The Research and Implementation of the High Performance CD - Mirroring Server

Dezhi HAN, Xiaogang ZHANG

Department of Computer Science, JiNan University,
Guangzhou, Guangdong, P.R.China, 510632

dezhihan88@sina.com.cn

Abstract

A high performance CD - mirroring server （for short HPCMS）has been designed and implemented in the paper. Firstly, the HPCMS can mirror the multi-CD with multi CD drives in parallel, improving the slowly - mirroring speed for a single CD drive; secondly, the HPCMS enhances the utilizing efficiency of the I/O resource and the CPU resource by a parallel mirroring and compressing algorithm, instead of mirroring and compressing to occupy the system I/O resource and CPU resource respectively which may greatly affect the system performance of the CD server; thirdly, the HPCMS greatly improves the file I/O speed by the direct data access (Zero Copy) between the RAID controller and the user-level memory. Fourthly, the HPCMS integrates the multi-RAID for a single storage pool by a multi-stage striping device driver, and it implements the storage virtualization; finally, the HPCMS implements automatically - allocating I/O bandwidth for different users and applications by enriching the metadata operating semantic of the CD server, and allocate a wider bandwidth for applications that has higher priority, changing the old average allocating bandwidth pattern. As is shown in the experiments introduced in this paper, the HPCMS has a superb high speed for the CD mirror, and has ultra-high-throughput for file I/O request.
Keyword: CD Mirroring, Zero Copy, Optimizing Bandwidth Allocation

I. Introduction

With the development of the Internet/intranet and digital libraries, the network-attached storage (for short NAS) has become more and more important than ever before. Now there have appeared many network storage devices, such as the network-attached storage RAID, network-attached storage CD server, network-attached storage security disk and network-attached storage tape library. With all these NAS devices, the pressure of network storage has been relieved to a great extent.

In applications, there are some disadvantages: (1) For example, the Network-Attached Storage CD server usually has one CD drive, mirroring one CD each time, which results in the slow mirroring speed and a mirror process separated from compression. Because the mirror and compression take

up the system I/O resource and CPU resource respectively, the system performance may be seriously impacted. When applying the traditional CD server to large-scale websites or TV stations or broadcasting stations or digital libraries, it cannot meet with the needs. Because in these cases the amount of the CDs may be very large, usually up to several thousands and even more than ten thousands. It takes a long time to finish mirroring so many CDs. (2) In the accessing speed aspect, as the tradition NAS, when some data are read from the tradition CD - Server, firstly, the data would be transferred between various I/O devices and the kernel-level buffer cache by the DMA, and the kernel-level buffer cache is used as a temporary buffer cache that is always mapped into the physical memory. And the data transfer between the kernel-level buffer cache and the user-level memory are executed by the CPU's memory-to-memory copy. When any data are written, they would go through a reverse process. In this case, the I/O performance is limited by the speed of memory-to-memory copy [2]. (3) In the resources integration and the system management aspect, as tradition NAS too, the tradition CD - Server can only integrate the disk resources in a single CD–Server device, and cannot span different CD-Server devices. So it is difficult to converge several different CD-Server devices to a large CD-Server, and the system has to manage those devices individually.

To speed up mirroring CDs, a concurrent mirroring CD server with multi-drives has been designed. To integrating the mirror and compression, a kind of parallel algorithms which can enable the CD mirror and compression to be carried out simultaneously has been given. Another is a Zero Copy mechanism to realize the direct data transfer between the RAID controller and the user-level memory in the HPCMS, it can greatly improve the I/O speed of HPCMS. With a MSS device driver, the HPCMS can be mounted to the multi-RAID simultaneously, and the multi-RAID can be virtualized a high-speed large RAID. Furthermore, according to characteristics of different users and applications, we reasonably arrange prior level and take optimizing bandwidth allocation strategy to assure discriminating storage service quality

The HPCMS is designed on the basis of a CD mirroring server designed by ourselves, which can mirror VCD, Audio CD, DVD video and data CDROM automatically [1][2]. And we have appended our product with a high-speed cache, an intelligent pre-fetch, Zero Copy and multi-stage stripping [3][4].

II. Operation System Choice

Because source codes for the LINUX are opened to all users, we may modify it according to our requirements. To reduce the system run overhead, on the one hand, we have simplified the file system, the network protocol, and the system services, etc; on the other hand, we have appended our product with a high-speed cache, an intelligent pre-fetch, a load balance, a distributing file system, etc, so the performance of the system is greatly improved and a storage-oriented and embedded system integrating many new techniques is achieved.

The LINUX uses the kernel-level buffer cache mechanism as the PC-Unix, which mediates the data transfer between the disk I/O system and the user's process. The kernel-level buffer cache is always used, because the LINUX has two process modes i.e., the kernel mode and the user mode. In the data transfer between various I/O devices and the kernel-level buffer cache by the DMA (direct memory access), the kernel-level buffer cache is used as a temporary buffer that is always mapped into the physical memory. And the data transfer between the kernel-level buffer cache and the user-level memory is executed by the CPU's memory-to-memory copy. The reason for this transfer sequence is

that the user's memory might be in the 'exchange' state. Although the data transfer between the kernel-level buffer cache and the I/O devices is by the DMA, the DMA is transferred between the physical memory areas. And the data transfer between the kernel-level buffer cache and the user-level memory is executed as a software process, which is a data transfer between virtual memories. The memory-to-memory copy would limit the I/O performance of the system. To solve this problem, we have introduced the Zero Copy mechanism[2]

Ⅲ. CD mirroring and compression parallel algorithms

A. The principle of design

A traditional CD mirror server compresses CD content after mirroring the CD, of which the workflow is shown as Fig. 1(a). It can be seen that the process of mirror and that of the compression are independent from each other and they are serially executed. Firstly, the mirroring process is carried out, in which CD data are read into the memory, and then written into the hard disk. And secondly, in the compressing process data are read into the memory from the hard disk, and compressed in it, and then written into the hard disk, with the old uncompressed data deleted out. During this process, both the system I/O and CPU resources will be occupied repeatedly, which will decrease the efficiency of the server. In the testing, we find that the mirroring process mainly uses the system I/O resource, with almost less than 1% CPU resource; while the compression process mainly use the CPU resource and little system I/O resource. If the two processes are parallel implemented, both the system I/O resource and the CPU resource will be fully utilized, and the system performance will be greatly improved. Consequently the serial operation has been taken placed by a parallel one, with no data stored in the hard disk. Also the process has been replaced by the thread, which needs less resource than that of process, and it is more convenient to communicate between threads; moreover, the synchronization and mutex are quite mature between them. The workflow is shown as Fig.1(b).

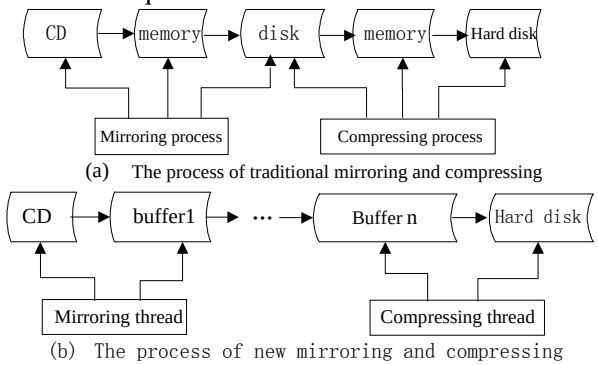

Figure 1 The process of mirroring and compressing

As is shown in the workflow, the mirroring thread reads the CD data into the memory, then the compressing thread compresses them thereof, and finally writes them into the hard disk of the server. The entire process is parallel executed in a multi-buffer hierarchy. Once the mirroring thread fills a buffer, it transfers the buffer data onto the compressing thread and compresses them, and simultaneously the mirroring thread applies another buffer to continue mirroring the CD data. In this way, this new process functions as the former mirroring process, without any data written on the mirroring thread and returned to the hard disk any more, which ignores the

process of writing the CD data from the memory into the hard disk. As a result, the CD mirroring time is shortened.

On the other hand, in order to make full use of a server's hardware resources, the main thread dynamically adjusts the number of the compressing thread according to the resources of each server so that a better performance can be achieved.

B. Design for program structure

The entire program of implementing the CD mirror and compression parallel algorithms includes three modules: the main thread module, the mirroring thread module and the compressing thread module. Each module can be executed in the memory of the sever with independent thread, and the three modules work parallel and coordinatively. The whole structure is shown as Figure 2.

As is shown in Figure 2, the data structures of the mirroring threads are arranged as a queue so that the main thread manages the mirroring threads effectively, and the queue head pointer is saved in the main thread data structure so that the main thread can access the data structures of mirroring threads and monitor the work of the mirroring threads. Similarly, the data structures of the compressing threads are arranged as a queue to facilitate the main thread to manage the compressing thread effectively.

To execute the above- mentioned parallel mirror and compression, the mirroring data are transported onto the compressing thread from the mirroring thread, and compressed by the compressing thread in the memory of the server. And the mirroring thread and compressing thread must be kept independent from each other in order to be implemented in parallel. So, a compression task queue has been designed to separate the mirroring thread from the compressing thread completely. And the mirroring thread only needs to continuously append the mirroring data buffer onto the compressing task queue tail, being not aware of the existing of the compressing thread at all. While the compressing thread just continuously fetch data buffer according to compressing task queue head information, the data in buffer will be compressed firstly then written to MPEG file in hard disk.

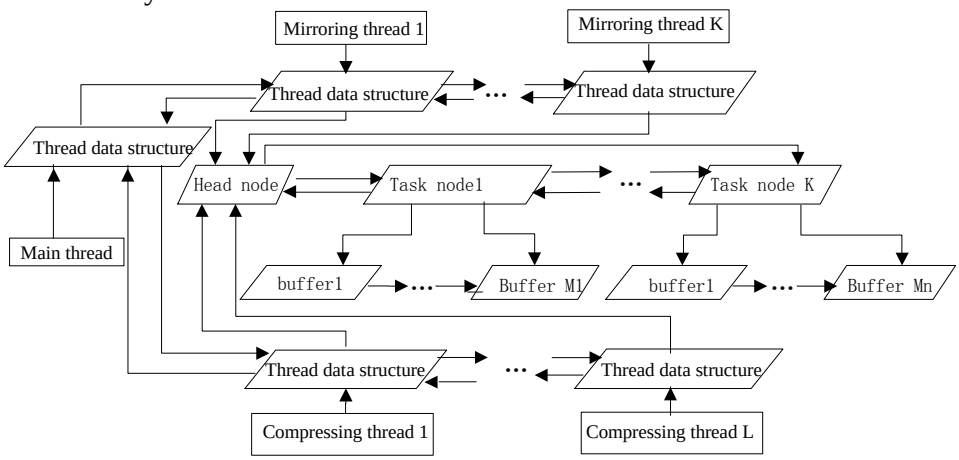

Figure 2 The program structure of the mirroring and compression parallel algorithms

C. The synchronization of the mirroring thread and the compressing thread

In the above-mentioned server, a mirroring thread can be corresponded to multiple compressing threads and vice versa. They don't correspond one-to-one respectively, synchronizing by compressing task queues. For a compressing task, it only has a mirroring thread and a compressing thread, and they need synchronize with each other. To implement

synchronization, a variable buffer_number is defined in the compression task queue nodes, which is used to record the data queue length of current node. Once the mirroring thread mirrors an entire buffer data, it appends this buffer onto the data buffer queue tail of current node, adding one to the buffer_number. If the buffer_number is less than 1, it indicates that the compressing thread is sleep because of waiting for data. Increasing the buf_number by 1 will awake the compressing thread. Before compressing thread take data buffer from current compressing task node, the buf_number must be decreased by 1, and if it makes buf_number is less than 1, the compressing thread will be sleeping because the data buffer is empty. Thus, the mirroring thread can realize synchronization with the compressing thread.

Ⅳ. I/O Optimization

A. *Zero Copy Mechanism*

The conventional and proposed processing stacks are shown in Figure 3, and the traditional read/write information transfer flows are shown in Fig.3(a). When data and commands are transported between the RAID controller memory and the user memory, the kernel-level buffer cache acts as a temporary buffer cache. Reading proceeds in two steps: the DMA transfer from the RAID controller to the kernel-level buffer cache, and the memory-to-memory copy from the kernel-level buffer cache to the user-level memory.

As for the proposed processing stack, see Fig. 3(b). To avoid using the buffer cache, a new data flow and a command flow are established in the LINUX and the device driver. In this way, the LFS can directly access to the device driver via the Zero Copy entry, and send block read/write commands including the initial sector number, the sector count and the virtual memory addresses (user memory addresses). We call the command request and the data transfer via this flow the Zero Copy mechanism.

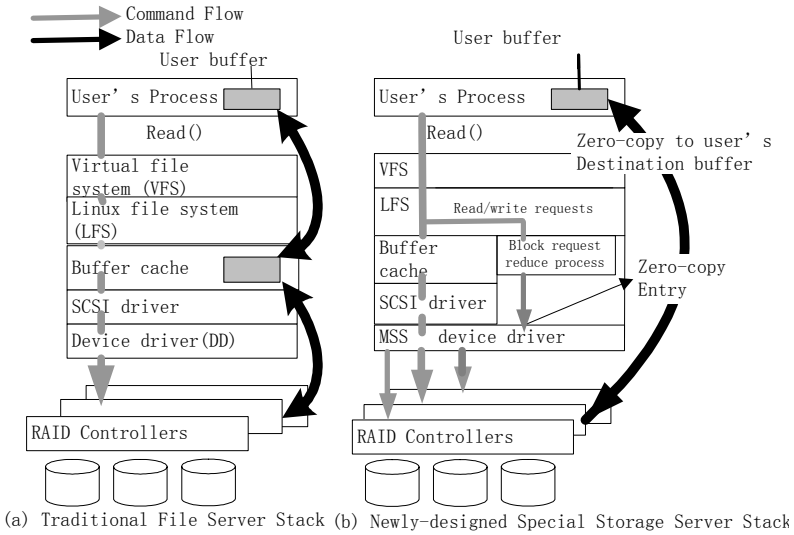

Figure 3. Traditional File Server Stack and Special Storage Server Stack

The file-distribution policy of HPCMS is based on inode numbers, the inode number is a unique identifier of Linux file systems and has one-to-one correspondence with the file. When the LFS receives a file read/write requirement, it maps the file to block according to its inode number, and a file request transforms into multi block requests. In the HPCMMS, the request

reduce process will find out the distributed blocks and combine all adjacent blocks together as a large block, and forms the Zero Copy Entry which includes the initial sector number, the sector count and the virtual memory addresses (user memory addresses), then transport the Zero Copy Entry to MSS device driver, finally the written/read data is transported directly between the user memory and the RAID I/O cache. For other file operation, such as creating a directory or accessing to the disk information, the conventional flow via the buffer cache is used.

B. Disk Fragment Clear-up Program

To enhance the HPCMS performance, reading or writing a file would run best when it is in contiguous disk blocks. So we develop a program to clear up disk fragment on schedule to assure that the blocks of a file can be mirrored at contiguous sectors of a disk.

Ⅴ. The Multi-stage Striping (MSS) Device Driver

In the proposed system, the LINUX device driver for the RAID controller PCI cards has two special functions. One is the storage virtualization function, offering an access to two or more RAID controllers, and constructs a large MSS RAID device with high-speed performance. The other is the Zero Copy mechanism.

In the Zero Copy mechanism, the LFS directly sends the read/write command to the MSS device driver. The command includes the virtual memory addresses (user memory addresses), in addition to the starting sector number and the number of sectors. The device driver should convert the address from virtual form to physical one, confirming that the page tables are contiguous or not and scattering contiguous pages and command request to the RAID controllers with chaining Zero Copy function to reduce the overhead of the scattering pages.

Ⅵ. The optimizing bandwidth allocation strategy

According to characteristics of different users and applications, we reasonably arrange prior level and take optimizing bandwidth allocation strategy to assure discriminating storage service quality [5]. The concrete implementation is like this: By enriching the metadata operating semantic of a CD server, the history recorders and operating attributions of each authorized user will be recorded in the metadata operating semantic set. The system assigns a prior level to each user's I/O request according to the metadata operating semantic set, and the high prior level I/O request is sent to the high prior level queue and the low one is sent to the low prior level queue. High prior level queue will be allocated with more bandwidth, while low prior level queue acquires less, but bandwidth allocation is still affected by the total I/O request number of the CD server. The principle is shown in Fig.4 (b).

The traditional bandwidth allocation is shown in Fig. (a), which only has a single prior level queue, and each I/O request is allocated with the same I/O bandwidth. The optimized bandwidth allocation is shown in Fig. 4(b). The data link layer (for short DLL) intercepts the I/O request and processes it according to the metadata operating semantic, then the high prior level request is sent to the high prior level queue and the low one is sent to the low prior level queue, and the high prior level queue will be allocated with more bandwidth, while the low prior level queue acquires less.

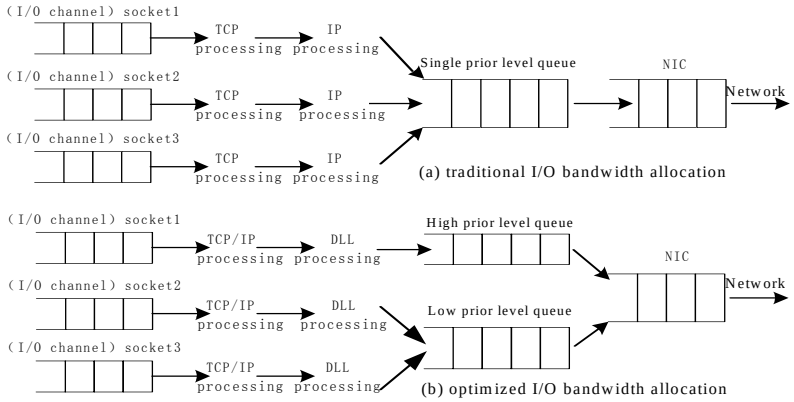

Figure 4. The optimized bandwidth allocation and traditional bandwidth allocation

Ⅶ. HPCMS hardware organization

The HPCMS hardware organization is shown in Table 1, in which the HPCMS is configured with 1GB memory and 2 RAID controllers, each of which can be plugged with two 80GB hard disks, one Pentium 4 2.66GHz CPU, Linux 9.0 OS and four Sony DVDROM. The elstorage-2000 is a RAID control card with two SCSI HDD interface developed by Wuhan East Lake Storage Corporation and our laboratory together.

Table 1. the HPCMS hardware configure

processor	Pentium4 2.66GHz
memory	1GB EDO-DRAM
PCI bus	Two PCI buses with 3 slots each
RAID controller	elstorage-2000
RAID level	RAID0、1 and 5 supported
#Controllers for RAID	2
Hard Drive	IBM 80G
#Drives	4 (2 drives at each RAID controller)
CDROM	4 Sony DVD ROM
OS	Linux 9.0

Ⅷ. Experiment Evaluation

We have evaluated the HPCMS performance by the following tests. The experiments are divided into two groups: the mirroring speed is tested by the first group experiment, and the I/O throughput and I/O average response time are tested by the second group. In the first experiment, when the HPCMS only uses a single CD - ROM driver, and the data mirror is separated with the data compression, we test the time for mirroring and compressing 12 pieces of Audio CD and 4 pieces of DVD respectively. Then we respectively test the time for mirroring and compressing 12 pieces of Audio CD and 4 pieces of DVD with the concurrent mirror of 4 CD drivers when the mirror and compression are parallel executed. In the second experiment, when the user accesses the HPCMS and ordinary CD-server via an IP switch, we test the I/O throughput rate and average responding time with IOmeter. The ordinary CD-server is made up of the HPCMS, but uninstall the optimization modules.

A. Experiment environment configuration

The next group experiment environment configuration is shown in Table 2. Host1 and Host2 can send a file I/O request to the CD mirroring server. The HPCMS is a high performance CD mirroring server. The elstorage-RAID is a RAID with two SCSI hard disks. The Host 3 is a ordinary CD-server.

B. Mirroring and compressing test

In the first group experiment, when we use a single CD drive, and the data mirror is separated with data compression, the time for mirroring and compressing the 12 pieces of Audio CDs is about 59”41’, and 72”12’ for 4 pieces of DVDs; when we use 4 CDs drives to concurrent mirror CD with parallel compressing, the time is about 18”32’ for 12 pieces of Audio CDs, 22”51’ for 4 pieces of DVDs. Thereby it can save 68.9% for 12 pieces of Audio CDs and 68.35% for 4 pieces of DVDs in time.

Table 2 The Configuration of Experimental Machine

	CPU	memory	OS	Hard disk	NIC/HBA）	RAID
Host 1	Pentium3 730	256MB	Linux 9.0	ST318437LW(SCSI)	AGE-1000SX (NIC)	
Host 2	Pentium3 730	256MB	Linux 9.0	ST318437LW(SCSI)	AGE-1000SX (NIC)	
HPCMS	Pentium4 2660	1GB	Linux 9.0	IBM 80G	AGE-1000SX (NIC)	elstorage-RAID
Host 3	Pentium4 2660	1GB	Linux 9.0	IBM 80G	AGE-1000SX (NIC)	elstorage-RAID

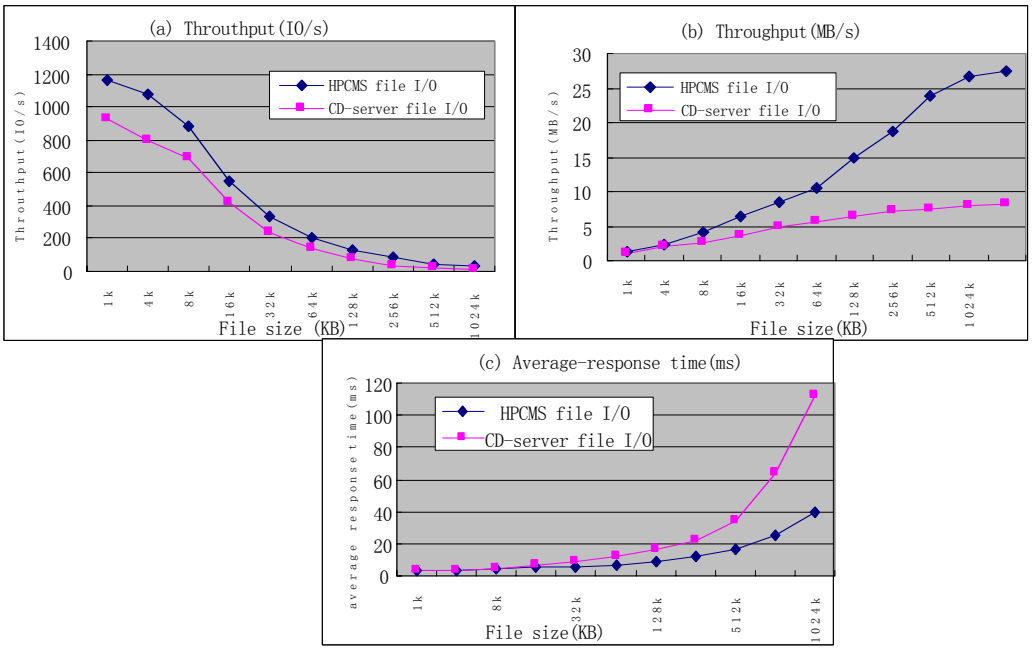

Figure 5. The Curve of Mean-response Time and Throughput

C. File I/O request test

In the experiment, we test the throughput and average response time for the file I/O request of the HPCMS and ordinary CD-server with the IOmeter, and the curve of the mean-response time and throughput is shown in Figure 5.

As shown in Figure 5, we have loaded the MSS device driver, PFTP and block request reduce module to CD-server, and we have appended our products with a high-speed cache, an intelligent pre-fetch, a distributing file system, etc. The file I/O speed of HPCMS is as high as 27MB/s, but the file I/O of the ordinary CD-server has only 8.1MB/s.

Ⅸ. Conclusion and prospect

There are two parameters which may greatly influence the performance in the parallel mirroring and compressing algorithms of the HPCMS, one is the number of the compression threads. Because a track in the CD can be compressed only by one compression thread, too much compression threads may lead to system resource competition and decrease the entire system performance. If there are too few compression threads, the utilizing rate of the system resource will decline and the whole system performance will be decreased, too. The other parameter is the size of the data buffer. If the data buffer is too small, it will frequently transfer data to the buffer between the mirroring thread and the compressing thread, so the system synchronization mutex cost will be increased. If the data buffer is too big, the parallel degree of the mirroring thread and the compressing thread will decrease, for example, when the data buffer size is very big, even mirroring thread has mirrored much data, the compressing thread is still free until the mirrored data buffer is full, so the system performance is influenced. To solve the problem, we use a dynamic measure to regulate the two parameters, and make them relate to the system hardware configuration. The two parameters are set to initial default value when the HPCMS is produced, then the main thread records the mirroring and the compressing statistic information every time. With these history records, the main thread will adjust the two parameters step by step until they reach a stable setting. When the hardware configuration changes, the main thread will also adjust the two parameters accordingly.

The optimizing bandwidth, Zero Copy and request reduction mechanism are a part of our optimizing work to the HPCMS. In the future, we will improve the HPCMS performance, including: (1) we will make up RAID, RAID cache, and CD server cache to a multi-level storage architecture. Based on the above-mentioned works, we plan to design and implement a kind of effective spatial pipeline algorithms to decrease the delay of the user I/O request responding and incease the user I/O bandwidth requirement; (2) The HPCMS cluster will be built by the storage virtualization. The HPCMS cluster can provide greater capacity and higher performance; (3) We will adopt the iSCSI technology to the HPCMS, which can make the HPCMS provide both of the file I/O and the block I/O services at same time, and combine the advantages of the NAS and the SAN together.

References

[1] D.Z.Han, “The design and implement of WEB management for a network-attached CD-R and CD mirroring server,” in *Computer Engineering*, 2001,vol. 27, pp. 55–58.

[2] D.Z. Han, J.G.Wang, “The design and implement of the automating mirror system a network-attached CD server”. In *Computer Engineering*, 2001, vol. 27, pp.36-39.

[3] D.Z. Han, C.S.Xie, “The design and implementation of An iSCSI-based Network Attached Storage server,” *Research and Development of Computer*, 2004, vol.41, pp. 207–213.

[4] M. Thadani, A.K.Yousef, “The Parallel I/O Architecture of the High-Performance Storage System (HPSS), ” *14th IEEE Symposium on Mass Storage System*s, 1995, pp.27-54.

[5] H.B. Mor, B. Schroeder, “Size-based scheduling to improve web performance”. In *ACM Transactions on Computer Systems*, Vol. 21, No. 2, May 2003, pp.207–233.

D.Z.Han.

In 1990, he graduated from Hefei Industry University, Hefei, P. R. C. with a BS degree in computer science and in 2001, he graduated from Huazhong University of Science and Technology(HUST), Wuhan, P. R. C. with a MS degree in computer architecture and a PhD degree in computer from the same university in 2003, and his research interest includes optical data storage, network - attached storage, and the IP-based storage area network. E-MAIL:dezhihan88@sina.com.

X.G.Zhang.

In 1998, he graduated from Inner Mongolia Normal University, Inner Mongolia, P. R. C. and a MS degree in computer architecture from the South China Normal university in 2003.His research interest includes the distributed computing and database system