Using DMA Aligned Buffer to Improve Software RAID Performance

Zhan Shi[1], Jiangling Zhang[1], and Xinrong Zhou[2]

[1] National Storage Lab, Computer Science Department
Huazhong University of Science and Technology, 430074 Wuhan, P.R.China
jlzhang@hust.edu.cn
[2] Turku Centre for Computer Science, Åbo Akadeni,
20520 Turku, Finland
xzhou@abo.fi
http://www.abo.fi/~xzhou

Abstract. While the storage market grows rapidly, software RAID, as a low-cost solution, becomes more and more important nowadays. However the performance of software RAID is greatly constrained by its implementation. Varies methods have been taken to improve its performance. By integrating a novel buffer mechanism – DMA aligned buffer (DAB) into software RAID kernel driver, we achieved a significant performance improvement, especially on small I/O requests.

1 Introduction

Most business computers running critical applications use SCSI disks with hardware RAID controllers. With the power of processors keeping on growing, more and more developers place greater loads on system CPUs. Nowadays, there are companies that provide software ATA RAID using cheaper IDE disks instead of the expensive SCSI disks. Lots of motherboard manufacturers are also including ATA RAID chips on board. This means that even a home user can afford a RAID solution today. Provided with benefits of any RAID system [1] such as increased write speeds while writing to a striped partition, software ATA RAID is now been widely accepted. [2].

However, there are some drawbacks to implement RAID in software. First, the issue of portability. Since a software implementation will have some OS-specific components, those components have to be re-written for each different OS. Second, the kernel-mode program. This is the main problem that troubles kernel-mode software developers. Unlike applications, kernel-mode program can execute privileged instructions and manipulate the contents of any virtual address, the whole system is exposed to these programs without any safeguards against programming errors. A simple mistake may even lead to a system crash. The third, performance. As we have already stated, software RAID solutions are typically implemented as kernel mode components. In fact it is embedded into the kernel itself in Linux. [3]. Many OS already have build-in buffers and caches to improve its I/O performance [4][5], but they may not cooperate with software RAID well.

M. Bubak et al. (Eds.): ICCS 2004, LNCS 3038, pp. 355–362, 2004.

The remainder of this paper is organized as follows. Section 2 introduces the related works. Section 3 presents how we can improve the performance using DMA-aligned buffer in detail. Section 4 describes the implementation of DAB in Windows 2000 based software RAID. Section 5 evaluates DAB under a variety of circumstances. Section 6 ends this paper with a conclusion .

2 Related Work

Caches and buffers are widely used to improve I/O performance in modern operating systems. Data prefetching is also a popular method to process data inside the cache and buffer. Lots of work have been done by many researchers. Sequential prefetching [9] simply look ahead to prefetch a variable number of sequential blocks past the current reference. Track-aligned prefetching utilizes disk-specific knowledge to match access patterns to the strengths of modern disks[10]. By allocating and accessing related data on disk track boundaries, a system can avoid most rotational latency and track crossing overheads. Avoiding these overheads can increase disk access efficiency by up to 50% for mid-sized requests (100–500KB). To be the most successful, prefetching should be based on knowledge of future I/O accesses, not inferences. Such knowledge is often available at high levels of the system. Programmers could give hints about their programs' accesses to the file system. Thus informed, the file system could transparently prefetch the needed data and optimize resource utilization. This is what Transparent Informed Prefetching (TIP) has done[11].

Currently, our problem is that I/O requests generated by most applications such as database, daily transaction, etc, are fairly small, always lesser than 64KB. The prefetching methods mentioned above do not work well in our situation. How to improve the small I/O performance while keeping larger I/O least affected in our software RAID implementation is the main topic of this paper.

3 DMA Aligned Buffer

When surveying the characteristic of access patterns of standard software RAIDs, we found that there is a significant change in transfer rate while we adjust the I/O packet size between its minimum value and maximum IDE DMA transfer length. If we take a close look at many Operating Systems, we will find that every data request is sent to the low-level drivers such as port/miniport driver. In modern IDE disks, every device I/O requires a DMA operation to complete. [12] Some applications are likely to generate a lot of small I/O requests, for example, database application, web transaction, etc. Operating Systems have to handle these requests by sending them to their driver stacks, from the highest file system drivers to the lowest HBA drivers [6]. File cache in most operating systems can mitigate the problem when processing most application-level I/O requests, but it cares little about low-level I/O efficiency. Low-level fragmental requests can hardly benefit from that.

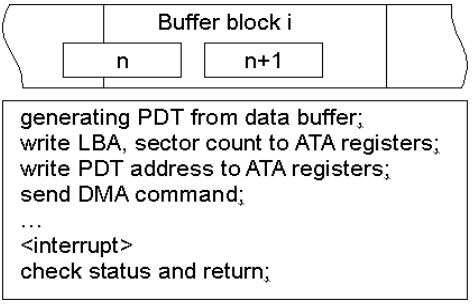

Fig. 1. DMA aligned buffer and DMA working process from the programmers point of view, PDT: Physical Descriptor Table. LBA: Logical Block Address

A typical DMA operation in OS kernel requires several steps to complete, which is shown in Figure 1. Although our program is running on Windows 2000, the steps are the same. PDT is an array of descriptor data structure which stands for physical memory blocks used to perform DMA. According to ATA specification [13], the sector count register has 8 bits currently, thus the DMA data transfer of IDE interface is restricted by a maximum length of 256 sectors by now (set the SECTOR COUNT register to 0).

As we've mentioned above, a large number of fragmental requests sent to low-level drivers may cause too much device I/O, and every device I/O consists of generating PDT, writing ATA registers, starting DMA, seeking and rotational latency and waiting for interrupt, therefore they result in a performance decline. We proposed a buffer method to solve this problem. The buffer format is demonstrated in Figure 1. Buffer block i acts as a generic LRU buffer together with other buffer blocks. n and n+1 represent two sequential small requests sent to HBA miniport. If request n isn't in the buffer (missed), it'll then be buffered in one of these blocks and hoping it'll be visited again, request n+1 is the same. A prefetching method can be applied here so that request n can bring on a prefetch to benefit following requests, such as request n+1. For the small I/O we observed, larger prefetching may service more future requests. But this differs to typical prefetching, because larger prefetching can also cause multiple device I/O to complete, not only the longer data transferring. So the most economical way is to limit prefetching length to just one time larger than the maximum DMA transfer length (maybe multiple times according to application, and this needs further working). In order to implement the buffer in a more efficient way, the starting address we used to prefetch is aligned to the 256 sectors boundary, thus avoiding buffer block overlapping (In Figure 1, this means all the buffer blocks, including buffer i are 256 sectors in size and all their mapped starting address are 256 sectors aligned.). Since the buffer is composed of buffer blocks aligned with maximum DMA transfer length, we call this approach DMA-Aligned Buffer.

However the time spent on data copy in memory and excess device I/O may reduce the benefit. Additionally, new ATA standard may break DMA transfer barrier by extending sector count register from 8 bits to 16 bits, hence the maximum data transfer length in one DMA operation can be increased to 65536 sectors in the future [13][14]. Thereby more work is needed to improve buffer method in the future.

Besides, we may also use other access characteristics to adjust our buffer mechanism. Our experimental software RAID is a part of network storage platform,

we can also use special designed aligned-buffer to cut down possible fragmentary requests which may cause noticeable transfer overhead.

4 Implementation

4.1 Buffer Data Structure

The main buffer of software RAID consists of several fixed-size memory blocks which act as R/W buffers. The number and size of these memory blocks are described as BufferNum and BufferLen respectively. These memory blocks form up a buffer queue, and every buffer block is managed by a data structure called _BUFFERBLOCK, which is shown in the following structure.

```
typedef struct_BUFFERBLOCK
{ PVOID    PreReadBuffer;   //Buffer virtual address pointer
  PVOID    Phy_PreReadBuffer;//Physical address  pointer
  UCHAR    Adapter;         //HBA ID
  UCHAR    Device;          //Device ID
  PREGISTERS_ADDR BaseAddress; // Base address of PCI
configuration space
  ULONG    StartLBA;        //Start LBA of data stored in
this buffer block
  ULONG    SectorCount;     //Sector number contained in
this buffer block
  USHORT   HitCount;         //Buffer hit counter
  USHORT   WriteCounter;     //Dirty counter (how many
writes since last refresh)
}BUFFERBLOCK, *PBUFFERBLOCK
```

According to Microsoft Windows Device Driver Development Kit (DDK) documentation, we may call ScsiPortGetUncachedExtension to allocate BufferNum * BufferLen bytes memory while SCSI port/miniport driver loading for buffer use. Maximum allocable memory depends on available system resources and can influence buffer hit rate, in most cases, the larger the better. BufferLen is decided by device interface. For example, IDE uses an 8-bit register as sector count, hence the maximum read/write length is 256 sectors * 512 bytes per sector = 128KB.

4.2 The DAB Scheme

The operation of the DAB scheme can first be divided into four cases depending on whether the sector of requested block (whether a read or write) is found in the buffer.

(1) Both the first and the last sector of requested block are found in the buffer: The requested block is located inside of one buffer block since it's smaller than 128KB. So the request is hit and no actual device I/O is performed, we just return the buffered data to complete the request.

(2) When the first sector of requested block is found in the buffer and the last sector isn't: The former part of requested block is located in one of the buffer blocks. So the request is partially hit, device I/O is needed to fetch its latter part. This will cause a DMA operation in order to prefetch 128K data that begin with this missing part. After necessary buffer block replacement and prefetching, this request is completed.

(3) When the first sector of requested block isn't found in the buffer but the last sector is: The latter part of requested block is located in one of the buffer blocks. So the request is also treated as partial-hit, device I/O is needed to fetch its former part. This will cause a DMA operation in order to prefetch 128K data that end with this missing part. After necessary buffer block replacement and prefetching, this request is completed.

(4) Neither the first sector nor the last sector of requested block is found in the buffer: The requested block isn't in any buffer block. Act as a traditional buffer and use 128K-aligned address to prefetch.

The action of the DAB scheme is presented below in algorithmic form.

```
if (request beginning sector in buffer) then
if (request ending sector in buffer) then{
request is satisfied and LRU stack updated;
}else{
fetch 128K-aligned data that contains the latter part
of request to buffer;
}
elseif (request ending sector in buffer) then{
fetch 128K-aligned data that contains the former part
of request to buffer;
}else{
use 128K-aligned address to prefetch;}
```

All prefetching can be completed in one DMA operation. We may use either write through (WT) or write back (WB) policy to handle write requests. [15] Missed write request will cause its referenced 128K block be prefetched in WB buffer, which is then updated in place in the buffer, so that later read/write requests may hit, saving its DMA. For WT buffer, write requests will be sent directly to HBA, thus write performance will not be affected.

5 Performance Evaluations

We realize our DAB method under Windows 2000. Low-level raw I/O accesses with different request size are tested. Block-level I/O access is achieved through SRBs (SCSI Request Block) It is sent from disk class driver to SCSI port/miniport driver, which is fully complied with Windows internal standards [16].

The benchmark we used to measure system throughput is Iometer [17] which is a popular file system benchmark developed by Intel. Iometer is an I/O subsystem measurement and characterization tool for single and clustered systems. It measures performance under a controlled load. Iometer is both a workload generator and a

measurement tool. It can be configured to emulate the disk or network I/O load, or can be used to generate synthetic I/O loads. It can generate and measure loads on single or multiple (networked) systems.

Table 1. Iometer test options

Sequential Distribution	100% Sequential
Read/Write Distribution	100% Read
Transfer Delay	0ms
Burst Length	1 I/Os
Transfer Request Size	from 1KB to 16KB

Our experimental settings for the purpose of evaluating the performance of DAB are shown in Figure 2. We load our driver to the test machine for DAB testing with the Iometer options shown in Table 1. The test machine runs windows 2000 with two Tekram DC200 dual-channel UltraDMA IDE adaptors. Its configuration is described in Table 2 and the parameters of individual disks are summarized in Table 3.

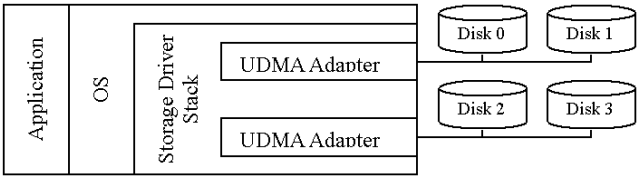

Fig. 2. Experimental platform –Software RAID based on Windows 2000 workstation. We have four hard disks connected via two dual-channel UDMA66 adapters. This software RAID can be configured as RAID0 with 2 or 4 disks, RAID1 or RAID1+0

Table 2. Test machine configuration

CPU	RAM	OS	FS
Intel Celeron 700	128M	WIN2000	NTFS

Table 3. Disk parameters

Disk Model	DJNA371350	RPM	7200
Interface	IDE ATA/66	Latency (ms)	4
Capacity	13.5G	Transfer rate (MB/s)	17
Data Buffer	2MB	Seek Time (ms)	9
Manufacturer	IBM		

We compared DAB software RAID with original software RAID and single disk in four aspects: IRP number, Read throughput, IRP response time and CPU utilization, which stand for total I/Os per second, total MBs data per second, request delay, host CPU load, respectively. As can be seen in Figure 3, compared to single disk (roughly the same as RAID1 configuration), both software RAIDs work better. If DAB isn't used, software RAID will suffer a significant performance decline on both the transfer rate and the number of request when the data block size is small. However, by using DAB, software RAID driver will consume more CPU cycles and the CPU utilization increases obviously on large data blocks.

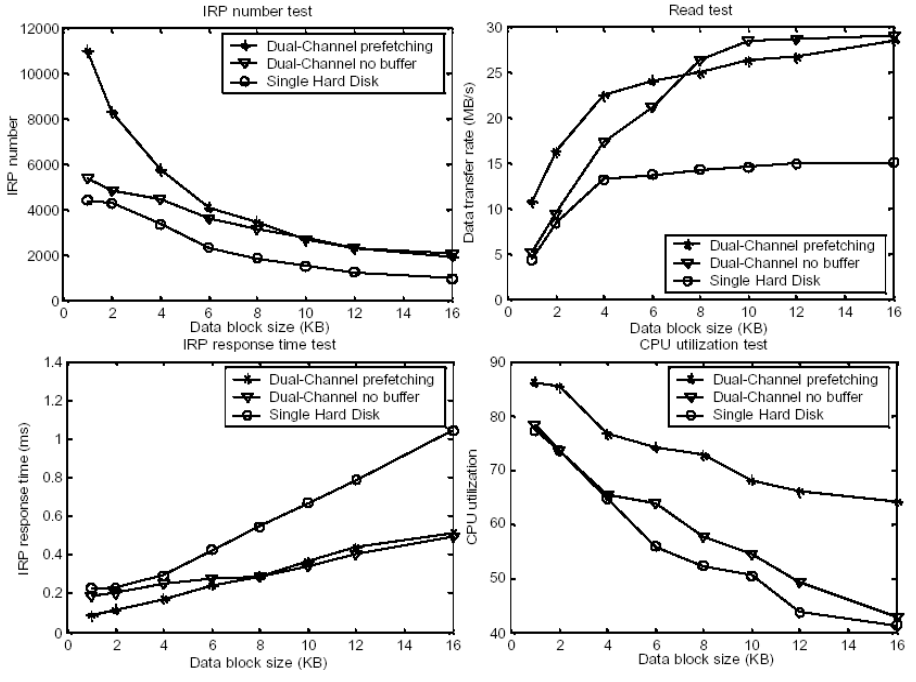

Fig. 3. Software RAID performance test results under windows 2000, using two-disk RAID0 configuration, based on dual-channel UDMA adapter. IRP represents an I/O operation

6 Conclusion and Future Work

According to our previous research on storage system, we noticed that I/O request's characteristic has a notable impact on I/O performance. As for software ATA RAID, adjusting its accessing block size around maximum DMA transfer length may result in transfer rate change. We proposed DMA-aligned Buffer to make use of this issue, by aligning small requests with DMA transfer boundary.

We have also shown that DAB can improve small block transferring, both the number of serviced request per second and the data transfer rate are improved when accessing block size is small. For larger accessing block size, the number of serviced request per second and data transfer rate are least affected by DAB, the downside happens only when CPU utilization is increased. The gap between the implementation with and without DAB grows wider as accessing block size increases.

One of our future work is to implement X-aligned buffer in our network software RAID, where X refers to various access pattern, such as network transfer block size. Especially for iSCSI protocol [18], its performance is greatly affected by different block sizes because a large number of fragmentary requests may result in considerable high overhead.

Although we have proved that our approach is able to enhance software RAID performance, the work presented in this paper is only the tentative step to the solution for storage system software implementation with better Cost/Performance Ratio.

References

1. P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson.: RAID: High-Performance, Reliable Secondary Storage. ACM Computing Surveys, (1994) 145–185
2. Software RAID vs. Hardware RAID, http://www.adaptec.com/ (2002)
3. Jakob Østergaard, “The Software-RAID HOWTO”, v0.90.8, http://www.tldp.org/HOWTO/Software-RAID-HOWTO.html, August 5, (2002)
4. David A. Solomon and Mark E. Russinovich.: Inside Microsoft Windows 2000, Third Edition, Microsoft Press (2000)
5. Alessandro Rubini, Jonathan Corbet, “Linux Device Drivers”, 2nd Edition, O'Reilly (2001)
6. Microsoft Windows2000 DDK documents, Microsoft Press (2000)
7. Kotz, D., Ellis, C.S., “Practical Prefetching Techniques for Parallel File Systems” Proc. First Int'l Conf. on Parallel and Distributed Information Sys, Florida, 1991, pp. 182-189.
8. Smith, A.J., "Disk Cache--Miss Ratio Analysis and Design Consideration". ACM Trans. on Computer Systems (1985) 161-203.
9. Knut Stener Grimsrud, James K. Archibald, and Brent E.Nelson.: Multiple Prefetch Adaptive Disk Caching, IEEE Transactions of knowledge and data engineering, Vol 5. NO.1 (1993)
10. Jiri Schindler, John L. Griffin, Christopher R. Lumb, and Gregory R. Ganger.: Track-aligned extents: matching access patterns to disk drive characteristics. Conference on File and Storage Technologies, Monterey, California, USA (2002)
11. Russel Hugo Patterson III.: Informed Prefetching and Caching. PhD Thesis (1997)
12. Leonard Chung, Jim Gray, Bruce Worthington, Robert Horst.: Windows 2000 Disk IO Performance, Microsoft Research Advanced Technology Division (2000)
13. Information Technology - AT Attachment with Packet Interface – 7. Revision 3e, http://www.t10.org/ (2003)
14. Serial ATA: High Speed Serialized AT Attachment, http://www.t10.org/ (2003)
15. Stallings, Wm.: Computer Organization and Architecture: Principles of Structure and Function, Macmillan, New York (1993)
16. Walter Oney.: Programming the Microsoft Windows Driver Model, Microsoft Press (2002)
17. Iometer project, http://iometer.sourceforge.net/
18. Kalman Z. Meth, Julian Satran.: Design of the iSCSI Protocol, MSS'03 (2003)