Building a Distributed Block Storage System for Cloud Infrastructure

Xiaoming Gao
Indiana University
gao4@indiana.edu

Yu Ma
Indiana University
yuma@indiana.edu

Marlon Pierce
Indiana University
mpierce@cs.indiana.edu

Mike Lowe
Indiana University
jomlowe@iupui.edu

Geoffrey Fox
Indiana University
gcf@indiana.edu

Abstract

The development of cloud infrastructures has stimulated interest in virtualized block storage systems, exemplified by Amazon Elastic Block Store (EBS), Eucalyptus' EBS implementation, and the Virtual Block Store (VBS) system. Compared with other solutions, VBS is designed for flexibility, and can be extended to support various Virtual Machine Managers and Cloud platforms. However, due to its single-volume-server architecture, VBS has the problem of single point of failure and low scalability. This paper presents our latest improvements to VBS for solving these problems, including a new distributed architecture based on the Lustre file system, new workflows, better reliability and scalability, and read-only volume sharing. We call this improved implementation VBS-Lustre. Preliminary tests show that VBS-Lustre can provide both better throughput and higher scalability in multiple attachment scenarios than VBS. VBS-Lustre could potentially be applied to solve some challenges for current cluster file systems, such as metadata management and small file access.

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 Storage as a Service such as Amazon Simple Storage Service (S3) [6], distributed file systems such as 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 and specifically on our VBS system. VBS implements similar Web service (WSDL) 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 the volumes as if they were local disks installed on their VMs.

Different from S3, VBS does not transport data through Web service invocations. Web services are only used for creating and attaching virtual volumes, and data storage is completed in the form of file systems or databases created on the volumes. Compared with HDFS, VBS is different in the sense that it gives users direct control over virtual block devices, which can be utilized in various ways – e.g., users can deploy a HDFS on a virtual cluster of VMs that are using VBS volumes as their storage devices. Finally, compared with the storage provided by VM instance images, virtual volumes have the advantages of persistency and extendibility. Volumes have life times that are independent of VM instances, and thus can be repeatedly detached from terminated VMs and attached to new VMs. Users can create more volumes on demand, not limited by the resources of the VM images or Virtual Machine Manager (VMM) nodes.

Going Beyond VBS: VBS is designed to work directly with VMMs, with the goal of more flexibility, meaning that it can be readily extended to support various VMM and cloud platforms. However, due to

its single-volume-server architecture, VBS has the problems of single point of failure and low scalability. To solve these problems, we need to build a new distributed storage architecture for VBS, either by extending its current architecture with multiple volume servers and implementing proper mechanisms for integrated storage management and high reliability, or by utilizing existing distributed storage technologies such as distributed file systems. We find the latter way preferable, since it allows us to take advantage of the storage management and reliability mechanisms provided by existing systems, and concentrate on the block storage service implementations. We choose to build a new distributed architecture for VBS based on the Lustre file system [10], because: (1) it has been successfully deployed on many top supercomputers in the world, providing high I/O performance and excellent scalability and reliability; (2) it is an open source project, and thus we can make necessary modifications to it to support the functionality of VBS. We call the system built on the new architecture "VBS-Lustre". By leveraging Lustre's distributed storage solution and fail-over mechanism, VBS-Lustre is able to achieve simpler implementation, as well as higher reliability, scalability, and I/O throughput than VBS. We also have 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.

2. Previous work and related technologies

2.1. VBS

VBS was our initial approach to building a block-level cloud storage system. It provides a set of block volume operations to cloud users, including 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 such as Nimbus, users can create virtual block volumes in VBS, and attach them to their (usually Xen-based) VMs. After the attachment is complete, they will be able to access the volumes from their VMs as if the volumes 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 VMs, 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.

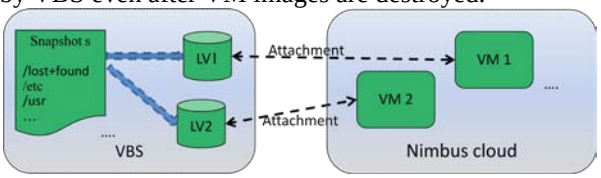

Figure 1. **Use of VBS: volumes and snapshots [9]**

VBS is designed to work directly with VMMs, 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 enabling remote access from VMM nodes to logical 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.

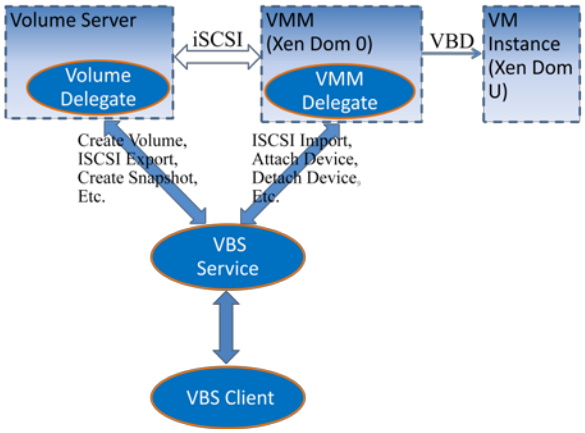

Figure 2. **VBS web service architecture [9]**

This architecture is simple, and can be readily extended to support other types of VMMs and various cloud platforms [9]. However, the single volume server can result in problems of single point of 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 I/O throughput 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. Based on the application scale of Amazon EBS, we hypothesize that it 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 currently owned by Oracle. 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, scale to 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 Server (MDS), Object Storage Servers (OSS), and Object Storage Targets (OST). MDS manages the metadata of all files in the file system, and answer all clients' namespace operation requests. OSSs are responsible for storing the actual data of files, and OSTs are storage devices connected to OSSs, such as disk arrays or storage area networks. A file system can have one MDS and one or more OSSs, and each OSS can be connected to one or more OSTs. The networking layer of Luster can support various network connections, including Elan, Myrinet, InfiniBand, and TCP/IP.

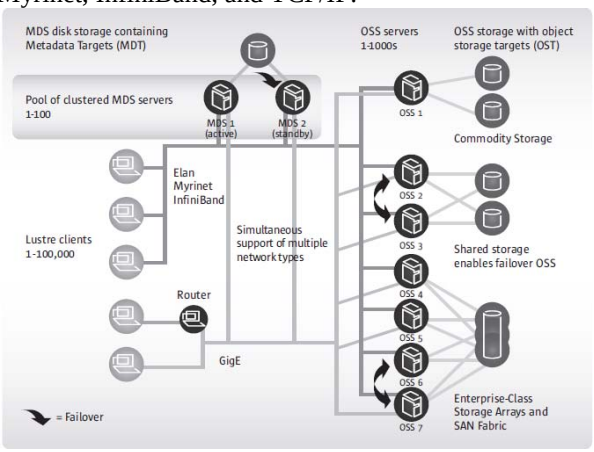

Figure 3. **Lustre architecture [10]**

The following features of Lustre make it attractive for being used as the basis for building a distributed architecture for VBS:

(1) *Distributed file storage:* Lustre uses an object-based storage model, and stores data in the form of objects on OSTs. File data is striped across objects on different OSTs, 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 OSTs, and the aggregate available bandwidth equals the sum of the bandwidth offered by OSSs to clients. Users can extend storage capacity by dynamically adding more OSSs and OSTs. Data striping balances work load among OSSs, leading to high I/O throughput and excellent scalability as the number of client increases;

(2) *High reliability mechanisms:* as shown in Fig. 3, MDSs and OSSs can both be configured 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. OSTs can be configured as RAID to handle disk failures better. These mechanisms can be utilized to improve the reliability of VBS.

3. VBS-Lustre architecture: a new approach

Leveraging the advantages of Lustre, we have built the distributed architecture as shown in Fig. 4 to solve the problems of VBS. We call the new 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. We call the file corresponding to a volume or snapshot a "volume file" or a "snapshot file". Therefore, all OSSs 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 OSSs; 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 longer 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.

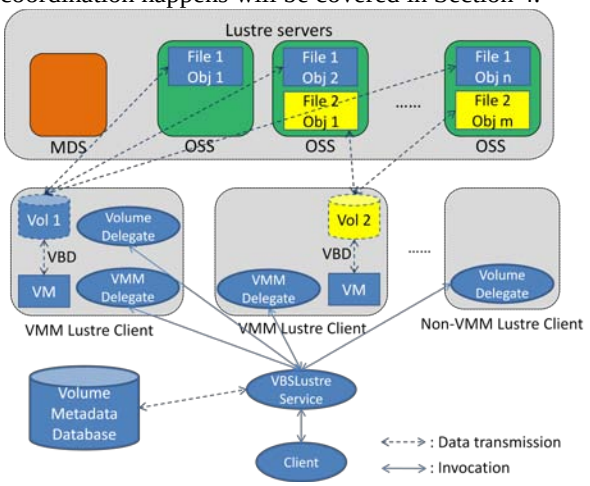

Figure 4. **VBS-Lustre architecture**

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 OSTs. 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, VBS-Lustre can get better volume throughput than VBS, as will be shown in Section 5;

(2) Accesses to volumes are now distributed across all OSSs, 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., OSSs) 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 OSSs, even the failure of a pair of volume servers is not necessarily a fatal problem for the whole system. To avoid a single point of 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

4.1. Workflows

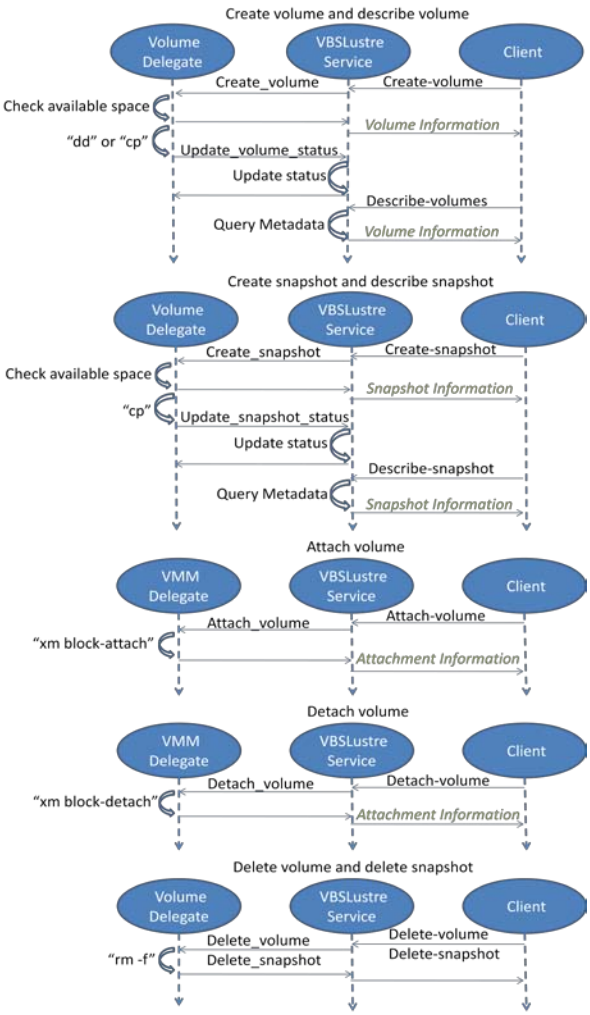

Figure 5. **VBS-Lustre workflows**

Workflows define the coordination between Web services in VBS-Lustre for competing clients' volume operation requests. The Web service APIs provided by VBS-Lustre are exactly the same as VBS, but due to the new architecture, the implementations 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 main 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 hosting 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 onto 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 main 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 Apache Axis2 [17] technology, and public key authentication is implemented by applying the Apache Rampart module. New accounts are created by adding users' certificates 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 potentially 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, there are situations where we have a large set of Global Positioning System (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 VMs, 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 VMs 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 complete initial validation of VBS-Lustre and compare it with VBS, we set up the test beds as shown in Fig. 6. In the VBS-Lustre test bed, Lustre 1.8 is installed on 1 MDS and 4 OSSs. The MDS 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. Each OSS has 2 AMD Opteron 2.52G CPUs, 2GB of memory, and 1 IBM 73GB 10K RPM Ultra320 SCSI hard disk. Each VMM has the same hardware configuration as an OSS, except the memory size is 1.6GB. All machines are running Red Hat Enterprise Linux (RHEL) 5.3 and using LVM 2.0 to manage the disks. A 20GB logical volume is created on the MDS and used for metadata storage. A 25GB logical volume is created on each OSS and used as an OST, leading to an aggregate storage space of 100GB. A stripe size of 4MB is used in Lustre, and each volume file is striped across 2 OSTs. Xen 3.1 is installed on both VMM nodes, and 1 VM is created on each VMM, which has 1 AMD Opteron 2.52G CPU, 256MB of memory, and a 4GB CentOS 5.2 disk image. The same VMMs and VMs are used in the VBS test bed and local volume test bed. The volume server in the VBS test bed has the same configurations as an OSS in the VBS-Lustre test bed. All nodes are connected to a 1Gb Ethernet LAN.

We created two 5GB volumes in VBS, two 5GB volumes in VBS-Lustre, and one 5GB LVM volume on the local disk of each VMM node – we call it a "local volume". An ext2 file system is created on each volume, and we tested the performance of VBS, VBS-Lustre, and local volumes in both single-volume and two-volume situations. In single-volume situations, one VBS volume, one VBS-Lustre volume, and one local volume were tested respectively by being attached to a VM. In two-volume situations, two VBS volumes, two VBS-Lustre volumes, and two local volumes were tested respectively by being attached to two VMs. 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 OSSs. A block size of 4KB is used in the block read/write tests. In each test, the testing process was repeated 10 times to alleviate the impact of accidental interruptions. In two-volume situations, the testing processes on two VMs were started at the same time.

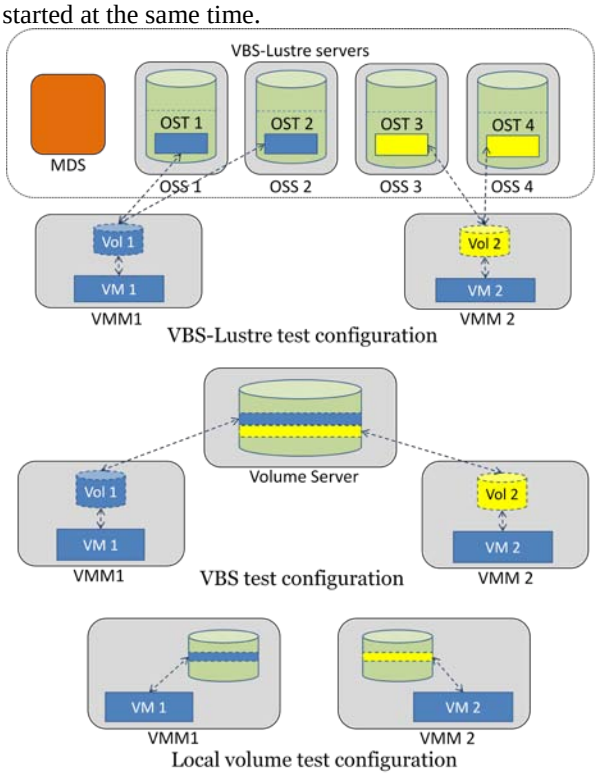

Figure 6. **Test bed configurations**

Fig. 7 shows the throughput difference between VBS, VBS-Lustre, and local volumes. 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, leveraging the distributed volume storage architecture, VBS-Lustre out-performs VBS on all kinds of operations in the single-volume test. VBS-Lustre also performs better than local volumes on block operations, although not as good on per-char operations, mainly because these operations are CPU-intensive, and the overhead of VBS-Lustre on them exceeds the benefits of distributed volume storage. Moreover, while VBS experiences a performance degradation of ~50% or

even more in the two-volume test, VBS-Lustre is able to make use of the bandwidth and disks on all related nodes in an aggregated way, and provide both consistent per-volume performance and aggregate throughput that is not limited by any single server. VBS-Lustre has a slight throughput degradation on block operations in the two-volume tests, but mainly due to different hardware performance on OSS nodes.

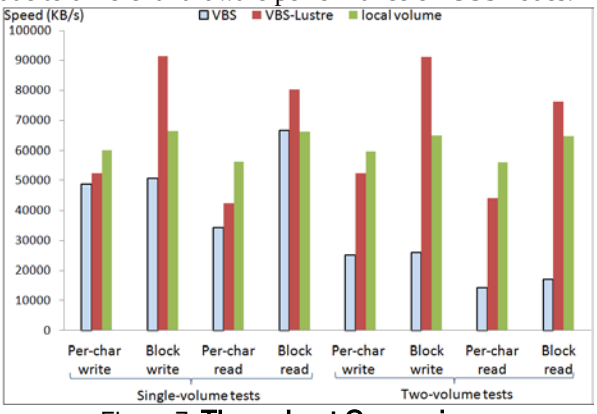

Figure 7. **Throughput Comparison**

Since a major use case of VBS-Lustre volumes is to host file systems, file system metadata operation performance is an important concern in our tests. Table 1 presents the metadata performance of VBS-Lustre in both single-volume and two-volume tests. The numbers are average values of 10 runs. As can be seen, the difference between the two situations is trivial, and the aggregate metadata operation throughputs in the two-volume tests are almost twice as high as in the single-volume tests.

Table 1. **VBS-Lustre metadata performance (files/s)**

Test type	Sequential create	Random create	Random delete
single-volume	6629	6654	23211
two-volume VM1	6510	6724	23312
two-volume VM2	6565	6771	23274
two-volume Agg.	13075	13495	46586

6. Conclusion and future work

The primary contribution of this paper is the description and initial evaluation of VBS-Lustre, an extension of our previous VBS system. Compared with VBS, the most significant difference with VBS-Lustre is its distributed architecture based on the Lustre file system. Leveraging Lustre's distributed storage and high availability mechanisms, VBS-Lustre avoids the problem of single point of failure, and provides higher I/O throughput and better scalability than VBS. VBS-Lustre also 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.

There are two directions that we will continue to work on in the future. On one hand, we will keep improving VBS-Lustre for better performance and more features. On the other hand, we will consider applying VBS-Lustre in other fields, such as distributed file systems.

6.1. Future improvements to VBS-Lustre

First, the tests in Section 5 are carried out for just validating the implementation of VBS-Lustre, and thus not large in scale. We plan to use FutureGrid [22] resources to test VBS-Lustre on larger scales in the next step.

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 OSSs and OSTs, 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.

6.2. Applying VBS-Lustre to build a new type of distributed file system

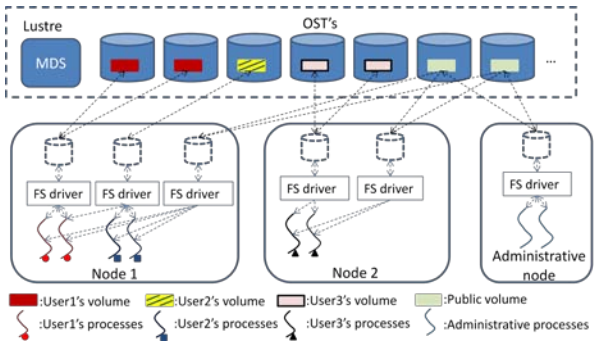

Figure 8. **VBS File System**

In the previous sections we reviewed the advantages of adopting Lustre as a substrate technology for VBS. In this section we review potential contributions of VBS-Lustre to 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, currently there is 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. Based on VBS-Lustre, it is possible to build a new type of distributed file system as shown in Fig. 8, which we call "VBS File System" (VBSFS). VBSFS 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.

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 containing 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 tries 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 only supports read-only sharing on the volume level, VBSFS cannot handle the situations where processes on different nodes are trying to write to the same home directory. But for the other cases that VBSFS can handle, it has the following advantages:

(1) The workload of Lustre MDS is tremendously relieved, since it only needs to maintain the volume files' metadata, which 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 potentially achieve a much larger aggregate metadata throughput than Lustre. Table 1 in Section 5 shows an example of this merit;

(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 mechanisms 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 that 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 these ideas of VBSFS are valuable for solving various challenges to current cluster file systems, and we look forward to combining them with traditional systems in our future efforts for conquering the challenges.

References

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

[2] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, D. Zagorodnov, "The Eucalyptus Open-Source Cloud-Computing System", *Proceedings of CCGRID 2009*, Shanghai, China, May 2009.

[3] The Nimbus project, http://www.nimbusproject.org/.

[4] B. Sotomayor, R. S. Montero, I.M. Llorente, I. Foster, " Virtual Infrastructure Management in Private and Hybrid Clouds", *J. IEEE Internet Computing*, vol. 13, no. 5, Sept.-Oct. 2009.

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

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

[7] K. Shvachko, H. Kuang, S. Radia, R. Chansler, " The Hadoop Distributed File System", *Proceedings of IEEE MSST 2010*, Incline Village, NV, USA, May 2010.

[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] A. Donnellan, J. Rundle, G. Fox, D. McLeod, L. Grant, T. Tullis, M. Pierce, J. Parker, G. Lyzenga, R. Granat, M. Glasscoe, " QuakeSim and the Solid Earth Research Virtual Observatory", *Pure and Applied Geophysics*, Vol. 163, 2006.

[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.

[22] FutureGrid, http://futuregrid.org/