International Journal of Database Theory and Application
Vol.9, No.10 (2016), pp.337-348
http://dx.doi.org/10.14257/ijdta.2016.9.10.29

REMB: Recoverable External Memory Bitmap of Software RAID

Wang Wei [1] and Yu Li-hua[2]

[1]Hangzhou Medical College, Computer Lab, Hangzhou 310053, China
[2]Hangzhou R&D Center, Netease Corporation, Hangzhou 310052, China
[1]weiw81@163.com, [2]peterylh@163.com

Abstract

Distributed block storage system is one of the fundamental components of cloud computing, and many important services, including cloud database, cloud queue, are built upon it. It is common practice to build block storage system based on reliable and efficient Linux open source software, i.e. software RAID, to meet the I/O requirements of cloud database. Bitmap is a critical data structure of software RAID, and hence is important to reliability and performance of this kind of storage system. We describe several existing software RAID bitmap management solution, and propose REMB (Recoverable External Memory Bitmap), which is reliable and efficient. Experimental results show that REMB improve cloud database performance by 30%~60%.

Keywords: *Software RAID, Bitmap, Cloud, Distributed System*

1. Introduction

The vigorous development of cloud computing has accelerated the changes of IT industry. Since 2011, the major Chinese internet companies such as Alibaba, Tencent and Netease, plus dozens of startups are in a mad rush to launch their cloud computing services. The booming cloud products promote industrial applications. The data form chinairn [1] shows that 76% companies have a plan to migrate local applications to cloud in future.

The virtualization of computing, block storage and network is the foundation of cloud computing. Block storage system provides block devices for virtual machine, which is compatible with local disks. Compared with Amazon EBS (Elastic Block Store), which is the most famous block storage device in industry, there are some corresponding open source software products such as Sheepdog [2] and Ceph [3]. Block storage is the foundation of Cloud Database System, for example, Amazon RDS (Relational Database Service) is built on EBS. The database transaction throughput depends on the speed of write-ahead log, and further the speed of write-ahead log depends on the response time of block devices. So, we can conclude that the response time of block storage has direct effect on the performance of the cloud database.

RAID with cache and SSD(Solid State Drive) have been used to reduce I/O response time in single computer environment. But in cloud, it is still a big challenge to build a distributed block storage system to satisfy the database's I/O need. Firstly, I/O always needs to transfer data across the network because the block device replicas locate in more than one remote server. Secondly, due to the large scale of the cloud, failure happens almost every day, and then fault recovery will also affect the I/O response time. The open source software, including Sheepdog and Ceph, is more or less inappropriate for cloud database. They run in user mode and thus the overhead is too large. Also, when a fault occurs, the system's response time usually became even worse. Sheepdog repair data during reading, hence user I/O is likely to be blocked by repairing workload Ceph allows data repair in parallel with user I/O, but when a user access the recovering data, the I/O write will be blocked until data recovery finished. Although the storage objects accessed by user have high recovery priority, Ceph's I/O response time will still be severely affected.

Open source software, such as iSCSI, MD(Software RAID), DM(DeviceMapper), have implemented core distributed storage functionality, such as remote block device, replication, data slice and combination. They are mature and stable, and run in kernel mode with high I/O efficiency. There are lots of block storage systems [4,9-12] based on those open source software. The distributed block storage of NTES(www.163.com) is also implemented based on those open source software, it has been running stably and reliably since putting into use. In this paper, we propose REMB(Recoverable External Memory Bitmap) to improve MD(Software RAID) performance, and implement it in distributed block storage of NTES. Experimental results show that the new schema can improve cloud database's performance by 30%~60%.

This paper is structured as follows: Section 2 gives an overview of open source storage software. Section 3 presents three distributed block storage mechanisms based on RAID: internal bitmap, reliable external bitmap, and Recoverable External Memory Bitmap, and discusses their advantages and disadvantages. Section 4 presents experimental setup and numerical analysis results. Section 5 introduces related works. Section 6 summarizes the results and discusses future directions.

2. Open Source Software

In this section, we introduce three open source software used in this paper: MD, open-iscsi/iet, and DM.

2.1. MD (MultipleDisk)

Multiple Disk(MD) is a software RAID in Linux, which consists of a kernel driver and a user mode administrative tool. MD support many types of RAID such as RAID0/1/5/6, and in this paper we will focus on RAID-1, which implements disk mirror.

RAID-1 makes up of several disks which are mirror images of each other. When a power off, a network error or other failure occurs, a write request may success in some disks, while fail in other disks. Therefore, these disks may have different data, and the RAID device is inconsistent. When inconsistency detected, MD starts data recovery to make sure all the disks in same state. Data recovery runs in background, hence won't block user IO requests. The default data recovery strategy is full copy, that is, copying the data of selected disk to all other disks. Obviously, this method is costly and time consuming. To solve the problem, MD designs in_sync flag and WriteIntentionLog.

There is an in_sync flag in MD's super block, in_sync=1 means that the disks are in a consistent state, while in_sync=0 means that the disks may be inconsistent. When RAID device restarted, it checks in_sync flag, and skips data recovery when in_sync = 1. Therefore, in_sync flag can reduce the chance of data recovery. In_sync is initialize to zero at the creation of RAID device. Before MD serves a write request, it must ensure in_sync=0, because any write requests may cause inconsistency. If there is no write operation in a period of time, MD will set in_sync flag to 1, because the device is consistent now. in_sync flag of a heavy write RAID deivce usually equals 0, so it is only helpful to devices with low write workload.

PaulClaments *et al.* [7] implement write intention bitmap in MD to accelerate data recovery. There are two types of write intention bitmaps, *i.e.* internal bitmap and external bitmap. Internal bitmap is stored redundantly on each member disk of RAID, while external bitmap is a file in external file systems. The RAID is divided into equal-size data blocks, each bit of the bitmap corresponds to a data block, and 1 means dirty, 0 means not dirty. The write intention bitmap helps to reduce the amount of data to be recovered, as we just need to deal with the blocks tagged 1. However, the disadvantage of the write intention bitmap is also obvious: Each write request has to set and reset the corresponding bits, so the write performance drops down to 1/3. Several optimization methods are proposed to improve write performance: 1) Batch submissions of bitmap write requests. When concurrent write requests set several bits dirty, MD writes these dirty bits to disk in a batch to reduce bitmap writes. 2) Asynchronous reset. When a dirty bit is reset, MD doesn't write bitmap to disk immediately. It waits until other write operations modify the bitmap again, and piggy-back reset operations to disk. This optimization may cause reset bits not persistent in failure, but it is safe because the only result is that some unnecessary data blocks are recovered during RAID recovery. Although the above optimization is obviously effective, however, the experimental results show that MD bitmap still doubles the I/O access. Thus, the behavior of the bitmap plays a key role in software RAID performance.

2.2. iSCSI

iSCSI is a standard protocol published by IETF(Internet Engineering Task Force) in February 2003. It can literally be interpreted as SCSI over TCP/IP, or Ethernet SCSI. The SCSI commands are encapsulated into TCP/IP package, and transported by IP network.

The iSCSI consists of iSCSIinitiator and iSCSITarget. When application sends an I/O request, the operating system generates a corresponding SCSI command, and sends the command to iSCSIinitiator driver. The iSCSIinitiator encapsulates SCSI commands and sends the package to remote equipment. The remote iSCSItarget receives the message and fetches SCSI command according to iSCSI protocol. Then the command is executed by the real SCSI storage device. The I/O response process is just reverse.

iet(iSCSI Enterprise Target) is the most famous open source implementation of iSCSI target, which runs in kernel mode. Open-iscsi is the only open source iscsi initiator under Linux.

2.3. DM

Device Mapper(DM) is a block device mapping mechanism used in Linux for mapping physical block devices onto higher-level virtual block devices. It runs in kernel mode and implements IO requests filtering, routing, replication and load balance by modularized targetdrive plug-in. It also provides dmsetup command as administrator tool for kernel driver.

The Linux kernel provides a lot of target driver plug-ins, such as linear, mirror, snapshot. The linear target maps a linear range of the device onto a linear range of another device. In this paper we use linear target concatenate chunks as a single volume.

3. Distributed Block Storage System

3.1. System Architecture

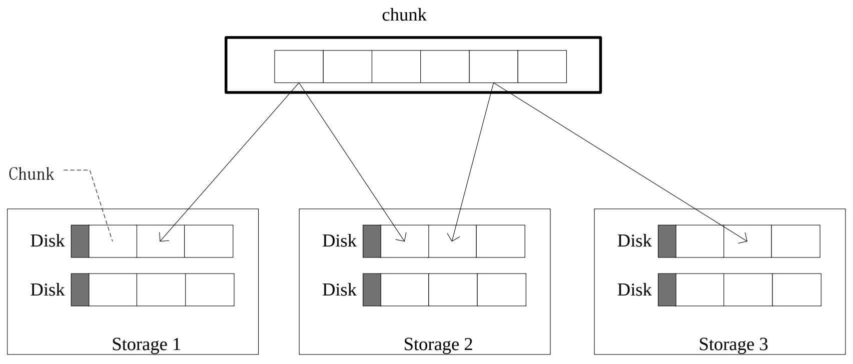


Figure 1. The Distributed Block Storage System Architecture

Netease, Inc. is a leading China-based internet technology company, the company starts to build its own private cloud system since 2012. Most of its internet application has deployed on cloud now. Netease Block System (NBS) provides block devices for the private cloud, its architecture is shown in Figure 1. There are two important concepts: Volume and Chunk. Volume is a block device provided by the system in host computer, and is mapped to user's KVM virtual machine by QEMU. From the user's perspective, there is no difference between a volume and a local disk, users can create file systems on a volume and read/write the files.

Chunk, normally 10G in size, is the unit of storage allocation and load balance. Allocation bitmaps are used to manage disk space. Each disk on host machine is divided into many chunks and has a corresponding allocation bitmap, 1 means the chunk is allocated and 0 means free. A volume is concatenated by multiple RAID-1 devices based on DM linear. Each RAID-1 devices consist of two chunks which is mirror to each other. A RAID-1 device is the same size as a chunk, which is also 10G. To ensure reliability, chunks from single RAID-1 device are located on different servers.

The system is made of storage servers, host machines and metadata server. The storage server manages dozens of physical disks and is responsible for chunk allocation and release. When the volume is mounted, the storage server exports chunks as LUN(Logical Unit Number) of iet. The host machine then mounts these chunks remotely to compose RAID device, and multiple groups of RAID-1 devices are concatenated to form a volume for users. Storage server and host machine are just two kinds of process, they usually collocate in a same physical server which provides both storage space and remote data access at the same time. The metadata server manages all kinds of metadata, including the mapping relationships between chunks and physical disks and mapping between volumes and chunks, *et al*. The metadata server is also responsible for global scheduling, failure detection and recovery, and providing APIs for up-level applications.

NBS has been online for several years in the Netease private cloud, and thousands of cloud database instances have been deployed on top of NBS volumes. NBS is very stable with reasonable performance, we have migrated most of our database to cloud, but there are still some write heavy database can't run on top of NBS. In Section 2.1 we have concluded that write intention bitmap has a great influence on software RAID performance. Now we will analyze bitmap management solutions in NBS, and propose our new bitmap management solution to improve NBS write performance.

3.2. Internal Bitmap

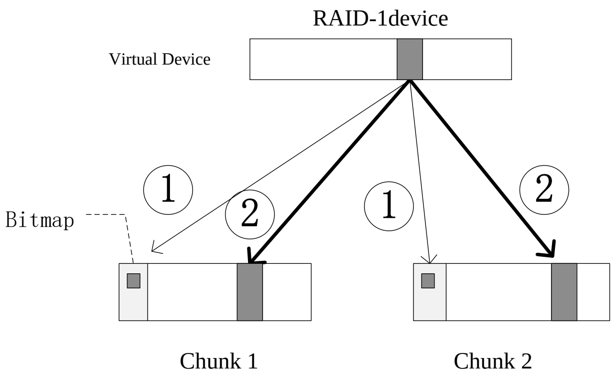


Figure 2. Internal Bitmap

The default MD bitmap mechanism is internal bitmap. As Figure 2 shows, MD stores a replica of the internal bitmap in the header of each Chunk. All of the bitmap is updated synchronously to maintain consistency. MD's write flow is shown in Figure 2: (1) if the bit of the bitmap corresponding to write request is not dirty, it is set to 1. Then MD sends two write requests simultaneously and writes the bitmap to the header of each Chunk; (2) After the bitmap modification, MD writes data in parallel to each Chunk; (3) MD resets the bit of each replica bitmap to 0 asynchronously.

The advantage of this mechanism is that the bitmap solution is mature and reliable, and no code changes are required. The disadvantage is that bitmap update is expensive because two cross-network disk I/O accesses are needed. Due to its poor performance, internal bitmap is not adopted by NBS.

3.3. Reliable External Bitmap

Because internal bitmap's poor performance, NBS employs a reliable external bitmap method. It uses MD external bitmap and stores external bitmap on reliable RAM disk. Each server creates several 64M sized RAM disks. Some of these RAM disks are used locally, and others are used by remote hosts through iSCSI protocol. As Figure 3 shows, a local RAM disk on the host machine and another RAM disk on the remote server make up a memory RAID-1 device which doesn't have write intention bitmap. A file system is created on memory RAID-1 device to store volumes' external bitmap files.

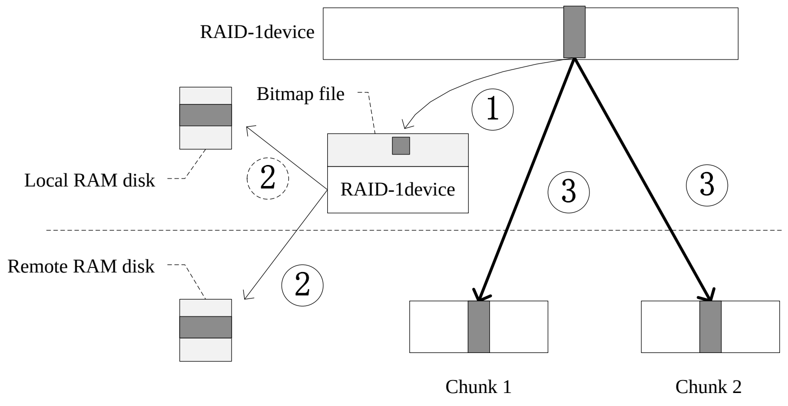


Figure 3. Reliable External Bitmap

The write flow is as follows: (1) If the corresponding bit of the write request is not dirty, it is set to 1, then MD flushes bitmap file.(2) The file system submits I/O to memory RAID device, and then MD updates local RAM disk and remote RAM disk concurrently.(3)MD writes data to the two Chunks after the bitmap modification.(4)MD set the bitmap to 0 asynchronously.

The most distinguishing feature of this approach is local RAM disk and remote RAM disk form a RAID-1 device to store the volumes' external bitmap. This approach outperforms internal bitmap, achieves 50% cost reduction of bitmap modification from 2 cross-network I/O accesses to 1 cross-network memory access. In compare with using two remote memory devices, it also has advantages: it reduces network communication cost, and has better reliability, because half remote device means half failure rate.

The disadvantage of this approach is that bitmap may be lost if the host machine and the remote machine where remote RAM disk locates go down in the meanwhile. Luckily, losing bitmap will not result in the volume becoming inaccessible or the data being lost. It may only lead to full copy data recovery. But full copy data recovery will happen only when the following three conditions apply simultaneously: the bitmap has lost; the MD device composed by the two Chunks is not in a consistent state; at least one Chunk is inaccessible. Thus full copy data recovery is a rare event.

3.4. REMB: Recoverable External Memory Bitmap

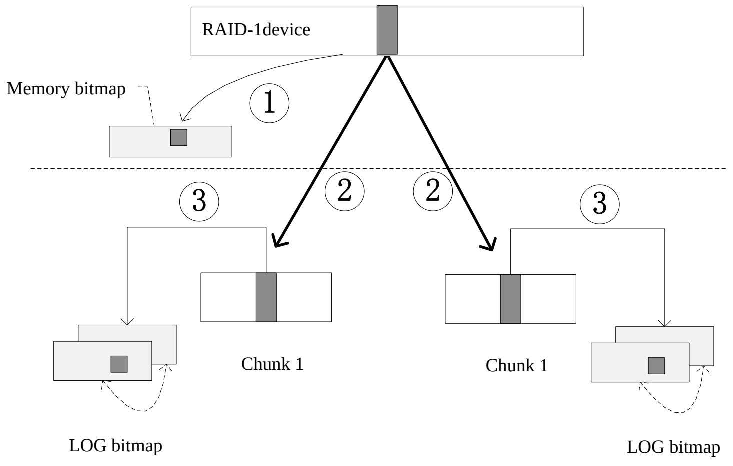


Figure 4. REMB: Recoverable External Memory Bitmap

We design REMB(Recoverable External Memory Bitmap), a new bitmap management mechanism for NBS which is more efficient without sacrificing durability. As Figure 4 shows, bitmap is stored on local RAM disks. Because memory access is fast, bitmap will no longer affect the write performance and minimize I/O response time in theory. Obviously, this approach obtains high performance at the expense of sacrificing bitmap reliability, so we have to figure out ways to ensure that the bitmap is recoverable. A new data structure LOG bitmap is allocated for each Chunk, and each bit of the LOG bitmap corresponds to a data block with 64MB size. Just like write intention bitmap, 1 means dirty, 0 means not dirty. A LOG bitmap keeps a record of write operations on a Chunk. the LOG bitmaps on the Chunk replicas together record all write operations on software RAID. Therefore, LOG bitmaps can be used to reconstruct the write intention bitmap and perform data increment replication.

Like write intention bitmap, the corresponding bit in LOG bitmap must be set to 1 before writing data. The difference is that, LOG bit must wait till the write request completed on all Chunk replicas, and then it is set to 0. Take this scenario: the host sends a write request to a certain Chunk, and then the host machine goes down too suddenly to send a write request to another mirror Chunk. In this case, the block should be dirtied in write intention bitmap, because the Chunk replicas are not consistent. If Log bitmap was set to 0 just after write operation, then all the LOG bitmap should be not dirty, and we can’t reconstruct write intention bitmap using LOG bitmaps.

Chunk replicas in software RAID don’t communicate with each other, and further, they don’t even know each other's existence. Storage servers can’t determine when a write request is finished at all replicas, and when to reset the corresponding bits in LOG bitmap. The solution to this problem is to delay the reset operation until the storage server makes sure that all the Chunk copies are written completely. LOG bitmap is divided into two parts: the current LOG bitmap which records data blocks modified most recently, and the history LOG bitmap which records blocks modified some time ago. Every T seconds, the current LOG bitmap overwrites the history LOG, and then all bits of current LOG bitmap reset to zero overall. The current LOG bitmap bitwise or the history LOG bitmap results LOG bitmap. Write intention bitmap can be reconstructed by bitwise or of LOG bitmaps at all replicas. We can prove that as the interval time T is much longer than iSCSI timeout, the recovered write intention bitmap must be right. That means all inconsistent data blocks have been marked in write intention bitmap.

Let T be the LOG bitmap interval time, and t be iSCSI timeout, $T >> t$. C_1 and C_2 indicate two Chunk repicas. Chunk C_i has the current LOG bitmap C_{i1} and the history

LOG bitmap C_{i2}. Let CT_{i1} be the time to reset C_{i1}. Algorithm 1 describes the bitmap update algorithm.

Algorithm1 update_bitmap

```
Require:
    I/O offset;
    I/O size;
Ensure:
    blocksize←64M
    now←gettimeofday()
    for index = offset/blocksize→ (offset + size)/blocksize
do
        bitmap_setbit($C_{i1}$, index)
        if $CT_{i1} + T < now$ then
            $C_{i2} \leftarrow C_{i1}$
            $C_{i1} \leftarrow 0$
            $CT_{i1} \leftarrow now$
        end if
    end for
```

Theorem 1: $C_{i1} \mid C_{i2}$ records all write operations for Chunk C_i in the period of T.

According to the bitmap update algorithm, C_{i1} records all write operation in the period of [CT_{i1}, *now*], and C_{i2} in the period of [CT_{i1}-T, CT_{i1}]. So $C_{i1} \mid C_{i2}$ keeps a record of write operations in [CT_{i1}-T, *now*). And because of $CT_{i1}<now$, $C_{i1} \mid C_{i2}$ contains all write operation in [*now*-T; *now*). In other words, $C_{i1} \mid C_{i2}$ records all write operation for Chunk C_i in the period of T.

Theorem 2: $C_{11} \mid C_{12} \mid C_{21} \mid C_{22}$ keeps a record of all possible inconsistent data blocks.

Only the data blocks being modified may be inconsistent. According to the bitmap update algorithm, the corresponding bit of the LOG bitmap must be set to 1 before the data is modified. Thus all modification operation must be recorded on some bitmap replica before modification.

In addition, we can prove that the bit will not be set to 0 if the data block is possible inconsistent. Because iSCSI timeout is t and $t << T$, a write request must be terminated in $\underline{T}$ seconds whether it fails or succeeds. We reset history LOG bitmap every T seconds, so a dirty bit of current LOG bitmap will remain dirty for at least T seconds. Therefore, when we reset history LOG bitmap, all corresponding write requests are finished, that is the bits will not be set to 0 if the data blocks are possible inconsistent.

As a conclusion, $C_{11} \mid C_{12} \mid C_{21} \mid C_{22}$ keeps a record of all possible inconsistent data blocks.

According to the theorem 2, we can also conclude that there will not be data inconsistency when recovering write intention bitmap from LOG bitmap. Now let's look at failure handling in distributed block storage devices.

Scene 1: temporary failure occurs on a storage server. When a storage server failure happens due to network disconnection or server restart, the soft RAID will degrade and I/O will not be affected. By the time the storage server recovered, the software RAID will perform increment data recovery based on write intention bitmap, that is, copying dirty data blocks from a valid replica to an invalid one. If the LOG bitmaps on failed storage server is not lost, data recovery will bring LOG bitmap up-to-date, because recovery write requests will update LOG bitmap just like user write requests. If the LOG bitmaps have been lost, new LOG bitmaps must first be created from a valid replica before data recovery.

Scene 2: the host machine fails. When failure occurs on a host machine, the VMs running on it would have been terminated, and the software RAID write intention bitmap would have lost. The reasonable approach to deal with this situation is mounting the volume to another host machine and restarting the VMs. The bitmap recover method is to copy LOG bitmap from the storage machine, and set the write intention bitmap to $C_{11} \mid C_{12} \mid C_{21} \mid C_{22}$. After recovering bitmap, incremental replication will be performed for the software RAID.

Scene 3: the host machine and the storage server are in trouble at the same time. In this case, the write intention bitmap and LOG bitmap are both missing. We have to create a new LOG bitmap and perform full copy data recovery for the software RAID.

4. Experiments

4.1. Experiment Environment

As Figure 5 shows, the experiment environment includes two servers, one as storage server, and another as host machine. We create 12 double replica volumes each with a size of 600G to simulate the cloud environment. Each volume is divided into 120 Chunks (size of 10G), and is deployed to 24 physical disks on storage servers randomly. The two Chunk replicas will be placed on two different physical disks. We start 4 virtual machines each with 3 volumes mounted. All the VMs are used to run test programs.

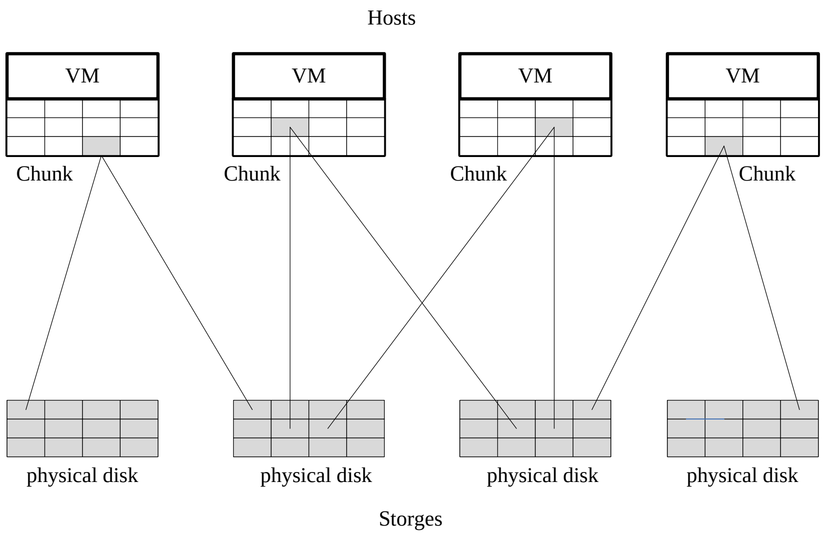


Figure 5. Experiment Environment to Simulate a Cloud

Table 1. Server Configuration

Server	Dell R720XD
CPU	E5-2650v2×2
Memory	16GB DDR3-1600Mhz×16， 256GB
SAS	2.5 inch, 900G×24 (10k RPM)
RAID	H710P(1G)
Network	2 × 10Gbps

4.2. I/O Response Time

I/O write response time is the most important factor for database transaction throughput. As internal bitmap has the worst response time, it is rarely used in real-world application. Therefore our experiments mainly focus on REMB and reliable external bitmap.

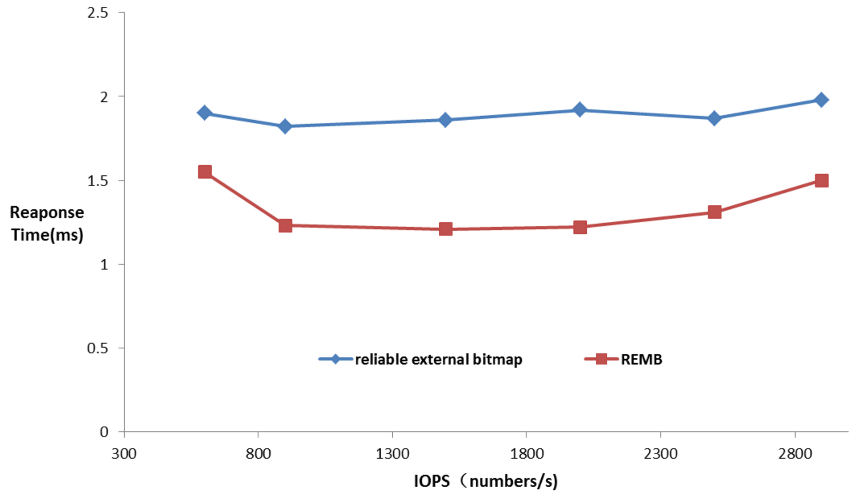
The experiments follow two principles: 1. simulating a real-world load environment. We start 3 fios on each VM, a total of 4 VMs and 12 fios involved to test 12 volumes. 2. testing the relationship between response time and system workload.

The results about I/O response time are shown in Figure 6(a). REMB has a better performance than reliable external bitmap in each I/O pressure scenarios, and reduces average I/O response time by 0.5~0.8ms.

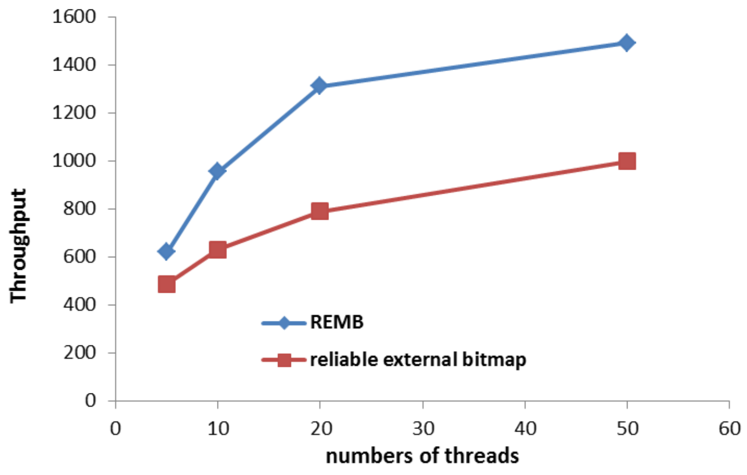
4.3. Database Performance

The mostly common cloud databases are based on MySQL in industry. In our experiments, the MySQL database is running on distributed block device, and the database performance of REMB and external bitmap is studied. We use MySQL benchmark sysbench [6] and OLTP as test program.

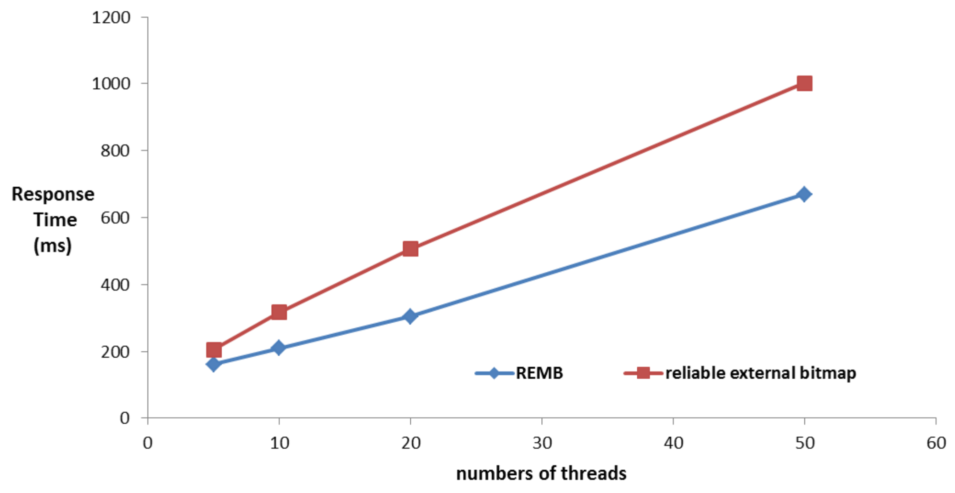
Figure 6(b) shows the database throughput of REMB and reliable external bitmap with a different number of threads. It can be seen that the database throughput of REMB increases by 30%~60% than reliable external bitmap.



(a) I/O Response Time



(b) Cloud Databases' Throughput



(c) Cloud Databases' Response Time

Figure 6. Experiment Results

Figure 6(c) shows the response time with a different number of threads. We can also conclude that the database response time of REMB reduces 20%~40% than reliable external bitmap.

5. Related Works

Distributed block storage systems require high reliability, high availability and high performance. Therefore, these systems are difficult to implement and there are few published research papers about the system implementation. The main open source distributed block storages we can find in cloud industry are Ceph [3], Sheepdog [2] and HLFS [8].

Ceph is an open source PB-scale storage system. It was born as a research project at the University of California, and later acquired by Redhat and used for Openstack as its standard block storage. Ceph uses CRUSH algorithm to map storage objects to PG(placement group), and PG to storage servers. It is an excellent scalable system,

Sheepdog is a distributed storage system for QEMU, iSCSI clients and RESTful services. It provides highly available block level storage volumes that can be attached to QEMU-based virtual machines. Sheepdog scales to several hundreds of nodes, and supports advanced volume management features such as snapshot, cloning, and thin provisioning. Sheepdog is a decentralized and scalable system in a peer-to-peer architecture.

However, neither Ceph or Sheepdog is suitable for high-performance cloud database. The basic shortcoming is that response time is uncontrolled and uncertain. Firstly, both Ceph and Sheepdog are running in user mode, and interaction between the kernel mode and the user mode is frequent and inefficient. Secondly, the IO path is long. Ceph's write requests are forward by primary replica to secondary replica. Similarly, a Sheepdog IO request must go through the gateway server. Thirdly, when a fault occurs, Sheepdog recovers data during reading, hence leading to long read response time. If a write request accesses data chunk to be recovered, it must wait until chunk recovery finished, hence the write response time is intolerable. Analogously, Ceph's write request must wait until replica recovery finished, and also brings unbearable response latency.

HLFS(Hadoop Log Structured File system) is another example of distributed block system. It is built on top of HDFS and implemented based on LFS. It inherits the features of Hadoop, such as high reliability, availability and scalability. However, Hadoop is an offline batch processing system with poor real-time performance, and the garbage collection mechanism of LFS causes further deterioration of response time. Therefore, HLFS is not fit for critical database applications.

Blizzard [12] is a high-performance block store proposed by Microsoft. By using a novel striping scheme, an out-of-order write committing mechanism, and CLOS network, Blizzard exposes high disk parallelism to workloads and provides high performance and crash consistency to applications. Blizzard is suitable for big data service but not OLTP database service.

There are also some distributed storage cases based on open source software in research area. Gao [4,11] implements a storage system based on IET and CLVM. Han [9] integrates iSCSI and RAID to implement a system. ORTHRUS [10] is based on IET and CLVM, and uses a load balance algorithm to improve storage throughput. Compared to these systems, this paper focus on response time and system stability, and proposes REMB to improve software RAID performance. Compared to DependableNetworkedRAID [13], which designs a memory bitmap for database incremental replica and implements the system based on software RAID, the bitmap in this paper is recoverable and more reliable.

6. Conclusion

Distributed block storage system is the core infrastructure of cloud computing, and the foundation of cloud database. The system must have rapid I/O response and proper fault handling to avoid an unbearable I/O response time when fault occurs. However, there are few open source distributed block systems that meet all of these requirements. The Software RAID is usually used to implement distributed block storage system because of its stability, high efficiency and mature fault handling. In these systems, the key factor of I/O delay is software RAID bitmap. This paper discusses software RAID bitmap methods in cloud environment, and then proposes REMB to improve software RAID performance. Experimental results show that the new approach is reliable and efficient, and achieves the least I/O delay in theory and improves cloud database performance by 30%~60%.

Acknowledgments

This paper is supported by CHB No. 2011ZX01039-001-002, and Hangzhou Medical College No. 2013XZA03.

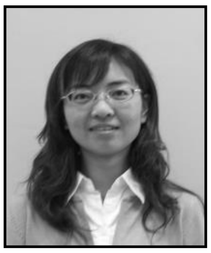
References

[1] Chinese Industry Research Network, http://www.chinairn.com/
[2] Sheepdog, https://github.com/sheepdog/
[3] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long and C. Maltzahn, "Ceph: A Scalable, High-Performance Distributed File System", Proceedings of OSDI, Seattle, **(2006)**, pp. 307-320.
[4] X. Gao, M. Lowe, Y. Ma and M. Pierce, "Supporting Cloud Computing with the Virtual Block Store System", Proceedings of Proceedings of e-Science 2009, Oxford, UK, **(2009)**, pp. 71-78.
[5] Fio, http://freecode.com/projects/fio/
[6] sysbench, https://launchpad.net/sysbench
[7] P. Clements and J. E. J. Bottomley, "High availability data replication", Proceedings of the 2003 Linux Symposium, Ottawa, Canada, **(2003)**, pp. 109-116.
[8] HLFS, http://code.google.com/p/cloudxy/
[9] H. D. Zhi, X. C. Sheng, F. X. Lin and Y. F. Ling, "Design and Implementation of an iSCSI-Based Network Attached Storage Server", Journal of Computer Research and Development, vol. 41, no. 1, **(2004)**, pp. 207-213.
[10] J. Wan, J. Zhang, L. Zhou, Y. Wang, C. Jiang, Y. J. Ren and J. Wang, "ORTHRUS: a lightweighted block-level cloud storage system", Cluster computing, vol. 16, no. 4, **(2013)**, pp. 625-638.
[11] X. Gao, Y. Ma, M. Pierce, M. Lowe and G. Fox, "Building a distributed block storage system for cloud infrastructure", Proceedings of Cloud Computing Technology and Science (CloudCom), Indianapolis, USA, **(2010)**, pp. 312-318.
[12] J. Mickens, E. B. Nightingale, J. Elson, K. Nareddy, D. Gehring, B. Fan, A. Kadav, V. Chidambaram and O. Khan, "Blizzard: fast, cloud-scale block storage for cloud-oblivious applications", Proceedings of

the 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14), Seattle, **(2014)**, pp. 257-273.

[13] T. B. Peter and A. Wiebalck, "Dependable networked RAID for the open source community", Proceeding of 11th Parallel and Distributed Systems, Fukuoka, Japan, **(2005)**, pp. 627-633.

Authors



Wei Wang, Computer Labs of Hangzhou Medical College, Research Interests are distributed system, cloud computing.

Lihua Yu, technical director of Netease Inc., Research Interests are distributed storage system, cloud computing.