Ninth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing

ESnapII : A Writable Dependent Snapshot System with Shared Cache

Guangjun Xie, Lu Qi , Feng Wang, Gang Wang, XiaoGuang Liu, Jing Liu
College of Information Science and Technology,
Nankai University, Tianjin 300071, China
Xieguangjun1980@163.com

Abstract

Snapshot technology, which has been widely used in data storage system for data protections and other tasks such as data mining and data cloning, is becoming one of the key technologies in storage area. ESnap[16] improved the performance, resource consumption and reliability compared with traditional LVM snapshot implementation. But it still has some problems: (i) high memory consumptions caused by snapshot metadata, (ii) not support writable snapshot. A novel optimization of Esnap called ESnapII, which alleviates all the above four difficulties is proposed in this paper. In detail, a global metadata cache shared by all snapshots belongs to the same VG is proposed to reduce the memory consumption; Future more, a writing mechanism is presented for both dependent and independent snapshot. Experiment results show that ESnapII has higher performance and lower resource utilization rate than previous work..

1. Introduction

In recent years, snapshot is becoming a key technology in data backup and disaster recovery. Just as its name implies, snapshot is the instantaneous image of storage system in specific time. It can help the storage management system to perform online backup and other task just like data mining and data cloning.

LVM (Logical Volume Manager) is a popular storage virtualization system on Linux platform. The main idea of LVM snapshot is COW (Copy-on-Write), which means that the data are only copied to snapshot before they are updated for the first time. But there are many problems in LVM snapshot in its resource occupation, performance and reliability, which depress the availability of systems. The main defects of the traditional LVM include (i) the size of the snapshot metadata is linear with its storage space; (ii) the write performance on origin which has multi-snapshots can be poor caused by the extra COW operations; (iii) the snapshot fails if the COW data overflows in its storage space. To solve these problems, ESnap[16] was developed in previous work, which used a novel metadata cache scheme to reduce the memory occupation. Esnap also applied dependent snapshot to improve the COW performance of origin.

In this paper, an improved ESnap, called ESnapII, is presented. In order to reduce the memory occupation of snapshot, a shared cache scheme of multi-snapshots belongs to the same VG is designed. According to this scheme, snapshot metadata is stored on its local disk space initially. A dynamic schedule algorithm is used to load metadata into memory if needed, and all the metadata in memory from different snapshot shares a global metadata cache pool rather than local cache space applied in ESnap. This strategy can improve the I/O performance on snapshot under most kinds of workload because of the improvement of metadata cache hit rate. We also present a writing mechanism for both common snapshots and dependent snapshots.

2. Related work

Snapshot has been widely used in storage area such as data backup and version management. There are four main snapshot technologies: split-mirror[1][3], copy on demand[5], virtual view[2][4] and incremental snapshot[8][9]. Shah proposed an optimization of LVM snapshot [10], which improved performance 18% - 40% compared with the traditional method. The Blue Whale system implemented iterative snapshot mechanism [11]. ESnapII can achieve the same object by creating two simultaneous dependent snapshots simply and setting the older one as a read-only volume.

Yong Feng[12] introduced a snapshot facility at the block level, which minimized disk space requirement and write penalty of master volume. Our design of dependent snapshot in ESnap is similar to this work.

978-0-7695-3263-9/08 $25.00 © 2008 IEEE
DOI 10.1109/SNPD.2008.94

But we presented a novel metadata cache method in [16]. Zhen[13] recommended a system SSDCD (snapshot system on disk caching disk) implementing snapshot on the block device. This system implemented the function of snapshot on the block device with the method of COW. Sitaraman[14] built a consistent snapshot of file which used a logging mechanism to record the modifications to each disk block, and then employed fast algorithms to reconstruct the contents of the file as it existed sometime in the past. Xiao et al [1] proposed a quantitative performance evaluation of different snapshot methods at block device level. The layout and operation of snapshot metadata is one of the key problems in snapshot implementation. To the best of our knowledge, we are the first one to design and implement metadata cache management mechanism on snapshot.

3. Overview of Esnap

3.1 Metadata organization in ESnap

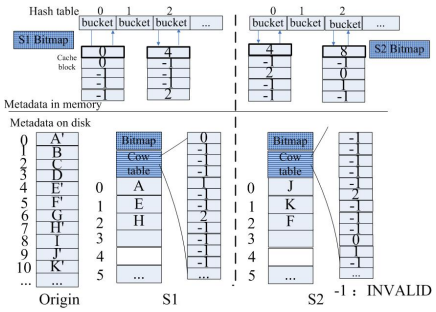

Figure 1. Metadata layout of Esnap

Esnap[16] use a novel metadata organization to reduce the memory consumption, The main idea is that only place the whole metadata on disk, and a cache mechanism like file systems is used. As shown in Figure 1, the whole COW table is stored in the start area of the snapshot storage space instead of scattering it over the space. The COW table entries are not arranged according to the snapshot volume address, but the original volume chunk number. So the original volume address of each chunk need not be recorded because it can be obtained from the table index. the COW table is spited into fixed size (such as 4KB) segments as cache blocks. The cache blocks are partly cached in memory and linked into a hash table to accelerate the metadata search. Request can be easily computed to which COW table segment it belongs based on its address. If the segment is not in the hash table, the segment needs to be read from disk to main memory and to be linked into the hash table. And then can find the COW entry easily to determine whether this block has been COWed. Two different metadata cache replacement strategies were implemented in ESnap: round robin and LRU.

3.2 Dependent snapshot

If there are many snapshots based on the same origin, the write performance may be very poor, and the COW data chunks stored in different snapshots are redundant. In ESnap, dependent snapshot algorithm was proposed to solve this problem. The main idea of dependent snapshot is that it only copies the old data chunk to the latest created snapshot for multi-snapshots based on the same original volume for COW operation. So one original volume write operation can only cause at most one COW operation. In Esnap, the snapshots which belong to the same origin were organized as a chain. As shown in Figure 2, the write operations on original volumes can only cause COW operation on the recent created snapshot volume. For example, the user updates chunk A at 9:15am, ESnap only copies A to S2 – the newest snapshot (LVM copies A to S1 and S2, see Figure 2).

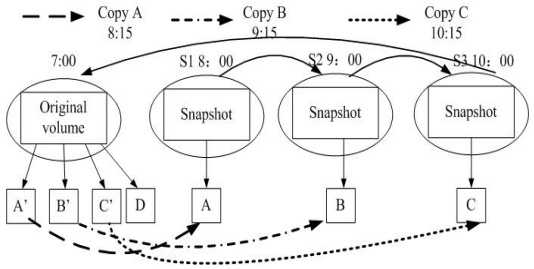

Figure 2. Dependent snapshots

4.The design of ESnapII

In order to solve the memory problem of ESnap, we designed a novel global cached exception table, which can be shared by all the snapshots in the same VG. We also implement the snapshot write function for both common snapshot and dependent snapshot.

4.1.Global Metadata Cache

In ESnapII, we use the same snapshot disk metadata layout as ESnap, which stores the whole metadata in the start area of the snapshot storage space and the remained space is fully used for storing COW chunks. The metadata of snapshot is divided into two parts: an initializing bitmap and a COW table. For large origin, there will be many COW entries stored in the snapshot metadata area. But in memory, we use a global

metadata cache layout instead of the local cache provided in ESnap. As shown in Figure 3, we split COW table into many fixed size segments, which are used as cache blocks. In addition to the continuous COW entries, each cache block contains the device number of snapshot which it belongs to and the start index of the COW entries. If we cache this block, we store it in memory and link into a global hash table according to the snapshot device number and the start entry index of this block. For example as shown in Figure 3, after the original data of chunk H (chunk 7 in origin) has been COWed to the Number 2 chunk of S1, ESnapII will record the index 2 into the 7th entry of the S1's COW table and cache it in memory. Because the 7th COW belongs to the COW entries segment start with entry index 4, so the cache block contains the device number of S1, start entry index(4) and the content of entry 4-7. The initializing bitmap of each snapshot is also preserved in main memory. The key algorithm in snapshot system is for a given virtual address, determining whether this address in origin has been copied to the snapshot. This algorithm is described as Algorithm 1:

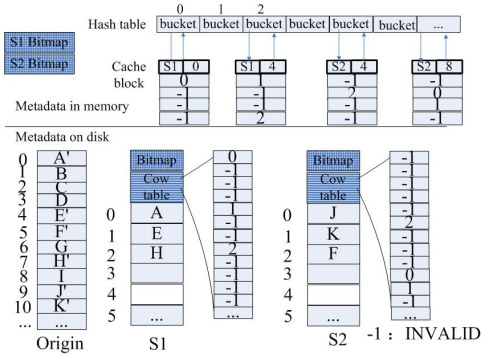

Figure 3. Metadata layout of ESnapII

```
Procedure: find_exception(snapshot, addr, exception)
    if !test_bit(snapshot, init_bitmap,
addr_to_segno(addr)) then
        initialize corresponding metadata segments
        read metadata segment or do replacement
        link the segment into hash table
        return fail
    else
        hash_table.search(snapshot,addr, &exception);
        if fail then
            read metadata segment or do replacement
from this snapshot
            link the segment into global hash table
            hash_table.search(snapshot, addr,
&exception);
        endif
        if exception.addr = INVALID then
            return fail;
        elseif exception.tag = COW then
            return succ;
        else
            return WRI;
        endif
    endif
end procedure
```

Algorithm 1. metadata search algorithm

Compared with ESnap which allocated a hash table for each individual snapshot, this design has some advantages when there are many snapshot volumes in system. First, the memory for snapshot metadata is predictable. Second, using this design, system can support more snapshot of massive origin and the memory resource consumption is not high. More important, if the workload on each snapshot is not balanced, the snapshot which becomes hotspot can occupy more global metadata space, so that the hit rate of the metadata cache can be improved. Obviously in most condition, the balance of I/O requests on each snapshot is impossible.

4.2 Snapshot Write implementation

Snapshot write function is complicated to design because of the dependent chain. in ESnapII we have implemented this function by which attaches a tag to each COW table entry to distinguish between COW chunks and snapshot updated chunks. In this way, a snapshot write function is divided into 3 conditions. If COW chunk is found in local snapshot S, it checks the COW entry status. If the status is UPDATE, it writes the request directly on S. If the status is COW, it copies the COW chunk to S's predecessor T first. If not found, walk through the dependent chain and ignore the COW chunk with status UPDATE, if found the COW chunk at a newer created snapshot, copy it to S, otherwise do a COW to copy the original data chunk to S, and then write to S. The algorithm implemented in ESnapII is described as follows:

```
Procedure: write_snapshot(buf, snapshot, addr)
    origin = snapshot.origin;
    snap = snapshot;
    snap1 = snapt.prev;
    do_COW = find_exception(snap, addr,
&exception);
    if do_COW = succ and
exception.status=UPDATE then
        write(buf, snapshot, exception.addr,
chunk_size);
        return;
    endif
```

```
    while do_COW <> succ and snap.next <> NULL
do
          snap = snap.next;
          do_COW = find_exception(snap, addr,
&exception);
          if exception.status = UPDATE then
                continue
          endif
    enddo
    if snap1 <> NULL then
          do_COW1=find_exception(snap1,addr,
&exception);
          if do_COW1 = fail then
                if do_COW = succ  then
      copy COW chunk from snap to snap1;
                else
                   find exception and do COW for
snap1;
                endif
          endif
    endif
    do_COW = find_exception(snapshot, addr,
&exception);
    if do_COW = fail then
          exception.status = UPDATE;
          call cow_data(origin, snapshot, addr,
&excpetion);
          find_exception(snapshot, addr,
&exception);
    endif
    write(buf, snapshot, exception.addr, chunk_size);
end procedure
```

Algorithm 2. Snapshot writing algorithm.

5 Experiments

5.1 Experiment Setup

We setup our experiment on a PC with an AMD Sempron 2500+, 1GB DDR memory and 2 Seagate 160G hard disks. The operating system is Redhat Enterprise Linux AS 4 update 3 with Linux kernel 2.6.9-43. In all experiments we create a 10GB original volume and the same size snapshot volumes. The snapshot chunk size is 64KB. A benchmark tool, Sysbench[17] is used to evaluate the performance of ESnapII. the total file size in each experiment is 8GB. The request size of work load generated by Sysbench is 16KB. To analyze the influence of the cache mechanism, we set an 800K shared metadata cache space for ESnapII. And the metadata memory occupation threshold for ESnap is set as 8KB / 1GB original volume space in every snapshot. We test four snapshot systems in our experiments: the original LVM1, ESnap round robin cache replacement strategy, ESnap with LRU strategy and ESnapII with round robin strategy, ESnapII with LFU strategy which are represented by “LVM1”, “I-RR”, “II-RR”,“I-LRU”, “II-LRU” respectively.

4.2 Evaluation and analysis

The origin sequential write performance was first tested by creating an original volume and a snapshot volume, and then writing 8GB data to the original volume sequentially. Figure 4 shows the result of throughput. We can see that RR, LRU and LFU have comparative performance. This result is comprehensible. The number of cache miss of sequential writing to an empty snapshot can be calculated easily. In this experiment, the COW chunks is 8G / 64K = 128K, So 128K COW entries were read and modified, and they are all continuous. Each cache block contains 4K / 4B =1K COW entries, so 128K / 1K = 128 cache misses happened. ESnap and ESnapII in each cache strategy must do 128 metadata initializing, 128 cache block reading, and 128K metadata segment writing. LVM doesn’t need metadata initializing and reading, so the performance is a little higher than ESnap and ESnapII.

Figure 5 shows the experiment result of snapshot random reading test for a single snapshot. We also create an origin and a single snapshot with 10GB. There is no difference between ESnap and ESnapII for reading a single snapshot. So we did not test reading performance of ESnap in this experiment. From the evaluation result, we can see the performance of LVM is higher than others. Since LVM holds all the metadata in memory, no metadata reading operations is performed.

This phenomenon can be analyzed through a simple calculation: As we say above, 8GB request data may influence 128K COW entries, the total size of this COW entries is 128K * 4 = 512K. But the total metadata space we set is 200K, so more than half of the metadata must be replaced from disk to memory. But for LVM, the entire metadata can be stored in the memory, so no extra operations which decrease the performance are performed. We can see the performance of LFU is almost equal to the RR. This is caused by the coarse grain of the metadata which we have illustrated above.

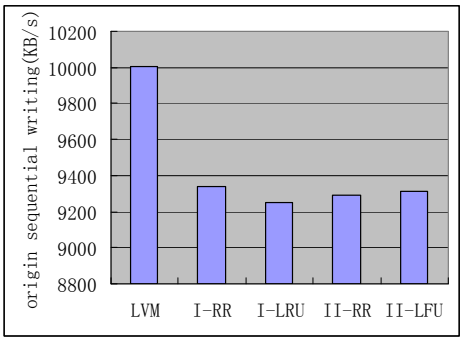

Figure 4. original volume sequential writing performance

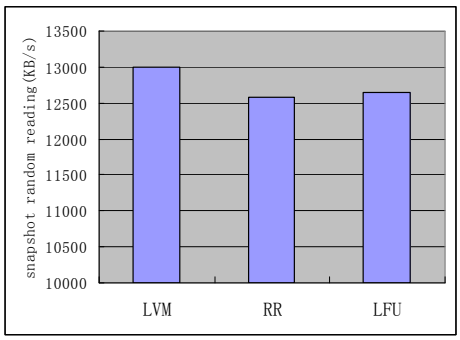

Figure 5. snapshot random reading performance

In order to observe how the dependent snapshot impacts the performance of original volume writing, we measured the origin writing under both 10 common snapshots and 10 dependent snapshots. The benchmark results are shown in Figure 6. The origin writing performance under 10 dependent snapshots is much higher than the traditional snapshots .The reason is that each write request only causes 1 COW operation rather than 10 COWs under 10 tradition snapshot. We also test the rewrite performance of original volume write under 10 snapshots. From Figure 10 we can see the rewrite performance is a little higher than the write condition under 10 dependent snapshots. The rewriting operation in this two conditions can not cause COW operations, so the performance can be close to the performance of origin writing without snapshot.

We also test the sequential writing performance of origin with both with 10 traditional snapshots and 10 dependent snapshots. The test results are shown in Figure 7. Similar to the experiments above, the performance is improved greatly under dependent snapshot. And it is close to the writing performance on origin under 1 snapshot. Of course, the performance is not improved as theoretic analysis because of the impact of the cache of hard disk and the file system.

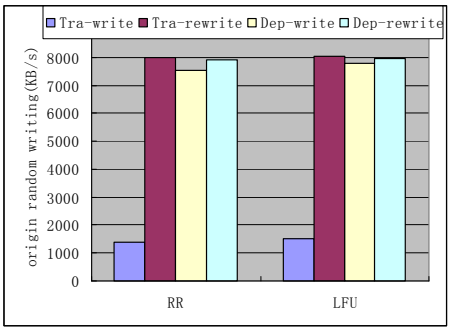

Figure 6. Original volume(with multi-snapshots) random writing performance

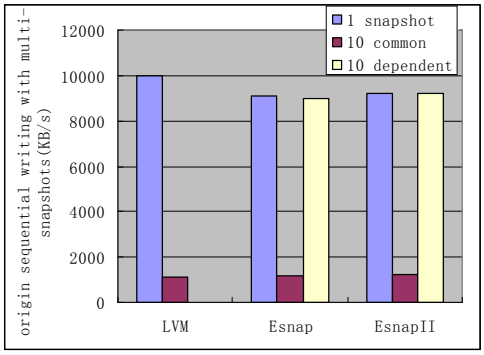

Figure 7. Original volume (with multi-snapshots) sequential writing performance

We measured the snapshot reading performance under 10 snapshots to observe the influence of the global shared metadata cache mechanism. We created 1 origin and 10 snapshots, after sequential writing on the origin. We perform this reading test on one of the snapshots. From the test results shown in Figure 8 we can see the reading performance on ESnapII is higher than ESnapII because of the shared global metadata cache. From the illustration in section 3 we can calculate that the snapshot to be read can use a metadata cache space as large as 800K in ESnapII, so every metadata can be preserved in memory without being replaced out. But in ESnapII, We can easily calculate the metadata cache threshold used by each snapshot is only 80K. So only 1/8 requests can be found in memory and cache replacement can take place for 7/8 times. We also test the reread performance of snapshot read in this condition to eliminate the influence of the overhead caused by metadata initialization and reading. We can see the rereading performance for ESnapII is almost equal to LVM1 because its cache-hit-rate can be 100% as we calculated above.

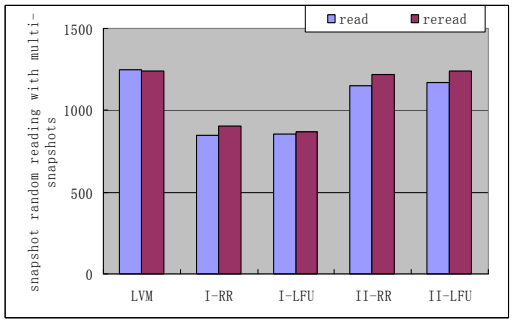

Figure 8. Snapshot random reading performance with multi-snapshots

5. Conclusions

In this paper, we have proposed the design and implementation of ESnapII, a writable dependent snapshot system with a shared global metadata cache. The shared metadata cache scheme can improve the hit rate of metadata cache greatly. And using this strategy, user can build more quantity and larger size volume than previous work because of its low memory consumption. We also present the implementation of snapshot write both for traditional snapshot and dependent snapshot. In addition, a flexible dependent snapshot deleting algorithm is presented in this paper. This improves the reliability and flexibility efficiently. Experimental results show that the read performance of the snapshot under multi-snapshots is improved compared with ESnap. The new metadata cache organization improves the scalability and availability greatly and pays a little in read/write performance of LVM.

6. Acknowledgements

This paper is sponsored by NSF of China (90612001), 863 program of China(2008AA01Z401), Education Ministry Doctoral Research Foundation of China(20070055054), Science and Technology Development Plan of Tianjin, (08JCYBJC13000).

7. References

[1] EMC Time Finder[M]. EMC corporation . http://www.emc.com/products/product_pdfs/ds/timefinder_1700-4.pdf,2000

[2] RAMAC Virtual Array http://www.redbooks.ibm.com/redbooks/pdfs/sg244951.pdf

[3] Hitachi:ShadowImage. : http://www.hds.com/pdf/shadowimageR6.pdf, 2001-06

[4] StorageTek(tm)Snapshot Software. http://www.storagetek.com

[5] HP Storageworks Business Copy EVA, http://h18006.www1.hp.com/

[6] Hasentein M.LVM Whitepaper.SuSE Inc. http://www.sistina.com,2001

[7] AJ Lewis.LVM HOWTO. Sistina Software, Inc, 2002-2003, Red Hat, Inc,2004-2005

[8] Xu Guangping, Wang Gang and Liu Jing," "Design of repetitious points incremental snapshots based on same snapshot volume" , Computer engineering and applications, vol,no3, pp 413,113-115, January 2005.

[9] Li Zhong, Wang Gang and Liu Jing, "A Technology of Implementing Sequential Points Snapshot in the Storage Subsystem," Computer engineering and applications, vol. 40, no. 9, pp. 18-20, 32, March 2004.

[10] Bhavana Shah, "Disk Performance of Copy-On-Write Snapshot Logical Volumes" master degree THESIS, The University Of British Columbia, 2006.

[11] Liu Zhenjun, Xu Lu ,Feng Shuo and Yin Yang, "The Design and Implementation of an Iterative Snapshot System," Jisuanji Gongcheng Yu Yingyong, vol. 42, no.14, pp. 11-15, May, 2006.

[12] Yong Feng, Yan-yuan Zhang, Rui-yong Jia: SnapChain: A Shared Snapshot Method for Data Version Management. ISCA PDCS 2004: 264-269

[13] Zhao Zhen, Xie Chang-Sheng, Li Huai-Yang, Dong Xiao-Ming, DCD-based block device snapshot system; Mini-Micro Systems vol.26, no.12 : 2168-71,12,2005

[14] Sitaraman, S.; Krishnamurthy, S.; Venkatesan, S. Srimani, P.K. "Byteprints: a tool to gather digital evidence"; Proceedings. ITCC 2005 International Conference on Information Technology: Coding and Computing : (Vol. 1) 715-20 Vol. 1, 2005

[15] Weijun Xiao, Yinan Liu, Qing (Ken) Yang, Jin Ren and Changsheng Xie "Implementation and Performance Evaluation of Two Snapshot Methods on iSCSI Target Storages," In NASA/IEEE MSST2006.

[16] Guangjun Xie, Genxi Fu, Yusen Li, Gang Wang, Xiaoguang Liu, Jing Liu , "ESnap - A Cached Dependent Snapshot System" , 2007 IEEE International Conference on Integration Technology: pp 783-788 March,2007

[17] Sysbench, "SysBench: a system performance benchmark" http://sysbench.sourceforge.net/