1998 International Computer Symposium
Workshop on Computer Architecture
December 17-19, 1998, N.C.K.U., Tainan, Taiwan, R.O.C.

Performance Evaluation of Intelligent I/O on File Servers

Chih-Yung Peng[✦], *Kuo-Pao Fan*[†], *and Chung-Ta King*[✦], *Yuan-Bin Tsai*[†]

[✦]National Tsing-Hua University, Hsinchu, Taiwan, R.O.C

[†]Communication Research Laboratories Industrial Technology Research Institute, Chutung, Hsinchu, Taiwan, R.O.C.

ABSTRACT

The intelligent Input/Output (I_2O) architecture is a standard to develop protable device drivers that offloads the host CPU for I/O operations. In this paper, we study the effectiveness of I_2O on file servers using the NetBench benchmark. We compare the total throughput of the file server with and without I_2O supports. The evaluation results show that I_2O indeed improves throughput of the file server, and allows the server to handle more number of clients. Our evaluation also shows that I_2O is especially useful when the server CPU load is heavy.

1. INTRODUCTION

In recent years, the Internet is growing at an exponential rate. The rapid growth has enabled numerous Internet-based applications. Many applications follow the client/server paradigm, in which clients in different parts of the Internet send requests to various servers to obtain services. Example services include video-on-demand, world wide web, printing, database, etc. More and more services will become possible in the future.

The performance of a server depends not only on its processors, but also on its I/O capability. Especially services over the Internet is that they often involve a lot of I/O operations, e.g., network communication, file accesses, etc. To do these operations, the processors have to spend a lot of time serving interrupts and running protocols such as TCP/IP. These in turn greatly reduce the processor efficiency. How to devise a scalable, efficient I/O subsystem is thus critical to server systems. The Intelligent I/O (I_2O) architecture [9] was proposed for this purpose.

The I_2O architecture offers a solution to provide a flexible, powerful I/O subsystem. Adding an extra I/O processor (e.g. Intel i960) to the I/O subsystem can offload I/O functions from the main CPU, and increase CPU efficiency by reducing the number of interrupts. The I_2O architecture adopts a split driver model, in which a device driver is partitioned into an OS-Specific Module (OSM) and a Hardware-Specific Module (HDM). The OS vendors provide OSMs for each different class of devices, e.g., LAN OSM for network devices. The device vendors on the other hand concentrate on developing the HDMs. This split device model thus increases the portability of drivers.

This work was supported by CCL of ITRI under grant G3-87062

In I_2O, a device driver can be split more than once, called *stackable drivers*. This enables an independent software vendor to add more functions to an I/O subsystem, independent of both the OS and the hardware. A typical example is to stack a RAID driver on top of a disk driver to increase the reliability of the disk subsystem.

In this paper, we study the effectiveness of I_2O in improving the performance of servers. We concentrate on the file server and use the NetBench [11] to evaluate the server performance with and without I_2O. NetBench 5.0 is a portable Ziff-Davis benchmark program that measures the performance of file servers as they handle I/O requests from DOS, Windows 95, Windows for Workgroups, Windows NT Workstations, or Mac OS clients.

From the evaluation results, we find that file servers with I_2O have a higher throughput. The improvement is more obvious when the load of the server host CPU is heavy. Using I_2O, the file server can also handle more number of clients.

The rest of this paper is organized as follows. In Section 2, we give an overview of the I_2O architecture and describe an I_2O development platform. Section 3 introduces our evaluation environment. In Section 4, we present the evaluation results and discuss the parameters affecting the performance of an I_2O-based file server. Finally, Section 5 gives the conclusions and future works..

2. PRELIMINARIES

In this section, we give some background on I_2O. Section 2.1 describes briefly the I_2O architecture. Section 2.2 introduces an I_2O development platform.

2.1 Overview of I_2O Architecture

The I_2O Architecture Specification describes an open architecture for developing device drivers in networked environments. The architecture is independent of the operating system, processor platform, and system I/O bus. It defines a standardized development so that portions of the driver can be offloaded to an embedded processor on the I_2O adapter card.

2.1.1 Split Driver Model

In I_2O, a device driver is split into several modules, as shown in Figure 1. The OS-Specific Module (OSM) provides the interface to the operating system. Typically, the OS vendor supplies this module. The Hardware Device Module (HDM) on the other hand provides the interface to the I/O adapter and its devices. The hardware vendor supplies this module, which contains no OS-specific code. A third component, the Intermediate Service Module (ISM), further splits the driver and adds more functionalities between the OSM and HDM. An independent software vendor can supply ISMs. HDMs and ISMs are often referred collectively as the Device Driver Modules (DDMs).

I_2O provides a software framework to integrate different drivers from multiple vendors, and an arbitration mechanism to let them share processing resources. The framework also allows the I/O processing system to bypass the main CPU altogether for peripheral-to-peripheral communications, e.g., loading multimedia files off the disk directly to the network, without causing any interruption to the main processor.

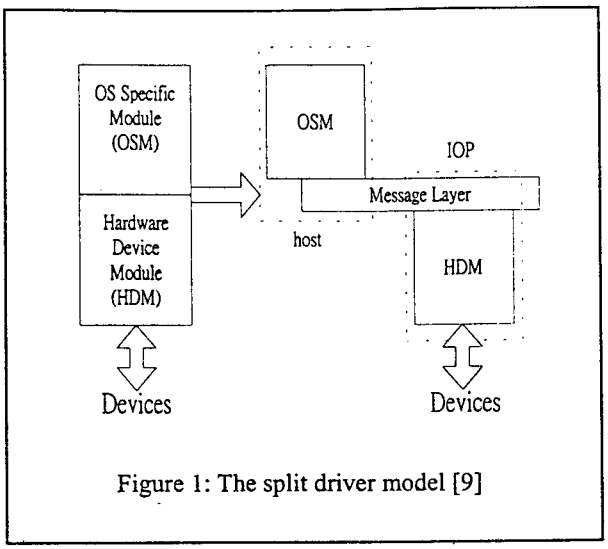


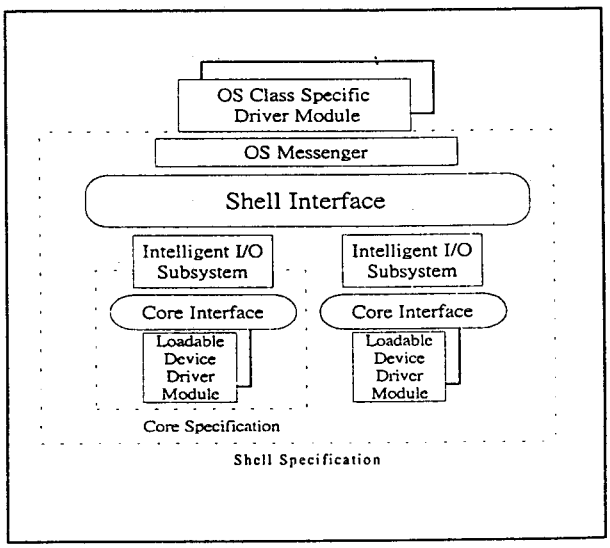
Figure 1: The split driver model [9]

2.1.2 Message Passing Interface

In I_2O architecture, communications between different modules are carried out by passing messages. The I_2O interfaces include shell interface and core interface, as shown in Figure 2.

The shell specification defines the interface that an I/O subsystem presents to the host. It specifies the behavior of both the system and subsystem when initializing and managing intelligent I/O subsystems. The shell interface provides both OS and I/O subsystem independence.

The core specification defines the interface between a loadable device driver and the I/O platform. It provides an operating environment for device drivers that, like the shell interface, is independent of both the OS and the I/O platform. This enables any real-time operating system to host device drivers produced by third-party hardware vendors.



2.2 Overview of I_2O Development Platform

An I_2O development platform facilitates the development of I_2O drivers. In the following, we describe briefly a development platform based on the IxWorks real-time operating system from the Wind River System [12]. The standard configuration for developing I_2O drivers with IxWorks includes: a development system, an I_2O host, and an I_2O-conforming I/O processor. The configuration is shown in Figure 3.

The development system is a PC running a visual development environment called Tornado. Tornado is executed on top of Windows NT or 95. Its collection of integrated host-resident tools allow developers of intelligent peripherals to:

(1) build I_2O drivers

(2) test and debug I_2O drivers

(3) package and deliver I_2O drivers and products

(4) monitor and optimize I_2O driver performance.

One dedicated serial port is required to connect the Tornado development system to the IOP. Also, since the components of the Tornado environment communicate using TCP/IP, the development system must have TCP/IP installed with an assigned IP address.

An I_2O host is a PC hosting the I/O device. Its operating system, e.g., Windows 4.0 or Novell Netware, must have suitable OS Service Modules (OSMs) installed for the class of drivers under development.

An I_2O-conforming I/O processor (IOP) executes ISMs and HDMs through the IxWorks operating system. It controls the adapter card that plugs into the PCI bus in the I_2O host. Of course the adapter cards for development must be supplied with a serial cable and connectors to connect to the development system.

As mentioned IxWorks is a real-time operating system conforming to the I_2O architecture. It provides a multithreaded, prioritized framework that allows drivers from multiple vendors to coexist safely. It is fully scalable across all I_2O configurations, from dedicated on-card i960RP processors to open "on-motherboard" implementations to complex distributed I/O systems that service multiple CPUs. IxWorks also includes a number of features for advanced I/O subsystems. It supports peer-to-peer communications, enabling two IxWorks-based systems to talk to one another without host-CPU intervention. It also manages ISMs, which are appropriate for applications like network management or RAID control algorithms that are a step or two on top of the hardware.

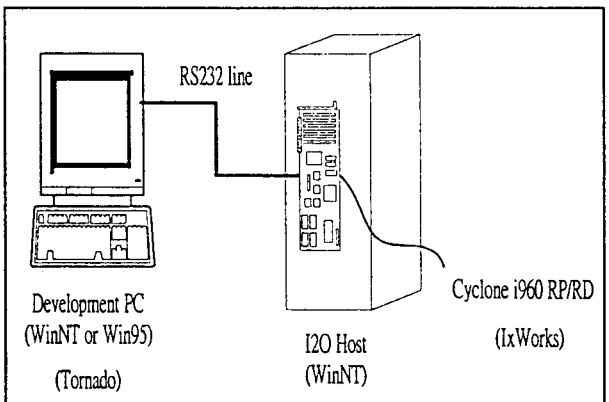


Figure 3: The IxWorks I_2O development platform

3. EVALUATION ENVIRONMENT

In this section, we describe our evaluation environment. We first give an overview of the NetBench in Section 3.1. Then in Section 3.2, we describe our configuration for evaluation.

3.1 Overview of the NetBench

NetBench® 5.0 [11] is a portable Ziff-Davis benchmark program that measures the performance of file servers as they handle I/O requests from DOS, Windows 95, Windows for Workgroups, Windows NT Workstations, and Mac OS clients. To run NetBench, at least three machines are needed, the file server, the controller, and the client, as shown in Figure 4.

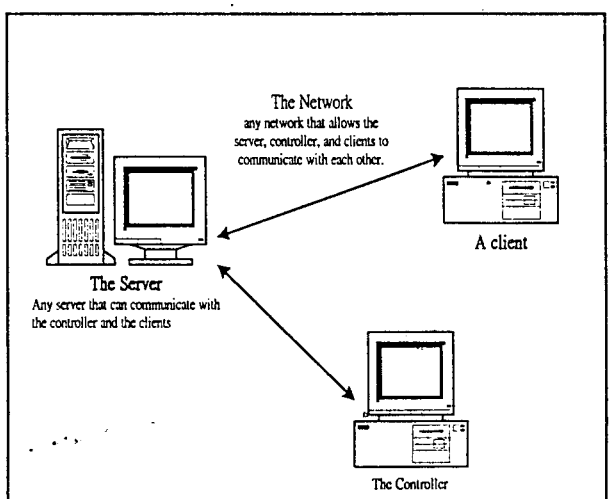


Figure 4: Configuration of NetBench test environment [11].

File Server: File server provides the shared file storage facilities to the clients and on which the clients create files used in the NetBench tests. It is also the machine on which the NetBench controller and client programs are installed. When NetBench is executed, each client sends requests to the server for network file operations on files that exist on the server. NetBench's results reflect the server's performance as a whole, including the server operating system and everything from the disk controller to the network in-

terface card to the number of processors to the network protocol.

Controller: The controller is a PC running Windows 95 or Windows for Workgroups 3.11. NetBench does not run tests on the controller nor does it count the controller as a test client in the tests. Instead, the controller controls which tests NetBench runs, monitors how the tests are going, and views the test results.

Clients: These are PCs and Mac OS systems that actually request the network file operations of the server during the NetBench tests. NetBench can accept up to 1,000 clients; but to get meaningful results from NetBench, only a few clients are needed to stress the server. This can be determined by checking the result curves.

The NetBench test suites include several tests. They are summarized as follows:

The Disk Mix: The Disk Mix is a synthetic application that mirrors the way leading PC applications perform network file operations on a file server. The Disk Mix uses multiple files, different request sizes, and different network file operations to place a diverse load on the server. The Disk Mix was obtained by first profiling applications to determine what sort of file requests they performed and how frequently they performed them. A list of the applications profiled for each client type is shown in Table 1. After profiling the applications, scripts were created to mimic the network file operations of those applications and were turned into the Disk Mix test. When Disk Mix is executed, NetBench tallies the scores for all the individual clients and compiles the overall score for the server.

The I/O Throughput Tests: NetBench includes five I/O Throughput Tests: Sequential Read, Sequential Write, Random Read, Random Write, and Random Read/Write. For each test, each NetBench client creates its own private test data file. It then reads from and writes to the file based on the test type and parameters. Each of these tests includes a think time. Think time is a parameter which tells a client how long to wait after performing one chunk of work before it performs the next. In general, the smaller the value for think time, the more the test stresses the server.

The NIC (Network Interface Card) Test: NetBench uses the NIC Test to measure the peak throughput of network interface cards in both servers and clients. This test attempts to isolate NIC performance by having all the clients sequentially read data from a shared file that fits into cache on the server. In this way, the server does not access the disk but handles all the sequential read requests from memory. Thus, each client throughput reflects the throughput capability of the network interface cards and the network physical layer.

3.2 Our Evaluation Configuration

Our evaluation was conducted on a NetBench platform as follows. The client machine(s) had a Pentium Pro-200 CPU and 64M RAM, running Windows 95. The controller machine had a Pentium-233 CPU, 64M RAM, and Windows 95. The server machine had a Pentium-90 CPU, 32M RAM, and Windows NT 4.0. It was configured following the IxWorks I_2O development platform as shown in Figure 3. The I/O adapter was Cyclone i960 RD with two 10 Mbs Intel 82557 controllers and two Symbios SCSI controllers. The NetBench was running off an IBM ultra-wide 2.1G SCSI HD.

Table 1. The applications profiled for each client type in Disk Mix

DOS Clients	32-bit	16-bit	Mac OS
Borland®	Adobe®	Adobe™	Claris
DBase IV®	PageMaker® 6.0	PageMaker™ 5.0a	Works™ 3.0
Borland Paradox®DOS	Borland® Paradox® 7.0	Claris® FileMaker® Pro 2.12	FileMaker Pro 2.1v2
Harvard Graphics®	CorelDRAW! 6.0	Lotus 1-2-3 Release 4.01	Finder™
Lotus® 1-2-3® for DOS	Lotus® Word Pro™ 96	Microsoft Excel 5.0	Microsoft Excel 5.0
Lotus cc:Mail™ for DOS	Microsoft Access 7.0	Microsoft Word 6.0a for Windows	Microsoft Word 6.0
Microsoft® C compiler	Microsoft Excel 7.0	Windows File Manager	WordPerfect 5.0
Microsoft MAIL for DOS	Microsoft® PowerPoint® 7.0	WordPerfect 6.0a	
WordPerfect® for DOS	Microsoft Word 7.0		

In our configuration, the 82557 HDM and Symbios HDM were running on the IOP. The other file access procedures were running on the host. The LAN OSM and block storage OSM were from Microsoft. In the experiments, the server ran in two different loading conditions. A lightly-loaded server did not run any other programs except serving file requests. A heavily-loaded server, on the other hand, ran a MPEG decompression program for the duration of the experiment, which induced a heavy loading on the server CPU.

4. EVALUATION RESULTS

In this section, we present the results of our performance evaluation. Note that the parameters affecting the performance of a file server include the total number of client machines, the number of client programs in each client machine, the test suites, the load of the server host CPU, and the think time. We use the total throughput on the file server as the performance metric. In each experiment, we compare the obtained throughput on a file server with and without I_2O.

4.1 Effects of Think Time

In this subsection, we study the effect of think time on the throughput of the file server. The think time is the time interval between each request to the file server. In this experiment, there is only one client machine, and in the client machine, only one client program is activated. The test suite used is the Disk Mix, performing only one iteration. Figure 5 shows the result when the load of the server CPU is light, while Figure 6 shows the result for a heavily-loaded server.

From Figure 5 we find that when there is no think time, the throughput is the highest. A longer think time reduces the total throughput, because when the think time is longer, the total number of requests to the server becomes smaller. Thus, the total throughput is reduced even though the host CPU still has extra capacity to process more requests. Comparing the file server with and without I_2O, we can see that the one with I_2O can obtain a higher throughput, because the I/O processor can reduce the number of interrupts to the host CPU.

From Figure 6, we find that when the server load is heavy, increasing the think time does not change the total throughput much. This is because a heavily-loaded server has reached the upper limit of its throughput, which is thus independent of the think time. Again, the throughput of the one with I_2O is higher than the one without I_2O.

4.2 Effects of Varying Clients

In this subsection, we study the effects of increasing the number of clients on the total throughput. In this set of experiments, we used the Disk Mix test suite, set the think time of each mix to zero, and ran one iteration of disk mix in each client program. The results shown in Figures 7 and 8 are based on a lightly-loaded and a heavily-loaded server, respectively. Intuitively with a modest increase in the number of clients, a lightly-loaded server should have enough processing capability to handle the increased file requests.The resultant throughput should increase also.

However, from Figure 7 we can see that this is not the case -- when the number of client programs increases, throughput of the server drops. Note that the client programs are running on the same client computer. Thus one explanation is that the client programs compete with each other for the resources on the client computer, especially at the receiving side. As a result, the rate at which the client computer can move data in and out becomes the limiting factor to the overall throughput. When the server is equipped with I2O, requested data will be replied to the client computer sooner, which in turn intensify the contention among the client

programs. This explains why the throughput in the I2O case drops faster than the one without I2O.

On the other hand, when the number of client programs increases to a certain degree, the excessive file I/O requests sent to the server increase the server overhead in handling the I/O operations and make the latter a bottleneck of throughput. The overall throughput become flattened.

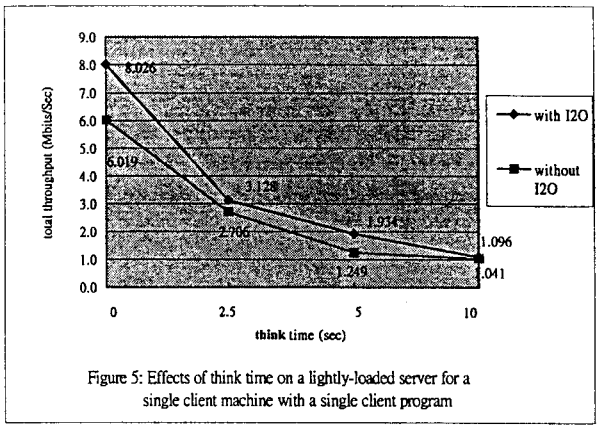


Figure 5: Effects of think time on a lightly-loaded server for a single client machine with a single client program

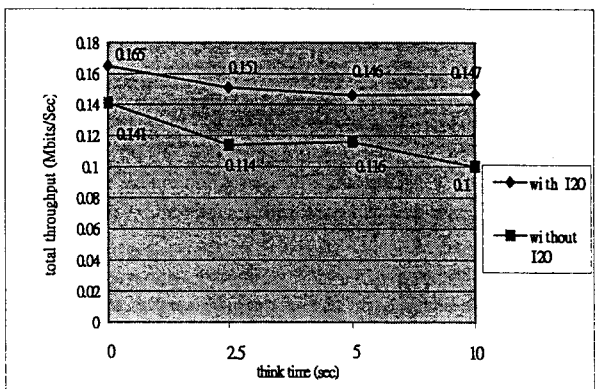


Figure 6: Effects of think time on a heavily-loaded server for a single client machine with a single client program

Figure 8 shows the resultant throughput when the server is heavily-loaded. Again, since the server is now the bottleneck and I2O can off-load the main CPU with reduced interrupts, the server with I2O can achieve a higher throughput. Note however that no matter whether I2O is used, the heavily-loaded server can handle at most two client programs. If more client programs are added, the server cannot keep up with the requests and may drop packets to cause faults.

4.3 Performance under Random Reads

In this subsection, we study the file server throughput under random reads. Each client program performed the requests 150 seconds using a chunk size of 1024 byes. The results are shown in Figures 9 and 10. From the figures, we find that the file server with I2O performs worse than the one without I2O. This is perhaps because the chunk size was too small and the initialization overhead in the server CPU dominated the whole performance. In this case I2O

did not help but added extra overhead to the datapath, which resulted in a degraded performance. Another possible cause for the performance degradation in I2O is the access patterns of random reads. We are still investigating this cause.

Note that when the CPU load is light, the curves in Figure 9 are quite flat. This means that the server had enough computing capability to handle requests, which is another indication of the light workload exercised by the random read test suite. Apparently, I2O is not most efficient when operating under such a loading condition.

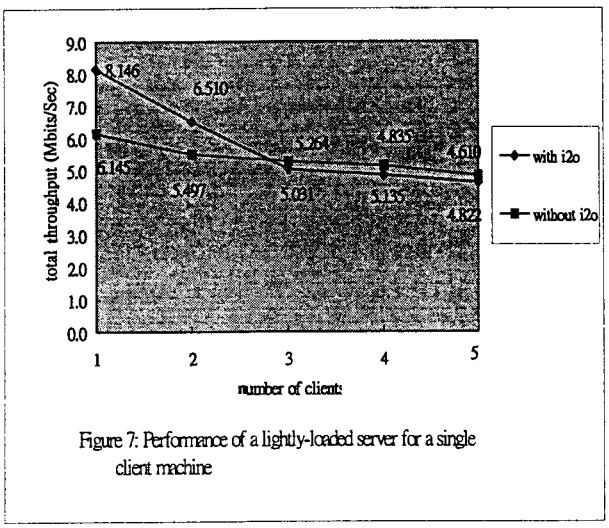


Figure 7: Performance of a lightly-loaded server for a single client machine

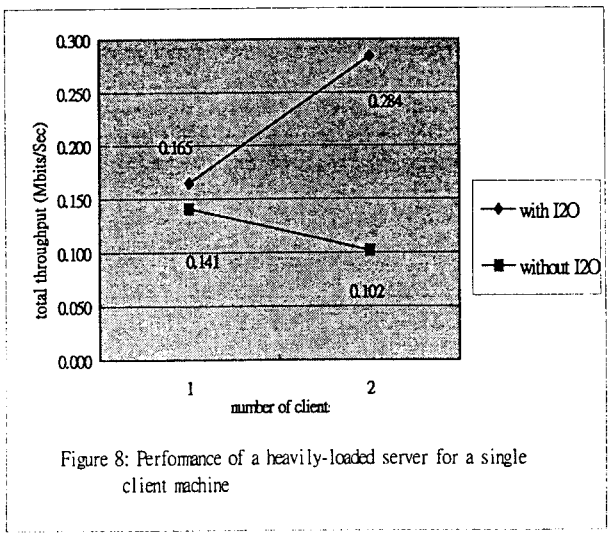


Figure 8: Performance of a heavily-loaded server for a single client machine

5. CONCLUSIONS AND FUTURE WORKS

In this paper, we study the performance of a file server with and without I_2O. We used NetBench to perform the evaluation. From the evaluation results, we find that when the load of the host CPU on the file server is heavy, the system with I_2O obtains a higher throughput. In addition, the number of clients that the file server can handle increases when I_2O is used. This indicates that the I/O processor can effectively reduce the interrupts to the host CPU. In the future, we will conduct more experiments to evaluate the performance of I_2O and identify its bottlenecks. Then we will develop some ISMs to see if these bottlenecks can be eliminated.

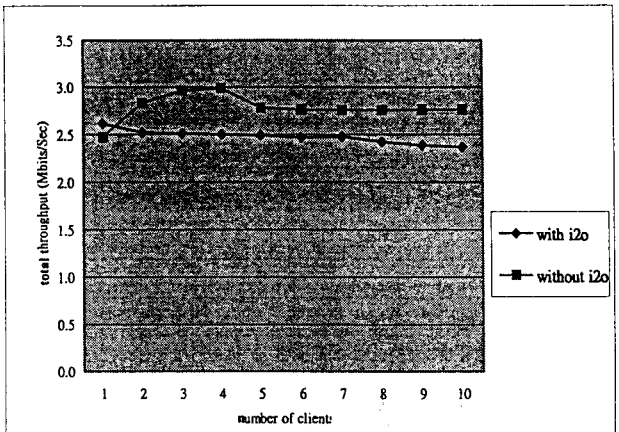


Figure 9: Performance of a lightly-loaded server under random reads by a single client manchine

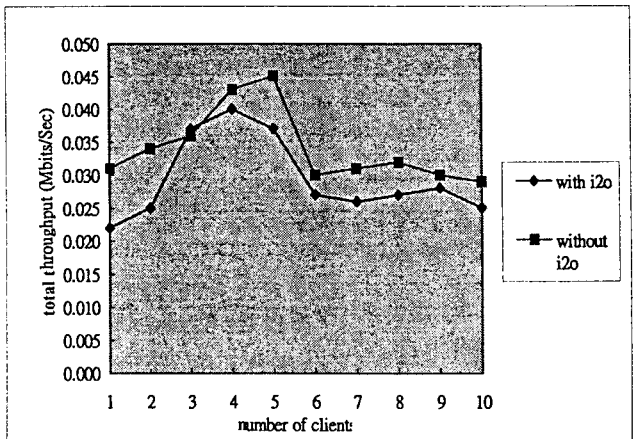


Figure 10: Performance of a heavily-loaded server under random reads by a single client machine

6. REFERENCES

[1] Intel Corp., *Multiprocessor Specification*, version 1.4, July 1995.

[2] S.S. Mukherjee and M.D.Hill, "A Survey of User-Level Network Interface for System Area Networks," *Tech. Report*, University of Wisconsin-Madison.

[3] N.J. Boden and et al, "Myrinet-A Gigabit-per-second Local Area Network," *IEEE Micro*, vol. 15, pp. 29-36, Feb. , 1995.

[4] David H.C. Du and et al, "Performance Study of Emerging Serial Storage Interfaces: Serial Storage Architecture(SSA) and Fibre Channel-Arbitrated Loop(FC-AL)", *Tech. Report*, University of Minnesota.

[5] Yuan-Bin Tsai, "Trends in Cluster Computer System Architectures," CCL. Confidential.

[6] Byon Gillespie and Mark Bronn, "Implementing Intelligent I/O in the PC Cluster Server," *Proc. of the Symposium on High Performance Interconnects*(Hot Interconnects'96), Aug. 15-17. 1996.

[7] SSA Industry Association, *Serial Storage Architecture: A Technology Overview*, version 3.0, 1995.

[8] "I_2O Beats I/O Bottlenecks", *Byte*, Aug. 1997.

[9] *Intelligent I/O Specification,* version 1.5, Mar. 1997.

[10] Byron Gillwspie, "PCI Intelligent I/O Design for High Performance Servers," Intel

[11] NetBench documents

[12] http://www.wrx.com