A Study of End-User SCSI Disk Performance with Parallel and Non-parallel Files

Robert D. Russell

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

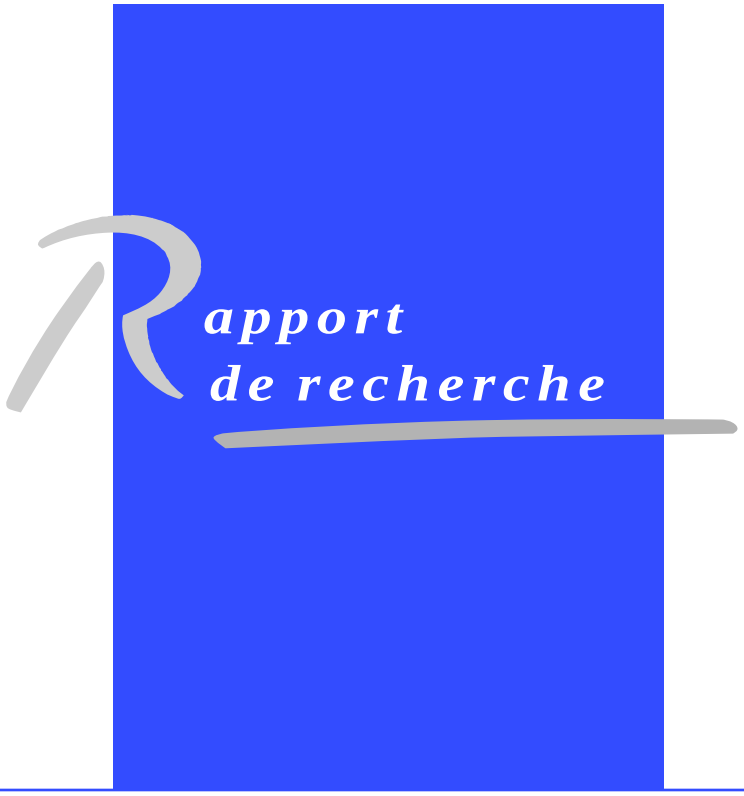
A Study of End-User SCSI Disk Performance with Parallel and Non-parallel Files

Robert D. Russell

No 3494

September 1998

THÈME 1



ISSN 0249-6399

RHÔNE-ALPES

A Study of End-User SCSI Disk Performance with Parallel and Non-parallel Files

Robert D. Russell *

Thème 1 — Réseaux et systèmes
Projet ReMaP

Rapport de recherche n ° 3494 — September 1998 — 28 pages

Abstract: This report describes a series of experiments to investigate the performance of SCSI disks as seen by the application programmer. The results show a significant performance loss, on the order of 50%, when two or more files are simultaneously read from the same disk. Performance is also reduced when files are stored in an interleaved manner, and when small block sizes are utilized.

Key-words: SCSI disk performance. File system performance. Parallel file system performance.

(Résumé : tsvp)

* On leave during the 1997-98 academic year as an Associated Professor at the Laboratoire de l'Informatique du Parallélisme, École normale supérieure de Lyon. Permanent address: Department of Computer Science, Kingsbury Hall, University of New Hampshire, Durham, NH 03824-3591, USA. Email: rdr@unh.edu

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN (France)
Téléphone : 04 76 61 52 00 - International: +33 4 76 61 52 00
Télécopie : 04 76 61 52 52 - International: +33 4 76 61 52 52

Une étude des performances des disques SCSI observées pour des fichiers paralléles et séquentiels

Résumé : Ce rapport décrit une série d'expériences réalisées pour étudier les performances des disques SCSI telles que l'on peut les observer au niveau de l'interface de programmation. Les résultats indiquent une réduction significative des performances, de l'ordre de 50%, lorsqu'au moins deux fichiers sont lus simultanément à partir du même disque. Les performances sont aussi réduites lorsque des fichiers sont stockés de façon entrelacée et lorsque des blocs de petite taille sont utilisés.

Mots-clé : Performances des disques SCSI. Performances des systèmes de fichiers. Performances des systèmes de fichiers parallèles.

1 Introduction

This report describes a series of experiments to investigate the performance of SCSI disks as seen by the application programmer. These experiments include measurements of I/O performance in both sequential and parallel programs. They include measurements of the performance of BPFS, a basic parallel file system [4] that utilizes SCSI disks.

In this report, the following common, although inconsistant, conventions are used when refering to units. When refering to a unit of storage, such as the size of a block or the size of a file, "Kilobyte" means 1024 bytes and is abbreviated as "KB", "Megabyte" means 1,048,576 bytes (i.e., 1024×1024) and is abbreviated as "MB", and "Gigabyte" means 1,073,741,824 bytes (i.e., $1024 \times 1024 \times 1024$) and is abbreviated as "GB". However, when refering to a rate, such as the bandwidth of a network or disk, then "Kilobyte per second" means 1000 bytes per second and is abbreviated as "KBps", "Megabyte per second" means 1,000,000 bytes per second and is abbreviated as "MBps", and "Gigabyte per second" means 1,000,000,000 bytes per second and is abbreviated "GBps". If the rate is in bits instead of bytes, "b" is used instead of "B" in the abbreviations (e.g. 1 Gbps).

1.1 Platforms

The following two platforms were available for these experiments.

- PoPC, a cluster of 12 PCs running the Linux 2.0 operating system. The hardware of each PC consists of a 200 MHz Intel Pentium Pro processor, 64 MB of RAM and one 4 GB SCSI disk (SEAGATE ST52160N Rev 0285). The PCs are interconnected by a 1 Gbps Myrinet. The networking software is TCP/IP on top of BIP [2].
- a SUN i86pc with 64 MB of RAM and one 4 GB SCSI disk running the SunOS 5.6 (Solaris) operating system.

Both platforms were used for the experiments conducted with sequential programs accessing sequential files, as reported in Section 2. Only the PoPC platform was used for the remaining experiments, since these required the use of a parallel file system and PoPC was the only platform on which the communications network was reasonably fast and could be isolated during the tests.

1.2 File Caching

In all the measurements taken in this report, the size of the file read or written is an important parameter, because today's operating systems automatically cache file system I/O. Consequently, programs that repeatedly read the same small file, or even a sequence of programs that read the same small file, will find that they seem to get very good performance from the file system because they are actually getting their data from memory cache rather than from disk. Clearly such programs are not measuring disk performance but rather cache performance. Furthermore, the operating systems used for the present studies provide no method by which the user can turn off this automatic caching. One way to defeat it is to utilize files that are too big to fit in the cache. This forces the file system to go to disk for at least some of the data, and if the file is always read completely from beginning to end before it is reread, current caching systems will be forced to always go to disk for all the data (i.e., they are not smart enough to realize that they could keep a fixed part (i.e., the first x bytes) of the file always in memory and go to disk only for the other part).

To use this approach, since all machines have 64 MB of RAM, files have to contain at least 64 MB of data to defeat the automatic caching effect. For parallel files, this means that each component of the file that is stored on a separate node has to contain at least 64 MB of data. For example, the minimum acceptable size for a file distributed across eight server nodes is 512 MB. Similarily, if the application is dealing with several files simultaneously, the above considerations apply to the aggregate size of the files, not each file individually, provided that one iteration is completely finished for all files before the next iteration is started for any files. For example, if an application is reading eight files simultaneously, each file must contain at least 8 MB for the total to be larger than the memory available for caching, and all eight files must be completely read on one iteration before the next iteration starts to read any of the same files.

Another way to defeat the automatic caching is to utilize sequences of files such that the total size of all files in the sequence exceeds the maximum possible cache size (64 MB). This approach also requires that files always be accessed in the same sequence, and that no file be accessed a second time before all files are completely accessed the first time.

A final step was also necessary when files were written in order to ensure that we were measuring disk accesses and not just cache access. This was to flush the cache when the program had finished writing. This was done using the UNIX "fdatasync" function (or "fsync" on those systems where "fdatasync" was not available), which delays until the system has forced all data previously written to an open file to be transferred to the associated storage device.

1.3 Methodology

When these tests were run (usually overnight), no other user-level processes were running on the platform. In spite of this, some perturbations in measured times were observed. It is believed these were caused by activation of daemon processes in response to network activity.

Times were measured using the unix function "gettimeofday", which is capable of reporting times accurate to one microsecond if the hardware clock and operating system software support it. The difference between a start time and a stop time was reported as the "elapsed" time. When files were written, the stop time was not recorded until the cache had been flushed. Tests were in general long enough so that the elapsed time was at least several seconds.

The results of a test are always expressed as a rate in terms of total megabytes per second (MBps). For one process, this result is computed as the total number of megabytes of data written to or read from all files accessed by that process, divided by the elapsed time. For more than one process, the total result is computed as the sum of the results for all the processes involved.

Each test was repeated some large number of times (usually fifty). The average and standard deviation of the rates was computed, and then a second computation of the average was made utilizing only those rates within plus or minus two standard deviations of the first average. This precaution tended to eliminate extreme rates that may have been caused by factors external to the process being measured (e.g. daemons activated by network activity, or unexpected logins).

The results are plotted in a standard manner, with the total rate in MBps on the y-axis. The x-axis is the independent parameter that was varied during the test. This is typically the file size, the block size, the number of files, the number of client processes, the number of servers, etc. Points are plotted as the average rate with an error bar indicating plus or minus two standard deviations around that point. The averages at all points are connected to form a line.

1.4 Organization

This report is organized as follows. Section 2 investigates the performance of sequential programs reading and writing sequential files stored on a single SCSI disk. Section 3 investigates the performance of sequential client programs that use BPFS to read and write parallel files stored on multiple SCSI disks. Section 4 investigates the performance of parallel programs with multiple clients using BPFS to read and write parallel files stored on multiple SCSI disks. Section 5 summarizes the conclusions which can be drawn from the various tests.

2 Accessing Sequential Files from Sequential Programs

2.1 Establishing a Baseline

The first tests determine the "baseline" performance that can be achieved by a single user-level process reading and writing a single file on a single SCSI disk.

2.1.1 Tests on an Intel Pentium Pro with Linux

The first test is a program that simply writes a large file to the SCSI disk using the standard UNIX "write" system call. Each write operation produced 1 MB of data. The results are shown in Figure 1. For 128 MB

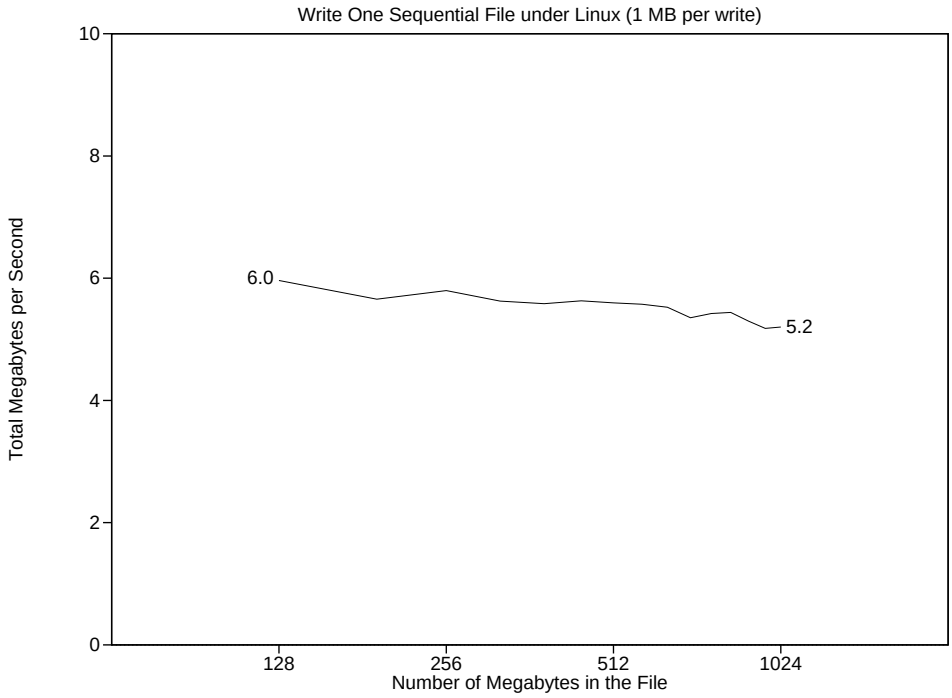


Figure 1: Total MBps for one process writing one sequential file under Linux on a Pentium Pro.

files the average rate is 6.0 MBps. As the file size increases (in multiples of 64 MB), the rate tapers off smoothly to only 5.2 MBps for a 1 GB file. The average rate for the 15 measurements is 5.5 MBps.

The second test is a program that simply reads the large file written by the first test. The standard UNIX "read" system call was used to perform the input operation. Each read operation requested 1 MB of data. The results are shown in Figure 2. For 128 MB files the average rate is 6.7 MBps. As the file size increases (in multiples of 64 MB), the rate tapers off slightly to 6.3 MBps for a 1 GB file. The average rate for the 15 measurements is 6.6 MBps.

2.1.2 Tests on a SUN i86pc with Solaris

In order to confirm that the tests on an Intel Pentium Pro running Linux were typical of SCSI disk performance in general, we repeated them on a SUN i86pc running the Solaris operating system.

The first test again writes a large file to the SCSI disk using the standard UNIX "write" system call, each write operation producing 1 MB of data. The results are shown in Figure 3. For 128 MB files the average rate is 7.5 MBps. As the file size increases (in multiples of 64 MB), the rate tapers off to 6.5 MBps for a 512 MB file, then rises slowly back to 7.3 MBps for a 1 GB file. The average rate for the 15 measurements is 7.0 MBps.

The second test reads the large file written by the first test. The standard UNIX "read" system call was used to perform the input operation, each read operation requesting 1 MB of data. The results are shown in Figure 4. For 128 MB files the average rate is 7.5 MBps. As the file size increases (in multiples of 64 MB), the rate tapers off to 6.4 MBps for a 512 MB file, then rises slowly back to 7.0 MBps for a 1 GB file. The average rate for the 15 measurements is 6.8 MBps.

2.1.3 Comparing the Baselines on Linux and Solaris

Comparing the Solaris readings with the Linux readings we note the following:

- For both reading and writing, all Solaris rates are slightly higher than the corresponding Linux rates for the same experiment. This could be due to a faster processor, a faster bus, a faster SCSI controller,

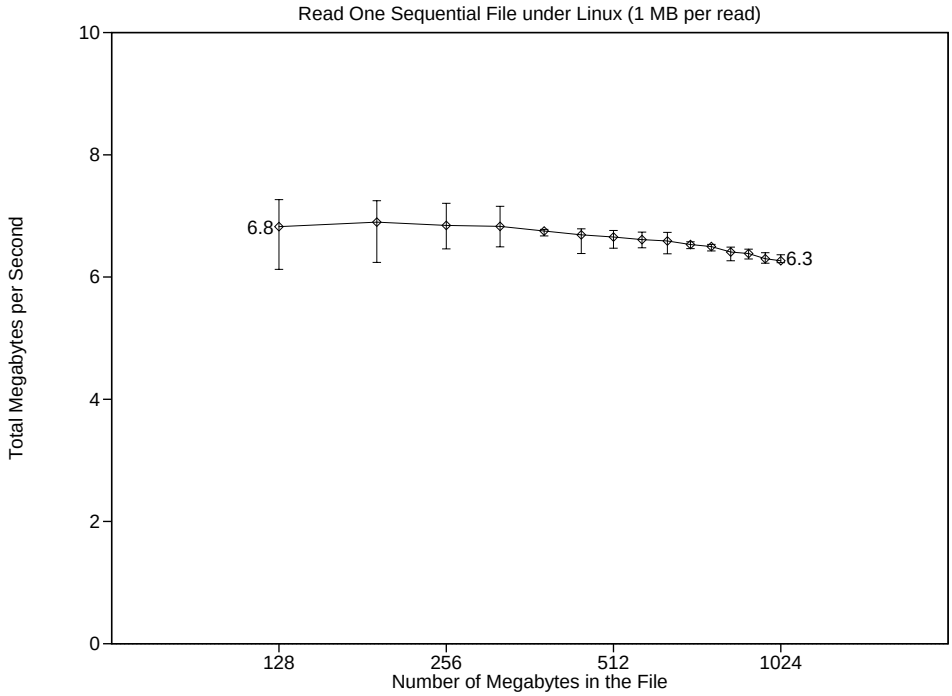


Figure 2: Total MBps for one process reading one sequential file under Linux on a Pentium Pro.

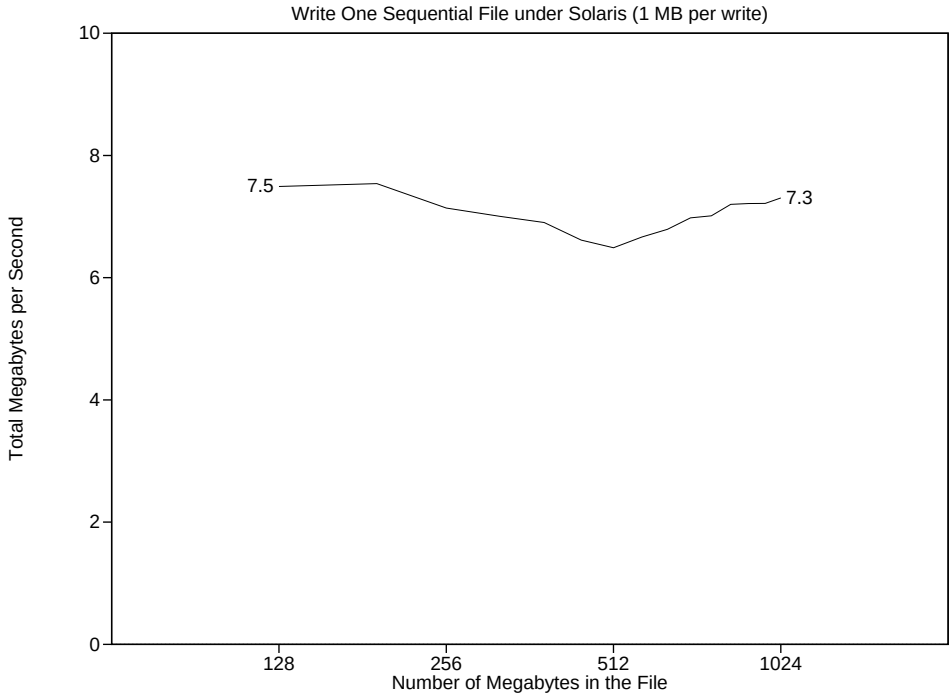


Figure 3: Total MBps for one process writing one sequential file under Solaris on a SUN i86PC.

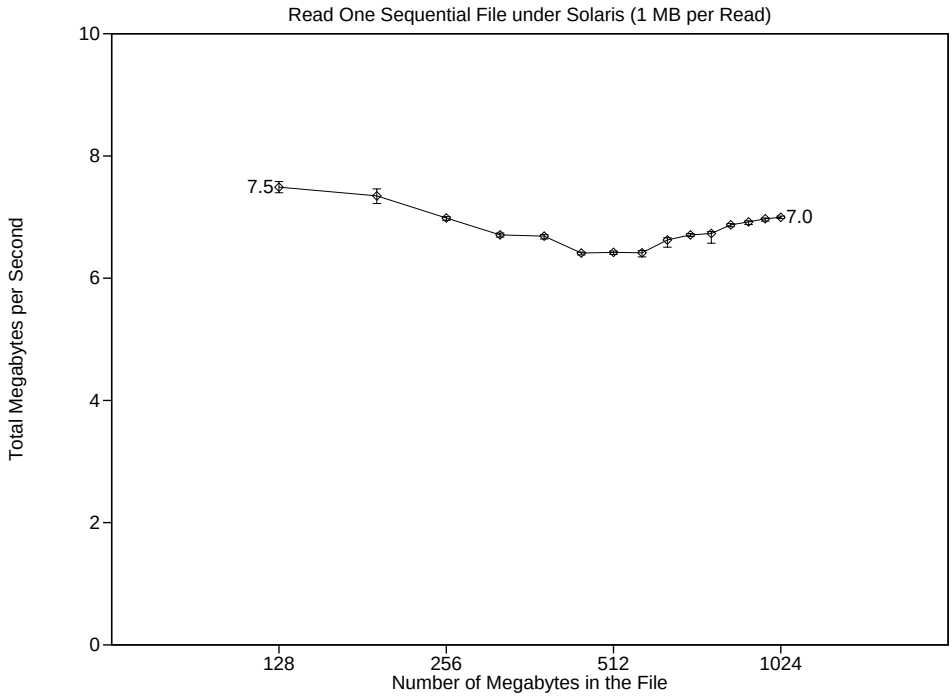


Figure 4: Total MBps for one process reading one sequential file under Solaris on a SUN i86PC.

a faster SCSI disk, less software overhead in Solaris compared to Linux, or some combination of these factors.

- The rates are higher on both machines for "small" (128 MB) files, and they tend to get lower as the size of the file increases. This effect is mitigated a bit under Solaris, because for files larger than 512 MB performance improves again, whereas on Linux machines it continues to degrade. On the other hand, the curves are flatter for Linux than for Solaris, indicating a more consistent performance from the file system that is independent of file size.

These tests establish the following baselines for reading and writing a large file on a SCSI disk (all values in MBps):

Platform	Writing	Reading
Linux	5.5	6.6
Solaris	7.1	6.8

2.2 Effects of Accessing Multiple Files Simultaneously

The next series of tests investigates the effect on SCSI disk performance of accessing multiple files simultaneously.

2.2.1 Accessing Files from Multiple Linux Processes

In the first test, several simultaneously active processes time-sharing one processor wrote separate files onto a single shared SCSI disk using the standard UNIX "write" system call. These processes were all spawned as children by a single parent process that controlled the test. However, once spawned, each child was free to progress at its own rate, with no explicit synchronization between children. Each child process terminated after writing its file, so that multiple iterations of the test involved multiple spawnings of all the child processes.

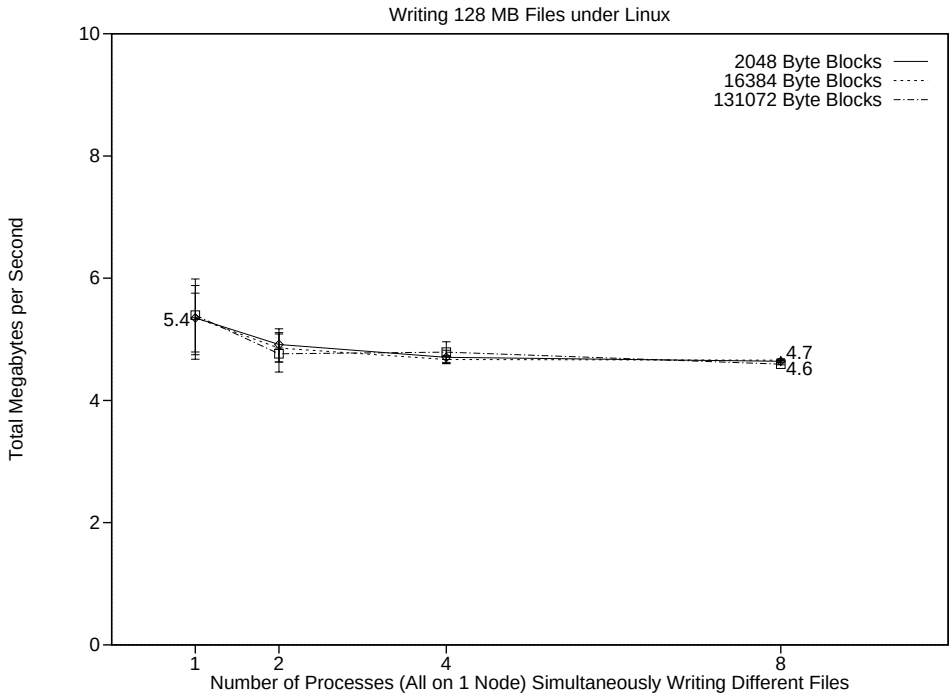


Figure 5: Total MBps for multiple processes each writing one sequential file under Linux on a Pentium Pro.

Figure 5 shows the total MBps (i.e. summed over all processes) obtained from the disk as the number of simultaneously active processes is varied from 1 to 8. The three curves, which are nearly identical, correspond to write units of 2048, 16384, and 131072 bytes. This graph shows an 11% drop in the rate, from 5.4 MBps when just one process is active, to about 4.8 MBps when more than one process is active. Furthermore, the number of simultaneously active processes doesn't seem to effect performance once there is more than one, since the total rate remains fairly flat as the number of active processes increases beyond two.

The next test consists of several simultaneously active processes reading separate files from a single shared SCSI disk. The standard UNIX "read" system call was used. As with the write test described above, one iteration of this test consisted of a single parent process spawning a number of child processes, each of which completely read an independent sequential file and then terminated. The parent process waited until all children from one iteration had terminated before starting the next iteration. Within any given iteration, there was no explicit synchronization between the children.

One factor that must be considered in this test is the "degree of interleaving" of the files being written or read. This is the number of files that are written or read simultaneously. For example, if eight files were written one after the other, their degree of interleaving is 1. If the eight files were written simultaneously, their degree of interleaving would be 8. Clearly the degree of interleaving during writing effects the layout of the files on the disk and therefore may have a substantial effect on the time taken to read them back. Files written with 1-way interleaving can be expected to be stored roughly contiguously on the disk, whereas files written with 8-way interleaving can be expected to be stored in groups of blocks that may be widely separated on the disk. It is also clear that the degree of interleaving during writing can be different than the degree during reading, and that various combinations may exhibit different performance.

Figure 6 shows the total MBps (i.e. summed over all processes) obtained from the disk when the number of simultaneously active processes varies from 1 to 8 and the read unit is held constant at 2048 bytes. The four curves, which are nearly identical, correspond to files that were written with 1-, 2-, 4- and 8-way interleaving. (The degree of interleaving during reading is just the number on the x-axis.)

This figure shows a very significant drop in the rate from about 6.0 MBps when just one process is active, to about 3.5 MBps when more than one process is active. As was the case in Figure 5 for writing, performance is flat once the number of active processes is at least two.

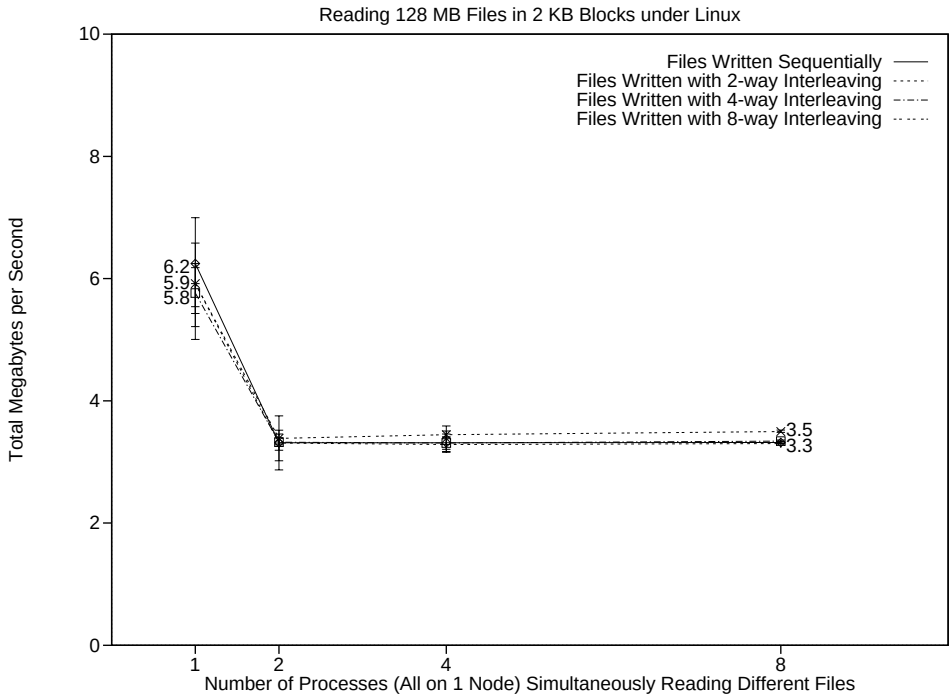


Figure 6: Total MBps for multiple processes each reading one sequential file using 2KB blocks under Linux on a Pentium Pro.

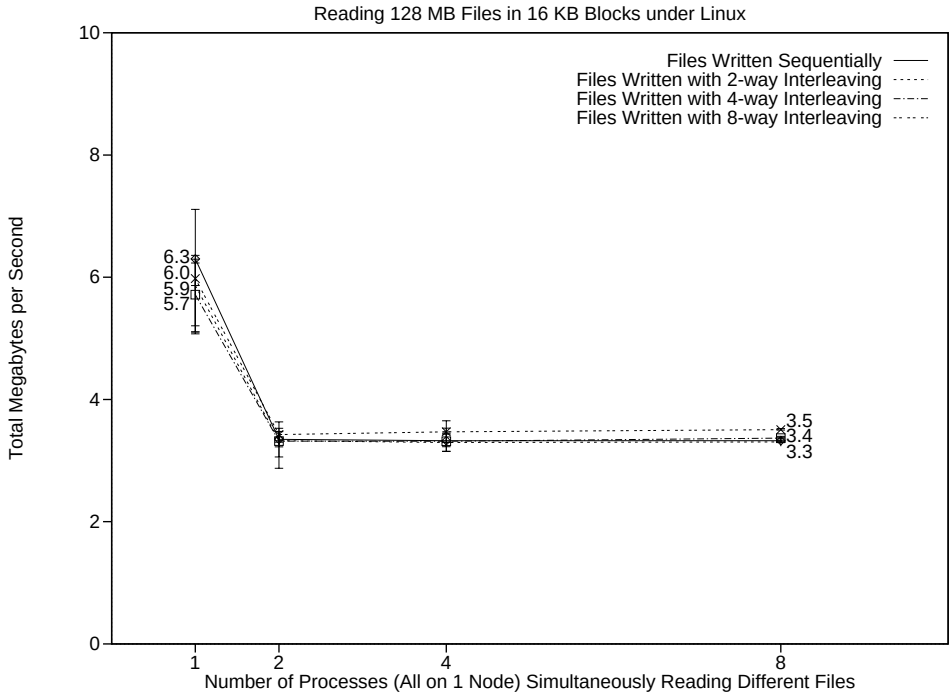


Figure 7: Total MBps for multiple processes each reading one sequential file using 16KB blocks under Linux on a Pentium Pro.

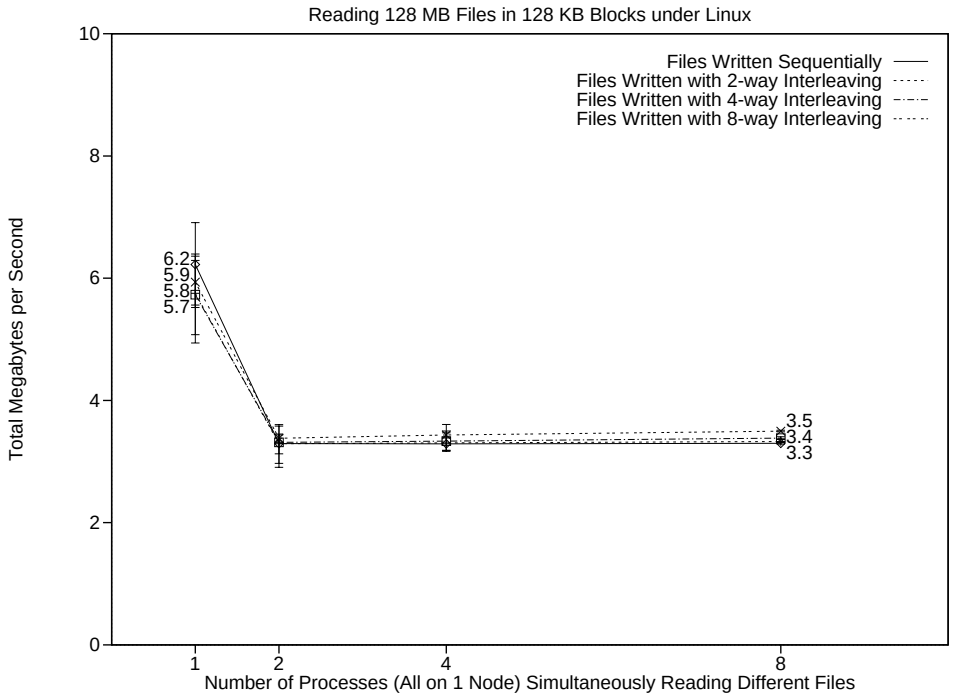


Figure 8: Total MBps for multiple processes each reading one sequential file using 128KB blocks under Linux on a Pentium Pro.

The drop in performance shown by Figure 6 is huge, to say the least, since 3.5 MBps is only 58% of the 6.0 MBps rate for a single process. This figure also shows that this drop is independent of the degree of interleaving with which the file was written, since all the curves are virtually identical. Furthermore, this drop is independent of the size of the block used in each write, as is shown by the next two figures. Figure 7 shows the results of a set of tests similar to those used to obtain Figure 6, but using a block size of 16384 bytes instead of 2048 bytes. Figure 8 shows the results when a block size of 131072 bytes was used. Both these additional figures show essentially the same results as Figure 6: a significant drop in performance when two or more processes simultaneously access separate files on a single shared SCSI disk, independent of how those files are interleaved on the disk. The performance loss is such that less than 60% of the single process disk bandwidth is available in aggregate to multiple processes.

There is therefore a significant conclusion to be drawn from these tests:

- Simultaneously reading more than one file from the same SCSI disk significantly reduces the total bandwidth that a user can obtain from that disk.

It is not obvious what is causing this performance drop. The prime suspect is the additional arm motion needed to simultaneously access two files stored in separate areas of the same disk. However, if this were the sole cause, it seems strange that accessing more than two files does not cause even more arm motion and hence even greater performance degradation. Perhaps the file system software, or the SCSI subsystem, is optimizing arm motion when it has many simultaneous requests to work on, and this prevents further degradation of the performance. Another possibility is that the disk is performing some sort of internal prefetching that works well when only one file is read, since the blocks of that file can be expected to be physically contiguous on the disk, but that is disrupted (or simply bypassed) when "jumpy" (i.e., to non-successive tracks) arm motion becomes necessary to access two (or more) files simultaneously.

It is also not clear from just these tests whether the performance drop is due solely to simultaneous access to multiple files or whether the multiprocessing aspects of the operating system contribute in some way. The next section addresses this issue.

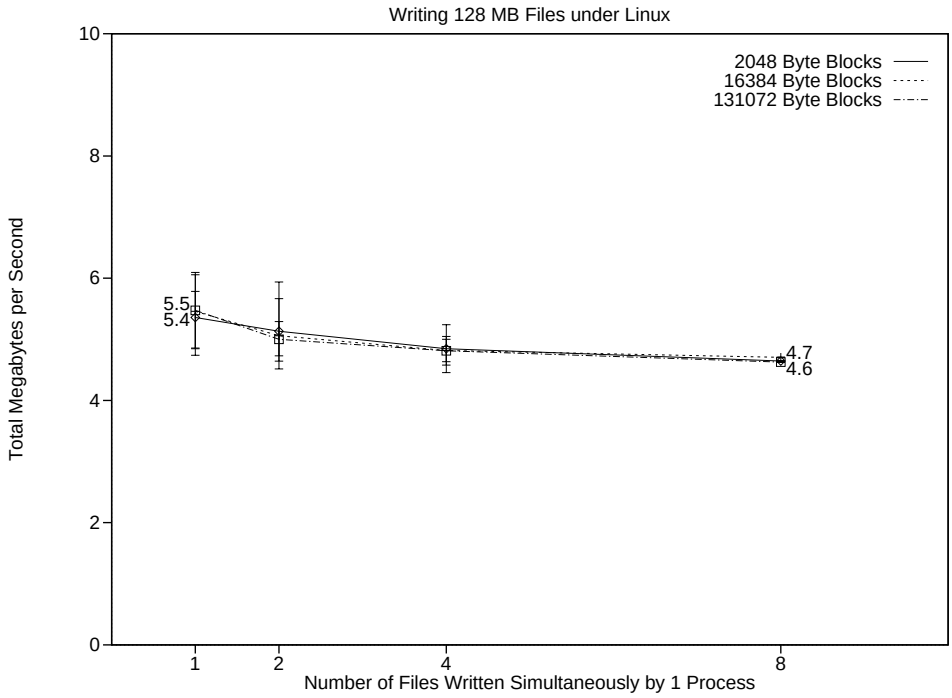


Figure 9: Total MBps for one process writing multiple files simultaneously under Linux on a Pentium Pro.

2.2.2 Accessing Multiple Files from a Single Linux Process

To investigate whether or not the severe performance loss experienced when multiple processes simultaneously access different files on the same SCSI disk is due to the effects of multiprocessing itself, a series of tests were run in which a single process accessed multiple files simultaneously in a simple "round-robin" fashion (i.e. the first block from all files was read, then the second block from all files, then the third, etc.). Figure 9 shows the total MBps (i.e. summed over all files) obtained from the disk when the number of simultaneously written files varies from 1 to 8. The three curves, which are nearly identical, correspond to write units of 2048, 16384, and 131072 bytes. This graph is nearly identical to that in Figure 5 for multiple processes. It shows the same 11% drop in rate from 5.4 MBps when just one file is being written, to about 4.8 MBps when more than one file is being written. Furthermore, the number of simultaneously written files doesn't seem to effect performance once there is more than one, since the total rate remains fairly flat as the number of files increases beyond two.

Figure 10 shows the total MBps (i.e. summed over all files) obtained from the disk when the number of simultaneously read files varies from 1 to 8 and the read unit is held constant at 2048 bytes. The four curves correspond to files that were written with 1-, 2-, 4- and 8-way interleaving. Clearly the degree of interleaving is significant in these tests. For a file written with 1-way interleaving, this figure shows the same significant drop in the rate from about 6.3 MBps when just one such file is read, to about 3.0 MBps when more than one such file is read. Performance is flat once the number of simultaneously read files is at least two, and represents less than 50% of the baseline rate for reading a single sequentially written file.

However, performance is very different for higher degrees of interleaving: it becomes worse! When reading a single file that was written with 2-way interleaving, the bandwidth is 0.7 MBps, only about 11% of the rate of 6.3 MBps for reading a single sequentially written file. The rates for reading a file written with 4-way (1.6 MBps) and 8-way (0.8 MBps) interleaving are also very poor. There is a slight improvement as the degree of read interleaving increases, but performance is always significantly less than the already reduced rate of 3.0 MBps for reading a sequentially written file. Clearly the higher degree of interleaving during writing causes dramatic loss of disk performance during read back, probably due to the greater amount of disk arm motion necessary to access the interleaved pieces of the files.

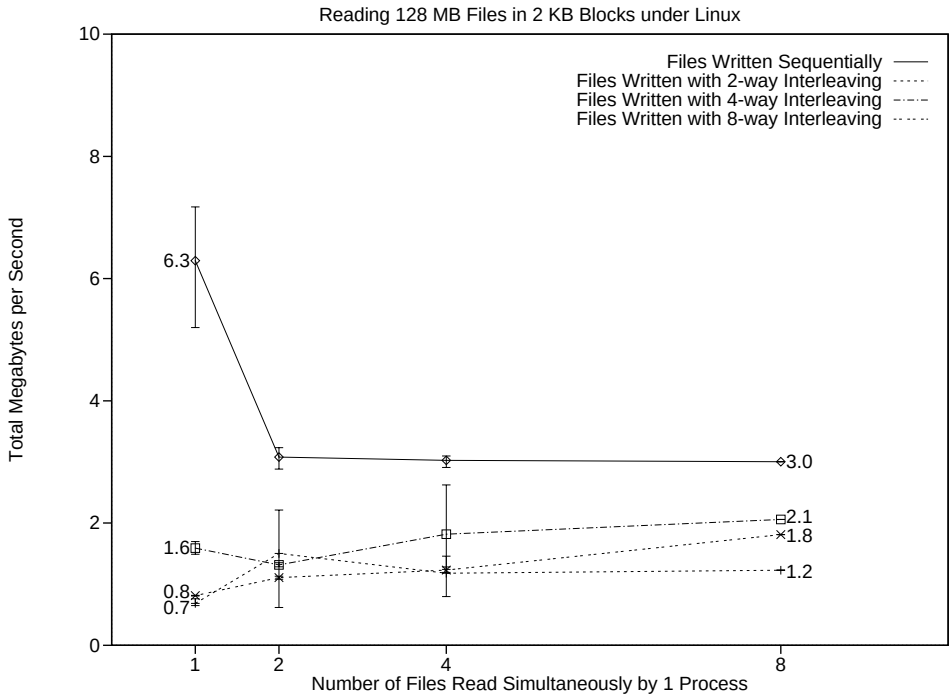


Figure 10: Total MBps for one process reading multiple files using 2KB blocks under Linux on a Pentium Pro.

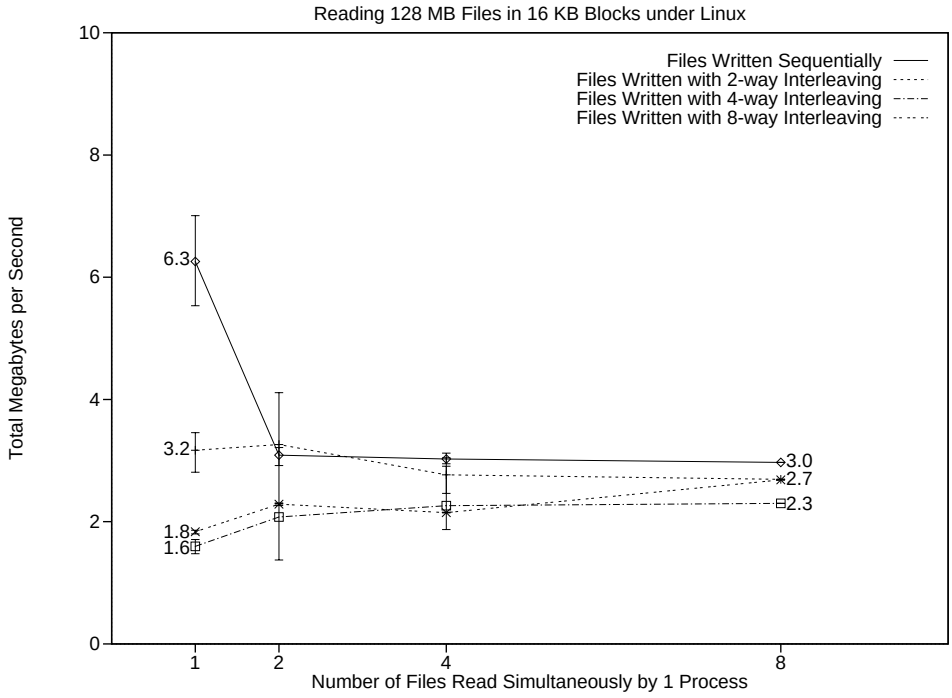


Figure 11: Total MBps for one process reading multiple files using 16KB blocks under Linux on a Pentium Pro.

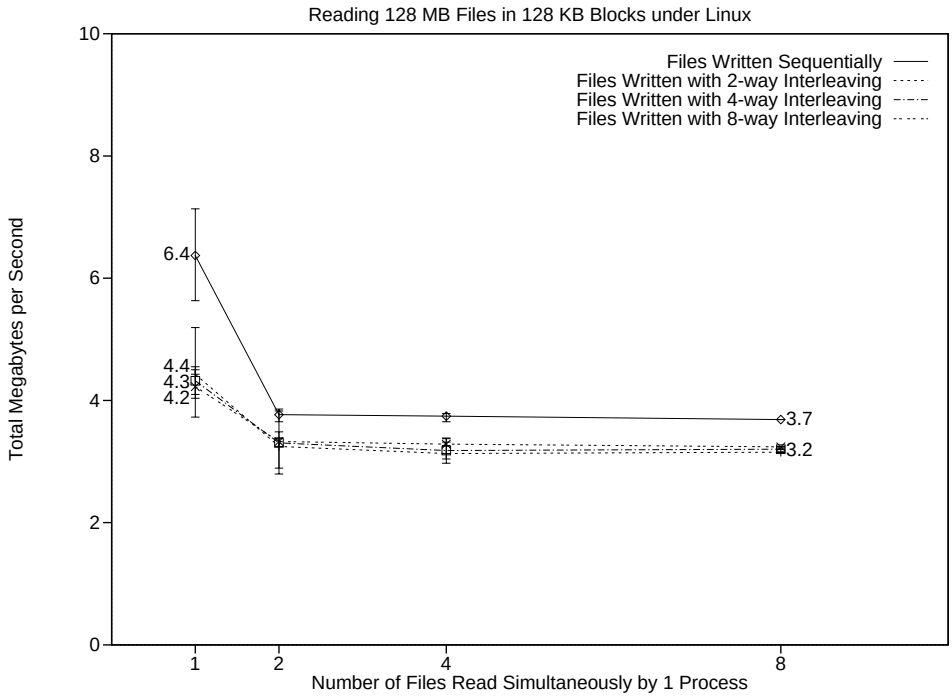


Figure 12: Total MBps for one process reading multiple files using 128KB blocks under Linux on a Pentium Pro.

Figure 10 shows the results when the block size was only 2048 bytes. Figure 11 shows the results when the block size is increased to 16384 bytes, and Figure 12 shows the results for a block size of 131072 bytes. As the block size increases, the performance improves somewhat when the degree of write interleaving is greater than 1 (it essentially remains constant for files written with 1-way interleaving (i.e., sequentially written files)). Furthermore, as block size increases, the degree of interleaving used during writing becomes less important during reading, so that for 131072 byte blocks, files written with 2-, 4- and 8-way interleaving all produce essentially identical results during reading. As shown in Figure 12, reading back a single file that was written with multi-way interleaving achieves a rate of about 4.3 MBps, about two-thirds the rate of 6.4 MBps achievable with no interleaving. When multiple multi-way interleaved files are read simultaneously, the rate drops by about 25% to 3.2 MBps, but this is only about 14% less than the rate of 3.7 MBps when reading multiple files written with 1-way interleaving.

There are three conclusions which can be drawn from these tests:

- The degree of interleaving with which a file was written has a significant effect on the rate with which that file can be read back. Best performance during reading, at all degrees of read interleaving, is obtained from files that were written sequentially (i.e., with 1-way interleaving).
- The higher the degree of interleaving when a file was written, the lower the rate at which that file can be read back at any degree of interleaving.
- The block size is significant for files written with multi-way interleaving. If files are written with multi-way interleaving, it is best to use bigger block sizes in order to obtain better performance when these files are read back.

The performance degradation seen when reading files written with multi-way interleaving is undoubtedly due to the fact that consecutive logical blocks of the same file are stored in non-consecutive physical blocks on the disk, necessitating additional arm motion on successive reads. Writing files sequentially, without any interleaving, mitigates this effect because most operating systems attempt to store consecutive logical blocks into consecutive physical blocks on the disk so that they can be accessed without additional arm motion.

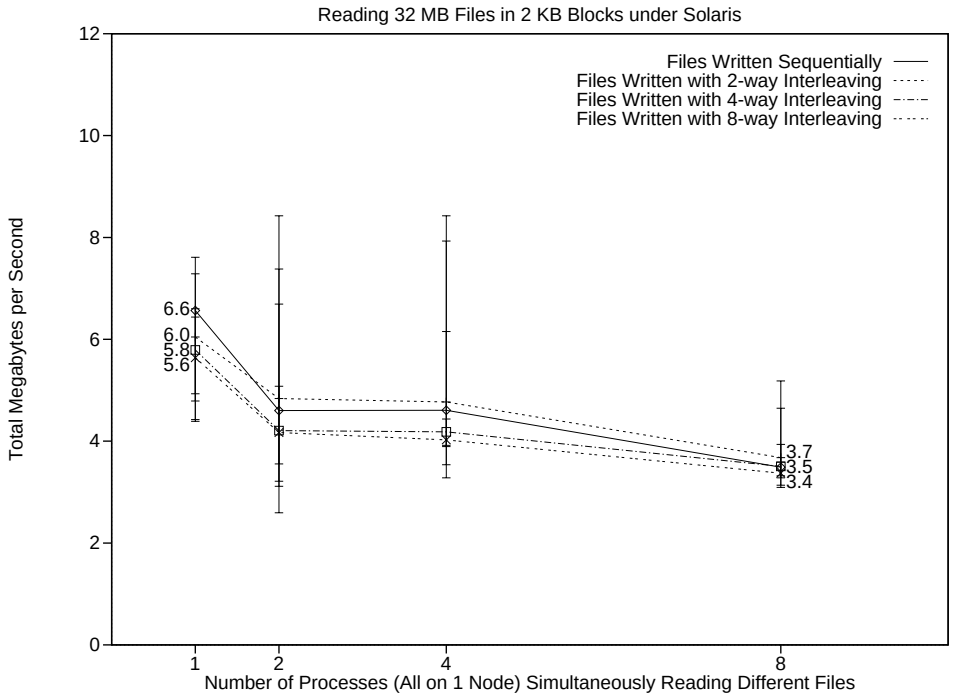


Figure 13: Total MBps for multiple processes each reading one sequential file using 2KB blocks under Solaris on a SUN i86PC.

It would seem to follow that if multiple files had to be written simultaneously, due to the constraints of the application producing the files, it would be better to preallocate a contiguous sequence of physical disk blocks for each file before any of the files are written. This of course assumes that the application has some way of predicting the final size of the file it is about to write before writing starts, and that the operating system provides a mechanism for preallocating contiguous sequences of physical disk blocks. Unfortunately, most UNIX systems do not provide such a mechanism, so the effects of this strategy could not be tested. We hypothesize that performance during writing would be degraded (since the writing between files would no longer be to physically contiguous blocks and would therefore introduce additional arm motion during writing that is presently lacking), but that performance during reading would improve. It is quite probable that curves generated by such write tests would look like those generated by current read tests, and vice versa.

2.2.3 Accessing Files from Multiple Solaris Processes

The tests of the previous two sections were rerun under Solaris on an i86PC in order to verify the conclusions reached about performance under Linux on a Pentium Pro. In these tests each file contained only 32 MB; however, the tests were run in a sequence such that many files were processed before any file was accessed a second time. Therefore, the system cache would have no effect on the measurements.

The first test consists of several simultaneously active processes reading separate files from a single shared SCSI disk. The standard UNIX "read" system call was used.

Figure 13 shows the total MBps (i.e. summed over all processes) obtained from the disk when the number of simultaneously active processes varies from 1 to 8 and the block size is held constant at 2048 bytes. The four curves, which all have the same basic shape, correspond to files that were written with 1-, 2-, 4- and 8-way interleaving. This figure shows a very significant drop in the rate from about 6.0 MBps when just one process is active, to about 3.5 MBps when eight processes are active. This figure is similar to Figure 6 for Linux. The only differences are that on Solaris there is more of a spread in performance between curves corresponding to different degrees of interleaving during writing, whereas Linux showed no such effect, and the Solaris curves are not as flat as the Linux curves, since they drop more as the degree of read interleaving

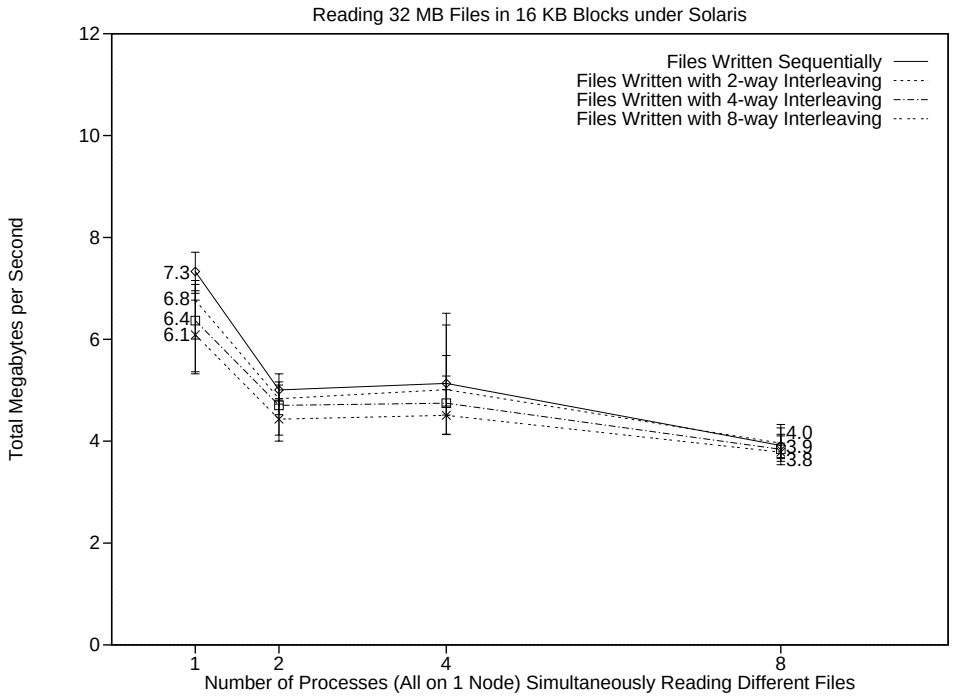


Figure 14: Total MBps for multiple processes each reading one sequential file using 16KB blocks under Solaris on a SUN i86PC.

increases (perhaps because Solaris is not optimizing arm motion over many requests, and Linux is). However, the basic conclusion remains the same: there is a significant drop in performance (on the order of 50%) when two or more processes simultaneously read different files from the same shared SCSI disk.

Furthermore, as was the case for Linux, this drop is independent of the size of the block used in each read, as is shown by the next two figures. Figure 14 shows the results of a set of tests similar to those used to obtain Figure 13, but using a block size of 16384 bytes instead of 2048 bytes. It should be compared with Figure 7 for Linux. Figure 15 shows the results when a block size of 131072 bytes was used, and should be compared with Figure 8 for Linux. Both these additional figures show essentially the same results as Figure 13 and the comparable Linux figures: a significant drop in performance when two or more processes simultaneously access separate files on a single shared SCSI disk, independent of how those files are interleaved on the disk.

2.2.4 Accessing Multiple Files from a Single Solaris Process

The last series of tests in this section was to duplicate on Solaris the tests run on Linux in which one process simultaneously accessed multiple files in "round-robin" fashion on a block by block basis. Figure 16 shows the total MBps (i.e. summed over all files) obtained from the disk when the number of simultaneously written files varies from 1 to 8. The three curves, which are similar, correspond to write units of 2048, 16384, and 131072 bytes. This graph is nearly identical to that for Linux shown in Figure 9. It shows the same drop in rate of about 11% from when just one file is being written, to when more than one file is being written. Furthermore, the number of simultaneously written files doesn't seem to effect performance once there is more than one, since the total rate remains fairly flat as the number of files increases beyond two.

Figure 17 shows the total MBps (i.e. summed over all files) obtained from the disk when the number of simultaneously read files varies from 1 to 8 and the read unit is held constant at 2048 bytes. The four curves correspond to files that were written with 1-, 2-, 4- and 8-way interleaving. As shown in Figure 10 for the comparable Linux case, the degree of interleaving is significant in these tests. For a file written with 1-way interleaving, Figure 17 shows the same significant drop in the rate by about 50% from when just one such file is read to when more than one such file is read. And as was the case for Linux, performance is generally worse for higher degrees of interleaving. We see the same slight improvement as the number of

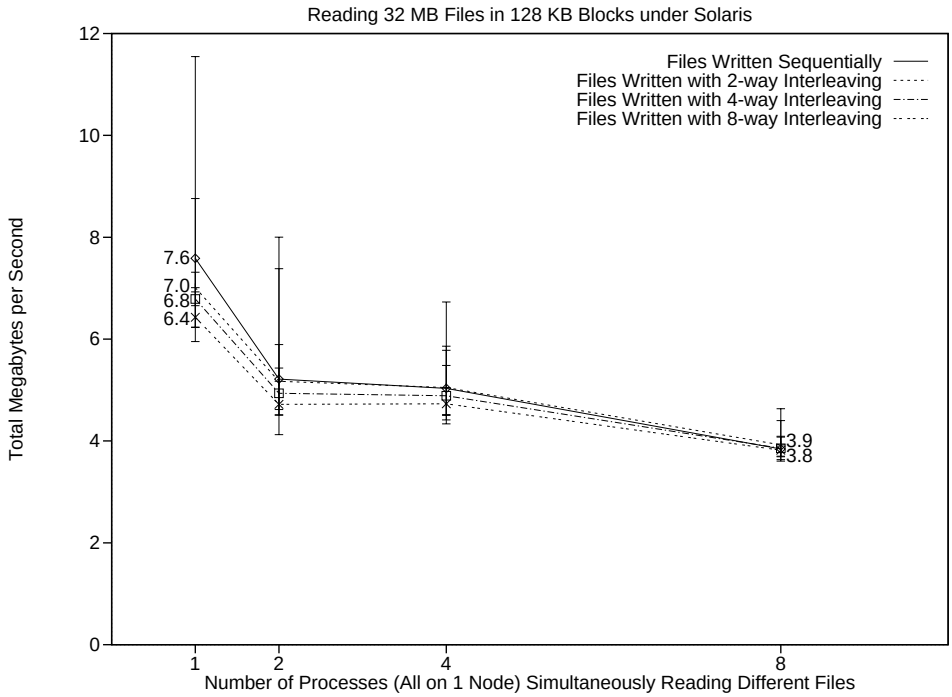


Figure 15: Total MBps for multiple processes each reading one sequential file using 128KB blocks under Solaris on a SUN i86PC.

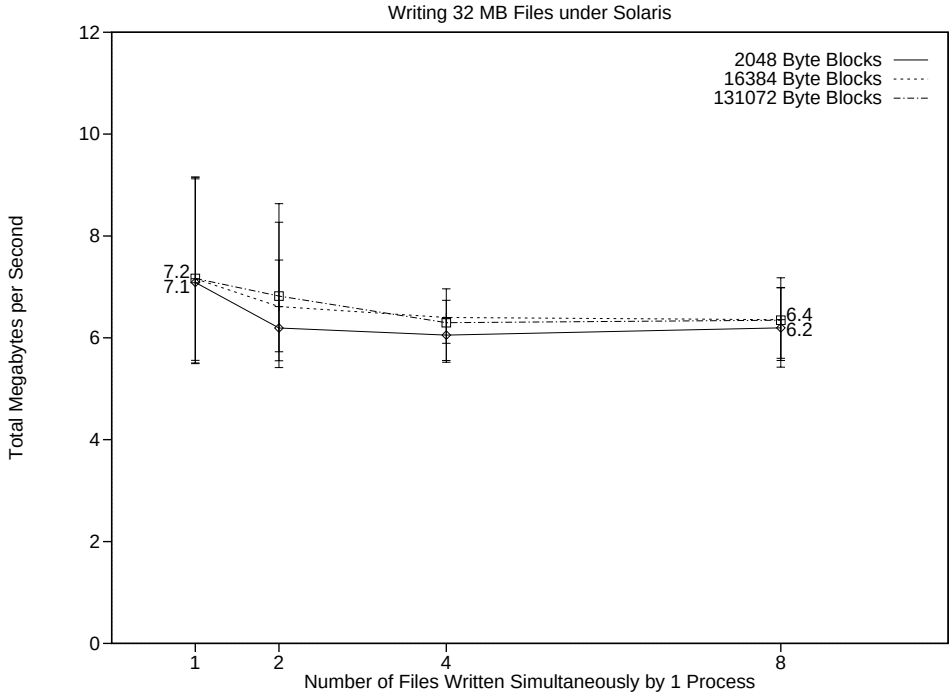


Figure 16: Total MBps for one process writing multiple files simultaneously under Solaris on a SUN i86PC.

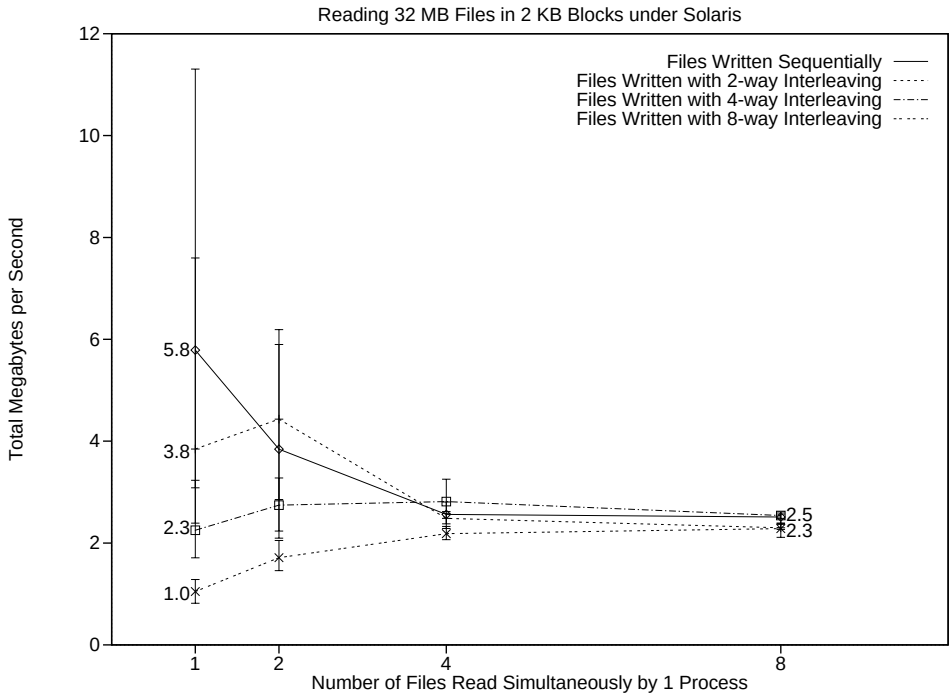


Figure 17: Total MBps for one process reading multiple files using 2KB blocks under Solaris on a SUN i86PC.

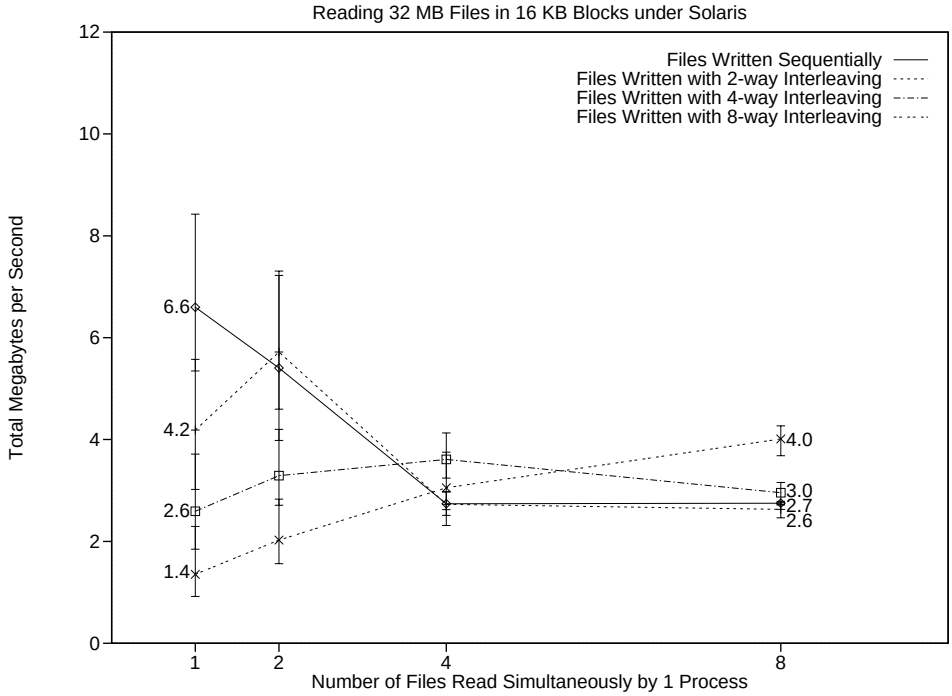


Figure 18: Total MBps for one process reading multiple files using 16KB blocks under Solaris on a SUN i86PC.

simultaneously read files increases, and on Solaris this performance does occassionally get better than the reduced rate for a sequentially written file.

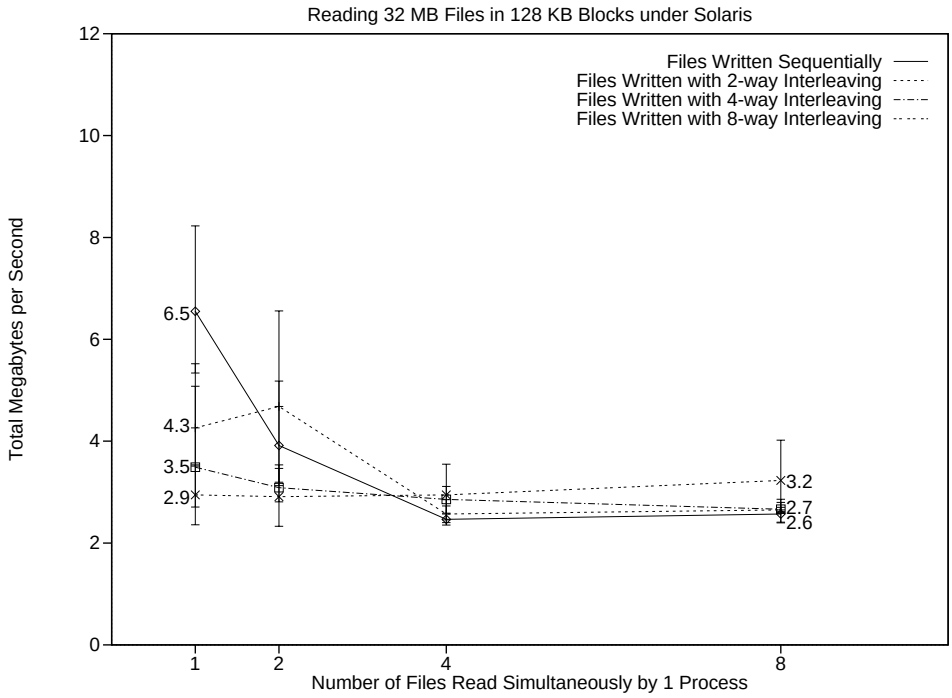


Figure 19: Total MBps for one process reading multiple files using 128KB blocks under Solaris on a SUN i86PC.

Figure 17 shows the results for Solaris when the block size was only 2048 bytes, and is comparable to Figure 10 for Linux. Figure 18 shows the results for Solaris when the block size is increased to 16384 bytes, and is comparable to Figure 11 for Linux. Similarly, Figure 19 shows the results for Solaris with a block size of 131072 bytes, and should be compared to Figure 12 for Linux. As was the case with Linux, when the block size increases, the Solaris performance improves. Unlike with Linux, the degree of interleaving remains important in Solaris as the block size increases. Indeed, Figure 18 shows an interesting feature of the interleaving: the best read-back rate is achieved when the degree of interleaving during reading exactly matches that used during writing. (The same result occurs for all but 8-way interleaving in the other two figures for Solaris.) There is a simple explanation for this: the sequence of disk arm motions caused during reading with x-way interleaving exactly duplicates that which occurred during writing with x-way interleaving, so that there is no extra, time-consuming arm motion involved.

2.3 Conclusions about Accessing Multiple Files Simultaneously

The results described in the previous subsections support the following conclusions about simultaneous access to multiple files on SCSI disks, whether by multiple simultaneously active processes or by a single process.

- Simultaneously writing more than one file to the same SCSI disk slightly reduces the total bandwidth of that disk as seen by a user application.
- Simultaneously reading more than one file from the same SCSI disk significantly reduces the total bandwidth of that disk as seen by a user application. This reduction is on the order of 50%.
- The degree of interleaving with which a file was written has a significant effect on the rate with which that file can be read back. Best performance is obtained by reading files that were written sequentially. If that is not possible, it is best to read files back with the same degree of interleaving as was used when they were written.
- In general, the higher the degree of interleaving when files are written, the lower the rate at which those files can be read back, regardless of the degree of interleaving during reading.

- The block size is significant for files with multi-way interleaving. Best performance is obtained by using bigger block sizes.

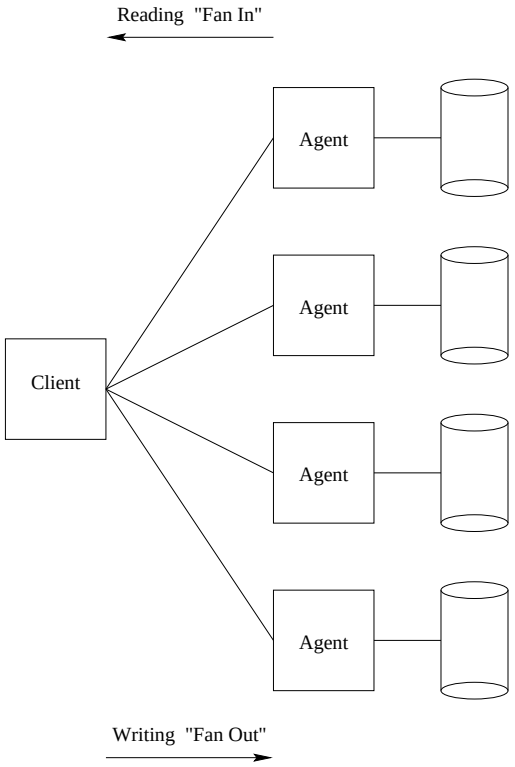


Figure 20: BPFS configuration when one non-parallel client process accesses a parallel file.

3 Using Parallel Files from Sequential Programs

In this section we investigate the performance of BPFS, a Basic Parallel File System [4], when used from sequential (i.e., non-parallel) programs. This configuration corresponds to a "fan-out" on writing, and a "fan-in" on reading, as shown in Figure 20. All of these tests were run on the PoPC platform, using the 1-Gbps Myrinet as the communication network. The client process was run on one physical node, and each server/agent was run on a different physical node. (The BPFS manager process was also run on its own physical node.) Note that in this configuration each server node will have only one agent process accessing a single file, so that the degradation of SCSI disk performance documented in Section 2 will not occur.

All the results are shown only for reading parallel files, since that is the "fan-in" situation in which a bottleneck appears, and also because it is expected to be the more critical situation in real applications.

The first test consists of reading a single large file that is distributed across a varying number of server nodes. Figure 21 shows the results for a fixed blocksize of 16384 bytes. The size of the file is adjusted so that the number of bytes stored on each server is a constant 256 MB. This means that for 8 servers the amount of data actually read by the client is 2 gigabytes.

The rate of 6.6 MBps achieved when using a single server is exactly the baseline rate for reading a file using the standard UNIX "read" system call, as demonstrated in section 2.1.3. This means that the overhead of communicating through the network is negligible for just a single server and a single client.

As the number of servers increases to 2 and 3, the rate increases almost linearly to 12.0 MBps and 17.1 MBps respectively. However, when there are 4 servers the total rate is 18.6 MBps, which is only about 3 times the base rate of 6.6 MBps, not the expected 4. Furthermore, as the number of servers increases

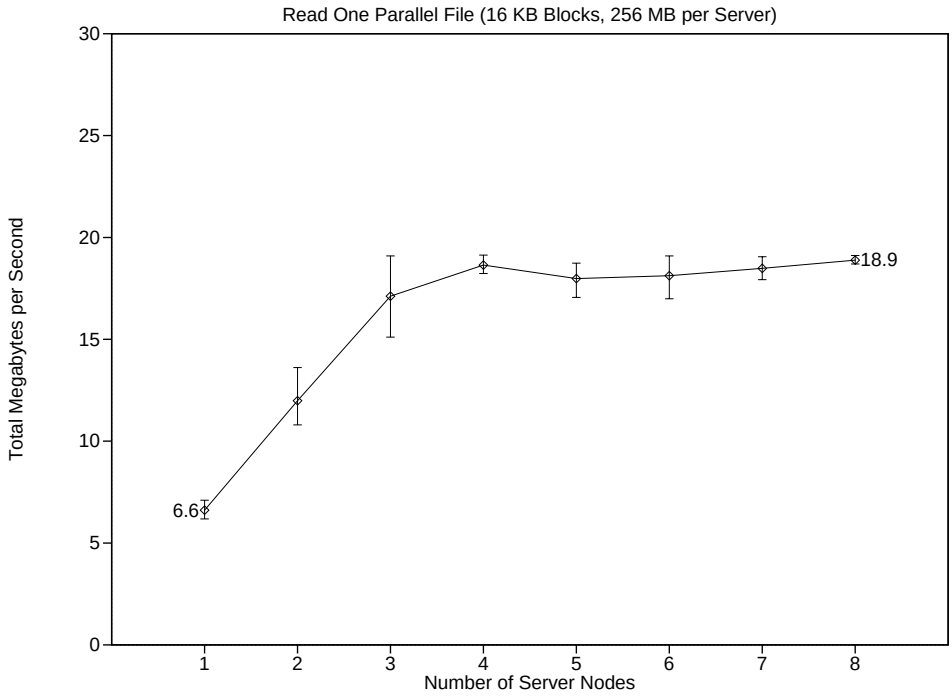


Figure 21: Total MBps for one client reading one parallel file using 16KB blocks.

beyond 4, the performance drops slightly and then increases slightly to only 18.9 MBps for 8 servers. The average rate for 4 or more servers is 18.4 MBps.

We believe this flattening of the performance curve after 3 servers is due to a communication bottleneck caused by the fan-in of the data paths from multiple servers to a single client. There are several factors that may contribute to this bottleneck, as discussed next.

The first obvious factor which may limit performance is the bandwidth available to a user program from the communications network. Although the raw hardware speed of Myrinet is rated at 1 Gbps (125 MBps), there are several layers of software between the hardware and the application program that each contributes its share of overhead.

The lowest level layer is BIP, a Basic Interface for Parallelism [2]. This is a message-passing protocol carefully designed to utilize the underlying Myrinet hardware in the most efficient manner. On top of this is the IP-BIP interface that configures a Myrinet platform for use by a standard IP stack. The next layer up is IP, followed by TCP. These are the two standard networking protocols provided by Linux. BPFS adds two more layers: the q-interface [5], which provides an asynchronous message-passing interface to UNIX and TCP/IP; and API0 [3], which is the client-side library for BPFS that is implemented on top of the q-interface.

In order to assess the network bandwidth provided to an application program sitting on top of all these layers, a simple "blast" test was designed that uses API0 to send 512 MB of data from a single client node to a single server node as fast as possible. Figure 22 shows the total MBps recorded by this test as the block size seen by a user application varies between 1 and 512 KB of data. Note that in this figure the x-axis is plotted on a logrithmatic scale. As can be seen from the figure, the effective bandwidth of the system seen by a user application is 25.6 MBps when using 1 KB blocks. This rate increases to 31.2 MBps for 4 KB blocks, and then drops off slowly to only 23.9 MBps for 512 KB blocks.

Comparing this to the results reported in [1], there are two unexpected phenomena:

1. The peak rate shown in Figure 22 is only 31.2 MBps, whereas the peak rate reported in [1] is 35 MBps.

2. The rates reported in [1] increase rapidly and then level off as the block size increases. They do not decrease for larger block sizes as they do in Figure 22.

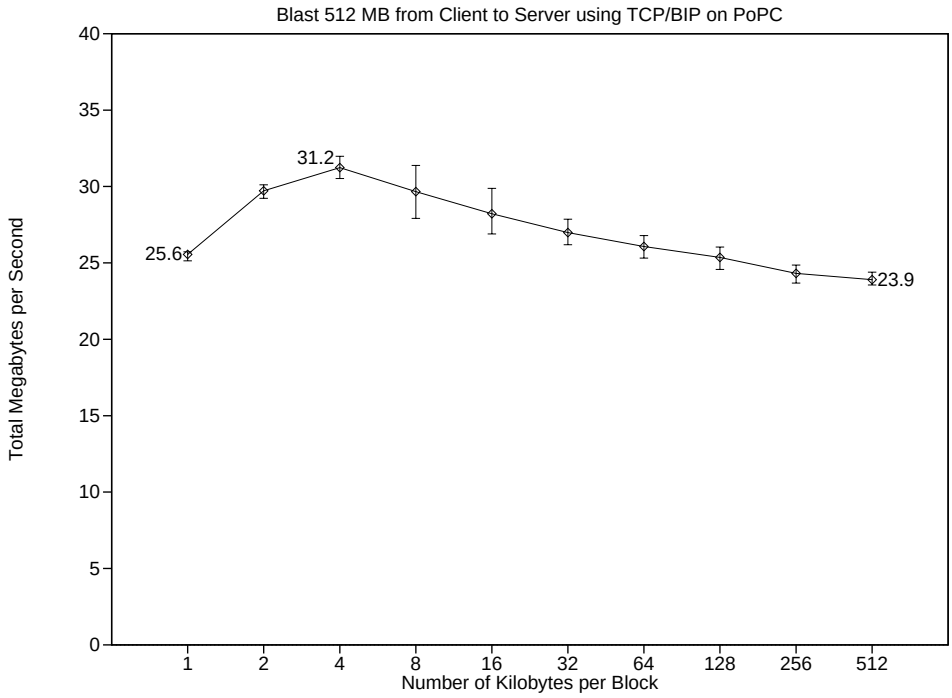


Figure 22: Total MBps for one client blasting 512 MB of data to one server using TCP/IP/BIP on PoPC.

These discrepancies are apparently due to a number of factors. First, the Linux kernel on PoPC does not contain two patches that are necessary to utilize the TCP "no delay" option. When this option is not used, under certain circumstances TCP messages are held up on the sending side awaiting an acknowledgement from the receiving side when, in fact, they could safely be sent. This obviously introduces delay, and could explain why, as the message size increases the bandwidth decreases.

A second factor is that the version of the BIP-IP software on PoPC performs software checksumming, whereas this was not the case on the test platform used in [1]. In general, checksums are not necessary because Myrinet provides reliable transmission. The time to compute a checksum over a large message can be significant.

A third factor is that the test program used in this report utilized the API0 and q-interface layers on top of TCP/IP, whereas those in [1] utilized TCP/IP directly.

These last two factors could explain why the peak rates are not as high in our tests as were observed in [1]. Unfortunately it was not possible to modify the kernel on PoPC, so factors one and two above could not be corrected.

A second experiment was performed to get a better idea of the total overhead imposed by the BPFS system independent of the disk I/O time, which was studied previously. The test shown in Figure 21 was rerun with a version of BPFS that did everything except actually read from the disk (i.e., the UNIX "read" statement in the code for BPFS was simply commented out). The results are shown as the solid line in Figure 23. As can be seen, performance is essentially flat as the number of server nodes is varied between 1 and 8, with a drop of only 3% from 25.6 MBps with 1 server to 24.8 MBps with 8 servers.

The dotted line at 28.2 MBps in Figure 23 is the "blast" value taken from Figure 22 for a block size of 16 KB. The average value represented by the solid line is about 10% less than the value represented by this "blast" line. This represents the effective overhead of BPFS, exclusive of the time to actually perform the disk reading. The flatness of the solid line in Figure 23 indicates that the BPFS system scales well, since as the number of servers is increased the total bandwidth remains essentially constant as limited by the communications network hardware and software. This gives us good reason to believe that if the network bandwidth available to a user program were increased, BPFS would be able to take advantage of it.

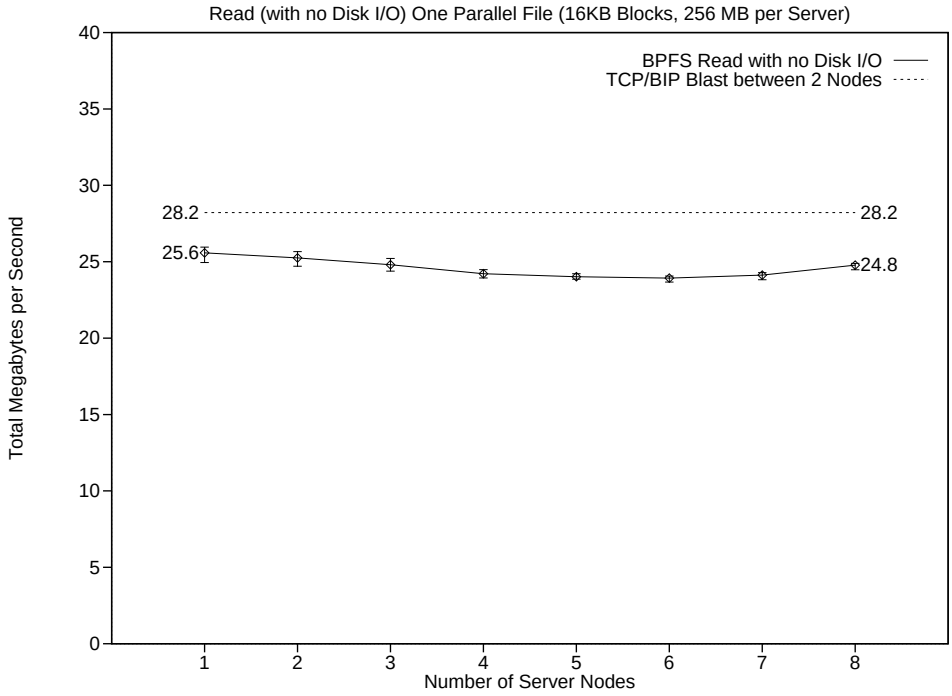


Figure 23: Total MBps for one client reading one parallel file using 16KB blocks.

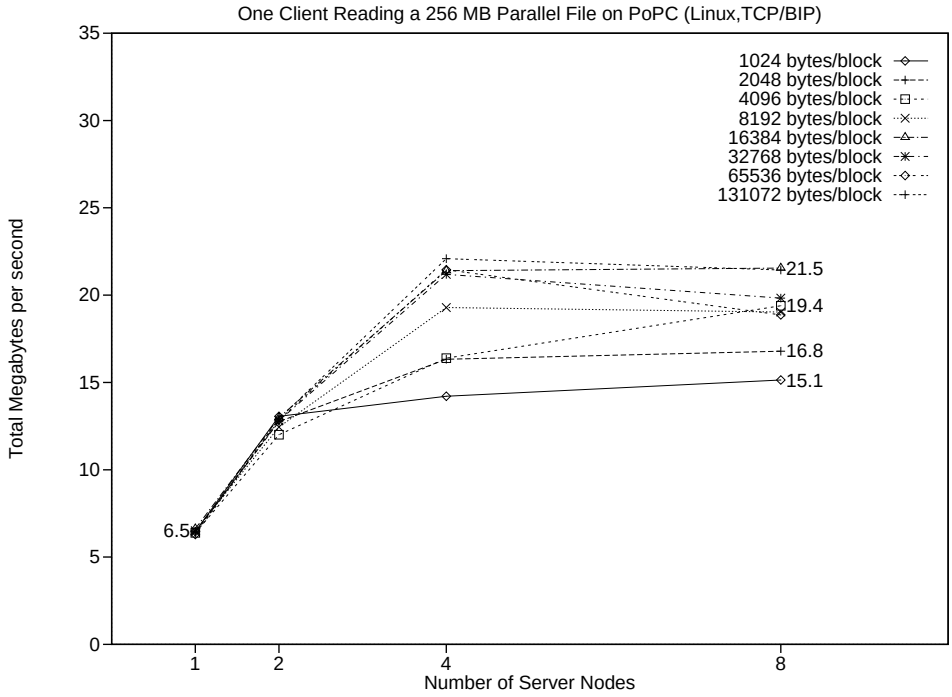


Figure 24: Total MBps for one client reading one parallel file at various block sizes.

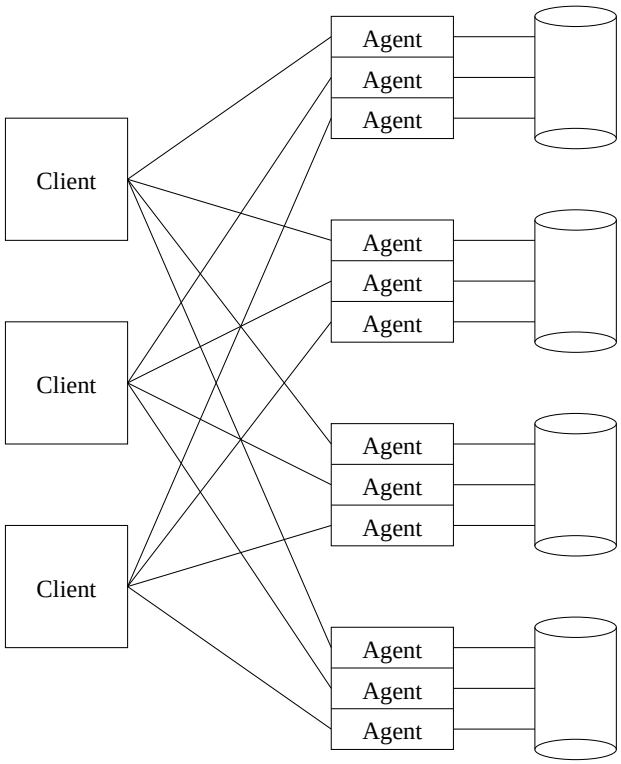


Figure 25: BPFS configuration when client processes access independent parallel files.

When disk reading is included, the nominal bandwidth provided by BPFS for 8 servers is about 18.9 MBps, as discussed earlier and shown in Figure 21. This is about 25% less than the 24.8 MBps achieved by BPFS without disk reading, as shown in Figure 23, and is apparently caused by the need to wait for the data to be delivered by the SCSI disks to the BPFS agent processes. This delay clearly impacts I/O bound tests such as these, but there is reason to believe that in truly parallel computations this I/O delay could be overlapped with computation, since data from parallel files can easily be prefetched using the features of BPFS.

One final test was run in order to asses the impact of blocksize on the performance of BPFS. Figure 24 shows the results of having one client read a parallel file distributed across a varying number of servers when block sizes between 1024 bytes and 131072 bytes are utilized. As can be seen, for all block sizes performance levels off at 4 server nodes. However, larger block sizes generally level off at a higher aggregate bandwidth than smaller block sizes. For very small blocks of 1024 bytes, the best that can be achieved with 8 servers is 15.1 MBps. With a large block size of 131072 bytes, the bandwidth provided by 8 servers is 21.5 MBps. The nominal block size of 16384 bytes used for most of the tests in this report appears to be a reasonable compromize.

4 Accessing Parallel Files from Parallel Programs

In this section we investigate the performance of BPFS, a Basic Parallel File System [4], when used from parallel programs. The configuration of a typical system with 3 processes each reading a parallel file distributed across 4 server nodes is shown in Figure 25. All of these tests were run on the PoPC platform, using the 1-Gbps Myrinet as the communication network. Each client process and each agent process was run on its own physical node. Since PoPC has only 12 nodes, the total number of processes was limited to 12, which is why none of the experiments were run with 8 clients and 8 servers (i.e., the "missing point" in all the graphs). Note that in this configuration each server node will have multiple agent processes accessing

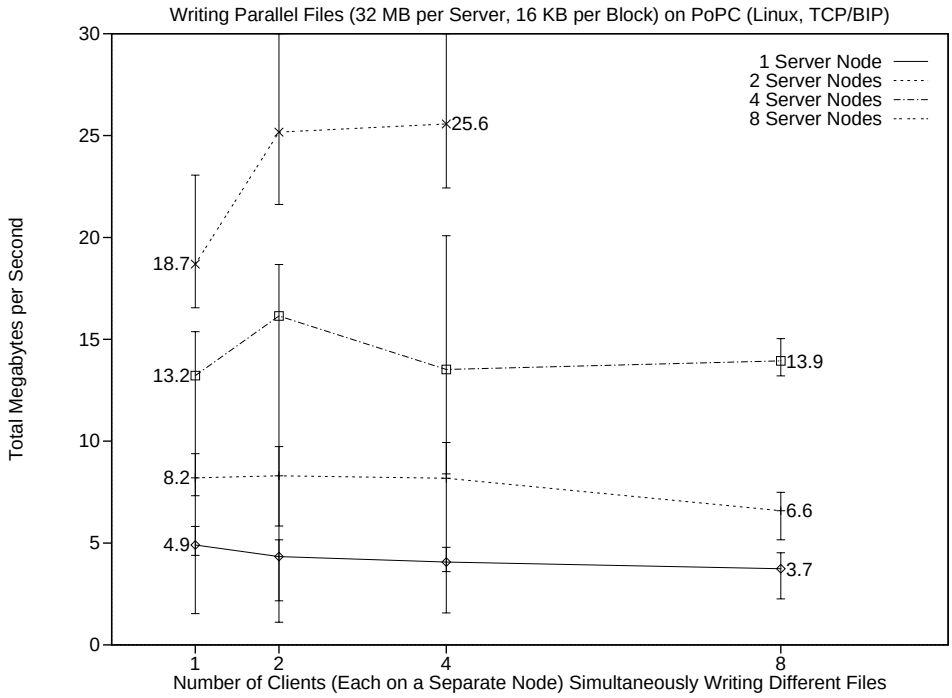


Figure 26: Total MBps for multiple clients each writing one parallel file using 16 KB blocks.

different files, so that the degradation of SCSI disk performance documented in Section 2 will also negatively affect performance here.

In order to control the client processes as one parallel program across the separate nodes, the LAM (Local area Multicomputer) version of MPI (Message Passing Interface) was used on PoPC.

In the first set of tests, various numbers of clients write independent parallel files distributed across various numbers of servers. The results for a fixed blocksize of 16384 bytes are shown in Figure 26. The size of the parallel file is adjusted so that the number of bytes stored on each server is a constant 32 MB. The graphs show that BPFS scales well during writing, both as the number of clients and the number of servers increase. As the number of clients increases, the total MBps achieved by the system for a fixed number of servers stays essentially flat or increases a bit, indicating that clients are sharing the bandwidth without an increase in overhead as their numbers increase. As the number of server nodes doubles from 1 to 2 to 4 to 8, the total MBps achieved by the system for multiple clients roughly doubles from 3.7 MBps to 6.6 MBps to 13.9 MBps to 15.6 MBps respectively. This indicates that BPFS is utilizing the additional capacity provided by additional servers with only slight loss due to overhead.

The next set of tests consists of reading back the parallel files written in the previous test. Figure 27 shows the results for a fixed blocksize of 16384 bytes and parallel files that were written with 1-way interleaving (i.e., they were each written by a single client sequentially). The size of the parallel file is adjusted so that the number of bytes stored on each server is a constant 32 MB.

The results when there is only one server node are graphed by the solid line at the bottom of Figure 27, which is nearly identical to the graph of Figure 7. This is to be expected, since the unique file being read by each client will be processed by a unique agent process, and all of these agents will be competing for access to the single disk on the same server node, exactly the situation tested in Figure 7.

The other lines in Figure 27 represent the performance for 2, 4 and 8 server nodes respectively. As expected, the curves for 2 and 4 server nodes demonstrate a sharp drop in bandwidth when the number of agents active on each server node goes from 1 to 2. This drop is missing from the curve for 8 server nodes because the performance with only 1 active agent on each server node is already limited by the network bandwidth, as described in Section 3. All the curves flatten out as the number of clients increases beyond 2,

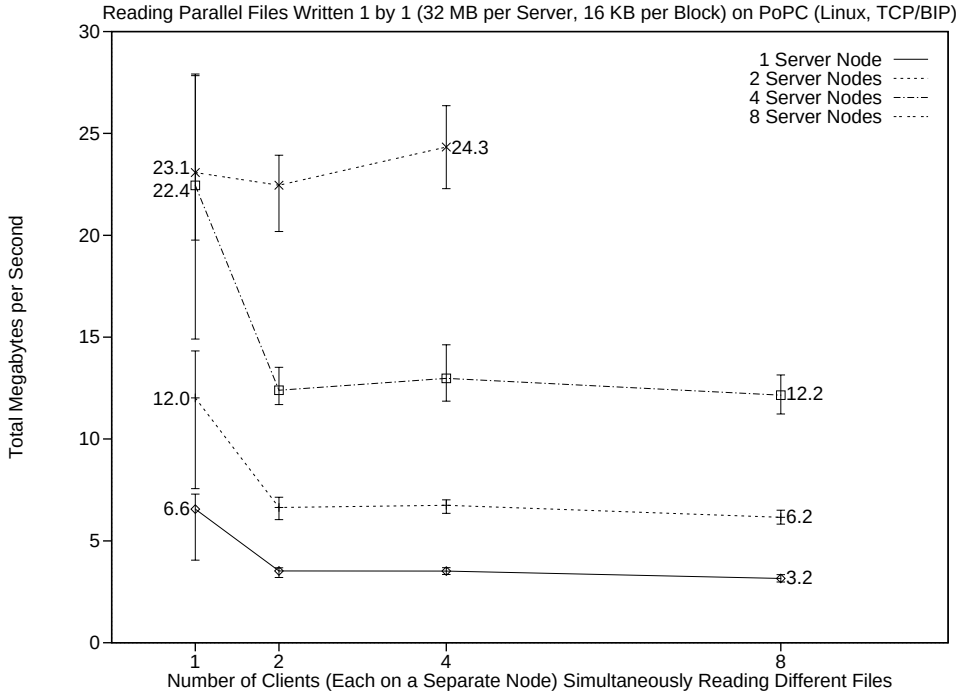


Figure 27: Total MBps for multiple clients each reading one parallel file written with 1-way interleaving using 16 KB blocks.

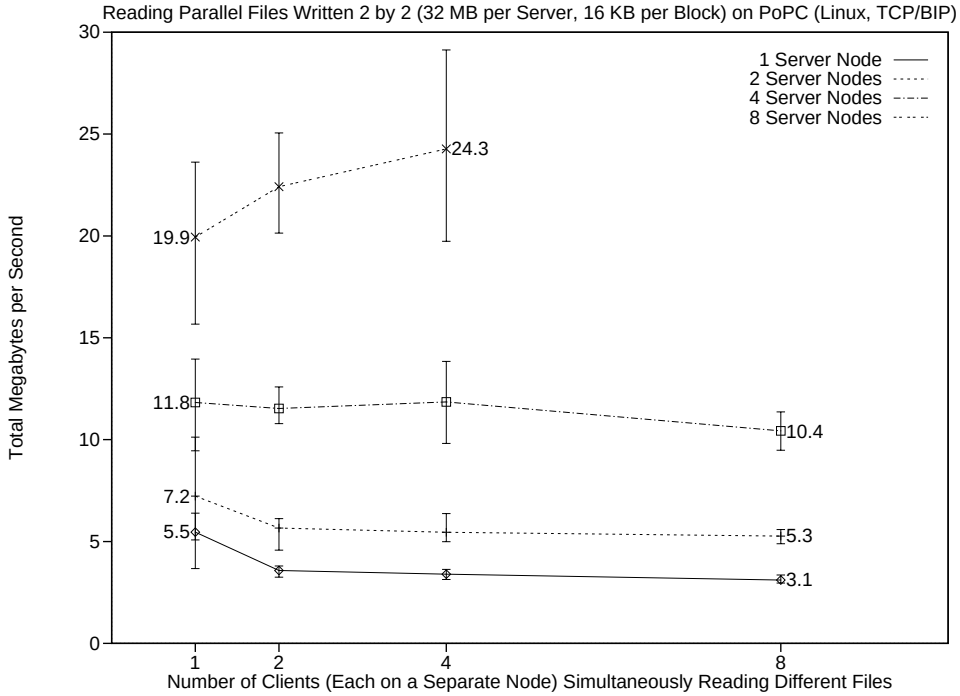


Figure 28: Total MBps for multiple clients each reading one parallel file written with 1-way interleaving using 16 KB blocks.

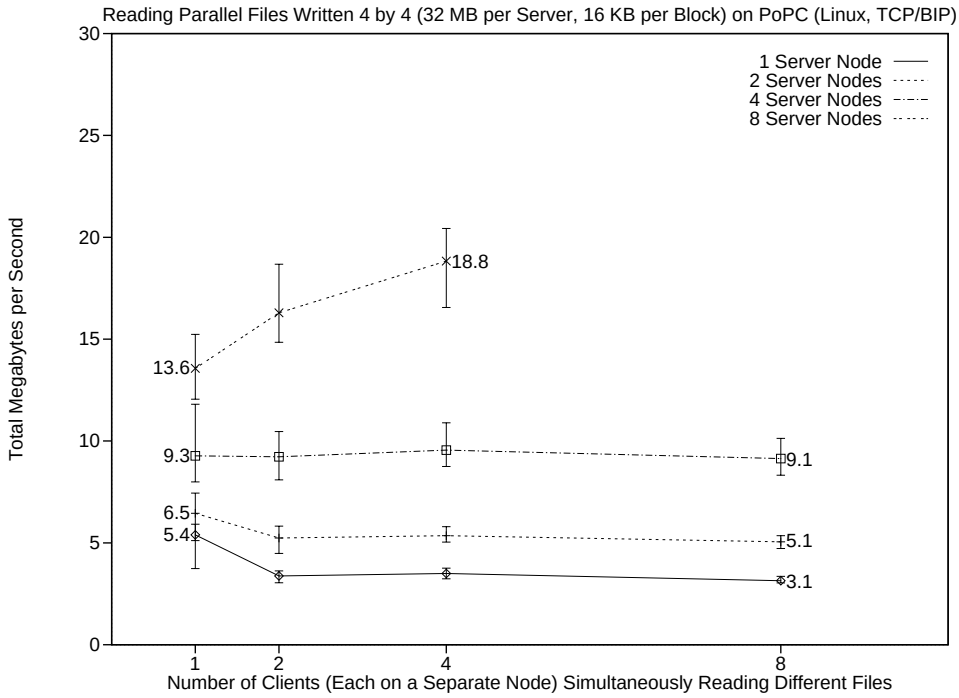


Figure 29: Total MBps for multiple clients each reading one parallel file written with 1-way interleaving using 16 KB blocks.

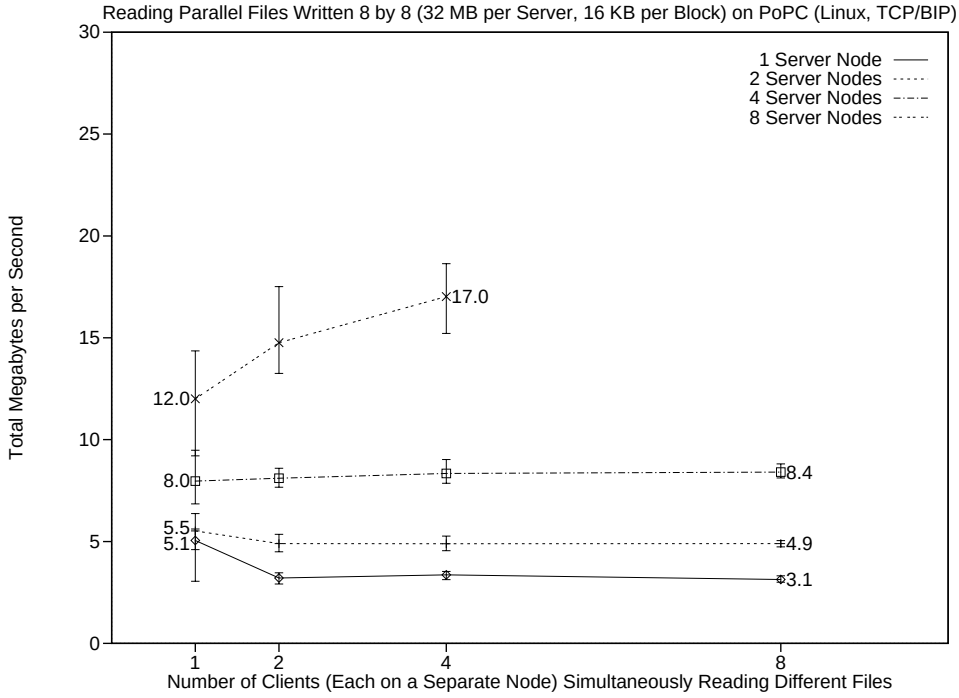


Figure 30: Total MBps for multiple clients each reading one parallel file written with 1-way interleaving using 16 KB blocks.

except for the curve corresponding to 8 servers, which shows a slight increase. This indicates that BPFS is scaling well, since the total system bandwidth remains constant as the number of clients increases.

Comparing the rates represented by the flat part of the curves, we see that BPFS also scales well in its use of servers: doubling the number of servers from 1 to 2 to 4 to 8 effectively doubles the available system bandwidth from 3.2 MBps to 6.2 MBps to 12.2 MBps to 24.3 MBps respectively.

The next set of tests reads back the parallel files written with 2-way interleaving (i.e., 2 clients simultaneously writing independent files). Figure 28 shows the results for a fixed blocksize of 16384 bytes. The curves in this figure show the same general shape and interrelationships as those in Figure 27. The differences are:

- they all represent correspondingly lower bandwidth (except for multiple clients with 8 servers),
- the drop in bandwidth when going from 1 to 2 simultaneous clients (which corresponds to going from 1 to 2 agents simultaneously active on each server node) is not as sharp.

Both these trends continue as the degree of interleaving used for writing increases to 4-way (as shown in Figure 29), and 8-way (as shown in Figure 30). Indeed, the 24.3 MBps rate achieved by 4 clients and 8 servers when reading files written with 1-way interleaving is reduced 30% to 17.0 MBps when reading files written with 8-way interleaving. There is also a 30% reduction in the rate when 4 servers are used, and a 20% reduction when 2 servers are used. Only when a single server is used is there essentially no change in the bandwidth.

These results for parallel files support the conclusions drawn in Section 2, as indeed they must, since the tests in Section 2 essentially perform the same operations as the agents in the tests of this section. The degree of interleaving used when a parallel file is written has a significant effect when that parallel file is read. In particular, the higher the degree of interleaving used when writing, the lower the rate at which that parallel file can be read.

5 Conclusion

This report has described a set of experiments which measured the SCSI disk performance seen by an end user. By performing the experiments on two platforms with different hardware and different operating system software, an attempt was made to factor out these variables as contributors to the observed behavior. Clearly further experiments on other platforms would be desirable to reinforce the conclusions drawn herein.

Based on these measurements, a number of conclusions were made in the previous sections of this report. These are summarized here.

- Simultaneously writing more than one file to the same SCSI disk slightly reduces the total bandwidth of that disk as seen by a user application.
- Simultaneously reading more than one file from the same SCSI disk significantly reduces the total bandwidth of that disk as seen by a user application. This reduction is on the order of 50%.
- The degree of interleaving with which a file was written has a significant effect on the rate with which that file can be read back. Best performance during reading, at all degrees of read interleaving, is obtained from files that were written sequentially (i.e., with 1-way interleaving). If that is not possible, it is best to read files back with the same degree of interleaving as was used when they were written.
- In general, the higher the degree of interleaving when files are written, the lower the rate at which those files can be read back, regardless of the degree of interleaving during reading.
- Block size is significant for files written with multi-way interleaving. For such files, it is best to use bigger block sizes in order to obtain better performance when these files are read back.

These conclusions describe the behavior a user application can expect to see from a SCSI disk system, and some steps which that application can take to improve its expected performance. It is not obvious what is causing this behavior. Clearly file layout on disk is important, since it can effect the amount of

arm motion necessary to access a file. Most of the consequences of multi-way interleaving have to do with the resultant layout of the file on disk. It also appears that SCSI disk systems may be implementing a prefetching or "read-ahead" scheme that improves performance when a smooth progression of disk blocks are being accessed (which would occur if only a single file is being accessed and that file were stored contiguously on the disk), but which is either by-passed or performs poorly when arm motion in a non-progressive fashion is required (which would occur if several files are being accessed such that successive disk accesses were to non-contiguous blocks). Unfortunately, without more detailed investigation of the inner workings of the disk and its software drivers, such statements are little more than speculation.

This report also investigated the performance of BPFS, a Basic Parallel File System, that is built on top of commodity components such as SCSI disks. The performance of BPFS directly reflects the performance of the underlying disk systems. It is therefore most disturbing to consider the huge loss of performance when two or more files are being simultaneously accessed on the same SCSI disk, because this brings into question one of the fundamental assumptions of parallel file systems, namely, that a parallel file system can provide good performance by providing parallel access to files that are distributed across shared server nodes. Our experiments show that BPFS scales well as both the number of client processes and the number of server nodes increases, once the underlying performance degradation caused by sharing SCSI disks has been discounted. Further experiments need to be done with other types of disks to see if they exhibit similar performance.

One feature that our experiments indicate would be beneficial when storing files on SCSI disks, but which is lacking from most operating systems, is the ability to preallocate an entire file as a single sequence of contiguous physical blocks on the disk. We hypothesize that doing this might decrease performance when several such files are written simultaneously, because of the additional arm motion that would be introduced, but should significantly increase performance when these files are read, regardless of the number of different files being read simultaneously.

References

[1] Prylli, L., and Tourancheau, B. Protocol design for high performance networking: a myrinet experience. Tech. Rep. RR97-22, Laboratoire de l'Informatique du Parallélisme, Ecole Normale Supérieure de Lyon, Lyon, France, July 1997.

[2] Prylli, L., and Tourancheau, B. BIP: A new protocol designed for high performance networking on myrinet. In *Workshop PC-NOW, IPPS/SPDP* (1998).

[3] Russell, R. D. Application interfaces to BPFS: a basic parallel file system. Tech. Rep. RR98-28, Laboratoire de l'Informatique du Parallélisme, Ecole Normale Supérieure de Lyon, Lyon, France, June 1998.

[4] Russell, R. D. The architecture of BPFS: a basic parallel file system, version 1.0. Tech. Rep. RR98-21, Laboratoire de l'Informatique du Parallélisme, Ecole Normale Supérieure de Lyon, Lyon, France, March 1998.

[5] Russell, R. D. The q-interface to UNIX. Tech. Rep. TR98-03, Laboratoire de l'Informatique du Parallélisme, Ecole Normale Supérieure de Lyon, Lyon, France, July 1998.

Unit´e de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unit´e de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unit´e de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unit´e de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unit´e de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399