Research and Implementation of a Snapshot Facility Suitable for Soft-Failure Recovery

Yong Feng, Yan-yuan Zhang, Rui-yong Jia

Computer Science & Engineering School, Northwestern Polytechnical University
710072 Xi'an, Shaanxi, China
{fengyong, zhangyy, jiary}@co-think.com

Abstract. Human error and incorrect software (a.k.a. soft-failure) are key impediments to dependability of Internet services. To address the challenge, storage providers need to provide rapid recovery techniques to retrieve data from a time-based recovery point. Motivated by it, a snapshot facility at the block level called SnapChain is introduced. Compared with former implementations, when managing different versions of snapshots, SnapChain minimizes disk space requirement and write penalty of master volume. In this paper, the metadata and the algorithms used in SnapChain will be explained.

1 Introduction

The Berkeley/Stanford ROC (Recovery-Oriented Computing) Project claims that rather than device failure (a.k.a. hard-failure), human error and incorrect software (a.k.a. soft-failure) are the largest causes of failures in Internet services [1]. To protect against soft-failure, creating point-in-time copy of data periodicity and maintaining different versions of point-in-time copies are necessary. Amazon.com, for example, is reported to create point-in-time copy of data as frequently as three times per hour [1].

Differ from the other two classes of point-in-time copies, split mirror and concurrent, snapshot requires much less storage and needs no advanced setting up prior to executing a point-in-time copy. Therefore, it is more suitable for soft-failure recovery. For snapshot facility, the expectation for continuous operation is commonplace. However, the ability to create and maintain different versions of point-in-time copies of the data efficiently, with minimal interruption and minimal overhead, is critical. This paper will only focus on the block level snapshot facilities.

Linux LVM and EVMS [2] are volume managers and support snapshot. In them, the snapshot of master volume, namely shadow volume, is achieved through a pseudo volume that contains pointers to two separate physical regions. One region is simply the unchanged blocks in the master volume. The other region, named private region, collects the original state of the master volume blocks just before they are updated, as well as any new changes made by the point-in-time consumer to the snapshot. Sun StorEdge [3] treats master volume and its related shadow volume as a volume pair. A bitmap is used to keep track of the differences of a volume pair that occur after the established point-in-time. In above implementations of snapshot, the shadow volumes of the same master volume have no relationship with each other. If they can share the old data just like master volume and shadow volume share unmodified contents, the mount of data, which is copied from master volume and kept in the private region of shadow volume, can be reduced. Based on such optimizations, SnapChain, the snapshot facility presented in this paper, can minimize the effort, time and incremental capacity, which are necessary to obtain point-in-time views. Especially in the environment supporting soft-failure recovery, where a master volume has a number of shadow volumes, the effects of optimizations are remarkable.

The remainder of the paper is organized as follows: The next section describes the metadata and algorithms used in SnapChain. After evaluating SnapChain in section 3, the conclusion will be drawn in the fourth section.

2 SnapChain

SnapChain is implemented in Linux, and designed to support up to 255 shadow volumes for a master volume. It uses a pool-and-pointer design, where metadata keeps the information of location and states of data chunk (a group of block) of shadow volumes. Shadow volumes of the same master volume share a large storage pool, named snapshot pool, to keep their private data. Every shadow volume owns a logic private region in snapshot pool. The snapshot pool is composed of one or multiple block devices in liner mode, and can be expanded through absorbing new devices on demand. Registering a volume in SnapChain will make it into a master volume. After registering, the content of master volume is kept untouched. Losing metadata or uninstalling SnapChain will not render data of master volume unusable.

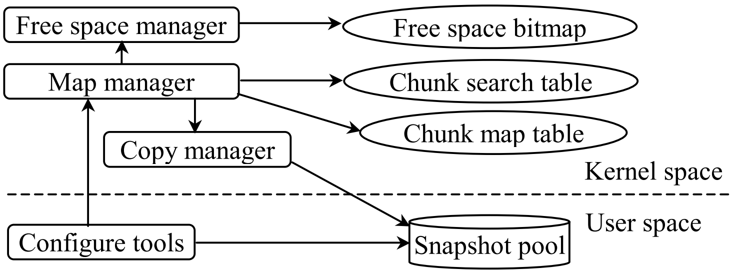

Fig. 1. Architecture of SnapChain

As Fig. 1 indicates, SnapChain has a mid-level block device driver and user space tools. The device driver can be inserted arbitrarily in a system's layered block I/O hierarchy. It consists of three components: map manager, free space manager, and copy manager. Map manager uses CMT (Chunk map table) and CST (chunk search table) to translate the virtual address of shadow volume into the corresponding physical address of master volume or snapshot pool. Free space manager uses FSB (free space bitmap) to allocate and reclaim disk space from snapshot pool for shadow volumes to store their private data. To maintain consistency of shadow volumes, copy manager controls copying data from master volume to snapshot pool. The user space tools used to configure device driver follow a set of operations.

2.1 Metadata

CMT is used to mark the current states of chunks in shadow volume and master volume. Each volume owns a CMT, and every chunk has a flag in CMT. For a chunk of master volume, the flag has two states: 0 means the chunk is not updated since the newest shadow volume is created; 1 means the chunk is updated. For a chunk of shadow volume, the flag has three states: 0 means the chunk is not stored in the private region of the shadow volume; 1 means the chunk is stored in the private region of the shadow volume and is the original state of master volume; 2 means the chunk is stored in the private region of the shadow volume and is updated by the point-in-time consumer. CST is used to map the virtual chunk of a shadow volume into the physical chunk in snapshot pool. Each shadow volume has a CST. In order to search rapidly, CST is organized into a hash table. The metadata of shadow volumes of the same master volume are chained by a bi-direction list, namely shadow volume list, in time order. The master volume keeps the head of the list. Extent (a group of chunks) is the unit during allocating disk space. In FSB, every extent of snapshot pool has one bit to record its allocation state.

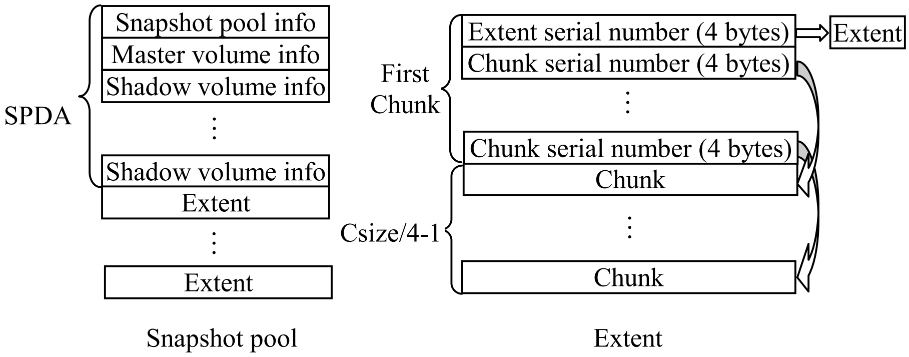

Fig. 2. Data layout in snapshot pool

It is not necessary to explicitly store CMT, CST and FSB on disk. Instead, enough information on snapshot pool can be maintained to allow them to be reconstructed. Fig. 2 shows the data layout in a snapshot pool. The first 16KB is SPDA (Snapshot Pool Descriptor Area), including snapshot pool info, master volume info, and shadow volume info. On each device of snapshot pool, SPDA is allocated for backup reasons. The remainder of snapshot pool is continuous extents. The extents are divided into several classes: free extent class and extent classes of each shadow volume. The PL method [4] is used to manage the extents. The first 4 bytes of each extent are used as a pointer to indicate the next extent of the same class. Thus the extents of the same class become a list. Shadow volume info keeps the head of the shadow volume extent list. In an extent, the first chunk, except the first 4 bytes, is used to record the serial numbers of the following chunks. The nth is the serial number of the virtual chunk, which the (n+1)th chunk in the extent corresponds to. The size of chunk (Csize) is chosen between 1KB and 8KB, and the size of extent is ($Csize^2/4$) MB. Large chunk size can reduce the footprint of the metadata and increase the addressing range of CST. However, it may result in significant performance overhead.

2.2 Commands and algorithms

SnapChain information can be created, displayed, and manipulated by the user space tools. The command "spcreate test_mv /dev/sda1 /dev/sda2" registers the master volume /dev/sda1 and creates its snapshot pool on /dev/sda2. The command spremove removes from the system all knowledge of the specified snapshot pool and releases the master volume from device driver. The similar commands svcreate and svdelete can be used to create and delete shadow volumes. When svcreate is executing, SnapChain stalls all incoming I/O requests for the time required to flush all outstanding writes to the master volume. When everything is synchronized on stable storage, SnapChain appends new metadata of shadow volume to shadow volumes list, and creates a new virtual block device. The command svrestore restores master volume from the specified shadow volume. SnapChain also provides to user programs an ioctl command interface and a /proc interface for device information and statistics.

The read request to master volume is directly made to master volume. When writing master volume, COFW (copy on first write) [3] is executed. Unlike former implementations, only the original date of the chunk with the CMT flag of 0 is copied to the private region of the newest shadow volume. To find the physical chunk that holds the request block of specified shadow volume, SnapChain first references CMTs of shadow volume list to locate the volume whose private region the specified chunk is stored in. It will take the following steps:

1. If the CMT chunk flag of the target shadow volume does not equal to 0, the targe shadow volume is what we want to find.
2. Otherwise, check the newer shadow volume. If the CMT chunk flag is 1, it is the volume that we want to find. If not, repeat until the volume is found. If no such shadow volume exists, the master volume is the volume that we want to find.

Then SnapChain uses CST of the found volume to get the physical address. When writing shadow volume, if the CMT flag of the target chunk is 1, the data of target chunk should be moved to elder shadow volume first. Deleting a shadow volume involves the similarly procedure. The command svrestore uses the algorithm of reading shadow volume to get the contents of the specified shadow volume, and uses the algorithm of writing master volume to update the master volume.

3 Evaluation

The goal of SnapChain is to minimize the write penalty and the capacity necessary to obtain point-in-time views. The response time and consumed capacity of SnapChain will be compared to Linux LVM. As expected, SnapChain achieves better performance and less consumed capacity.

Table 1. Consumed capacity and response time when writing to master volume.

Snapshot facility	SnapChain		Linux LVM	
Shadow volume number	4	8	4	8
Consumed Capacity (MB)	12	12	48	96
Response Time (ms)	2335	2338	5481	9767

Table 2. Response time when reading from shadow volume.

Snapshot facility	SnapChain		Linux LVM	
Shadow volume number	4	8	4	8
Response Time (ms)	180	179	178	180

The tests are run on a dual Pentium 700 MHz computer with 256 MB of RAM. NEC S2100 disk array (RAID 5) is connected directly to Emulex Lightplus 750 HBA card in the PC. The versions of Linux kernel and LVM are 2.4.20 and 2.00.07 respectively. The chunk size used by LVM and SnapChain is set to 4KB. To avoid the influence of Linux buffer cache, “raw” devices associated with the block devices are used. In the first test, the master volume has 4 shadow volumes. In the second test, it has 8 shadow volumes. Table 1 shows consumed capacity and response time, when copying 10MB data to the area of master volume, which has not been updated since the first snapshot is created. The memory consumed and response time of the SnapChain is much less than those of LVM, for only 10MB data is copied to snapshot pool in SnapChain rather than n*10MB data in LVM, where n is the number of shadow volumes. As the number of shadow volumes increased, the degree of superiority of SnapChain over LVM is increased. Table 2 shows response time, when reading 10MB data from a shadow volume. The response times of SnapChain are similar with those of LVM. Thus the optimizations do no harm to the performance of accessing shadow volume. Since restoring master volume from a specified shadow volume involves reading from shadow volume and writing to master volume, the recovery time of the SnapChain is better than that of LVM.

4 Conclusion

In this paper, we propose SnapChain – a block level snapshot facility. SnapChain enables master volume and its snapshots to share as many contents as possible. Consequently it can support more snapshots, and achieve better performance of accessing master volume, than its peers do. SnapChain supports writing request to snapshot. Besides, it can create snapshot transparent to application without disturbing current data accessing, and recovery a master volume from a specified copy-in-time in a short time. Therefore SnapChain is suitable for soft-failure recovery.

References

1. A. Brown: A Recovery-Oriented Approach to Dependable Services: Repairing Past Errors with System-Wide Undo. UC Berkeley Computer Science Division Technical Report UCB//CSD-04-1304, December 2003
2. D. Teigland, H. Mauelshagen: Volume managers in linux. Proc. the FREENIX Track of USENIX 2001 Annual Technical Conference, Boston, USA, 2001, 185-197
3. Sun Microsystems. Instant image white paper. http://www.sun.com/storage/white-papers/ii_soft_arch.pdf
4. Arun Iyengar, Shudong Jin, Jim Challenger: Techniques for efficiently allocating persistent storage, The Journal of Systems and Software 68(2), 2003, 85-102