PIOFS Measurement Data

****

How this Document is organized

****

This file consists of three sections. The first offers a summary of information collected from a series of measurements. It is intended to provide guidance to what performance can be expected from a given configuration. The next one tells where to find information on how to set certain tuning parameters to optimize performance. The third part is the set of measurements that form the basis for the recommendations in the first section.

****

Estimating the Number of Server Nodes you need.

****

The number of server nodes is a function of at least four application/hardware characteristics, plus the tuning parameters discussed below. The four characteristics are: (1) random or sequential processiing; (2) percentage of reads and writes; (3) the number of disks accessed and their speeds; and (4) the record length.

Reads and writes with 64K byte record lengths, or larger, are usually faster than with 4k byte records. Writes are often faster than reads, but can be slower for certain record sizes. Sequential accesses are usually faster than random ones. Two disks, instead of one, improve read speeds, even with a SCSI adapter. Write speeds can also benefit from a second disk, but it requires another SCSI adapter. SSA adapters improve performance with multiple disks until the total bandwidth reaches about 10 Megabytes/sec.

****

PIOFS tuning information.

****

This information is provided ″as is″ without warranty of any kind, either expressed or implied.

****

The observations above are based on GA code and the following data:

With one client and one server on two wide nodes, with two 9334 disks on one SCSI adapter, we measured:

sequential reads from 2 to 4 MB/sec for records from 4K to >16K bytes
sequential writes from 1 to 5 MB/sec for records from 4K to >128K bytes

Random reads from .2 TO 3.3 MB/sec for records from 4K to 256K bytes
(performance approaches that of sequential reads for records > 256k)
Random writes from .4 TO 4.4 MB/sec for records from 4K to 256K bytes
(performance peaks at or near multiples of 63K bytes)

The performance data was measured with a RS/6000 SP with the high- performance switch and no switch traffic except for PIOFS requests.

Similarly, with one client and one server, each on a thin node, with four 7332 disks on a SSA loop, we measured:

sequential reads from 2 to 7 MB/sec for records from 4K to > 256K bytes

sequential writes from 1 to 7 MB/sec for records from 4K to > 256K bytes

With one client and four servers, each on a thin node, where every server has four 7332 disks on a SSA loop, we measured:

sequential reads from 8 to 10 MB/sec for records from 64K to > 256K

sequential writes from 6 to 10 MB/sec for records from 64K to > 256K

One fourth of each record is stored or retrieved from each of the four servers. This data is an indication of performance at a single client node.

With one server and four clients, each on a thin node, where the server has four 7332 disks on a SSA loop, and each client reads and writes to a unique file at the server, we measured

sequential reads at 6 MB/sec for records from 128K to >256K bytes

sequential writes at 5.5 MB/sec for records from 128K to >256K bytes

This data is an indication of performance at a single server node. In all cases your performance on specific jobs will be faster or slower depending on many factors which are not described here, such as other switch traffic. Our read measurements were repeatable within about 6 %; writes varied by about 20%.

****

PIOFS tuning parameters

****

PIOFS uses the user datagram protocol (UDP) to send messages and data between client and server nodes. Several tunable parameters are available which have a direct impact on UDP performance. To get the best possible performance using UDP, it is important to understand the purpose of these tunable parameters so that they can be set correctly for a particular PIOFS configuration. See publication SH34-6065-02 (IBM AIX Parallel I/O File System: Installation, Administration, and Use) and the README file in the install image on how to set them.

Use the piofs.profile file or the PFS_CREATE command in pfsioctl() to create files with a bsu of 256K (262144) or larger.

****

Extending Parallel System Performance with the Parallel I/O File System

****

For certain types of engineering/scientific applications, I/O performance becomes critical in several ways. As processors become faster, it is important that they not be starved for data input. Also, it may be necessary to accommodate files larger than 2 gigabytes.

This paper explores the ways IBM′s Parallel I/O File System (PIOFS) can help applications achieve significant throughput improvement. PIOFS became available for general use in August of 1995.

Introduction

In parallel file systems, a distinction is sometimes made between server nodes and client nodes. Server nodes store the data to be processed; client nodes do most of the calculations. One advantage of parallel systems is that a client node can obtain data from multiple server nodes. The speed of data transmission benefits if there is a high-speed multi-channel switch connecting the nodes.

PIOFS Overview

PIOFS is designed to improve the application disk I/O bandwidth by allowing a client node to retrieve information from several server nodes at once. Data is striped across the server nodes and also across the disks within a node. PIOFS lets the user easily change the way the data is viewed by adjusting the appropriate parameters.(1) Plus, PIOFS allows one to specify files that are greater than 2 gigabytes.

PIOFS Performance with 1 Client and 1 Server

It is convenient to start with a single client and a single server, on separate nodes, as a basis for comparison with larger configurations. The nodes are connected with an IBM High-Performance Switch. Figure 1 shows the elapsed time to read or write a fixed amount of data under various conditions. The file size is a constant 750 Megabytes, while the record size varies from 4k to 256k bytes. Other variables are the type of access (serial or random), and the number of disks (1 or 2 9334s). Both client and server are wide nodes.

The server node actually has more than 2 disks attached, but it only has one adapter. Spreading the data over 3 or 4 disks does not improve performance because the single adapter is the bottleneck. Other configurations, with disks attached via SSA adapters, deliver better throughput, as illustrated later in this paper.

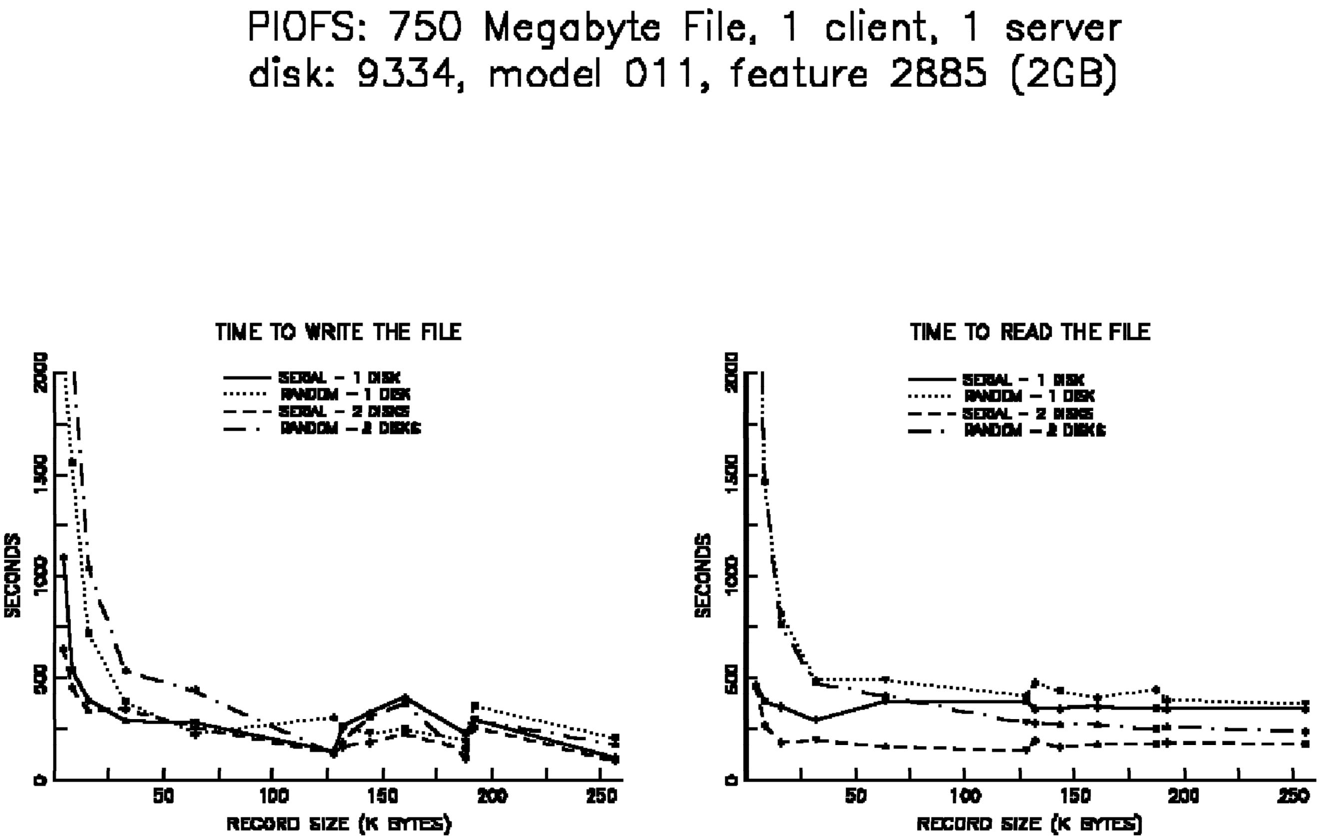

Figure 1.

Figure 1 demonstrates that the time required to process the file with serial accesses shows much less variation than the processing time with random reads or writes. The processing time is a function of the number of buffers processed, so smaller buffer sizes require a larger total of CPU time to move the data. Serial reads or writes can avoid some of this overhead by referring to two user-controlled parameters, minpgahead and maxpgahead. They control how many records will be retrieved with a single disk access.

Random accesses do not benefit from the the look-ahead parameters. However, Figure 1 does show that performance using random accesses approaches serial access rates with increasing record sizes for both reads and writes. It has been our experience that most engineering/scientific applications process data in sequential fashion.

Figure 1 leads to the conclusion that writing a file spread over 2 disks takes about the same amount of time as writing it on a single disk. The data is first cached in server node storage, and written out to disk if replaced by new information. A synchronization daemon makes sure that all updated pages are written out to disk, if they have not been forced out already. Writes to disks in can only occur in parallel if there are two adapters available. Here, there was only one.

The real advantage of 2 disks shows up when reading the file: processing time with 2 disks is approximately half as much as with a single disk. While seeking the starting position on one disk, information can be read from the other one. As we shall see later, the processing time with 4 disks should not be expected to be halved again compared to 2 disks, even if the adapter constraint is reduced.

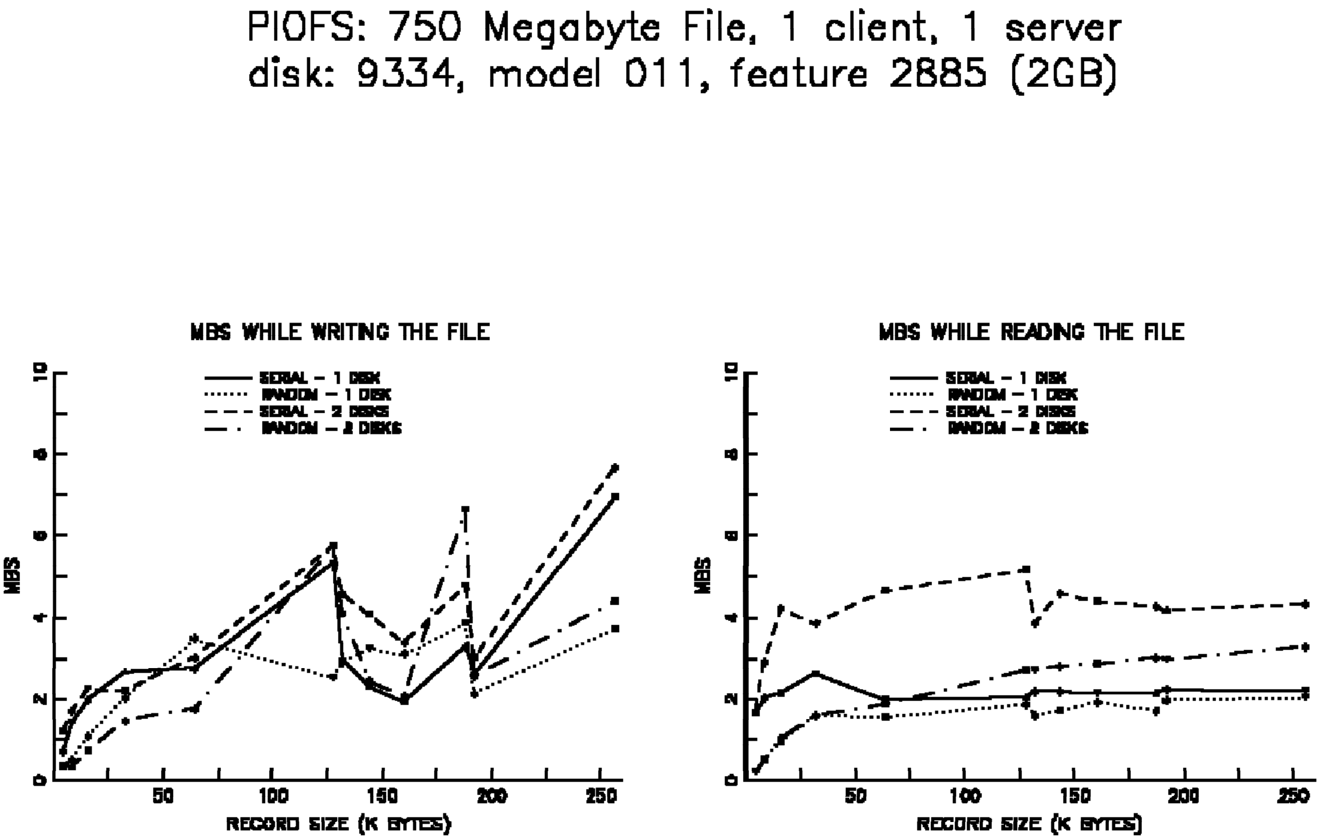

Figure 2.

A more general metric for comparing performance is the rate of moving data, expressed in Megabytes/sec. Figure 2 contrasts the data rates for the same combinations of parameters as Figure 1. Sequential accesses with 1 disk exceed 2 megabytes a second when the access size reaches 16k. With 2 disks, the total rate is about 4 Megabytes/second.

To make the performance measurements conservative, we forced the data to be read from and written to disk. To force writes to disk, we made the measurements last long enough for the synch daemon to write out the data. To make sure the reads access disk, we write over the data in main storage before starting to read the file. This way, when PIOFS looks for the data in cache, it does not find it, and uses the data on disk instead.

We found the reads to be more consistent than the writes. Repeated performance of the experiments showed that read performance is often nearly the same, with an occasional deviation as large as 6 percent. Write performance usually varied within about a 20 percent range. One reason for this behavior is that the file blocks are returned in no particular sequence.

Performance with random accesses varies considerably over the range of record sizes, but it approaches the rates for sequential accesses as the record size increases. If we were to plot the data rates for random accesses with record sizes that are multiples of 64k, we would see fairly smooth curves, but that would be misleading. For other record sizes, there are two separate effects that make the curves less than smooth. One is that record sizes other than multiples of 64k (which is usually equal to one track on disk), require another access. One can see this effect for record sizes of 132k, 138k, etc.

The second effect is caused by how PIOFS transmits the data between client and server. It divides large records into units of 63k bytes and transmits them individually. That gives an advantage to records that are multiples of 63k, or just below that. This effect is seen at 188k, where the next multiple of 63k is 189k. The performance of random accesses at 188k is clearly higher than for other nearby samples.

PIOFS CPU Utilization

To understand PIOFS performance in more detail, it is necessary to introduce some aspects of the software. PIOFS code consists of two layers. There is the PIOFS server code that figures out which disks need to be accessed and where to route the data and the acknowledgements. A second layer is PMP (PIOFS Message Passing), invoked when communication between nodes is required. PMP, in turn, calls the User Datagram Protocol/ Internet Protocol, or UDP/IP, to effect the data transfer.

UDP is a standard communication protocol, more efficient than the Transaction Control Protocol (TCP), but lacking the latter′s guaranteed delivery of messages to their destination. PMP adds back this function. Since PMP and UDP are so closely related, we will consider them as a single layer in this paper.

It is possible for both the client and the server function to reside in a single node. In that case, the UDP/IP code is not used, and PMP is barely referenced. If the client and the server are on separate nodes, PMP is used in both of them, in combination with UDP.

A third layer is the interface between the server code and the disks, Asynchronous I/O Support, or aios. Finally, there is another pathlength component that comes into play periodically. A synchronization daemon wakes up at 1-minute intervals, and checks if there are any pages with updated data that can be written out to disk. If so, the daemon carries out that function and goes back to sleep.

Before looking at each layer separately, Figure 3 shows the total CPU time to process the 750 Mbyte file within the 4 dimensions used earlier. The CPU time is measured with netpmon, an IBM Product Program. It accumulates the CPU time for all the active processes.

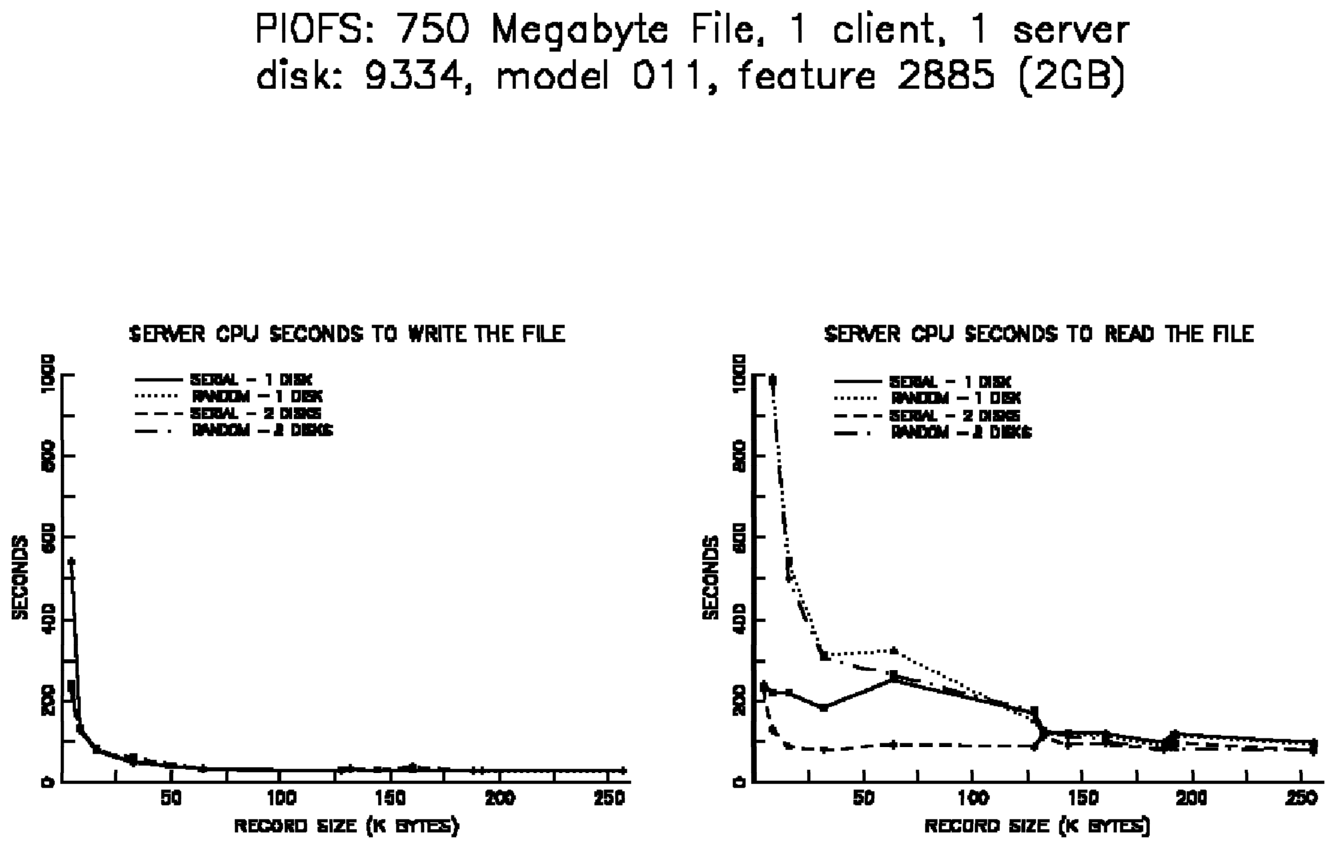

Figure 3.

Figure 3 clearly shows that the amount of work the server must do to service a write request is nearly the same, regardless of whether the access is sequential or random, or whether 1 or 2 disks are involved. For reads, the CPU overhead is much higher for random accesses with short records, but gradually approaches CPU times with sequential accesses as the record size increases.

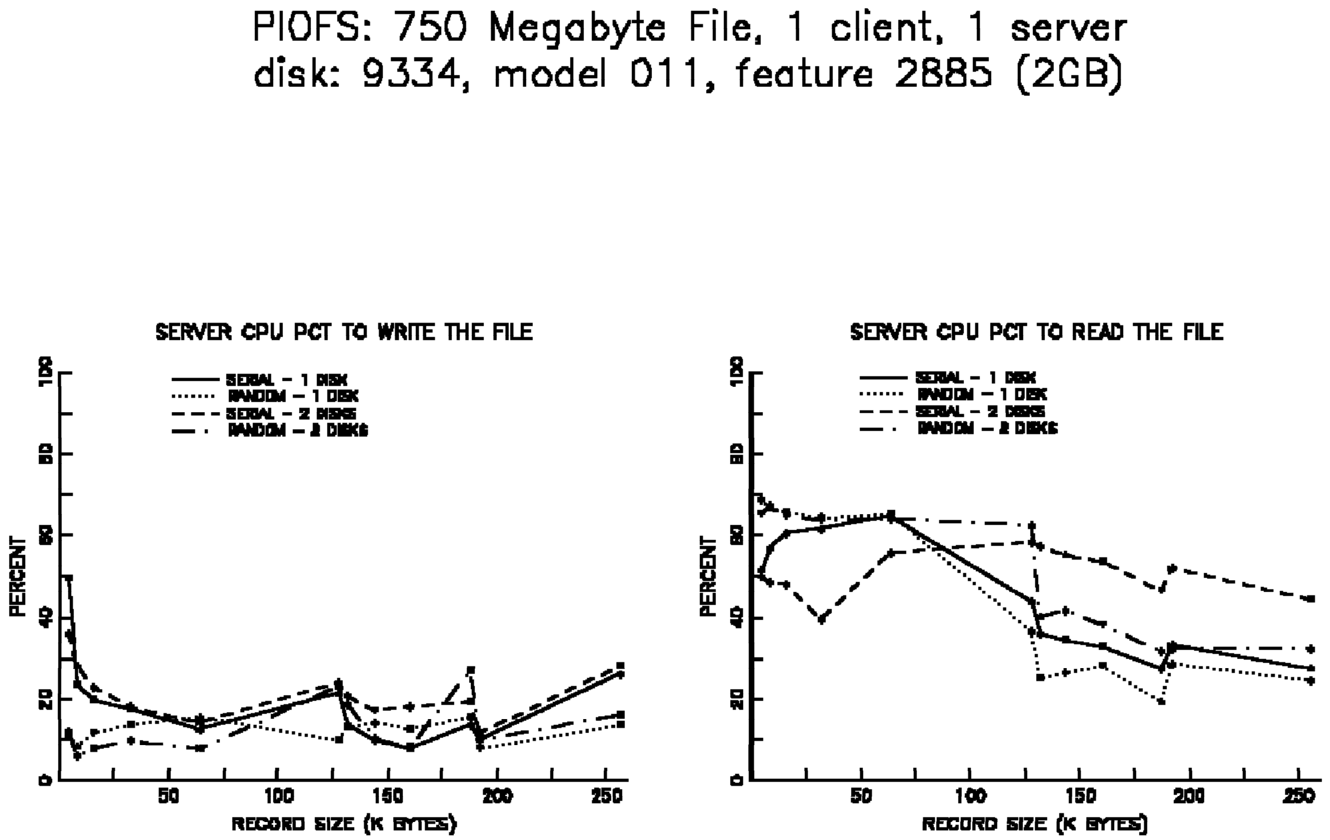

Figure 4.

It is a simple matter to divide the total CPU time in Figure 3 by the elapsed time in Figure 1, to give the CPU utilization in Figure 4. As one would expect, the CPU utilization for writes is lower than for reads. Although the server CPU utilization fluctuates, we know from Figure 3 that the CPU seconds only vary with record size for writes.

Since utilization is the ratio of CPU time to elapsed time, and the numerator is constant over much of the range of record sizes, the source of variability must be in the denominator, the elapsed time. Thus, the variation in CPU utilization must be because of factors that affect the elapsed time, such as disk arm movement, or transmission time over the switch.

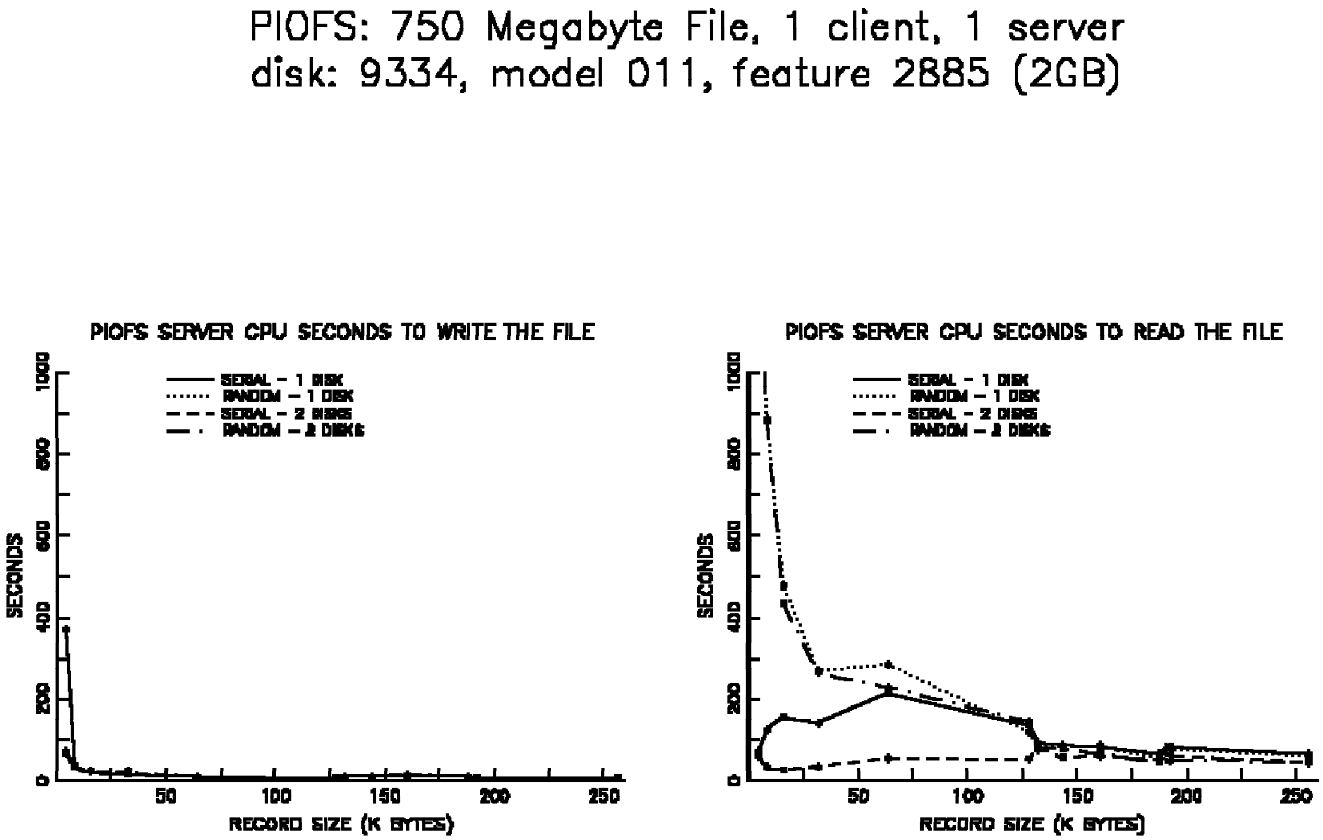

Figure 5.

PIOFS Server Layer

The largest contribution to total CPU time to read records is usually from the PIOFS Server layer, as displayed in Figure 5. Comparing Figure 5 to Figure 3 (the total CPU time), clearly this layer does most of the processing to read records. While writing records, on the other hand, CPU time is more evenly divided between the Server and PMP layers.

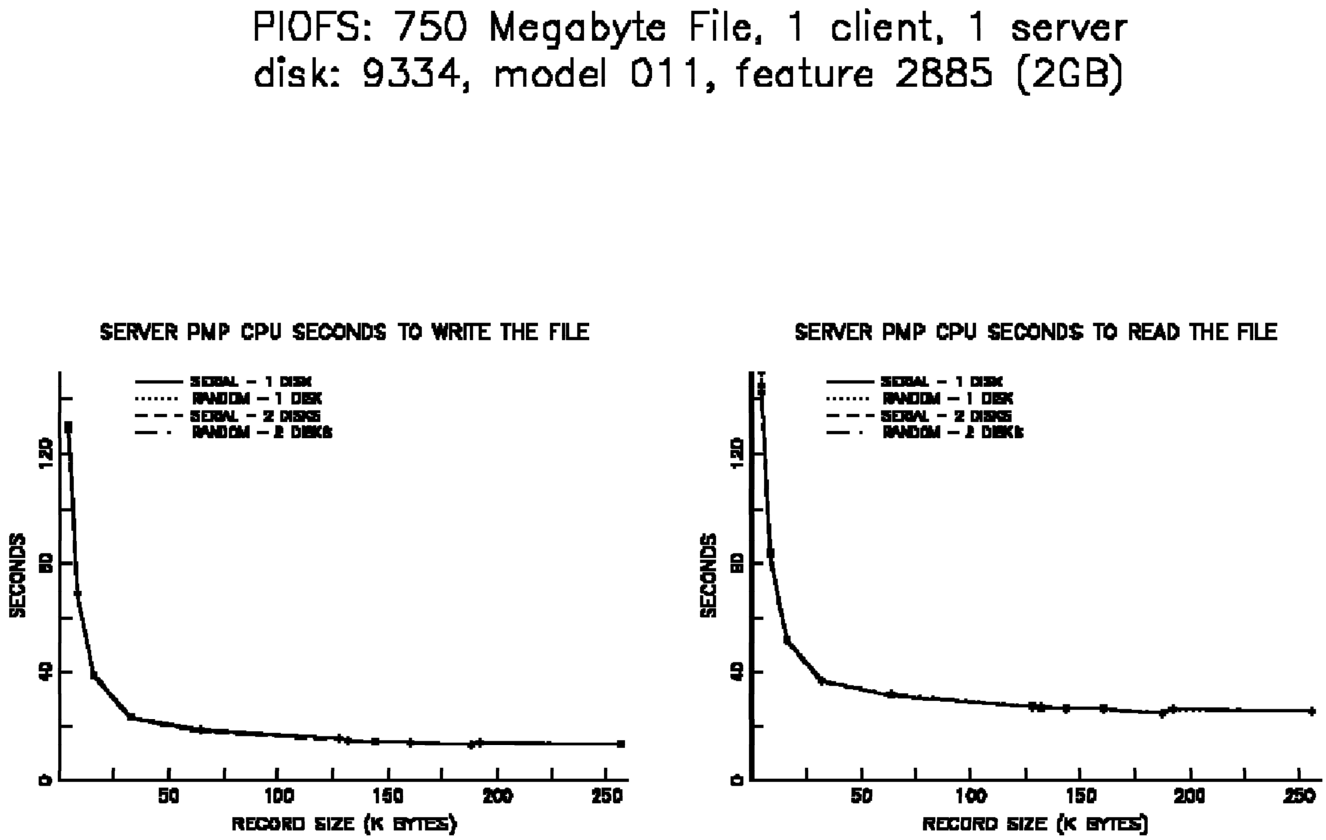

Figure 6.

PMP Layer

The PIOFS Message Passing layer (including UDP/IP) needs about twice as much CPU time to send data as it does to receive it. The measured CPU times are shown in Figure 6. Since the server node sends out the data after it has been read, it is the CPU time while reading the file that is the higher of the two.

The client node needs the same amount of time as the server node to send or receive data. Since the client is receiving when the server is sending, the corresponding graph for the client node is the same as Figure 6 with the labels for reading and writing interchanged.

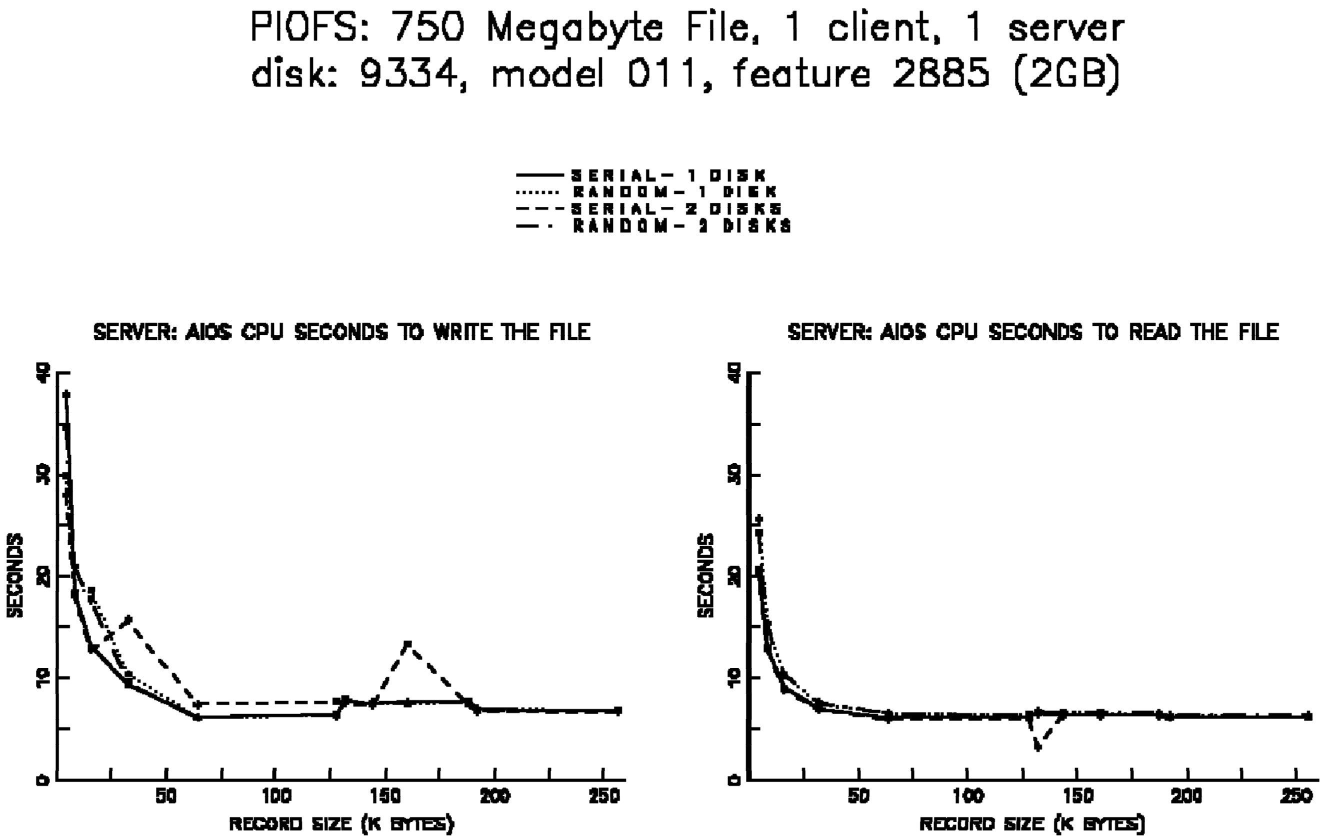

Figure 7.

AIOS Layer

The aios layer is the interface between the Server code and disk. Figure 7 shows that its contribution to the total CPU time is small. When a file is written, the data is initially cached at the server node, until overwritten by new data. Also, the synch daemon periodically updates the physical disk. When reading a file, on the other hand, an initial attempt is made to retrieve the data from cache. If unsuccessful, the data is read in from disk.

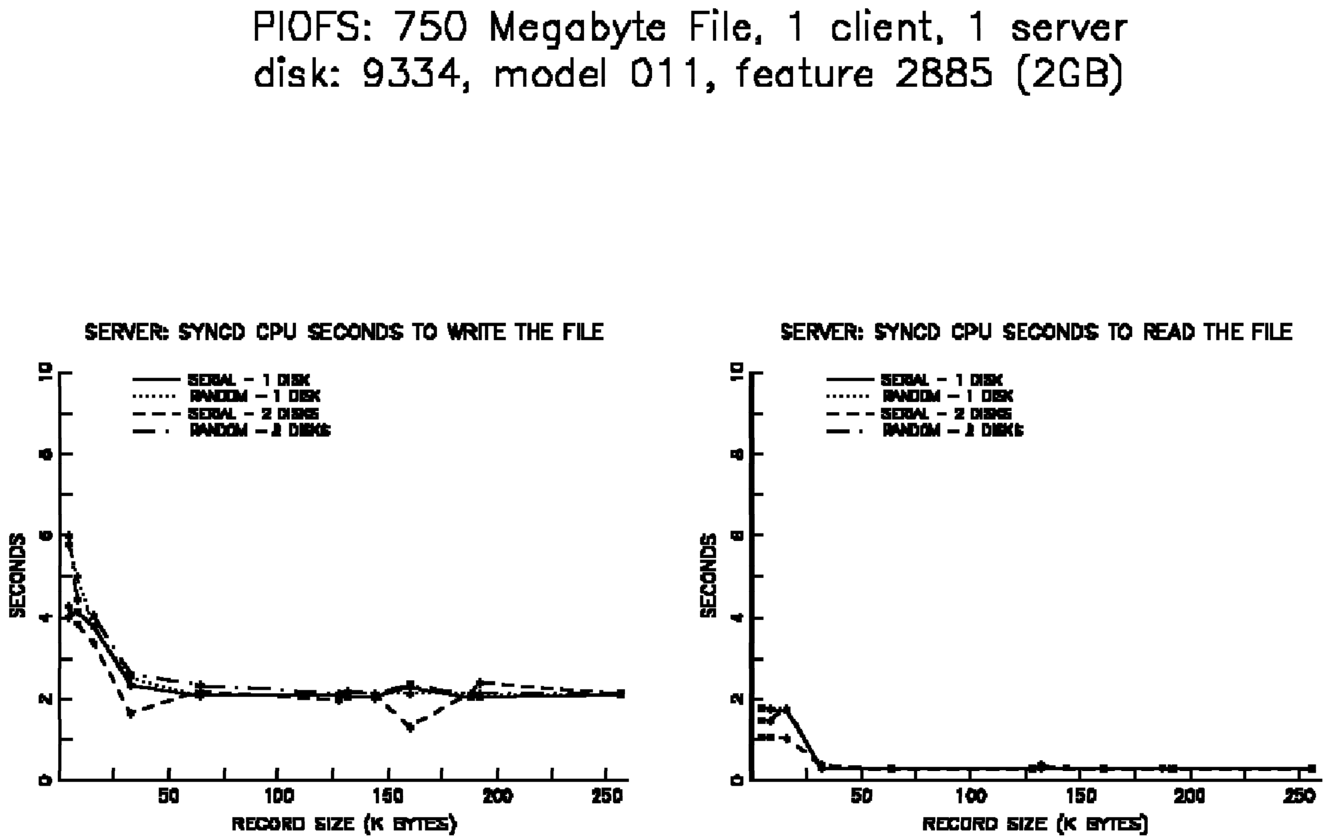

Figure 8.

Synch Daemon

There is another contributor to the total CPU overhead that cannot be classified as a layer. The synch daemon wakes up at one-minute intervals. It checks if there are any dirty pages to be cached on the server node. If there are, it does that function and goes back to sleep. The CPU time to do so is shown in Figure 8.

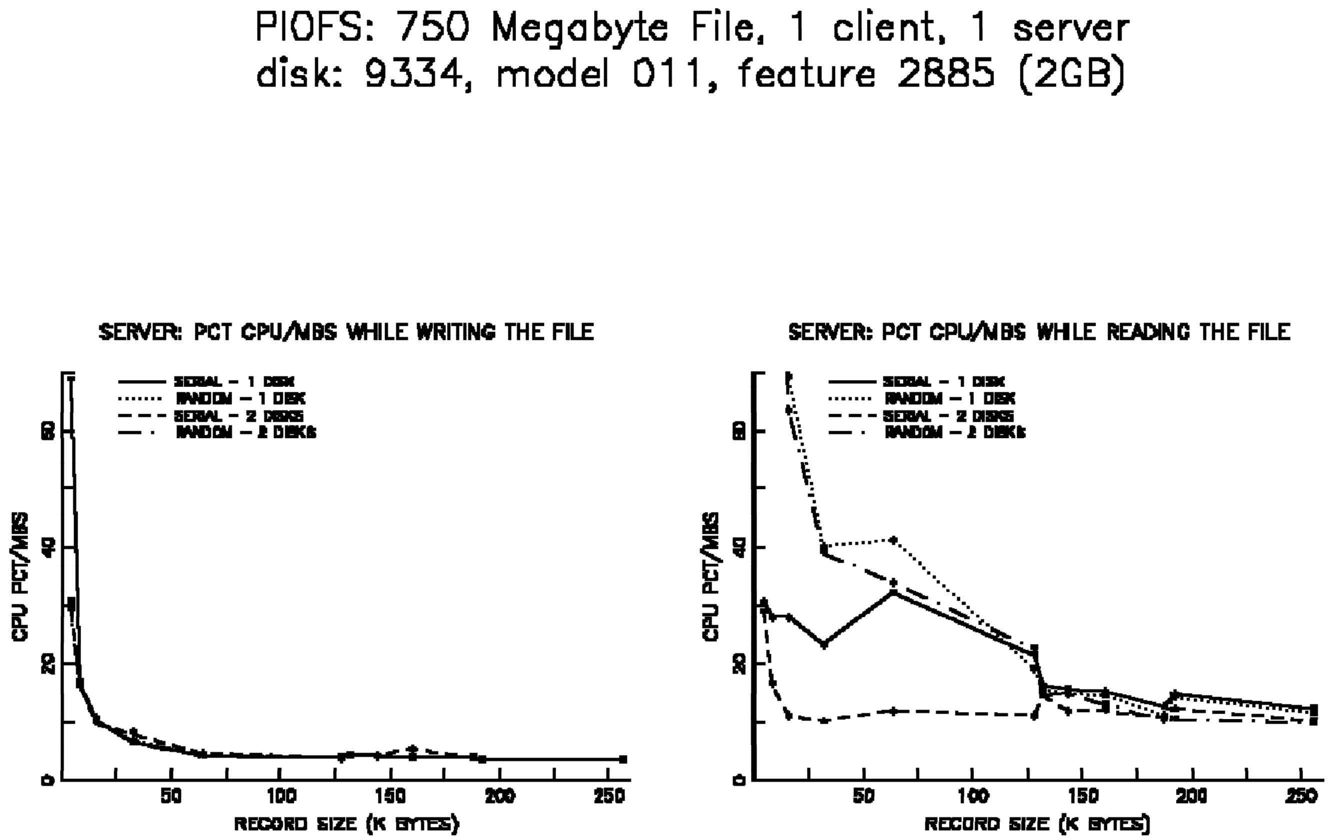

Figure 9.

Input Data to Performance Analysis Models

Once all the measurement data has been collected, the question becomes how to make the best use of it. In the performance community, there are several standard metrics in use. One is the CPU utilization per Megabyte/sec. This parameter is displayed in Figure 9.

It shows that PIOFS requires approximately 5% cpu utilization per Megabyte/sec to write a file over a wide range of operating conditions. The requirements are higher for small record sizes, especially with random accesses. To read a file, PIOFS needs at least 10% per Megabyte/sec (with 2 disks).

If we assume that CPU utilization can reach about 80%, then we expect that a server node can produce about 8 Megabytes/sec for reads, and absorb about 16 Megabytes/sec for writes. Reading a file from a single disk shows higher CPU utilization because of time spent polling the disk to monitor the data transfer from the device to main storage. Later, the write speed estimate is shown to be optimistic compared to actual measurements with 1 client and multiple servers.

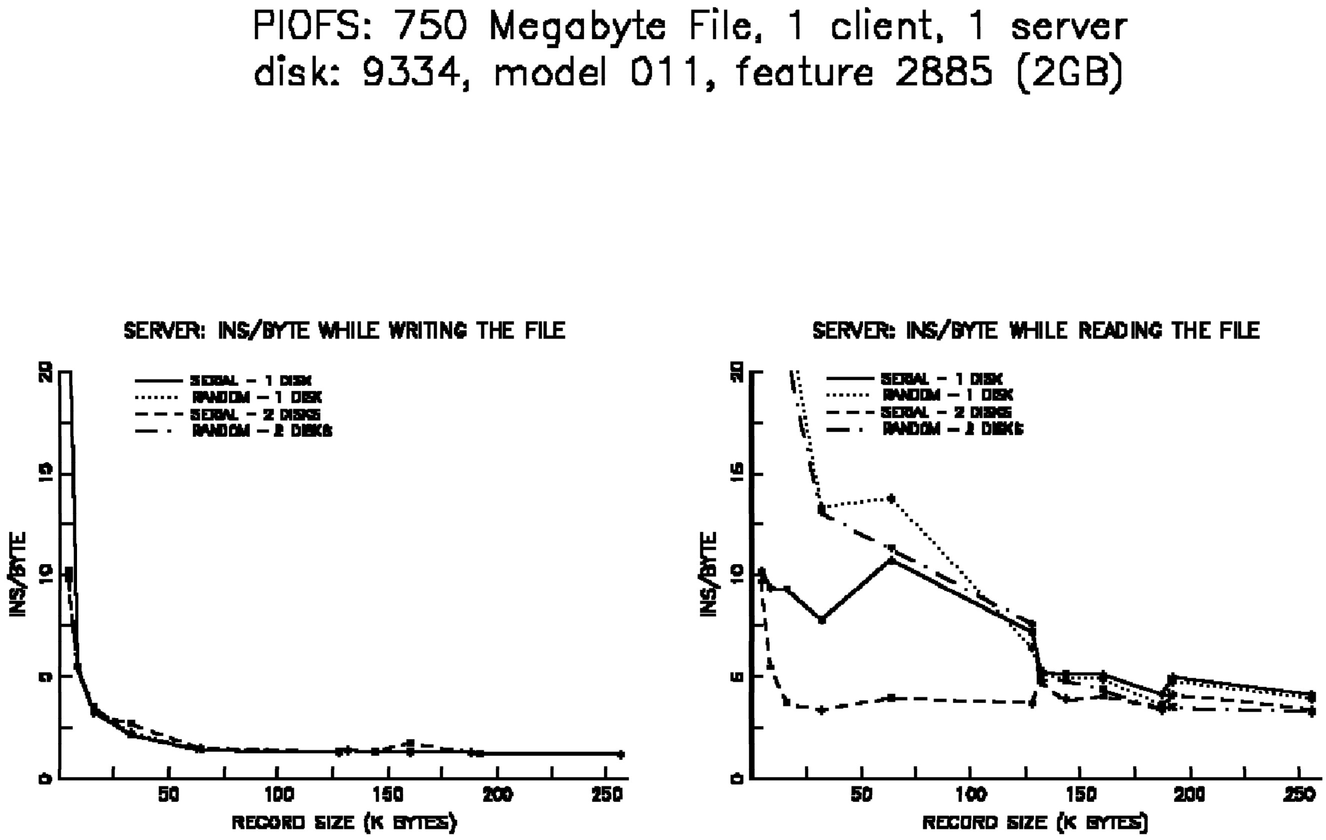

Figure 10.

Another common metric is instructions/byte. This one is requires an assumption (or a measurement), of cycles per instruction. If we assume 2 cycles per instruction, Figure 10 shows how this parameter varies over the range of measurements. The similarity between Figures 9 and 10 is easily explained. We can convert from one to the other by using simple conversion ratios. In fact,

$$1 \text{ ins/byte} \times 2 \text{ cyle/ins} \times (15\times10^{-9} \text{ sec/cycle}) \times 10^{6} \text{ bytes/Mbyte}$$

$$= 3 \text{ \% cpu/Megabytes/sec}$$

The conversion ratio is directly proportional to both the cycle time and the cycles per instruction.

One Client, Multiple Servers

Now consider what happens when a single client is allowed to use multiple servers. This set of measurements was obtained on a configuration different from the one used for a single client and a single server. It contains 16 thin nodes, each with 256 Mbytes of main storage.

Also, the server nodes have disks connected via a SSA loop, rather than a SCSI adapter. Since the SSA loop can sustain about 18 Mbytes a sec, it allows better use of the disk bandwidth. The disks themselves are 7332s, rather than 9334s.

The test case parameters allow us to specify that the file must be equally divided among the server nodes. We can specify the striping per node with the Basic Striping Unit, or BSU. To divide each record equally among the servers, we calculate the BSU by dividing the record size by the number of servers. With that assumption, the total throughput, as seen by the client, is shown in Figure 11.

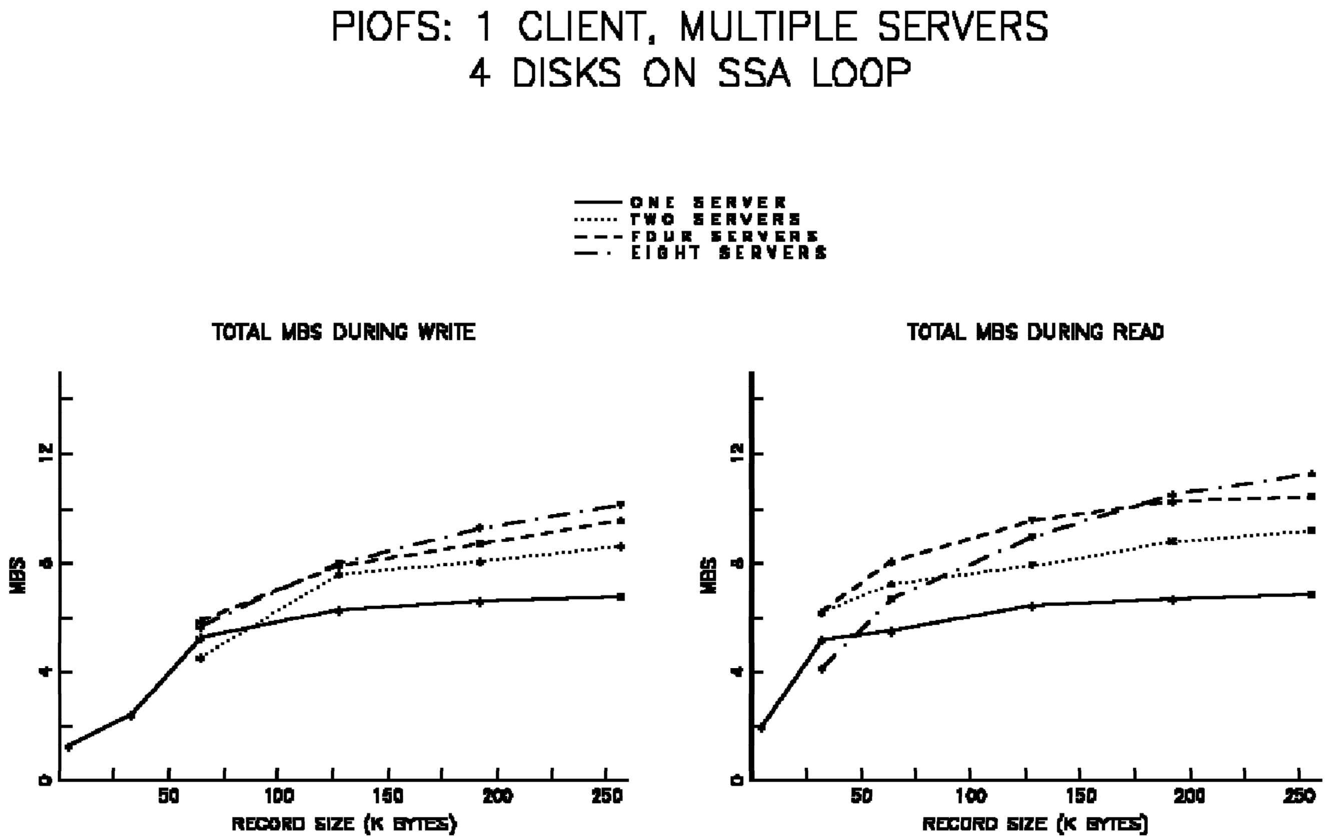

Figure 11.

The combination of a single client and single server can now attain almost 7 Megabytes/sec. With 2 servers, the maximum bandwidth is about 9 Megabytes/sec. Increasing the number of servers to 4 and 8 gives diminishing returns. Doubling the number of servers to 4 gives only about one more Megabyte/sec. Doubling the servers again, to 8, improves the maximum read bandwidth by almost another Megabyte/sec, but the write bandwidth only gains about .6 Megabyte/sec.

On the read side, we even see a decrease in performance at the low end of the record size scale. That reduction occurs because the effective record size per server has decreased.

One Server, Multiple Clients

Now consider what happens when more and more clients are sharing a single server. This time, we are not striping across multiple servers, so the BSU size does not matter. Since engineering/scientific applications rarely share data, we chose to have each client access a separate file. The results are plotted in Figure 12.

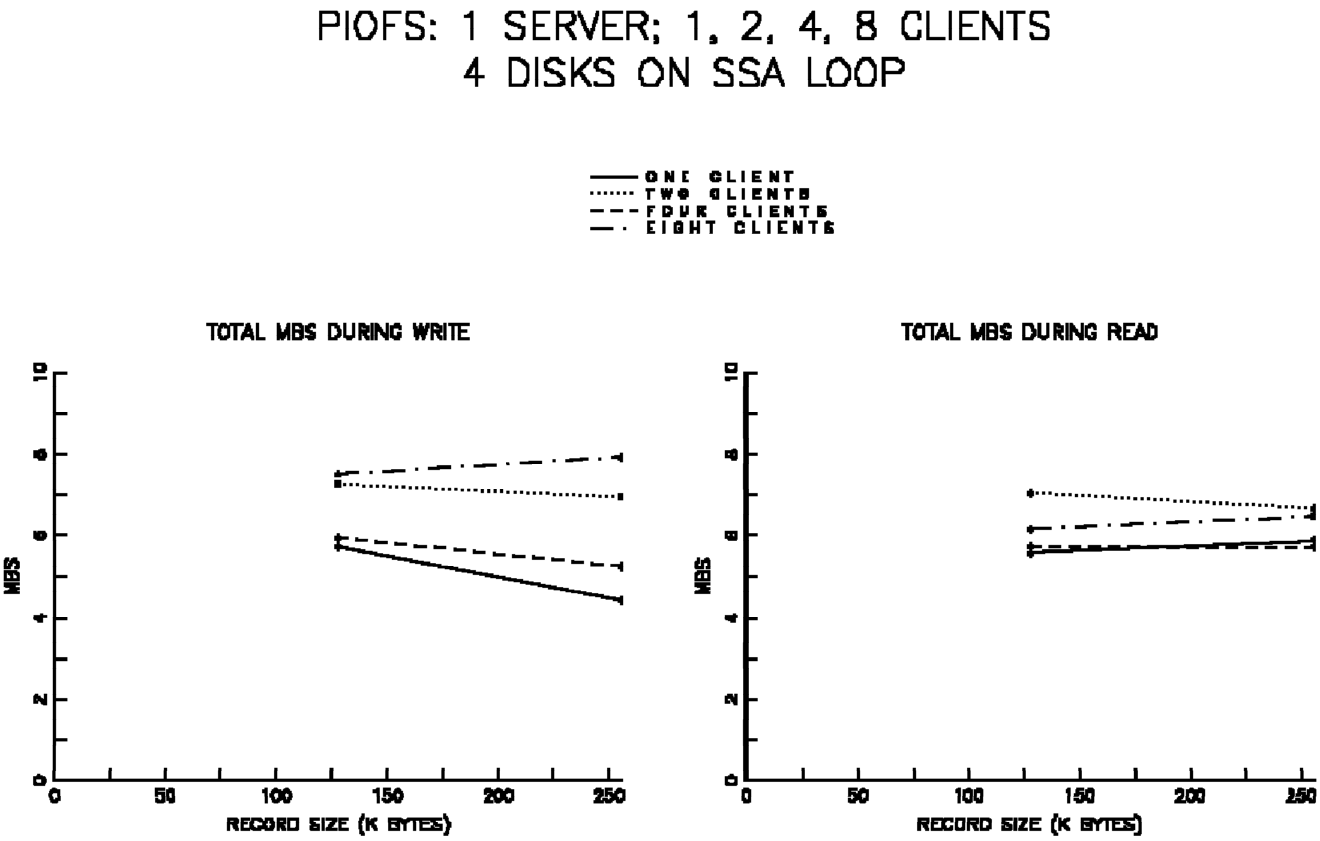

Figure 12.

There is a noticeable increase in throughput in going from 1 to two clients. However, for 4 clients, the total bandwidth is closer to a single client's. Then, with 8 clients, the total throughput improves again. Note that the best performance is close to the 8 Mbytes/sec derived from CPU utilization.

Multiple Clients, Multiple Servers

Finally, we present the performance of configurations where the number of clients matches the number of servers. This is where we see high bandwidths occurring: up to 49 Megabytes/sec for reads, and up to 39 Megabytes/sec for writes, as illustrated in Figure 13.

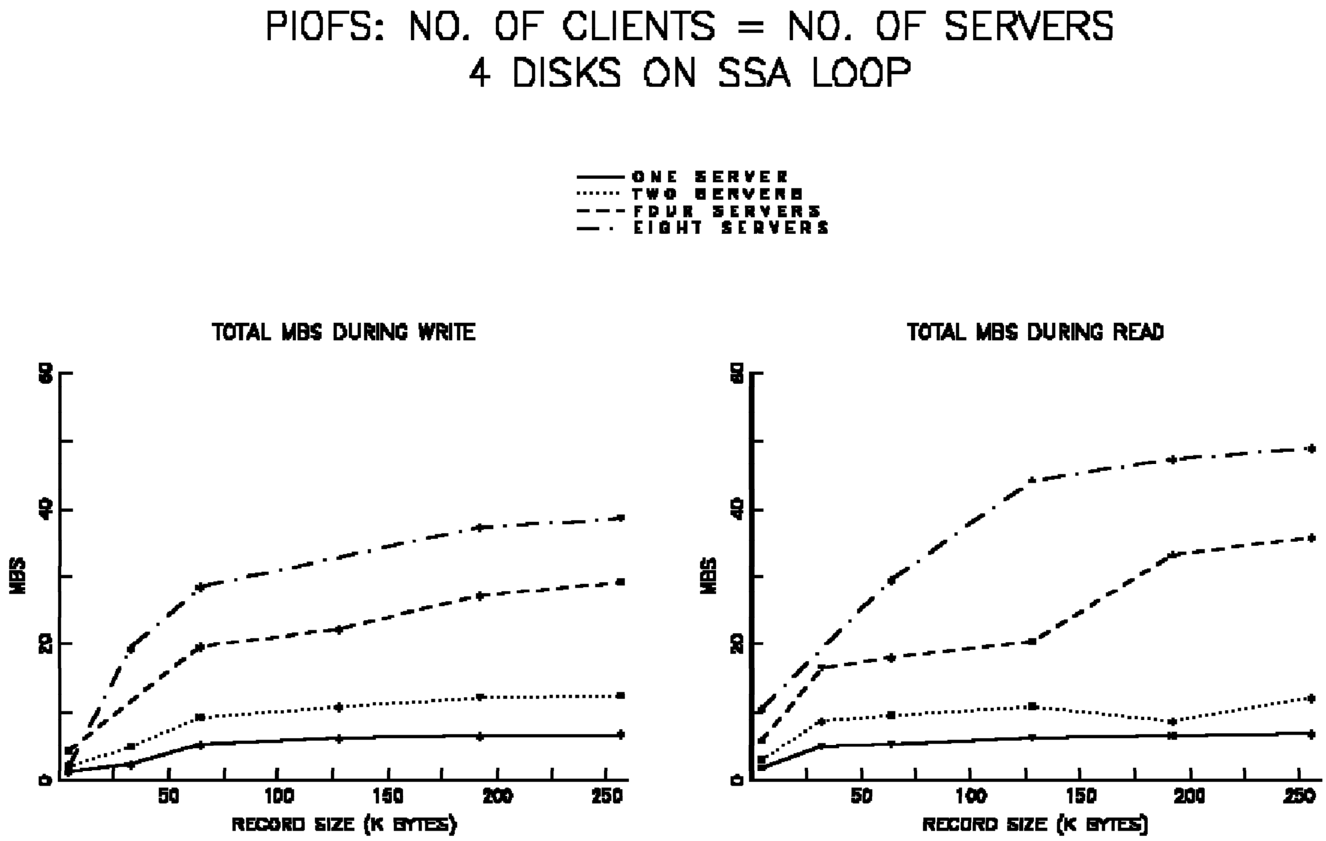

Figure 13.

Tuning Applications with PIOFS

The results clearly show that the adapter connecting the disks to the server can be a bottleneck. It is preferable to have no more than 2 disks per SCSI adapter. A good solution is to have the disks on a SSA loop. Still, it seems that more than 4 disks per server will not help performance much. It is also obvious that sequential accesses give better performance than random ones, especially for small access sizes.

Increasing the number of clients for a single server increases total read bandwidth to about 7 Megabytes/sec. From Figure 9, we expect the upper limit to be 10 Megabytes/sec at 100% CPU utilization. Similarly, increasing the number of servers available to a single client sees diminishing returns, but reaches as high as 11 to 12 Megabytes/sec. We expect the total throughput to be higher now because the PIOFS code running on the client is much less CPU intensive than that on the server. The client code consists of application code plus PMP, while the server runs PMP, the PIOFS Server code, and aios. To achieve high throughputs, both the number of clients and the number of servers must increase at the same time.

Conclusion

Usually, the best performance will be achieved with sequential accesses to servers with enough disk capacity to keep them busy. For instance, our measurements showed that 4 7332 disks on an SSA loop, gives close to optimum performance. The disks should be configured with either 2 SCSI adapters or a SSA loop. The total bandwidth will then be mainly a function of the number of servers.

Reference

(1) P. F. Corbett, D. G. Feitelson, J-P. Prost, G. S. Almasi, S. J. Baylor, A. S. Bolmarich, Y. Hsu, J. Satran, M. Snir, R. Colao, B. Herr, J. Kavaky, T. Morgan and A. Zlotek, Parallel File Systems for the IBM SP Computer, IBM Syst. J., Vol. 34, No. 2, pp. 222-248, 1995.

DSMBEG323I STARTING PASS 2 OF 2.
DSMAFP709E PAGE SEGMENT TIM2L EXCEEDS RIGHT PAGE BOUNDARY ON PA
GE 4.
DSMAFP709E PAGE SEGMENT MBS2 EXCEEDS RIGHT PAGE BOUNDARY ON PAG
E 5.
DSMAFP709E PAGE SEGMENT CPUSEC2 EXCEEDS RIGHT PAGE BOUNDARY ON P
AGE 7.
DSMAFP709E PAGE SEGMENT CPUPCT2 EXCEEDS RIGHT PAGE BOUNDARY ON
PAGE 8.
DSMAFP709E PAGE SEGMENT PFSPL2 EXCEEDS RIGHT PAGE BOUNDARY ON PA
GE 9.
DSMAFP709E PAGE SEGMENT PMPPL2 EXCEEDS RIGHT PAGE BOUNDARY ON P
AGE 10.
DSMAFP709E PAGE SEGMENT AIOPL2 EXCEEDS RIGHT PAGE BOUNDARY ON PA
GE 11.
DSMAFP709E PAGE SEGMENT SYNPL2 EXCEEDS RIGHT PAGE BOUNDARY ON P
AGE 12.
DSMAFP709E PAGE SEGMENT CPUMBS2 EXCEEDS RIGHT PAGE BOUNDARY ON
PAGE 13.
DSMAFP709E PAGE SEGMENT INSBYTE2 EXCEEDS RIGHT PAGE BOUNDARY ON
PAGE 14.
DSMAFP709E PAGE SEGMENT ONETON EXCEEDS RIGHT PAGE BOUNDARY ON P
AGE 16.
DSMAFP709E PAGE SEGMENT S1MULTC EXCEEDS RIGHT PAGE BOUNDARY ON
PAGE 17.
DSMAFP709E PAGE SEGMENT CLEQSRV EXCEEDS RIGHT PAGE BOUNDARY ON
PAGE 18.