Design and Implementation of an Out-of-Band Virtualization System on Solaris 10*

Yang Wang[1], Wei Xue [2], Ji-Wu Shu [3], and Guang-Yan Zhang [4]

Department of Computer Science and Technology, Tsinghua University,
Beijing 100084, China
iodine01@mails.tsinghua.edu.cn,
xuewei@mail.tsinghua.edu.cn,
shujw@tsinghua.edu.cn,
zhang-gy04@mails.tsinghua.edu.cn

Abstract. In out-of-band virtualization systems, it is typical that the virtual storage manager (VSM) maintains the metadata for all Agents and performs the address mapping. The Agent performs I/O access according to the mapped address. High network latency, poor performance and low scalability are the main problems of this method. This article introduces an improved design and implementation of an out-of-band virtualization system, in which each Agent maintains a copy of the metadata and performs the address mapping by itself, so as to improve performance, reliability and scalability. This article discusses the on-line extension of a logical volume and the related problems such as synchronization. This function improves the ability of uninterrupted service. The system was tested on the UFS of Solaris 10 and the results showed that the performance of a stripe volume consisting of four disks exceeded that of a linear volume by an average of 103.65%.

1 Introduction

Storage virtualization is an important technique in Storage Area Networks (SANs). It brings the following benefits: 1) Reducing machine downtime [1]. 2) Increasing storage resource utilization[1]. The disk utilization can be increased up to 80% through the administration of virtualization software [2]. 3) A virtual volume can become much larger than the size of a single disk or even than a single RAID-system [2]. 4) Offering new degrees of flexibility. Storage systems can be added to or removed from storage pools without downtime [2]. We designed and implemented a virtualization solution for better utilization of our SAN system, TH-MSNS[3].

Storage virtualization system can work in two ways: in-band and out-of-band [4]. In an in-band virtualization system, data and control information flow in the same path. Data transmission and address mapping are both done by the Virtual Storage

---

Manager (VSM). In an out-of-band virtualization system, the VSM is located outside of the data path, so as to improve performance and scalability, but different Agents are required on different platforms [5,6]. HP's OpenView [7] is a typical out-of-band virtualization system. It implements an Agent in the HBA driver on the host. Open-View is limited to a given HBA card and driver, and has poor compatibility. Our system implements an Agent on the volume manager level, universal for different kinds of HBA cards and drivers.

In a typical out-of-band virtualization system, the VSM maintains metadata and uses it to perform address mapping. The Agent has to query the VSM in each I/O access for address mapping. Our experiment has shown that this method brings too much communication latency, thus reducing the throughput of the disks. Therefore, we introduced an improved design of an out-of-band virtualization system [2]: the Agent gains a copy of the metadata from the VSM and conducts address mapping by itself. This method provides the following benefits: 1) High performance: the Agent communicates with the VSM only when getting the copy of metadata and does not require any communication in I/O accesses. The virtual volume can have a performance level comparable with the local disk. 2) High reliability: the Agent maintains a copy of the metadata. Even if the VSM fails for some reason, the Agent can still work. 3) High scalability: the work of address mapping is distributed to every Agent. The VSM has low overhead and is able to manage more Agents and logical volumes.

Volume Extension, which means enlarging a logical volume without damaging its data, is an important characteristic of virtualization systems. Volume Extension includes not only the updating of metadata, but also the refreshment of the file system. Therefore, Volume Extension requires support from both virtualization software and OS and its file system. Some third party software such as PowerQuest Partition Magic can change the size of a volume without damaging its data, but they normally work off-line, requiring rebooting or terminating of other applications. A modern storage system is required for 7×24 on-line time [8], so off-line extension cannot satisfy the requirement. Solaris provides a lot of support for on-line extension, but it is not integrated in the virtualization system and needs complex management. This article researches how to extend a logical volume automatically with support from both the virtualization and operating system. We implemented an on-line extension function on Solaris 10 and UFS (Unix File System).

This article introduces the design and implementation of the virtualization system on Solaris 10. Section 2 introduces the design of VSM. Section 3 describes the design and implementation of the Agent on Solaris 10. Section 4 presents the design and implementation of on-line extension. Finally, the test results of the linear and stripe volumes on UFS are shown.

2   Design of VSM

The VSM is responsible for the management of metadata. The design of the VSM is shown in Fig.1, consisting of five modules:
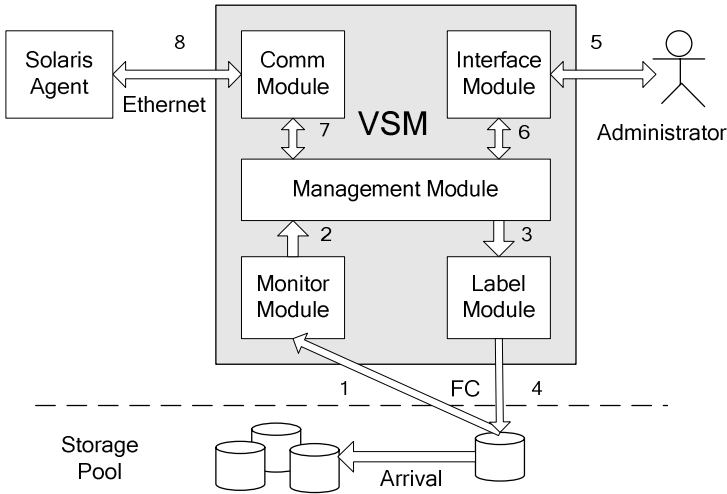
Fig. 1. Design of the VSM

The Management Module is the core of the whole system, maintaining metadata for all Agents and controlling and coordinating other modules.

The Monitor Module watches the status of all storage devices. When a storage device is added or removed, the Monitor Module will inform the Management Module (process 1 and 2).

The Label Module writes an identifier (UUID) at a specific location of the physical disk. It is used to identify the same physical disk among different machines. When the Monitor Module detects the arrival of a new disk, the Management Module will inform the Label Module to do this process (process 3 and 4). This UUID is used as a global name for this physical disk in messages between the VSM and the Agent.

The Interface Module processes the administrator's commands and displays the VSM's information (process 5 and 6). Two methods are provided: command line and GUI, and its main tasks are command parsing and information displaying.

The Comm Module is responsible for the communication between the VSM and Agent. The main content of communication is the metadata and the returning information after executing commands (process 7 and 8). Communication is based on Ethernet and TCP.

3   Design and Implementation of Agent on Solaris

The Solaris Agent consists of a Comm Module in user space and a Driver Module in kernel space as shown in Fig.2. The Comm Module receives commands from the VSM, translates them and sends them to the Driver Module. The Driver Module performs the address mapping and consists of three parts: the metadata of a logical volume—LV, the accessing point for the Comm Module—admin pseudo device and the address mapping and dispatching module—I/O.

Implementing the Comm Module in user space can improve the reliability and portability of the whole system, but it brings extra inter-process communication overhead. The Comm Module works only when updating metadata, and the frequency of metadata updates is much lower than the frequency of disk accesses, so this overhead will not affect the I/O performance of a logical volume.
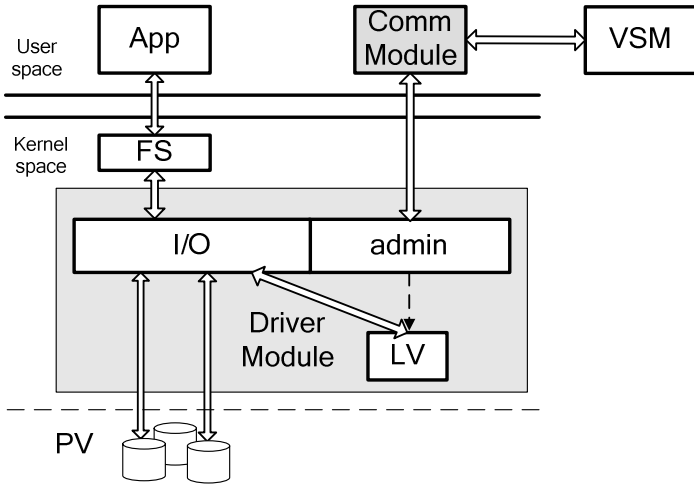

Fig. 2. Design of the Agent

3.1   Comm Module

The Comm Module receives commands from the VSM, checks their validity, and then translates them into formats that can be understood by Driver Module. It sends them to the Driver Module through the admin pseudo device and finally returns the executing result to the VSM. The commonly used commands include creating a logical volume, setting the metadata of a logical volume and removing a logical volume, etc.

The translation work done by the Comm Module includes: 1) Translation from the UUID to a local disk identifier. As described in Section 2, to identify a physical disk between the VSM and Agent, the VSM writes a UUID at a specific location on the disk, and this UUID is used in managing commands to identify this disk. It is the Comm Module's work to translate this UUID into an identifier which can be understood by the local machine. 2) Translation of formats: for performance consideration, the metadata transferred on the network is not suitable for the Driver Module, and the Driver Module can only receive data in a continuous memory space. It is also the Comm Module's work to do this translation.

The Comm Module works entirely in background and displays no message. For observation of the executing results, the Comm Module sends the messages back to VSM. The administrator can watch these messages through a command line or GUI.

3.2   Driver Module

When loaded, the Driver Module creates the admin pseudo device for the Comm Module and the Comm Module can use some special IOCTLs to send commands to the Driver Module through the admin pseudo device. After receiving metadata from the Comm Module, the Driver Module saves it and allocates some data structure for it such as locks or reference counts. When the metadata is updated on the VSM, the Driver Module also updates its copy according to the managing command.

The main task of the Driver Module is to perform address mapping when there is an I/O access. Our system supports two kinds of address mapping ─linear and stripe. A linear volume is used to enlarge the size and a stripe volume is used to improve performance. These two can be used in a mixed way.

Address mapping and dispatching is the key process in the whole system. Solaris uses a buf_t data structure to describe an access [9]. In the Driver Module, the disk accesses are accomplished in an asynchronous way: the Driver Module receives the original buf_t structure containing the logical address from the application and breaks it into several child buf_ts containing the physical addresses. Each physical buf_t is bound with the completion routine and then sent to the lower disk driver. When an access is finished, the OS notifies the Driver Module by calling the completion routine, which means that the Driver Module does not wait for the accomplishment of these accesses. When all the child buf_ts are finished, the Driver Module notifies the upper level of the accomplishment of the parent buf_t. Only through this method could the advantages of the stripe volume be realizedd to improve performance.
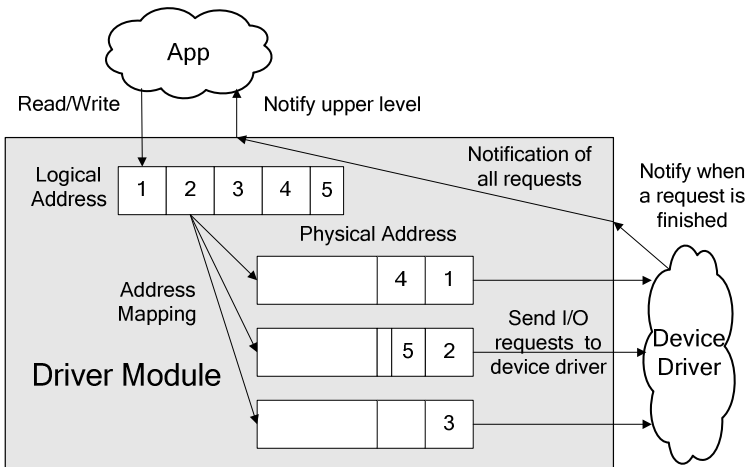

Fig. 3. Process of an I/O access

Fig 3 describes the process of an I/O access to a stripe volume consisting of three physical disks. At first, the application gives the logical address to access, and the Driver Module performs the address mapping. In this example, the whole logical address was mapped to five segments of physical addresses on three disks. The Driver Module sends these requests (buf_ts) to a lower disk driver for asynchronous I/O

access. Each time the disk driver finishes a request, it will notify the Driver Module. After the Driver Module has been notified of all the five requests, it returns the result to the application. The access from the Driver Module to the lower disk driver is asynchronous, but the access from the application to the Driver Module can be either asynchronous or synchronous, according to the requirement of the application.

4   Design and Implementation of On-Line Extension

On-line extension of a logical volume includes two steps: the update of metadata and the refreshment of the file system. The former is quite simple: the VSM sends the new metadata to the Agent through their Comm Modules. But the latter is quite complex. The file system level is higher than disk device level logically. Thus, the refreshment of the file system should not be implemented in the device driver, but should be done by the OS. So the extension of a logical volume requires support from both the virtualization system and the OS and its file system. Solaris 10 can achieve this job on the UFS (growfs) [10]. In the process of extension, synchronization operations such as lock and unlock are necessary . The locking time should not be too long for consideration of the time needed to respond to an application. Three types of extension processes are discussed as follows:

1) Logical Volume without a file system: Some database software can bypass the file system and use the block device directly. For a logical volume without a file system, the process is as follows: locking metadata in the Driver Module, updating metadata and unlocking metadata. This is a simple and fast process.

2) Logical Volume with a file system but not mounted: The refreshment of the file system requires access to the logical volume. In this process, the metadata cannot be locked. To keep data sychronization, it is necessary to instruct the OS to lock the file system first (a writing lock is sufficient), and then the refreshment of the file system can be done. Solaris provides such support (lockfs, growfs) [10].

3) Logical Volume with a file system and mounted: This is the most common status. On-line extension requires that the volume cannot be umounted. Support from the OS is also required. The process is similar to 2). Solaris 10 provides support for this status (growfs –M). If the size of a logical volume changes greatly or the machine is slow, the refreshment of the file system takes a long time. In the process, the volume is locked and all I/O accesses are suspended. To solve this problem, a logical volume can be enlarged in several steps: after the file system is locked, only part of the file system is refreshed and then the file system is unlocked. This process is repeated several times until the whole refreshment is accomplished. In this way, the response time to applications is reduced. Fig.4 describes this process.
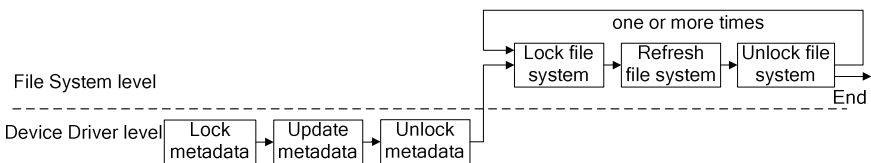

Fig. 4. Extension of a logical volume with a file system and mounted

5   Results and Analysis

We used Iometer [11] to test the performance of the out-of-band virtualization system. In the test, a Solaris Agent, a VSM and an FC disk array were used. The Solaris Agent was installed on a two-way 1.2 GHz UltraSPARC-III Cu machine with 4GB of memory and a Qlogic 2310 HBA card running SunOS Release 5.10 Version. The VSM was installed on a two-way 2.4 GHz Intel Xeon machine with 1 GB of memory and an Emulex LP982 HBA card running Linux kernel v.2.4.16. Via a Brocade Silk Worm 3800 fibre channel switch, these machines were connected with an FC disk array controlling five 146 GB Seagate Cheetah 10K disks.

The parameters of the Iometer were as follows: 4 worker threads; 80% reading access and 100% random access; the size of the logical volume (linear and stripe) was 8GB; every test lasted for 15 minutes. The file system used by Solaris was UFS. The size of I/O requests ranged from 8KB to 4MB. The stripe volume consisted of 4 FC disks with a stripe size of 64KB.
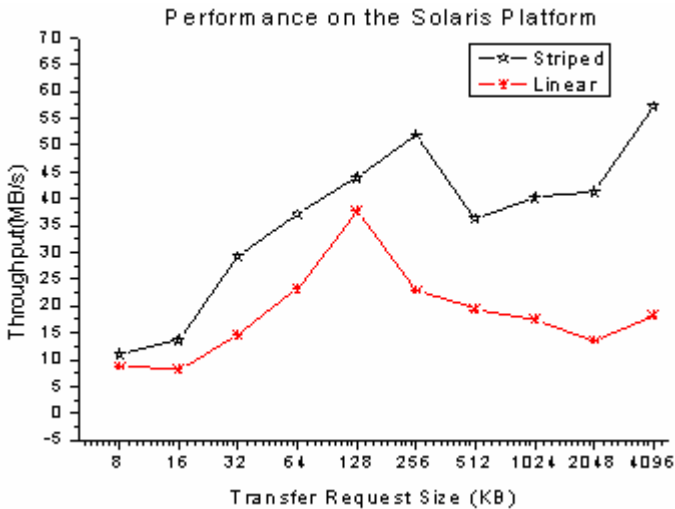

Fig. 5. Results of the performance test

The results of the performance test are shown in Fig 5. The stripe volume exceeded the linear volume to different extents. The stripe volume consisting of 4 disks exceeded the linear volume by a maximum of 215.01%, a minimum of 16.77% and an average of 103.65%. The advantage of a stripe volume is not distinct in small-size requests as a result of cache and file system influence, but it is quite significant in large-size requests when the volume performance becomes the key factor. At the last two points (2048KB and 4096KB), the throughputs of the stripe volume are both more than 3 times than those of the linear volume. The figure depicts the performance in a real system with different request sizes. The right part of the figure shows the performance of the logical volume itself.

6   Conclusion

This article introduces the design and implementation of an improved out-of-band virtualization system on Solaris 10. This system distributes the overhead of address mapping to every Agent, improving performance, reliability and scalability. It also implements on-line extension of a logical volume, providing better support for 7×24 uninterrupted service.

A performance test with Iometer on UFS showed that the performance of a stripe volume consisting of four disks exceeded that of a linear volume by an average of 103.65%. The advantage of the stripe volume was distinct when the request size was large, providing more than three times of throughput than the linear volume.

References

1. The Storage Networking Industry Association (SNIA). Storage Virtualization I: What, Why, Where and How.
2. Andr´e Brinkmann, Michael Heidebuer, Friedhelm Meyer auf der Heide, Ulrich R¨uckert, Kay Salzwedel, and Mario Vodisek. V:Drive–Costs and Benefits of an Out-of-Band Storage Virtualization System, In Proceedings of the 12th NASA Goddard, 21st IEEE Conference on Mass Storage Systems and Technologies (MSST), College Park, Maryland, USA, 13 - 16 April 2004.
3. Shu Ji-wu, Li Bigang, Zheng Wei-min, Design and Implementation of a SAN System Based on the Fiber Channel Protocol, IEEE Transactions on Computers, 54(4), 2005: 439-448.
4. Spinnaker Networks. White Paper: A New Approach to Storage Virtualization.2002.
5. Meng Ran, Shu Ji-wu, Xue Wei, Design and Implementation of an Out-of-Band SAN virtualization system Based on Windows NT Architecture, IFIP International Conference on Network and Parallel Computing 2005, LNCS 3779,371-378.
6. Zhang Guanyin, Shu Jiwu, Xue wei, Zheng Weimin, MagicStore: A New Out-of-Band Storage Virtualization System in SAN Environment, IFIP International Conference on Network and Parallel Computing 2005, LNCS 3779,379-386.
7. HP Open View Storage Operations Manager, http://h18006.www1.hp.com/products/ storage/software/som/index.html.
8. Charles Milligan, Sid Selkirk. Online Storage Virtualization: The key to managing the data explosion, Proceedings of the 35th Hawaii International Conference on System Sciences – 2002.
9. Sun Microsystems, Inc. Writing Device Drivers. Part No: 816–4854–10. Jan. 2005. http://docs-pdf.sun.com/816-4854/816-4854.pdf.
10. Sun Microsystems, Inc. Solaris Volume Manager Administration Guide. Part No: 816–4520–10. Jan 2005. http://docs-pdf.sun.com/816-4520/816-4520.pdf.
11. IoMeter Project, http://sourceforge.net/projects/iometer/.