A Content Delivery Accelerator in Data-Intensive Servers

Joon-Woo Cho, Hyun-Jin Choi, Seung-Ho Lim and Kyu-Ho Park

Computer Engineering Research Laboratory, EECS
Korea Advanced Institute of Science and Technology
{jwc, hjchoi, shlim}@core.kaist.ac.kr, kpark@ee.kaist.ac.kr

Abstract. The standard OS and server platform hardware have not been optimized for applications that transfer large multimedia files, resulting in poor server I/O performance. One source of the problem is that several redundant copies are introduces when the data is transferred from disks to a Network Interface Card. To solve the problem of redundant copies, we propose a Contents Delivery Accelerator that accelerates large file transfers by eliminating the redundant copies from disks to the NIC. To eliminate the redundant copies, the CDA introduces a new function, called a logical direct link, which provides the shortest path from the disks to the NIC. By using the shortest path, we can completely eliminate the redundant copies, thereby improving the I/O performance of server. The CDA architecture is a combined hardware-software approach. Thus, it comprises CDA hardware and a modified Linux kernel. We implemented the current version of the CDA on a Linux 2.4.18 kernel and an IXP1200 evaluation board. In the experiment, we compared the logical-direct path with a redundant path. For the transfer of data from disks to the NIC, our experimental results show that the average transfer latency of a direct path is as much as 30 percent less than a redundant path.

1 Introduction

Internet web servers deal with an enormous amount of multimedia data. This work is highly time-consuming and can increase the response time of the server. Consequently, clients that connect to the server might not get multimedia data in time. When a multimedia server operates a general-purpose operating system (OS) such as Unix and Linux, the multimedia data is often too large to be handled effectively because those systems have not been optimized for multimedia data.

There are two critical problems in conventional Web-servers that handle streaming multimedia data. The first problem is that there are many redundant data copies between the disks and the network interface card (NIC). These redundant copies are due to the long data path and the modern OS architecture that splits the OS space and the application space. Figure 1 shows the data flow of this programming model. First, the CPU initiates the disk controller to get

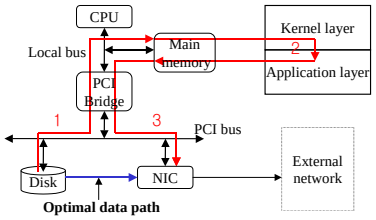


Fig. 1. Data-transfer path in general-purpose system

the data fragment that we wish to send. The data fragment is delivered to the main memory from the disks. The data in the main memory is then delivered to the NIC. During this path, redundant copying occurs between the disks and the NIC. Moreover, there is another redundant copy of the data, though it is not shown in figure 1. The data delivered from the disk is located in the kernel-space memory. The server application program then copies the data into the user-space memory region. A similar operation is required at the downside data path from the memory to the NIC. All these wasteful data flows are repeated until all the data is delivered [1]. If a server working set is small, the problem we just described is not critical because the operating system manages the buffer cache so that data can be located in the cache area in advance and these data can be directly delivered from the memory to the NIC rather than from the disks. However, if the server working set is large, especially a multimedia streaming server, these problems are critical.

Another problem is the TCP processing time. The processing of the TCP/IP protocol is difficult. Consequently, if the CPU of a Web server processes the TCP by itself, the CPU wastes many CPU clocks, and the response of the server therefore becomes slow. In a gigabit network, the problem is exacerbated [6]. Many researchers and companies therefore suggest various TCP offloading techniques to reduce the CPU overhead for processing the TCP packet. A well-known technique is to import TCP accelerating hardware such as a TCP offloading card [2][3]. However, to support these mechanisms, a new interface is required between the server application and the TCP offloading engine. This is critical problem for the system compatibility.

To solve these problems which are described above, we now propose a new architecture, which uses additional assisitant hardware Contents Delivery Accelerator. When the CDA is inserted into a PCI slot, it functions and interacts with the CPU to resolve the problems mentioned above. First, the CDA can make a direct path between the disks and the NIC, and between the memory and the NIC. By using this direct path, the CDA can eliminate the redundant copy and transfer large amounts of multimedia data without interfacing with the host CPU. Second, the CDA can manage the data cache for the hot data or frequently accessed data. As a result, Servers can improve their throughput and reduce the response time. Third, the CDA can offload the TCP job from the host. The CDA relieves the host from that heavy work so that the host can

reduce the CPU overhead for the TCP packet processing and use the CPU for other purposes. In our approach, the more important thing is that the CDA does not harm Linux compatibility, obviating the need to modify current applications. The CDA has this ability because we kept the Linux system call interface when we modified the source codes of the host Linux kernel.

The organization of this paper is as follows. The next section covers related works. In Section 3, we describe the architecture of the CDA, while in Section 4 we discuss how we implemented the CDA. In Section 5, we show how the CDA performs. Finally, in Section 6, we make a comparison and offer our conclusions.

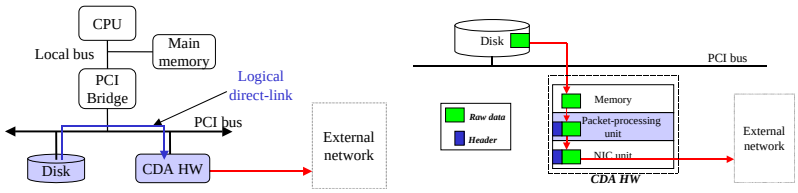
2 Related Works

There have been several works on eliminating the copy overhead and processing overhead of data intensive servers. The general approach to solving the redundant copy overhead involves optimizing the OS. To solve this problem, which is caused by a layered approach, general OSs introduce an 'mmap' and 'sendfile' system call. These system calls prevent one redundant copy between the OS and the application program so that the server can use its resources more effectively [7][8]. The sendfile system call is used in the Apache Web server to deliver requested data. Because this interface is only operated in the kernel-space region and sends data directly using the kernel copy operation according to the Apache document, the sendfile call enables Apache to deliver static content faster and with lower CPU utilization [5]. Alternatively, to enhance the data transfer and to increase the memory and cache efficiency, some OSs such as I/O Lite [11] unify the buffer of many subsystems in the OS to reduce the redundant copies between the subsystems. Moreover, someone has made an I/O-specific OS such as Hi-Tactix [12].

By using the approach of existing software, we can eliminate the problem of redundant copy in the main memory. However, the problem of producing redundant copies outside the main memory still exists. With the help of improvements to existing hardware techniques, many I/O-specific types of hardware have been introduced to solve the problem of redundant copy. The Xiran [3] offers I/O-specific hardware with a disk interface and a network interface. The two interfaces are connected to each other by a direct path that sends data without producing a redundant copy. However, to use this direct path, a new application was made by SDK provided by the Xiran company. As a result, this method has no software compatibility.

3 CDA Architecture

We suggest CDA architecture that uses the auxiliary hardware shown in figure 2. This architecture is similar to existing system architecture so that it can maintain its interface and compatibility. The purpose of the CDA is to improve the I/O performance of the server by eliminating the redundant copy problem and achieving zero copies between the disk and the NIC. To do this, the CDA



(a) Logical Direct Link (b) Network Packet Processing

Fig. 2. Proposed CDA Architecture

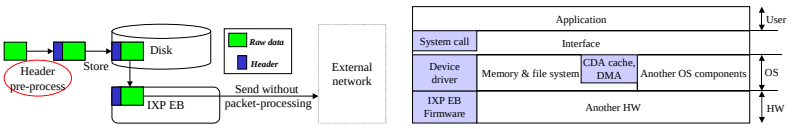
provides two functions: the logical direct link and network packet processing. To provide these functions, the CDA is composed of CDA hardware and a modified Linux kernel. The prototype of the CDA hardware is a kind of intelligent NIC. It has many units. To use the CDA hardware in the Linux system, we modified the Linux kernel, version 2.4.18. Specifically, we implemented a device driver for the CDA hardware and modified the system call, the memory management and the file system.

The logical direct link provides a zero-copy path between a disk and the CDA hardware. This link can be made up through the address that redirects them from the main memory of the host to the CDA memory region. Through this link, we can directly copy the data from a disk, as shown in figure 2(a). By using this, we can eliminate the redundant copy that occurs outside the main memory.

The copied data from a disk to the CDA hardware through the logical direct link is raw data. This means that the data packets have no processing header for the TCP or IP. As a result, data cannot be sent to an outside network. To send that data, the raw data stored in the CDA memory must be processed to get the network header shown in figure 2(b). The header is then packetized with the raw data. After these operations, the network packet made at this packet is sent to an outside network through the NIC unit similarly to an Ethernet card.

4 Implementation

We made the prototype of the CDA using Intel's IXP1200 evaluation board [14]. The evaluation board has architecture that is similar to our proposed hardware. By using this board, we checked the validity of the CDA architecture and, accordingly, the OS that we modified to fit with the CDA prototype. Because the IXP evaluation board has no packet processing unit, it cannot convert raw data into a network packet at the sending time. To solve this problem, we used a virtual packet-processing method, as shown in figure 3(a). Before storing new raw data, the CDA preprocesses the data to get the network header for the raw data and it stores them together on the disk. At the transfer time, the CDA gets the network header and the raw data from a disk. By using this method, the CDA



(a) Virtual Packet Processing (b) CDA module Design

Fig. 3. Virtual Packet Processing and CDA module Design

can send data without packet processing at the transfer time [13]. However, this method only supports the UDP because the TCP does dynamic processing; for example, controlling the fragments and flow during the transfer. In implementing the CDA mechanism, we considered conserving at the design stage Linux's I/O mechanisms such as page caching, reading ahead, and DMA, especially for fast file I/O operations [1]. If the CDA could not maintain these mechanisms, the I/O performance would be impaired. As a result, the CDA maintains these mechanisms to achieve a good I/O performance for general cases and for large cases. Figure 3(b) shows the CDA modules in the Linux OS. The inserted modules are the device driver of the IXP evaluation board, the CDA cache, the DMA, the firmware of the IXP evaluation board and the modified system call. We elaborate the implementation of the CDA module in detail.

Device driver To copy data from a disk to the memory of CDA hardware, a kernel is required to directly access the memory of CDA hardware. The role of the device driver, therefore, is to make a logical direct path that maps the hardware components such as the memory and registers into the kernel space. To do this function, Linux offers the following functions: *pci_read_config_dword* and *ioremap_no_cache*. A pseudo-code for using these functions is as follows:

```
pci_read_config_dword(&val);
bus address=val&MASK;
virtual address=ioremap_nocache(bus address);
```

After this operation has been completed, the kernel can transparently access and use components of the IXP evaluation board through the virtual address variable.

System call To send data to an outside network, we modified the sendfile system call provided by Linux. This system call is suitable for the CDA because it can explicitly send a file on a disk outside the network. In some cases, however, applications use this system call to quickly copy data from a disk to the memory of the host computer. We therefore modified this call to ensure it could be applied to both cases without destroying the interface of the system call. First, we chose

one disk from among the many disks in the system to be the CDA's own disk. In that disk, we stored a file to be sent to an outside network. We also modified the sendfile system call itself. When a file is called by the modified sendfile call, the OS first checks where the requested file is located. If the file is not in the CDA's own disk, the OS calls the original function *do_generic_file_read* to copy the requested file to the memory of the host computer. Otherwise, the OS calls the modified function *cda_do_generic_file_read* to directly copy the requested file from the CDA's own disk to the memory of the CDA hardware.

CDA cache With the help of the CDA cache, the data stored in the CDA hardware memory can be reused without having to access the disk. The CDA cache can therefore increase the overall I/O performance of the CDA. In this section, we describe in detail how we implemented the CDA cache. The CDA cache is based on the Linux page cache mechanism. Because the page cache uses a hash algorithm for a fast search, the CDA cache uses the same algorithm. When a file is requested by the sendfile call, the CDA first searches for the requested data at the cache table in the CDA hardware. If there is no descriptor at the cache table, the CDA calls the *descriptor_alloc* function. This function allocates a new descriptor for that file and inserts it into the cache table by calling the *add_to_cache_table* function. After this operation, the OS reads the requested file from the disk and copies it into the memory of the CDA hardware. Finally, the OS accesses the requested file at the memory of the CDA hardware. How the CDA cache operates when data is cached in the memory of the IXP evaluation board is described below. When the sendfile call requests data, the CDA also searches for the requested data in the cache table of the CDA. In this case, the CDA uses a page descriptor to confirm the existence of data in the memory of the evaluation board. The CDA can therefore use data in the memory of the evaluation board without having to access the disk.

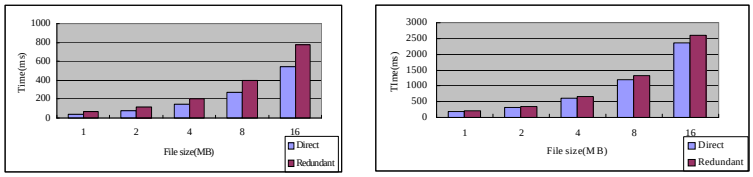
DMA operation To transfer data quickly, Linux uses a DMA mechanism to copy data between the memory and the I/O device. With DMA, the host CPU can do other tasks and significantly reduce the bus transactions when transferring large amounts of data. The DMA controller does the DMA operation with the aid of the bus address. The kernel, however, orders data to be copied from the I/O device to the memory through a virtual address. A translation function is therefore required to convert the virtual address to the bus address. When the OS orders a file to be copied from the disk to the memory, it sends a virtual address of the IXP memory to the DMA controller. The address translation function in the DMA module intercepts a virtual address and converts it into a bus address. It then sends the converted address to the DMA controller. Finally, the DMA controller uses the bus address for the copy operation.

5 Experimental Results

The logical direct link offers a zero-copy mechanism between a disk and a network interface. Using this mechanism, the amount of data on a PCI and a local bus can be reduced to half. If we can measure how much the bus is used, we can easily estimate the performance. However, there is no software tool or equipment for measuring how much the bus is used. We used the CDA system to compare how long it takes to transfer data between the logical direct path and the redundant path in a conventional system. For the experiment, we connected two computers via an Ethernet LAN.

First, to determine the benefit of a direct path, we measured the data transfer-time in the two types of system architecture, which compares the direct path from a disk to the memory of the CDA hardware and the redundant path that uses the system memory. We generated synthetic workloads that performed several file transfer iterations. The workload applications generated file transfer requests from the disk to the NIC, which models the file send mechanism in multimedia servers. The variation in the size of the requested files ranged from 1 MB to 16 MB. Figure 4(a) shows the result of this experiment. For all file sizes, the processing time was, on average, 30 percent less for the direct path than for the redundant path. Moreover, the files were transferred more efficiently for the direct path than for the conventional system. As the file size increases, the transfer time is more significantly reduced than in the conventional method. This result means that the direct path is more suitable for transferring the large files of multimedia streaming servers. Next, we measured the total data transfer-time from a disk to another computer by comparing direct path and redundant path. In this experiment, we used the client-server model and we measured the overall performance of the system with respect to data-intensive servers such as multimedia systems. The workload of the experiment was the same as in the first experiment. The files we used ranged in size from 1 MB to 16 MB. Figure 4(b) shows the result of this experiment. For all file sizes, the copy time at the direct path was, on average, 10 percent less than at the redundant path. The performance of this experiment was worse than the previous experiment because of the IXP evaluation board, which makes a prototype of the CDA hardware. Because the network system in the IXP evaluation board formed a bottleneck, this experiment showed less improvement than the previous experiment. In the next experiment, we therefore present an analytical model that performs better without the effect of a network system.

Because existing CDA hardware (which includes the IXP evaluation board) has many limitations, the results of previous experiment are disappointing. Consequently, in this section, we use an analytical model to predict the overall performance of the CDA. We used the following variables to predict the performance:
1. T_{dm}, the duration of copying data from a disk to the memory of an IXP evaluation board
2. T_{mn}, the duration of sending data from the memory of an IXP evaluation board to an outside network
3. T_{total}, the total transfer time from a disk to an outside network.



(a) Result for Exp 1 (b) Result for Exp 2

Fig. 4. Experimental Results for the data transfer using CDA

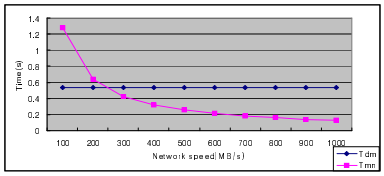


Fig. 5. Result of Experimental Model

For analysis of the model, we used the results of a 16 MB file. At this size, the bandwidth of the disk shows that 29 MB of data per second can be stably transferred from a disk to the memory of an IXP evaluation board. Unlike existing hardware, our hardware can support concurrent I/O activity. Only the larger absolute value between T_{dm} and T_{mn} affects T_{total}. In this case, one operation can proceed without waiting for another task to be completed, thereby ensuring that one task's operation time masks another task's operation time. Figure 5 shows the results of this prediction. When the speed of network hardware is greater than 300 Mbps, T_{dm} is larger than T_{mn}, indicating that only T_{dm} has any effect. Accordingly, when we use a disk that shows a speed of 29 MB/s when the speed of the network hardware is greater than 300 Mbps, T_{total} is 0.54 s.

6 Conclusion

We propose the design and implementation of fast I/O architecture called the CDA to solve the problem of redundant copies between the disk and the NIC. The CDA comprises CDA hardware, which is a substitute for the NIC, and a modified Linux kernel, which provides the CDA functions for the CDA hardware. Because existing versions do not have their own CDA hardware, we made a prototype of the CDA architecture using the Intel IXP1200 evaluation board, which has similar functions. With these two components, the CDA provides a logical direct link that can do zero copies between the disk and the NIC. Using this function, data can be copied directly from a disk to the CDA hardware so that the server can improve I/O efficiency when sending a large file. Furthermore, to achieve

I/O efficiency in general, as well as for large files, we considered using the fast I/O mechanisms of Linux such as page caching, reading ahead, and DMA. In our experiments, we verified that the CDA can transfer data faster than a general OS and regular hardware.

To take full advantage of the CDA, we need to use a fast network device such as a gigabit Ethernet process or a simple network process. On the other hand, because existing CDA hardware has no packet-processing unit, the existing CDA uses virtual packet processing to store packets rather than raw data in the disk. However, this method does not support the TCP. At present, only the UDP can be used. Many Internet applications use the TCP to guarantee the reliability of data transfers. For quick processing of the TCP, the packet processing unit must be added [9][10].

References

1. Daniel P. Bovet, and Marco Cesati, Understating the Linux Kernel, OReilly, 2001.
2. Alacritech, Inc. Delivering High-Performance Stroage Networking, white paper, 2001.
3. Xiran, A Division of SimpleTech Inc, http://www.xiran.com
4. Soam Acharya and Brian Smith, "MiddleMan: A Video Caching Proxy Server", *10th International workshop on Network and Operating Systems support for digital audio and video*, 2000.
5. Apache web document, http://httpd.apache.org/docs-2.0/misc/perf-tuning.html.
6. Evangelos P. Markatos, "Speeding up TCP/IP: Faster Processors are not Enough", *21st IEEE International Performance, Computing, and Communication Conference*, 2001.
7. Dragan Stancevic, "Zero Copy I: User-Mode Perspective", *Linux Journal*, 2003.
8. W. Richard Stevens, Advanced Programming in the UNIX Environment, Addison-Wesley, 1992
9. Evangelos P. Markatos, "Speeding up TCP/IP: Faster Processors are not Enough", *21st IEEE International Performance, Computing, and Communication Conference*, 2001.
10. Eric Yeh, Herman Chao, Venu Mannem, Joe Gervais and Bradley Booth, Introduction to TCP/IP Offload Engine (TOE) Version 1.0, 10 gigabit ethernet alliance.
11. V.S. Pai, P. Druschel, and W. Zwaenepoel, "it IO-Lite : A unified I/O buffering and caching system", *The 3rd USENIX Symposium on Operating Systems Design and Implementation*, New Orleans, USA, 1999
12. Damien Le Moal, Tadashi Takeuchi, Tadaki Bandoh. "Cost-Effective Streaming Server Implementation Using Hi-Tactix"", *ACM Multimedia 2002*, pp. 382-391.
13. Halvorsen, P., Plagemann, T., Goebel, V. "Network Level Framing in INSTANCE", *Proceedings of the 6th International Workshop on Multimedia Information Systems 2000 (MIS 2000)*, Chicago, IL, USA, October 2000, pp. 82-91.
14. Intel IXP1200 Network processor Hardware Reference Manual, Intel document 278303-008, Aug. 2001.