2013 International Conference on Cloud and Service Computing

PIOD: A Parallel I/O Dispatch Model Based on Lustre File System for Virtual Machine Storage

Zhou Lei, Zhebo Zhou, Bolin Hu, Wenfeng Shen, Luokai Hu
School of Computer Engineering and Science
Shanghai University
Shanghai 200072, P. R. China
{leiz, racezzb, hubolincn, wfshen}@shu.edu.cn
School of Computer, Hubei University of Education, Wuhan 430205, P.R. China
luokaihu@gmail.com

Abstract—**Virtualization is a key technique for cloud computing. One challenge in it is providing a scalable, high performance and high cost effective data storage for virtual machines. In this paper, we propose a parallel I/O dispatch model called PIOD based on Lustre parallel file system to address this issue. PIOD adopts queue mechanism to coordinate data access requests from clients, avoiding random and concurrent storage access. Doing so contributes to avoid random disk seek overhead, which in turn improves I/O performance of data access. The simulation results show that PIOD is effective and can improve I/O throughput about 26.3% on average.**

Keywords- Cloud Computing, Virtualization, Data Storage, Lustre, I/O dispatch.

I. INTRODUCTION

Virtualization is a key technique for cloud computing. Virtualization can make good use of economics of scale and elastically delivers almost any IT related services on demand [1]. Its main advantages include isolation, consolidation and multiplexing of resources [2]. This means that different applications running on different virtual machines (VMs) would not affect each another, and that one physical resource can be reused by different VMs hosting on the same physical machine.

Cloud data center, taking the advantages of virtualization, usually deploys a large amount of physical nodes and virtual machine nodes that reside on physical nodes. However, one key challenge cloud data center faced is providing a scalable, high performance and high cost effective data storage for VMs. This is because the data storage for VMs not only need to consider the capacity, scalability, data sharing and low-price, but also the performance bottleneck caused by particular I/O framework of virtual machine system [3, 4].

There are three kinds of data storage scenarios in cloud data center: local storage, centralized share storage and network storage. Local storage is convenient, but capacity is limited，data cannot be shared among different nodes and it is difficult to migrate VM. Centralized share storage, such as disk array connected to host by fiber channel or Infiniband [5] has excellent I/O performance and is easy to migrate VM. However it is very expensive. An alternative is adopting network storage that consists of a number of low-priced storage devices which are usually connected with TCP/IP-Ethernet networks and managed by distribute file systems or parallel file systems.

We primarily focus on the network storage scenario for VM, which is also the hot research area recently. To further research the network storage scenario, we first compare the I/O performance of three kinds of network file systems: MooseFS, HDFS and Lustre. The experiment indicates that Lustre behaves better than MooseFS and HDFS as far as I/O throughput. However, one drawback of Lustre is the I/O throughput would dramatically drop with the number of VM increases.

In this case, how to improve the I/O throughput is a key issue, and deserves our further investigation. As we know, disk seek is the slowest operation in computer I/O system. So, optimizing seek time is deemed a very crucial approach to improve I/O throughput.

In this paper, we optimize seek time by improving the data access parallel degree and decreasing the data access interference. More specifically, we propose a parallel I/O dispatch model called PIOD based on Lustre parallel file system by efficiently dispatching I/O requests from VMs to disk drive. Our experimental results demonstrate that the PIOD is effective and can improve I/O throughput remarkably.

The rest of this paper is organized as follows. In Section 2, we introduce the background knowledge about Lustre and Xen. In Section 3, we design some experiments to compare the I/O throughput of MooseFS, HDFS and Lustre. In Section 4, we analyze the reason which leads to the drop of I/O throughput of Lustre. In Section 5, we design and implement a parallel I/O dispatch model based on Lustre. In Section 6, we evaluate the performance of our parallel I/O dispatch model through several experiments. In Section 7, we introduce the related work about improving I/O throughput of virtual machine. Finally, we conclude the paper in Section 8.

II. BACKGROUND

Lustre [6] is a parallel network storage system built on top of a number of independent storage devices which accesses files in parallel way. Lustre can increase storage capacity and access bandwidth easily by adding storage devices without interrupting of file service. In Lustre, each file is divided into several sub-files and they are stored into different storage node. A data access request from client is disassembles into several small sub-requests, and then the storage servers concurrently access the pieces of data for each sub-request.

Lustre file system consists of Metadata Server (MDS), Object Storage Servers (OSS), and Object Storage Target (OST), as showed in Figure 1. The MDS stores the layout of each file, the number and location of the file's stripes. The clients obtain the file layout from the MDS. The OSS provides file I/O service and network request handling for one or more local OSTs. The OST stores file data as data objects on one or more OSSs. A single Lustre file system usually has multiple OSSs and OSTs. To optimize performance, a file may be spread over many OSTs.

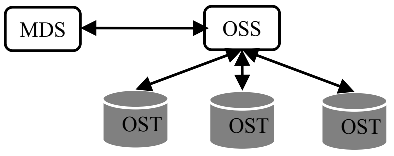

Figure 1. Architecture of Lustre

Xen is an open source virtual machine monitor (VMM) which is implemented by Para-virtualization technology and is used widely in cloud computing.

In Para-virtualization, all I/O operations in guest virtual machine need two virtual machine's cooperation: Domain 0 and Domain U. A backend driver resides in Domain 0 (a privileged VM, also called Driver Domain) to access physical device. And it provides special virtual interfaces to other VMs for I/O accesses. A frontend driver locates in Guest OS. The driver handles Guest's I/O requests and passes them to backend driver, which will interpret the I/O requests and map them to physical devices. Physical device drivers in Driver Domain will drive the devices to handle the requests. Using Para-virtualization, VMM doesn't directly drive I/O devices, so it is much simpler to implement than full-virtualization.

III. EXPERIMENT OBSERVATION

As mentioned in introduction, there are three kinds of data storage scenarios in cloud data center. The first one is storing data at local storage. The benefits of it are convenient, cheap and acceptable I/O performance. But its capacity is limited，and data cannot be shared and it is difficult for migrating virtual machine [7]. The second one is storing data by centralized share storage, such as disk array connected to host by fiber channel or Infiniband. The benefits lie in excellent performance and convenience to migrate virtual machine, because data is shared. But its disadvantage is high price. Figure 2 shows the centralized share storage. An alternative is adopting network storage which is built by a number of low-priced storage devices that are usually connected with TCP/IP-Ethernet networks and managed by distribute file systems or parallel file systems. Figure 3 shows the network storage with low-priced storage devices.

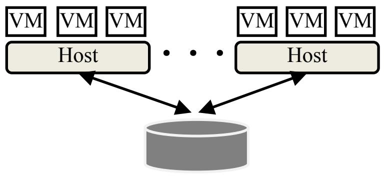

Figure 2. Centralized shared storage

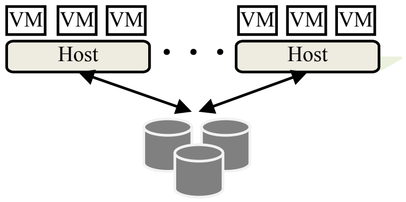

Figure 3. Network storage with low-priced storage devices

MooseFS, HDFS and Lustre are three typical network storage types. We tested the I/O throughput of MooseFS, HDFS and Lustre in virtualized environment. The test results showed in Figure 4 and Figure 5.

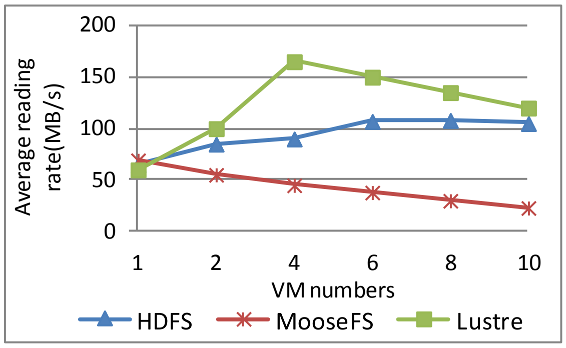

Figure 4. Average reading rate of HDFS, MooseFS and Lustre

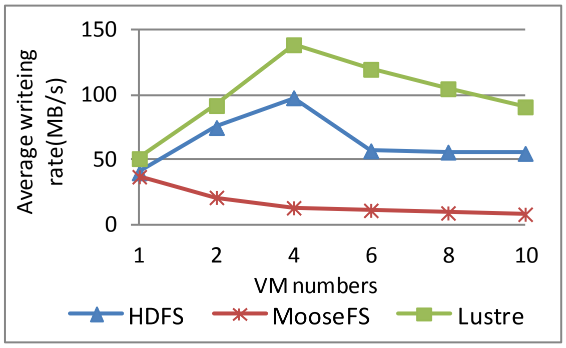

Figure 5. Average writing rate of HDFS, MooseFS and Lustre

From the test results, we can see that the average I/O throughput of Lustre behave better than that of HDFS and MooseFS. However, the I/O throughput of Lustre drops especially fast than that of HDFS and MooseFS with the number of VMs increases. We will analyze the reason that causes the I/O performance dropping dramatically in next section.

IV. DATA ACCESS INTERFERENCE

Assume there are four VMs and three OSTs in our storage model, as illustrated in Figure 6. Each VM has 20MB file to write with a stripe size of 3MB and a stripe width of three.

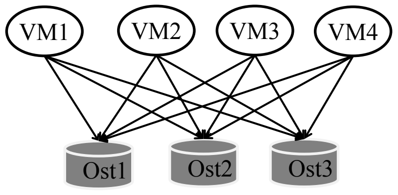

Figure 6. Parallel data access

Given these parameters, Lustre distributes the 28 stripes across the three OSTs in a round-robin pattern. In such a storage method, every VM communicates with every OST. Thus, there is a significant communication overhead. And the performance would dramatically degradate as the number of VMs increases. The primary reason is disk access requests from VMs are random, and they do not perform a series of contiguous disk I/O operations. This would cause random and disorder disk seek, which in turn incurs additional disk seek overhead, therefore reduces parallel potential of file access.

Thus how to improve the data access parallel degree and decrease the communication overhead is our research focus. For this issue, paper [8] gives us a good inspiration. The access pattern shown in Figure 6 is called as 'all-to-all' OST pattern because it involves all VMs communicating with all OSTs. The simplest approach reducing contention caused by such access patterns is to limit the number of OSTs across which a file is striped. However, reduce contention by limit the number of OSTs means limit the parallelism of file accesses, which in turn, limits parallel I/O performance. Therefore, the question is how we can do to implement an alternative data access pattern to both reduce contention and maintain a high parallel degree.

The basic idea behind PIOD is optimizing the communication overhead by replacing the 'all-to-all' OST pattern with a 'one-to-one' OST pattern, as showed in Figure 7, where the data is arranged by a model we called PIOD.

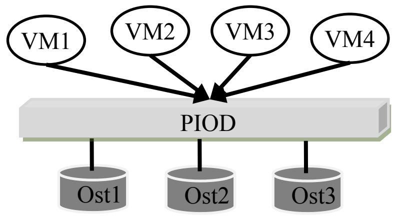

Figure 7. Parallel data access with a coordinator

In virtual machine system, both the scheduler within the virtual machine monitor (VMM) and the scheduler within the driver domain play a key role in determining the overall fairness and performance characteristics of the whole system. The first scheduler determines which VM's I/O request should be scheduled and the second scheduler determines how and where the request should be dispatched. In next section, we would analyze the I/O process in out architecture, and then design the PIOD based on Lustre, which belongs to the scheduler within the driver domain.

V. PIOD DESIGN AND IMPLEMENTATION

A. Analysis of I/O Process in our storage Architecture

As illustrated in Figure 8, the storage for VMs is managed by Luster, and VMs can access files in parallel. Frontend driver in Domain U is responsible for transmitting Domain U's I/O request to the backend driver in Domain 0, then backend driver analyzes the request received from frontend and maps the request to actual physical equipment. I/O ring is a shared memory region used by both frontend driver and backend driver. Each time when domain U accesses the device, the frontend driver will write a request to the I/O ring and notify the backend driver through the event channel. After the physical device access is completed, the backend driver will write a response to the I/O ring and notify the frontend driver by an I/O response event. Thus Domain 0 conducts I/O operations instead of domain U by this way.

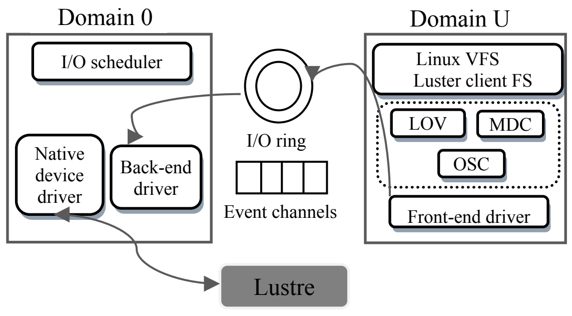

Figure 8. Virtual machine storage with Lustre

Further analysis of I/O process showed in Figure 9. There are three phases to process I/O operation in Xen virtualized environment. First, domain U sends an I/O request to the I/O ring and waits for domain 0 to receive it. Second, domain 0 receives the request and accesses the physical device. Third, domain 0 sends an I/O response to the domain U.

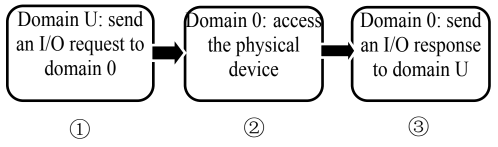
Domain U: send an I/O request to domain 0
Domain 0: access the physical device
Domain 0: send an I/O response to domain U
①
②
③

Figure 9. Process of I/O request in our storage mold

The time consumption at Phase 1 and Phase 3 depends on I/O event handling delay, which is determined by scheduling algorithm. The time consumption at Phase 2 mainly depends on disk seek and data access contention. However, those contentions increase rapidly with the increase of the number of domain U. Therefore, how to reduce the latency of I/O event handling caused by scheduling delay and I/O dispatch is the key to improve I/O performance of virtual machine storage.

B. PIOD Design and Implementation

In the configuration environment of Lustre, there is a client counterpart in each domain U: Metadata Client (MDC), Object Storage Client (OSC) and Logical Object Volume (LOV). MDC is an interface module at client side and we can access MDS through it. The relationship of OSC and OST is one-to-one, and a group of OSCs are wrapped into a single LOV, as showed in Figure 10.

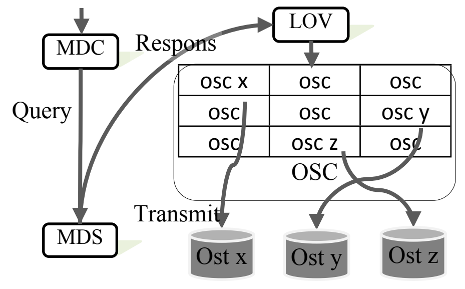

Figure 10. Process of open, write and read a file

When we create, open or read a file, we first acquire metadata information stored on MDS through MDC at client side. After MDS returns the location of the file, client will directly conduct I/O access operation with the relevant OSTs when it needs the stripes of this file. For example, before reading a file, client will query the MDS via MDC and be informed that it should talk to <ost x, ost y, ost z> for this operation. Then the LOV at the client side interpret this information so client can send requests to OST directly, as showed in Figure 10.

In Lustre, file is striped on several OST, and all data storage on one OST of this file is seen as one object. The object is presented by 128 bit called Object ID. Unlike the traditional Unix file systems, the inode in the MDS does not point to data blocks, but points to one or more Objects associated with the files. So clients side can read or write data to this file in parallel only need to know the Object ID. In other word, clients can exchange data with OST directly by Object ID.

The key part we implemented in PIOD is adopting queue mechanism called OST_Queue which maintained in a global hash table. The hash key is also known as OST_ID. In other words, PIOD maintains a queue for every OST by OST_ID. When a request arrives, the PIOD would queue it into related OST_Queue in enqueueing module. Then the requests in the OST_Queues will be dispatched into related OST by round-robin mechanism in dequeuing module. The advantage is all requests in one queue can conduct disk operation in order on one OST, which in turn avoids random and concurrent disk seek operation on OST, improving I/O performance of storage devices. Figure 11 shows our PIOD model.

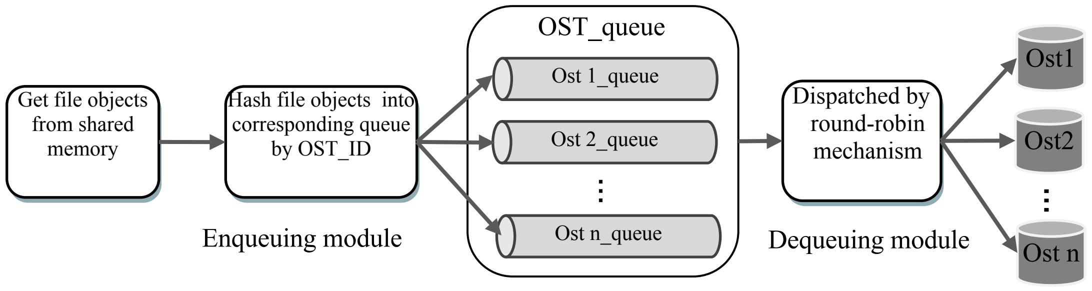
OST_queue
Get file objects from shared memory
Hash file objects into corresponding queue by OST_ID
Ost 1_queue
Ost 2_queue
⋮
Ost n_queue
Dispatched by round-robin mechanism
Ost1
Ost2
⋮
Ost n
Enqueuing module
Dequeuing module

Figure 11. PIOD model

In Xen, I/O request and I/O data are separated rather than in one package. So, after clients getting the position that the file data will be stored, assume < OST1、OST3、OST4>, Domain 0 will capture the data block and dispatch them into appropriate queue by OST_ID. In order to guarantee the fairness, we adopt round-robin mechanism in dequeuing module to avoid starvation. The time slices are assigned to each OST_Queue with the same portion and in circular order, handling all queues without priority. The PIOD algorithm is described in detail as following.

Algorithm : code for PIOD

Queue(i) is the ith queue, which stores data objects with OST_ID i. Queue are numbered 0 to (n-1), n is the maximum number of OST. PIOD is service once queue every time slice to choose the next queue to service. Insert_object(p, i) is add data object p into Queue(i). Dispatch_object(p, i) is dispatch data object p from Queue(i) into native device driver.

```
Begin
Enqueuing module: on arrival of data object p
1 Queue(i) = hash(object p);
2 if (there is no this object exist in this queue)
3     {if (no free buffers left Queue(i))
4         {Queue(i) = must_queue; return -1;}
5     else {Insert_objcet(p, i); return 1;}}
6 return 0;

Dequeuing module:
1 While (true) do{
2  if ((time slice for Queue(i) is enough) || (Queue(i) == must))
3     { if ( Queue(i) is not empty )
4           {get a data object from OST_queue;
5           Dispatch(p, i);}
6      else if (there is enough slice left to wait for one requests)
7              {do{wait for data object p;
8                   Dispatch(p, i);}
9            While(time slice for Queue(i) is enough)}
10 else {get next queue for service;}}
end
```

VI. Experimental Result

A. Experimental Setup

The computer hardware configuration of our experiment environment is DELL PowerEdge M610, Intel Xeon 5500 twenty-four cores processor, 96GB memory, two Broadcom NetXtreme II 5709 1Gbps Ethernet NIC, 500GB SAS6 hard disk.

Based on such a hardware configuration, our experimental platform consists of fifteen virtual machine nodes. One node is configured as Domain 0 (driver domain); ten nodes are configured as Domain Us (client); and the other four nodes are configured as storage nodes (one is MDS, three are OSS). All domains U are installed Debian5.0 Linux distribution with Linux-2.6.30 kernel with one VCPU and the volume of 10GB image file as virtual file system. To minimize the effect of memory caching, all virtual machine nodes were configured with a 256MB memory. Each storage node was allocated 100GB virtual disk space.

We run Xen-4.1.0 on test system and use Luster 2.0.0 as the file system. The testing tool IOzone [9] runs in each Domain U.

B. Key Results

Stripe size is how much data written to one OST before moving to the next OST. The default stripe size is 1 MB. Stripe count is how many OSTs are used to store one file. The default stripe count is the number of all available OSTs. This default setting is strongly recommended by Lustre, as it allows space and load balancing to be done by the MDS as needed. We also use this setting.

We used the IOzone benchmark for random read and the random write in the throughput mode. We can see from Figure 12 and Figure 13, that reading rate and writing rate are both improved by PIOD.

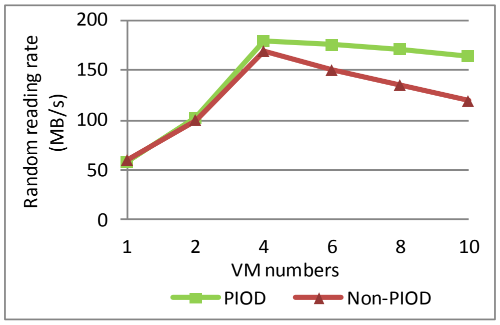

Figure 12. Random reading rate

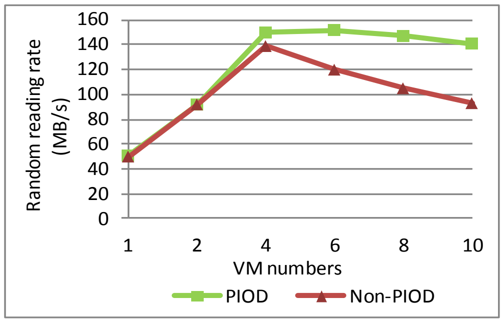

Figure 13. Random writing rate

We also tested the I/O performance under different record size. We can see from Figure 13 and Figure 14, that reading rate and writing rate are both improved by PIOD especially in small record size. This confirmed that PIOD improve the data access parallel degree and decrease the communication overhead efficiently.

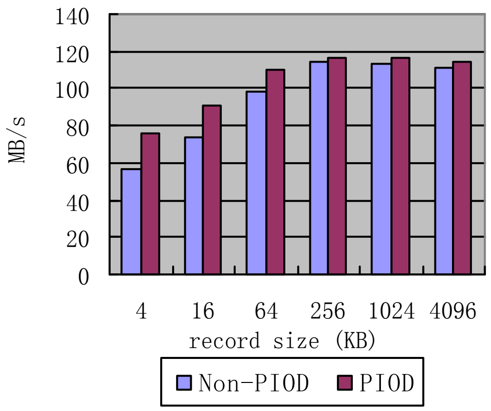

Figure 14. Lustre writing rate under different record size

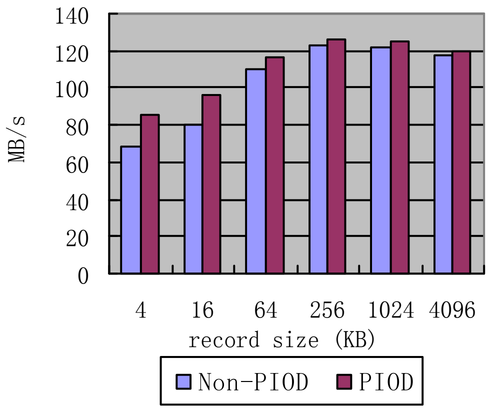

Figure 15. Lustre reading rate under different record size

VII. Related Work

There are a large number of researches focusing on how to improve I/O throughput of data storage for VMs. They can be summarized as storage architecture, I/O scheduling and VCPU scheduling.

Dickens et al. [8] proposed a new data redistribution model to effectively reduce the access contention of OST storage resources, which is resulted by 'all-to-all' communication model. However, he improved I/O throughput by limiting the number of OST which communicates with client access requests. The disadvantage is it reduces the I/O parallelism. Unlike this, we improve I/O throughput by coordinating storage access using queue mechanism. Doing so contributes to make storage access in order between client requests and OST, but not reduces the I/O parallelism.

Casale et al. [10] proposed a simple performance models to predict the impact of consolidation on the storage I/O performance of virtualized applications. Paper [10] also defined a simple linear prediction models for the throughput, response times, and mix of read/write requests in consolidation based only on information collected in isolation experiments for the individual virtual machines.

Ismail et al. [11] present virtualization benchmarking methods, results and analysis in various resource utilization scenarios. Such as performance overhead, the cost of virtualization compared to physical, the VM instance performance when more VM is instantiate and performance fairness or isolation. These studies provide us the fundamental understanding of virtualization technology.

Beside the above researches about scheduling algorithm, there are also researches about virtual machine I/O model. Hu et al. [12] proposed a virtual machine I/O scheduling model based on multi-core dynamic partitioning to eliminate the I/O performance bottleneck caused by scheduling delay, and implemented a prototype based on Xen virtual machine. Santos et al.[13] bridges the gap between software and hardware techniques for I/O virtualization. Raj et al. [14] effectively reduced the I/O performance degradation caused by scheduling delay using self-virtualization device and VM-bypass method, but they cannot be applied to traditional share device cases.

VIII. CONCLUSION

Virtualization is a key technique of cloud computing. How to improve I/O throughput is one of the challenging issue. In general, there are three ways to resolve this problem: 1)improve the scheduling algorithm which is responsible for scheduling the VCPU in VMM; 2)improve the scheduling algorithm which is responsible for data block scheduling and dispatching in Domain 0; 3)use more fast storage device and network device to reduce disk seek time and data transmission time. In this paper, we adopt a widely used parallel network storage called Lustre, which can increase storage capacity and I/O bandwidth easily by adding storage devices without interruption of data access service. We also design and implement a parallel I/O dispatch model based on Lustre called PIOD to improve the parallel degree and decrease the data access interference caused by parallel access without coordination. Our evaluation results demonstrate that PIOD is effective to improve the I/O throughput of VMs especially in the case of large storage capacity and large number of client access.

Acknowledgement

This work was supported by the Hubei provincial department of education science Research programs for youth project (No. Q20133003), Young College Teachers in Hubei Province in-depth enterprise action project (No. XD2012430), National Natural Science Foundation of China(No. 61073096).

REFERENCES

[1] I. Foster, Y. Zhao, I. Raicu and S. Lu, "Cloud Computing and Grid Computing 360-Degree Compared," Proc. IEEE Grid Computing Environments Workshop, pp. 1-10, 2008.

[2] T. Wood, P. Shenoy, A. Venkataramani and M. Yousif, "Black-box and Gray-box Strategies for Virtual Machine Migration," Proc. The 4th USENIX conference on Networked Systems Design & Implementation, pp. 1-14, 2007.

[3] J. Shafer, "I/O Virtualization Bottlenecks in Cloud Computing Today," WIOV'10 Proceedings of the 2nd conference on I/O virtualization, USENIX Association Berkeley, CA, USA, pp. 5-5, 2010.

[4] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica and M. Zaharia, "A view of cloud computing," Communications of the ACM, pp. 50–58, 2010.

[5] Jacob Gorm Hansen and Eric Jul, "Lithium: virtual machine storage for the cloud," Proceedings of the 1st ACM symposium on Cloud computing, New York, NY, USA: ACM, pp. 15-26, 2010.

[6] F. Wang, S. Oral, G. Shipman, O. Drokin, T. Wang and I. Huang, "Understanding lustre filesystem internals," Technical Report ORNL/TM-2009/117, Oak Ridge National Lab, National Center for Computational Sciences, 2009.

[7] H. A. Lagar-Cavilla, J. A. Whitney, A. Scannell, P. Patchin, S. M. Rumble, E. de Lara, M. Brudno and M. Satyanarayanan, "SnowFlock: rapid virtual machine cloning for cloud computing," Proceedings of the 4th ACM European conference on Computer systems, Nuremberg, Germany, April 01-03, 2009.

[8] PM. Dickens and J. Logan, "A high performance implementation of MPI-IO for a Lustre file system environment," Concurrency and Computation: Practice and Experience, dio: 10.1002/cpe.1491, 2010.

[9] IOzone File system Benchmark. http://www.iozone.org/, 2011

[10] G. Casale, S. Kraft and D. Krishnamurthy, "A Model of Storage I/O Performance Interference in Virtualized Systems," 1st Int. Workshop on Data Center Performance, DCPerf 2011.

[11] Bukhary Ikhwan Ismail, Devendran Jagadisan and Mohammad Fairus Khalid, "Determining Overhead, Variance & Isolation Metrics in Virtualization for IaaS Cloud," Computer science, data driven e-science, part 8, pp. 315-330, 2011

[12] Y. Hu, X. Long, J. Zhang, J. He and L. Xia, "I/O scheduling model of virtual machine based on multi-core dynamic partitioning," In HPDC, pp. 142–154, 2010.

[13] J. R. Santos, Y. Turner, G. J. Janakiraman and I. Pratt, "Bridging the Gap Between Coftware and Hardware Techniques for I/O Virtualization," In Proceedings of the USENIX 2008 Annual Technical Conference, Berkeley, CA, USA, pp. 29–42, June 2008.

[14] H. Raj and K. Schwan. High Performance and Scalable I/O Virtualization via Self-Virtualized Devices. In Proceedings of the 16th International Symposium on High Performance Distributed Computing (HPDC'07), New York, NY, USA, ACM, pp. 179–188, June 2007.