AoDI: An Allocation-On-Demand Incremental Volume based on LVM

Rui Cao, Caijun Zhen, Yan Gao, Guangzhi Xu, Xiaoguang Liu, Gang Wang
Nankai-Baidu Joint Lab, College of Information Technical Science, Nankai University
Weijin Road 94, Tianjin, China
caorui12001@yahoo.com.cn, zhenqiansha@126.com, monciel@163.com,
azhi.coder@gmail.com, liuxg74@yahoo.com.cn, wgzwp@163.com

Guangjun Xie
Baidu Inc.
Baidu Campus, No. 10, Shangdi 10th Street Haidian District, Beijing, China, 100085
xieguangjun1980@163.com

ABSTRACT

Cloud computing has become one of the hottest topics in both academia and industry. In cloud computing, virtualization is the fundamental technology, and the resource management is the key issue. Storage, an important resource, traditionally is allocated to users according to users' predictions, so may cause bad space utilization because of overprovisioning or underprovisioning. Dynamic adjusting is difficult because of the fixed address mapping used in the traditional storage systems. In this paper, we present an Allocation-On-Demand Incremental (AoDI) volume. It uses an appending rather than overwriting strategy to deal with the write requests. So we can obtain accurate storage space usage easily. Accompany with an automatic volume extension technique, AoDI can allocate storage space always matching users' real-time requirement, so avoids space waste. Based on appending strategy, we also design a Non-COW (Non Copy On Write) snapshot that is dramatically faster than traditional COW snapshots. Since snapshot is a frequent operation in cloud or virtualization systems, Non-COW snapshot can improve overall performance effectively. Another advantage of AoDI is translating random requests into sequential requests that obviously can speed up random access. We implement AoDI based on LVM (Logical Volume Manager) in Linux platform. Our experimental results show the great performance advantage of AoDI in snapshot and random access.

Categories and Subject Descriptors

D.4.2 [**Operating Systems**]: Storage Management

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SAC'11 March 21-25, 2011, TaiChung, Taiwan.
Copyright 2011 ACM 978-1-4503-0113-8/11/03 ...$10.00.

General Terms

Design, Experimentation, Performance

Keywords

storage resource management, on-demand storage allocation, appending strategy, Non-COW snapshot, automatic extension

1. INTRODUCTION

Cloud computing [4] refers to both the applications delivered as services over the Internet and the hardware and systems software that provide those services. Cloud users only pay for the services they are using, and need not be concerned about overprovisioning for a service that causes wasting costly resources, or underprovisioning for one that can not meet the requirements. Virtualization [8] is one of the key techniques to build cloud. Using virtualization, all resources are shared actually, but the virtual machines give cloud users the illusion of possessing private resources. Accordingly, resources management is a challenge to both virtualization and cloud computing [3].

How to improve both storage utilization and its performance must be considered to cloud provider. In traditional storage management, storage resources are allocated to users as their private resources. Overprovisioning or underprovisioning often happens that will cause resource wasting or refusing service in cloud computing when many users require services simultaneously. This situation is caused by the static allocation in traditional storage management [9]. The motivation of this paper is to implement an Allocation-On-Demand Incremental (AoDI) volume management, which can allocate storage resources according to users' real time requirements rather than one-off strategy.

The main ideas of AoDI are concluded as follows,

- Separating the user-oriented visible volume and system-oriented physical volumes by using double-level volumes structure. From user's view, they have the illusion of possessing enough storage resources. In fact, the physical storage space will only be allocated when users need them really.
- Appending rather than overwriting strategy is used to

deal with the write requests. Appending strategy both improves performance by converting random write requests into continuous write requests, and makes accurate estimation of storage space usage availably. Using automatic extension, AoDI can allocate storage space always meeting users' real-time requirements, which can avoid space wasting.

- Non-COW (Non Copy On Write), a zero copy snapshot accompany with AoDI, has dramatically better performance than traditional COW snapshots. Since snapshot is a frequent operation in cloud or virtualization systems [5], Non-COW snapshot can improve overall performance effectively.

We implement AoDI based on LVM (Logical Volume Manager) in Linux platform. Our experimental results show the great performance advantage of AoDI in snapshot and random access.

2. RELATED WORK

How to allocate resources effectively is a classic problem to system designer. Andrzejak et al. [3] found up to 50% savings could be obtained by dynamically redistributing resources among applications in utility computing. Dynamic memory allocation algorithms have also been applied in the VMWare ESX server [12]. This algorithm estimates the working set sizes of each virtual machine and periodically adjusts memory allocated to each virtual machine such that performance goals are met. Multi-resource partition [6] is an idea that multiple resources are partitioned to provide isolation and QoS for several competing applications. Wachs et al. [11] show the benefit of considering both cache allocation and disk bandwidth allocation to improve the performance in shared storage servers. Padala et al. [9] study methods to allocate memory and CPU to several virtual machines located within the same physical server. AoDI is mainly to solve the problem of dynamic allocation on storage resource at LVM level, and it is transparent to users.

Snapshot provides the ability to record the state of a storage device at any given moment. In Linux, LVM2 [2] adopts Copy-On-Write (COW) to maintain snapshot data. COW [10] is space efficient, because the snapshot volume only stores the original version of modified data after snapshot creation. The original volume can provide other snapshot data. However, COW impacts write performance, because write requests to the original volume must wait until original data is copied to the snapshot. In Parallax [13] [7], radix tree is used to maintains the dynamic mapping between virtual address space and physical address space. So COW operations are avoided. However, a snapshot inherits its original volume's mapping table lazily. That is, Parallax converts data COW to metadata COW. In this paper, in order to avoid COW completely, we develop a new Non-COW approach based on AoDI, which do not need copy both data and metadata during snapshot operations.

3. THE DESIGN OF AoDI

3.1 Fundamental Concepts

In order to achieve on-demand storage allocation, we need to get accurate estimation of storage utilization and have ability to adjust the physical size of the volume dynamically. Therefore, AoDI adopts the following strategies:

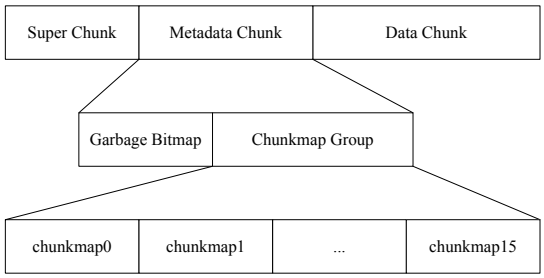

Figure 1: AoDI Disk Layout.

- Double-level volume structure. AoDI separates the user-oriented visible volume and system-oriented physical volume. So it can allocate physical space according to user's real-time requirement such that storage resource is used effectively.

- Appending strategy, used in AoDI, not only makes it easy to obtain accurate utilization ratio, but also converts logically random write requests into physically continuous disk write operations. Accordingly, address mapping between visible volume and physical volume must be maintained (see Section 3.2 and 3.3). A set of new read/write algorithms for this dynamic address mapping are designed (see Section 3.4). Since appending strategy is used, multiple versions of the same data block may simultaneously exist in the volume. A defragmentation mechanism is designed to collect garbage space produced by old versions (see Section 3.5).

- Automatic extension mechanism. An automatic physical volume extension algorithm preventing physical space overflow, and a visible volume extension algorithm for the growth of users' requirement to storage are designed (see Section 3.6).

3.2 Disk Layout

To achieve the goal mentioned in Section 3.1, a new disk layout different from LVM volume is designed for AoDI (as shown in Figure 1). It consists of three parts: *the super chunk*, *the metadata chunks* and *the data chunks*. Here, the chunk size is 4KB.

The super chunk just likes the one in EXT2 file system. It records the global information of AoDI volumes, such as the device number of the volume, the number of snapshots and the start physical chunk number of each snapshot.

The metadata chunks consists of garbage bitmap and chunkmap group. Chunkmaps record the mapping tables between visible and physical volumes. The i-th entry in a mapping table records the physical chunk number accommodating the i-th logical chunk, so the size of each mapping table is proportional to the visible volume size. Moreover, every chunkmap holds one version of metadata at a snapshot point in time (see Section 3.7). As its name suggests, garbage bitmap takes charge of defragmentation. A set bit means that the corresponding physical chunk is a garbage.

The data chunks are used to store the users' data. New data is stored into first unused chunk sequentially.

3.3 Metadata Organization And Cache Replacement

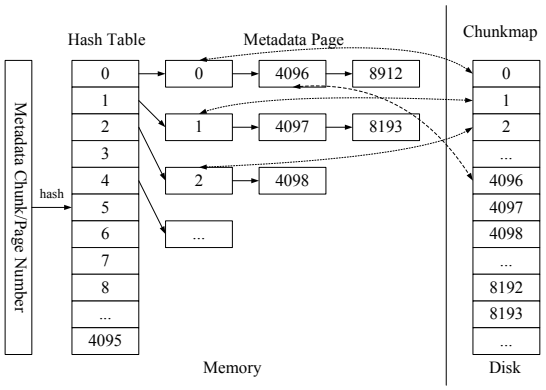

Figure 2: Metadata Organization And Cache Replacement.

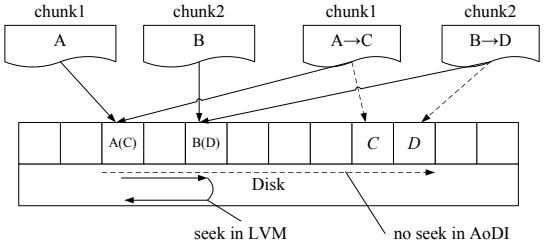

Figure 3: AoDI Write Strategy.

For the purpose of consistency, we must keep some disk space to contain the metadata (see the metadata chunks in Section 3.2). Moreover, since the mapping table of a very large volume may not fit into the main memory completely, we use cache mechanism to accelerate metadata access.

The cache is organized using hash technique as Figure 2 shown. Each hash bucket holds a linked list of metadata page. Each metadata page has a unique number that designates its bucket through Formula 1.

$$bucket\ number\ =\ metadata\ page\ number\ \mod\ 4096 \quad (1)$$

The metadata on disk is organized as a train of chunks in *chunkmap*. The metadata page size in memory is set to equal to the chunk size on disk. Moreover, FIFO is used as the cache replacement strategy. When a cache miss occurs, the oldest metadata page will be written back to the disk if there are no free space in the cache, and the metadata chunk containing the required mapping entry will be read from disk to cache.

3.4 Read And Write Strategy

As mentioned in 3.1, AoDI adopts appending strategy to deal with write requests. That is, when an update request arrives, the new data will be stored in the first free data chunk rather than overwriting the old version. Since random write requests are converted into sequential disk write operations, appending strategy dramatically reduces seek time that is the largest cost of disk operations.

Comparison between write strategy used in LVM and AoDI is shown in Figure 3.

LVM uses the traditional overwriting strategy. As shown in Figure 3, suppose that chunk1 and chunk2 contain A and B initially. If they are modified, LVM first seek to the sectors storing chunk1 and chunk2, and then write the new contents (as shown by solid arrows). By contrast, AoDI just appends the new contents after the last used sector (as shown by broken arrows). Since a dynamic instead of the traditional fixed mapping from user address space to physical address space is used, we must maintain a mapping table, chunkmap for each volume as described in the last subsection. So read and write algorithms must deal with mapping table lookup and update. If a cache missing happened, the metadata replacement will take place. In the write algorithm, before updating, we need to check whether the chunk has an old version. If the old version exists, we need to mark the chunk as a garbage chunk. Moreover, in the read algorithm, if the chunk has not been written, it will end this I/O operation directly.

3.5 Defragmentation

As mentioned in Section 3.1, AoDI performs write requests using appending strategy rather than overwriting strategy. It means multiple versions of the same logical chunk may simultaneously be stored in different physical chunks. Therefore, physical chunks occupied by old versions become garbage chunks. We need a defragment/garbage collection mechanism to deal with them. Otherwise, garbage chunks will exhaust physical space gradually and fragments will also hurt reading performance greatly.

To ensure data consistency, our defragmentation algorithm updates the metadata only after all data chunks are migrated successfully. Since data migration just copies valid data chunks to garbage chunks, even if it is interrupted, data consistency is guaranteed because all valid data chunks and metadata are not damaged.

To implement this idea, we design a *"Back Patching"* algorithm. This algorithm maintains two pointers, s and e, respectively pointing to the first garbage chunk and the last valid chunk. The algorithm repeatedly copies data from chunk e to chunk s, and then searches the next garbage chunk and the previous valid chunk until the two pointers meet. Finally, metadata is updated.

3.6 Automatic Extension

As mentioned in Section 3.1, AoDI adopts the "*Allocation-On-Demand*" strategy and double-level volume structure. The size of physical volume typically is not equal to that of visible volume. When a new volume is created, the size of visible volume is set to user's requirement. However, AoDI system does not allocate the same amount of physical space to the volume. The physical volume that actually occupies disk space, is initially much smaller than the visible volume. With the growth of user data, the allocated physical space may not meet the users' real-time demands. So we design an automatic physical volume extension algorithm to avoid space overflow.

A user-space daemon process real-time monitors the usage of the physical volume through the /proc file system. If the space utilization exceeds the predefined threshold, the daemon process will start extension. Since the human component is removed entirely, AoDI can eliminate risk of overflow timely.

Figure 4 shows the process of visible volume extension. Unlike physical volume extension, visible volume extension implies user address space extension, therefore metadata area

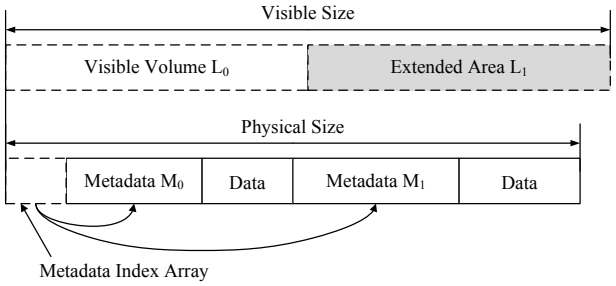

Figure 4: Visible Volume Extension.

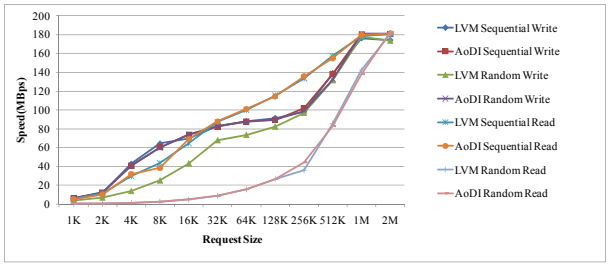

Figure 5: Write/Read Performance In LVM/AoDI Volume.

must be extended. We use a Metadata Index Array (MIA) in the super chunk to maintain multiple metadata areas for multiple user address segments as Figure 4 shows. The first slot of MIA stores the pointer to metadata M_0 for the original user address space L_0, and the second slot stores the pointer to metadata M_1 for the first extended user address space L_1, and so on. Our experimental results show that, using this structure, extended visible volumes achieve almost the same performance as original volumes.

3.7 Non-COW Snapshot

As mentioned in Section 1, AoDI is not only a kind of volume in LVM, but also a basic platform. We can develop a lot of applications on it. In this paper, we introduce an application named *Non-COW* (*Non Copy On Write*) snapshot.

Non-COW snapshot is named opposed to *Copy On Write* (*COW*) snapshot used in LVM. For a original volume with COW snapshots, write performance is impacted seriously by COW operations especially when multiple snapshots are active simultaneously. Since AoDI adopts appending strategy to deal with write requests, it natively keeps all revisions of data. We can design a snapshot technology avoiding COW based on this property. Each snapshot is just an area of continuous data chunks, and snapshot creating is just recording the last used data chunk as the end point of the last snapshot and the first unused data chunk as the start point of the new snapshot. Since user-physical address mapping in AoDI is dynamic rather than fixed, we must maintain a unique *chunkmap* for each snapshot. Each snapshot shares a part of data versions with its subsequent snapshots if these versions are not modified during subsequent snapshots. Therefore, each snapshot partially shares its *chunkmap* with its successors. To avoid metadata (*chunkmap*) copying when snapshot creating, we design a *dependent chunkmap chain*. For each snapshot, address mapping is done by looking up not only its own *chunkmap*, but also its predecessor snapshots' *chunkmaps* (*dependent chunkmaps*). We maintain a bit vector for each metadata chunk to record dependency between *chunkmaps* of different snapshot versions. For the i-th snapshot, when a mapping entry is looked up, AoDI finds the last set bit from the 0-th bit to the $(i-1)$-th bit of the bit vector corresponding to the metadata chunk containing the entry. The j-th bit is the last set bit means that the newest version of this metadata chunk is stored in the j-th snapshot. Then this metadata chunk is read from the j-th snapshot's chunkmap to cache as the metadata of the i-th snapshot. When the metadata chunk is modified, it is written back to the chunkmap of the i-th rather than the j-th snapshot. We can see that this method avoids metadata COW effectively.

4. EXPERIMENTAL RESULTS

4.1 The Experimental Platform

The experiments were done on 64-bit RedHat Linux AS 5, whose kernel version is $2.6.18-128.el5$. The benchmark tool is IOmeter, which can generate read or write requests to simulate real applications. Besides synthetic workload, we also tested the real-world performance using HP trace [1]. The hardware used in experiments is a DELL Power Edge 2850 Server with dual Inter Xeon CPU (2.66GHZ), 3GB memory and a hardware RAID-0 storage composed of six 37GB SAS disks.

On the experimental platform, two kinds of volume, LVM logical volume and AoDI volume, were created to compare their performance. The size of both volumes is 100GB.

4.2 Experimental Results

4.2.1 Throughput

As the most important criterion to storage systems, the throughput of LVM and AoDI were tested firstly. Figure 5 shows the results of sequential and random write performance with different request size. As expected, LVM and AoDI have almost the same sequential write performance. However, the random write performance of AoDI is much better than that of LVM especially under small requests. This result verifies the advantage of appending strategy in converting random write requests into continuous disk write operations.

Figure 5 also shows the results of read test. Since we used synthetic workload in this test, the random access is completely random rather than containing some repetitive access patterns. Therefore, AoDI does not show advantage in random read performance. However, if real-world workload is used, AoDI is expected to show its advantage.

4.2.2 Impact of Cache Size

Since metadata cache is an important factor to AoDI's performance, the impact of cache size has also been tested in experiments. Here, 20%, 40%, 60%, 80% and 100% mean the ratio of cache size to the size of chunkmap. Figure 6, Figure 7, Figure 8 and Figure 9 show the experimental results of sequential write, random write, sequential read and random read respectively. We can see that as the cache size decreases, throughput drops remarkably. In the experiments of sequential write, sequential read and random read, AoDI

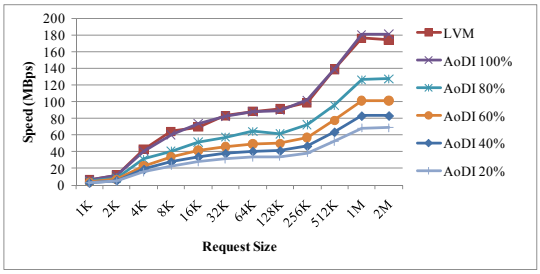

Figure 6: Sequential Write.

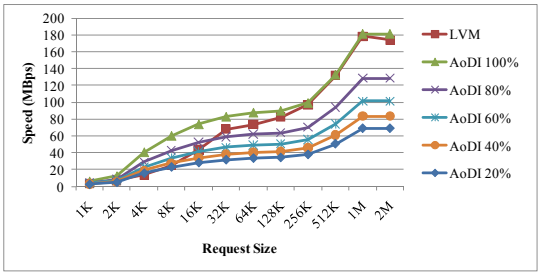

Figure 7: Random Write.

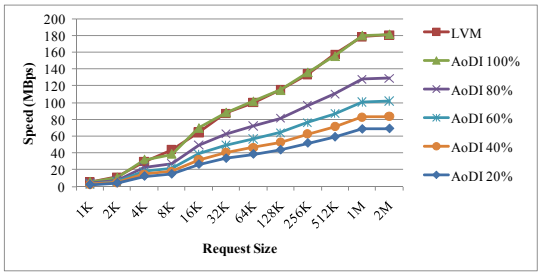

Figure 8: Sequential Read.

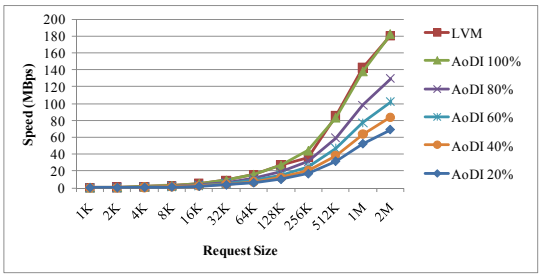

Figure 9: Random Read.

is inferior to LVM because of extra cache replacement. In random write experiment, especially under small requests, the throughput of AoDI is better than LVM. However, as the request size increases and the impact of disk seek weakens, LVM is better increasingly. So better cache replacement strategy is an important future works.

4.2.3 *Real-World Workload*

Figure **??** shows the performance of AoDI and LVM with real-world workload. HP TPC-D trace [1] was used in this test. Since this real-world trace is mainly composed of random requests, and read and write have some similar access patterns, AoDI shows great advantage in response time due to its appending strategy.

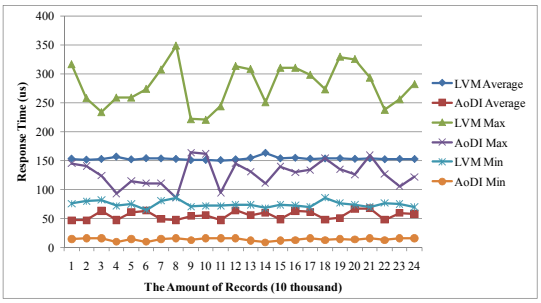

Figure 10: Responding Time.

4.2.4 *Automatic Extension Performance*

We also test the performance of automatic extension. First we test the speed of physical volume automatic extension. Test is divided into three groups that initial physical volume size is 1GB, 10GB and 100GB respectively. The extension time tested with the increment ratio from 10% to 100%. Figure **??** shows the result. We can see that the initial size and the increment ratio hardly impact the extension time that remains a very small value. This implies that automatic physical volume extension will not decrease user experience.

Two volumes are used to test the performance of extended visible volume. One is 200GB, another is initially 20GB and is extended 9 times to 200G. That is, the former volume has only one metadata area, the latter one has ten. We test the read and write performance for both of them. As the Figure **??** shows, we can see the performance of two volumes is basically the same. That is, multiple metadata areas structure caused by visible volume extension does not decrease performance perceptibly.

4.2.5 *Snapshot*

As we develop a Non-COW snapshot system based on AoDI, we also compare the degradation of throughput in LVM snapshot and Non-COW snapshot. We test the throughput after creating 3 snapshots for both kinds of volume. Here, we set the ratio of cache size to the size of chunkmap as 20% that is a very harsh setting to AoDI. Figure **??** show the results. Because of COW mechanism, when dealing with write requests, LVM snapshot performs much more extra I/O operations that cause seriously throughput degradation. Moreover, as the number of snapshots increases, the performance gap becomes wider. However, in AoDI, the throughput does not decrease after creating snapshots because appending strategy does not introduce extra I/O operations in snapshot. Since COW does not introduce extra I/O operation during dealing with read request, LVM snapshot does not show read throughput degradation, so does AoDI snapshot. Although Figure **??** show the advantage of LVM snapshot in read throughput, we note that LVM snapshot places metadata entirely in main memory. If AoDI uses cache size ratio 100%, it will obtain similar performance.

5. CONCLUSIONS

Storage management is one of the most important issues in

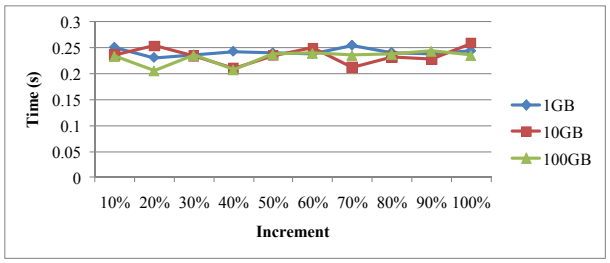

Figure 11: Physical Volume Extension Time.

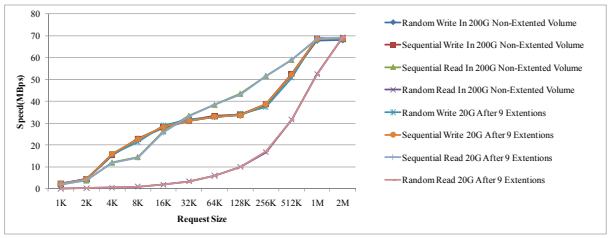

Figure 12: Write/Read Performance In Extended/Non-Extended Volume.

cloud computing. Traditionally, storage is allocated by users' predictions. It may cause bad space utilization because of overprovisioning or underprovisioning. In this paper, an Allocation-On-Demand Incremental (AoDI) Volume mechanism is presented. AoDI has double-level volume structure and uses appending rather than overwriting strategy to deal with the write requests. Both space utilization and write performance of storage can benefit from AoDI. Moreover, using an automatic volume expanding technique, AoDI can allocate storage space always matching users' real-time requirement to avoid space wasting. We also design a Non-COW snapshot based on AoDI that is dramatically faster than traditional COW snapshots. We implement AoDI based on LVM in Linux. Our experimental results show the great performance advantage of AoDI in snapshot and random access.

In future work, we will mainly focus on optimizing the metadata organization and defragmentation technology. As a platform, AoDI can also be used by other applications, such as remote replication, continuous data protection and so on.

Acknowledgments

We would like to thank the anonymous referees for their time and appreciate the valuable feedback. This work was supported in part by the National High Technology Research and Development Program of China (2008AA01Z401) , NSFC of China (60903028,61070014) , RFDP of China (20070055054) , and Science and Technology Development Plan of Tianjin (08JCYBJC13000).

6. REFERENCES

[1] TPCD traces. http://tesla.hpl.hp.com/public_software/tpcd97.

[2] Hasentein M.LVM Whitepaper. http://www.sistina.com, 2001.

[3] A. Andrzejak, M. Arlitt, and J. Rolia. Bounding the Resource Savings of Utility Computing Models. Technical report, HP Labs, December 2002.

[4] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the Clouds: A Berkeley View of Cloud Computing. Technical report, EECS Department, University of California, Berkeley, February 2009.

[5] A. Azagury, J. Satran, and W. Micka. Point-in-Time Copy: Yesterday, Today and Tomorrow. In *Proceedings of the 10th NASA Goddard, 19th IEEE Conference on Mass Storage Systems and Technologies*, pages 259–270, 2002.

[6] C. Lee, J. P. Lehoczky, D. Siewiorek, R. Rajkumar, and J. HANSEN. A Scalable Solution to the Multi-Resource QoS Problem. In *IEEE Real-Time Systems Symposium - RTSS*, pages 315–326, 1999.

[7] D. T. Meyer, G. Aggarwal, B. Cully, G. Lefebvre, M. J. Feeley, N. C. Hutchinson, and A. Warfield. Parallax: Virtual disks for virtual machines. In *ACM SIGOPS Operating Systems Review*, volume 42, pages 41–54, 2008.

[8] S. Nanda and T. cker Chiueh. A Survey on Virtualization Technologies. Technical report, Department of Computer Science, SUNY, Stony Brook, February 2005.

[9] P. Padala, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, and K. Salem. Adaptive control of virtualized resources in utility computing environments. *ACM SIGOPS Operating Systems Review*, 41:289–302, 2007.

[10] S. Quinlan and S. Dorward. Venti: A new approach to archival storage. In *FAST*, pages 89–101, 2002.

[11] M. Wachs, M. A. el malek, E. Thereska, and G. Ganger. Argon: performance insulation for shared storage servers. In *FAST, USENIX Association*, pages 5–5, 2007.

[12] C. A. Waldspurger. Memory resource management in VMware ESX server. In *OSDI '02: Proceedings of the 5th symposium on Operating systems design and implementation*, pages 181–194, 2002.

[13] A. Warfield, R. Ross, K. Fraser, C. Limpach, and S. H. Parallax: Managing storage for a million machines. In *Proceedings of the 10th Workshop on Hot Topics in Operating Systems*, volume 10, pages 4–4, 2005.

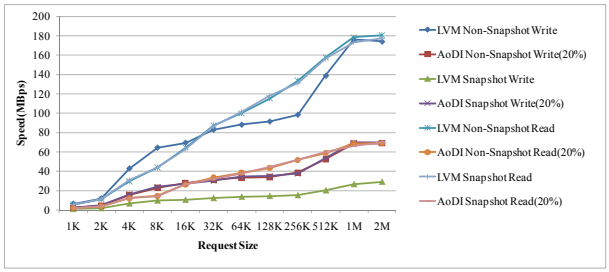

Figure 13: Write/Read Performance Before/After 3 Snapshots.