MDS's, but still the group of MDS's are shared among all processes. Besides, like most cluster file systems, Lustre is optimized for large file I/O's, and the performance of the whole system degrades when there is a large number of small files and most accesses are targeting small files. Based on VBS-Lustre, we propose a new type of distributed file system, which can provide the same functionalities as cluster file systems in certain use cases, and help solve these challenges by limiting the scope of competition for resources to a smaller number of concurrent processes. We call this new type of distributed file system "VBS File System" (VBSFS), whose architecture is shown in Fig. 7.

Figure 6. **VBS File System**

In VBSFS, all nodes can be attached to volumes in VBS-Lustre. Each user of VBSFS is provided with a private volume, which is used to create a file system as the user's home directory. VBSFS also provides a public volume which contains a file system where all public software and data are installed. The public volume is attached to all nodes in read-only mode, and updated by system administrators during maintenance time. When a user is trying to run a process on a node, that node will first be attached to the user's private volume, so that the process can access all the files in his/her home directory. Since VBS-Lustre does not support writable volume sharing among multiple nodes, and volume level consistency does not guarantee file system level consistency, VBSFS cannot handle the situations where processes on different nodes are trying to write to the same volume. Specifically, it cannot handle the use cases where one or more users' processes running on different nodes are trying to access a specific user's home directory. But for the cases VBSFS can handle, such as multiple processes trying to access one specific user's home directory on the same node, it has the following advantages:

(1) The workload of Lustre MDS is tremendously relieved -- it only needs to maintain one volume file for each user, and the metadata of volume files are mostly stable;

(2) User processes' metadata operations happen within their private virtual volumes, which are actually translated to I/O operations to volume files in Lustre. Lustre’s caching and parallel I/O mechanisms can make these operations much more efficient than the metadata operations taken on Lustre MDS. Therefore, VBSFS can achieve a much larger aggregate metadata throughput than Lustre. Table 2 in Section 5 has shown an example of this merit in a two-volume situation, and we could obviously expect even higher performance when there are a larger number of attached volumes;

(3) I/O operations to small files in VBSFS are translated to I/O’s to sections of big volume files in Lustre, and thus can benefit from the caching and parallel I/O mechanism of Lustre, which are specially optimized for access to large files;

(4) In Lustre, every process has to go through the MDS for synchronization, so the concurrency domain is the whole cluster. In VBSFS, the concurrency domains of users' processes are separated by the scope of the virtual volumes they access, mostly only the users' private volumes. Processes only compete with other processes which are accessing the same virtual volumes, and the synchronization is handled by the driver modules of the on-volume file systems, which are running on client nodes.

While traditional distributed file systems are trying to separate concurrency domains by namespace partitions or server nodes [10][20][21], VBSFS is actually trying to separate concurrency domains by users. We believe this idea is valuable for solving various challenges to current cluster file systems. Moreover, the ideas of VBSFS can be combined with cluster file systems to support all types of use cases. For example, in a Lustre file system, each user can have both a traditional home directory and a private volume file. The volume file can be configured as a loop device and mounted on any node the user is running processes. While the private volume file is accessed in the same manner as in VBSFS, the traditional home directory can be used to support concurrent accesses from different nodes.

# 7. Conclusion and future work

VBS is an open source block storage system that can provide persistent and off-instance block storage services to cloud infrastructures. It supports similar Web service interfaces to Amazon EBS and Eucalyptus EBS, but is designed for better flexibility and can be readily extended to support various VMM and cloud platforms. However, due to its single volume server architecture, it has the problems of single point failure and low scalability. To solve these problems, we built a new distributed architecture based on the Lustre file system, and we call the system based on this new architecture “VBS-Lustre”. VBS-Lustre does not have the problem of single point failure, and provides higher I/O throughput and scalability through its distributed volume storage mechanism. VBS-Lustre has simpler workflow implementations and many new features, including Web service security, user access control, and read-only volume sharing. Our preliminary performance tests show that VBS-Lustre can provide higher throughput than VBS in both single attachment and multiple attachments scenarios. Based on VBS-Lustre, we proposed a new type of distributed file system, which in a certain use cases can help solve some critical challenges faced by traditional cluster file systems, such as metadata maintenance and small file access. There are several directions that we will continue to work on in the future:

First, we need to test the performance of VBS-Lustre on larger scales of server and attachment numbers, in order to find more potential problems.

Second, the creation of new volumes and snapshots are completed with the “dd” and “cp” command, which could be a long process for large volumes. We will consider modifying Lustre to invent faster solutions.

Third, new users are now created by directly adding their self-signed certificates to the services’ trusted certificate store. We will add a certificate authority (CA) to VBS-Lustre and implement user creation by signing new user’s certificate with this CA.

Fourth, although Lustre supports commodity hardware as OSS’s and OST’s, it does not provide solutions for their reliability. Therefore, we need to find a good reliability mechanism for commodity hardware in order to use them in VBS-Lustre.

# References

[1] Amazon EC2, http://aws.amazon.com/ec2/.

[2] Eucalyptus, http://open.eucalyptus.com/.

[3] The Nimbus project, http://workspace.globus.org/.

[4] OpenNebulla, http://www.opennebula.org/.

[5] OpenStack, http://openstack.org/.

[6] Amazon S3, http://aws.amazon.com/s3/.

[7] The Apache Hadoop project, http://hadoop.apache.org/.

[8] Amazon EBS service, http://aws.amazon.com/ebs/.

[9] X. Gao, M. Lowe, Y. Ma, M. Pierce, "Supporting Cloud Computing with the Virtual Block Store System", *Proceedings of e-Science 2009*, Oxford, UK, Dec. 2009.

[10] Lustre file system white paper, Oct. 2008.

[11] LVM, http://tldp.org/HOWTO/LVM-HOWTO/.

[12] The Xen hypervisor, http://www.xen.org/.

[13] The iSCSI protocol, http://tools.ietf.org/html/rfc3720.

[14] S. Hopkins, B. Coile, “The ATA over Ethernet Protocol Specification”, *Technical Report*, The Brantley Coile Company, Inc., Feb. 2009.

[15] http://open.eucalyptus.com/forum/poor-performance-ebs.

[16] Jeffrey Shafer, "I/O Virtualization Bottlenecks in Cloud Computing Today", *Proceedings of the Second Workshop on I/O Virtualization*, Pittsburgh, PA, USA, Mar. 2010.

[17] Apache Axis2, http://ws.apache.org/axis2/.

[18] The QuakeSim project, http://quakesim.jpl.nasa.gov/.

[19] Bonnie++, http://www.coker.com.au/bonnie++/.

[20] F. Schmuck, R. Haskin, "GPFS: A Shared-Disk File System for Large Computing Clusters", *Proceedings of the 1st USENIX Conference on File and Storage Technologies*, Monterey, CA, USA, Jan. 2002.

[21] J. Xing, J. Xiong, N. Sun, J. Ma, "Adaptive and Scalable Metadata Management to Support A Trillion Files", *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, Portland, OR, USA, Nov. 2009.