Tuning STARFISH

David Kotz

Technical Report PCS-TR96-296

Department of Computer Science
Dartmouth College
Hanover, NH 03755-3510
dfk@cs.dartmouth.edu

October 14, 1996

Abstract

STARFISH is a parallel file-system simulator we built for our research into the concept of disk-directed I/O. In this report, we detail steps taken to tune the file systems supported by STARFISH, which include a traditional parallel file system (with caching) and a disk-directed I/O system. In particular, we now support two-phase I/O, use smarter disk scheduling, increased the maximum number of outstanding requests that a compute processor may make to each disk, and added gather/scatter block transfer. We also present results of the experiments driving the tuning effort.

1 Introduction

STARFISH is a parallel file-system simulator, based on the Proteus simulator [BDCW91]. It was originally developed by the author for research into the concept of disk-directed I/O [Kot94a, Kot94b, PEK96, Kot95b, KC95, Kot95c, Kot95a]. In the course of preparing a more complete paper about disk-directed I/O [Kot96], we made several modifications to both of the parallel file systems supported by STARFISH ("traditional caching," which we now call the "traditional parallel file system," and "disk-directed I/O"). This report details those changes, and the results of the experiments driving the tuning effort.

Most of the effort was applied to improving the performance of the traditional parallel file system, as disk-directed I/O needed little improvement.

This research was funded by NSF under grant number CCR-9404919, by NASA Ames under agreement number NCC 2-849.

This paper assumes that you are familiar with the disk-directed I/O papers, but not necessarily with the STARFISH code.

2 Major changes to STARFISH

The earlier disk-directed I/O papers report results from STARFISH version 2.0, with some small extensions. In 1996, we made several significant changes to STARFISH. STARFISH 3.0, released in conjunction with this report, includes all of these changes.[1]

2.1 Disk scheduling

The disk-scheduling algorithm only affects the traditional parallel file system, because disk-directed I/O builds a (possibly sorted) list of disk requests in advance, for each disk, and feeds that entire list to the disk driver at once.

STARFISH 2.0 used a simple FCFS disk-scheduling algorithm. Each disk had its own queue of disk requests, read by its own disk-driver thread (see Figure 1). The cache-management code on the I/O processors (IOPs) fed requests into the appropriate disk-request queue, one at a time, as they occurred. The disk-driver thread serviced requests, one at a time, in the order they were inserted into the queue. Poor performance resulted, especially in contiguous disk layouts, unless the requests happened to be enqueued in increasing disk-block order. Although, in the tested workloads, each compute processor (CP) requested its blocks in increasing file-block order, which in a contiguous layout translates into increasing disk-block order, the interleaving of requests from multiple CPs may not have resulted in an increasing disk-block order at the disk-request queue.

In STARFISH 3.0 we added a new disk-queuing module. Each disk has a single queue as before, but it is a priority queue rather than a simple FCFS queue. New disk requests were placed into the per-disk priority queue using the Cyclical Scan algorithm [SCO90]. The implementation actually uses two priority queues, each maintained in order of increasing disk-block number: one for requests "ahead" of the current disk-head position, which will be serviced in the current pass of the disk head, and another for requests "behind" the current disk position, which will be serviced on the next pass of the disk head.

2.2 Increased number of outstanding requests

This section applies only to the traditional parallel file system.

[1] The STARFISH code and papers are available at `http://www.cs.dartmouth.edu/~dfk/STARFISH/`.

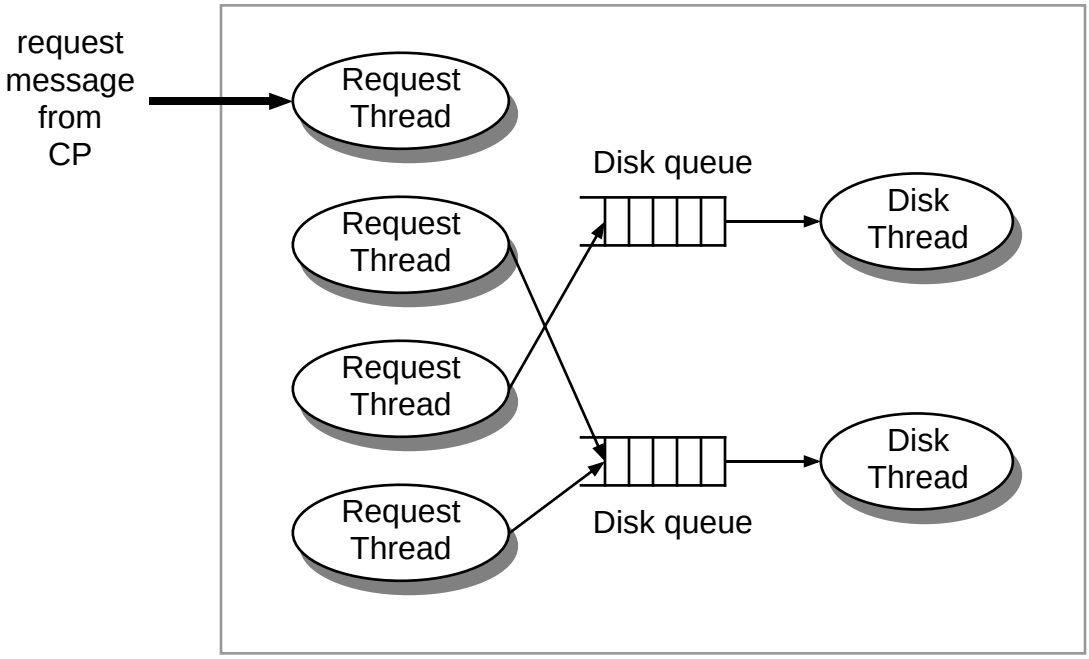

Figure 1: Structure of an I/O processor (IOP) in STARFISH, when running the traditional parallel file system. Each incoming request is handled by a separate thread. That thread examines the cache, and possibly inserts a request in one of the disk requests queues. There is one queue, and a thread to service it, for each local disk.

In our traditional parallel file system implementation, each CP receives requests from the application, each for some contiguous range of bytes in the file. Call these "file requests." The file is striped, block by block, across all of the disks. Thus, the CP file-system code must translate the file request into a sequence of "IOP requests," each requesting one block (or less) of data and each directed at a particular disk on a particular IOP.

In STARFISH 2.0 each CP was limited to a maximum of one outstanding request to each disk. This provided a simple static flow-control solution, because each IOP knew it would not receive more than $N_{\mathrm{CP}} \times N_{\mathrm{LocalDisks}}$ requests at any given time. Thus, each IOP cache contained twice that many buffers, each holding one block, allowing the IOP to double-buffer a separate stream of requests from each CP to each disk. The double buffering was necessary because the IOP caches attempted to prefetch on each read, and write behind on each write.

The CPs attempted to maximize their potential by working through the sequence of blocks necessary to satisfy the current file request, sending up to one IOP request to each disk, then waiting for a IOP request to complete. As such, one CP could keep all of the disks busy, if the request involved at least one block from each disk.

Nonetheless, with only one request from each CP, the IOPs' disk queues were fairly short, which

seemed to justify the use of FCFS disk queuing.

Few of our workload's access patterns presented the CP with requests larger than one stripe; many of them presented the CP with an 8-byte request! So it was not often possible to send more than one request to each disk, even if it were allowed.

Nonetheless, in STARFISH 3.0 we rewrote the CP code that services file requests. It now generates up K outstanding IOP requests to each disk, if the current file request so requires, where K is a constant integer ≥ 1. As soon as a disk replies that it has completed a given IOP request, another IOP request is sent to that disk. With a large request (larger than KD blocks on D disks), each CP can keep each disk supplied with K requests until it has made all of the IOP requests necessary for the current file request. That concurrency should fill the disk queues, and enable better disk scheduling.

Each IOP cache now contain K times as many buffers as before, to accommodate the potential flood of data and to retain the same static flow-control solution.

2.3 Queued Memput and Memget

This section affects only disk-directed I/O and two-phase I/O (we used Memgets and Memputs to implement the permutation phase).

STARFISH supports two "remote DMA" operations. *Memput* allows a processor to store a block of data into a remote processor's memory, and *Memget* allows a processor to fetch a block of data from a remote processor's memory. In both cases, the remote processor must be expecting such transfers, and must supply a base address in advance. The Memput or Memget request supplies an offset and a length; the offset is added to the base address by the remote processor's interrupt handler before transferring the data.

This arrangement is extremely convenient for disk-directed I/O. Each CP sets its base address to the address of its user-supplied buffer. Then the IOPs, using their knowledge of the access pattern, can compute for each byte of the file read or written, the location (CP number and offset within the CP) of the corresponding byte in memory. There is no need for every IOP to know the base address of every CP buffer.

Nontheless, we saw poor performance in some access patterns involving 8-byte records, because they used Memput or Memget operations to transfer only 8 bytes at a time. The overhead was overwhelming.

STARFISH 3.0 supports an alternative Memput and Memget system, which we call "queued

Memget" and "queued Memput." This system essentially supports gather/scatter block transfer. Queued Memgets and Memputs have the potential to be faster for small requests, although they require extra data copying and processing on both sides of the transfer.

With this arrangement, each processor has a collection of buffers, one for each other processor (allocated as needed). Each buffer is large enough to hold one file-system block. The processor may make a series of queued Memputs, to any set of processors, of any size. Requests larger than one block are broken into smaller requests. Each resulting Memput request (offset, length, and data) is appended to the buffer for the appropriate destination processor. Once the buffer fills, it is sent to the remote processor, where the requests are unpacked and processed: each offset is added to the base address, and then the data is copied into that address. An acknowledgement is returned.

Similarly, a processor may make a series of queued Memget requests. Requests larger than one block are broken into smaller requests. Each request (offset, length) is appended to the outgoing buffer for the appropriate destination processor. Once the outgoing buffer is full, that is, it would result in a full reply buffer, it is sent to the destination processor. Once there, a reply buffer is allocated and the requests are unpacked and processed: each offset is added to the base address, and the data is copied from that address into the reply buffer. The reply buffer is then sent to the requesting processor, where the data is copied into the desired location.

In both cases, the requesting processor restricts itself, for flow control reasons, to one outstanding request per destination processor. It avoids waiting as much as possible, by avoiding the check for a reply until it needs to send a new request.

3 Other features of STARFISH

STARFISH 2.0 included several features that were not used in the original disk-directed I/O papers. In this report, we experimentally re-examine those features to decide which features offer the best performance.

3.1 Memget writes

This section applies only to the traditional parallel file system.

When writing a file, each CP sent write requests to many IOPs. Each request could transfer up to one block of data. The data was included in the request message, so the IOP had to be prepared to accept the data. That requirement led to the static flow-control policies outlined above. Once the data was safely entered into the cache, the IOP replied to the CP, allowing it to make a new

request.

The alternate method did not include data with a write-request message. The IOP had to be prepared to accept the same number of requests, but the requests were substantially smaller. Once the IOP had a chance to process the request, it used Memget requests to fetch the data from the CP's memory. Once the IOP had all of the data from the CP, it replied to the CP, allowing it to make a new request.

We call this alternate "Memget writes". The potential benefits of this approach include reduced memory usage on the IOPs, and possibly higher throughput due to reduced memory-memory copies. (With Memget writes, the data arrived at the IOP in a Memget reply, and was deposited directly into the appropriate cache buffer; without Memget writes, the data arrived in a request message, was copied into a thread stack, and then was copied again into the appropriate cache buffer.)

3.2 Queued requests *vs.* thread requests

This section applies only to the traditional parallel file system.

The IOPs were multi-threaded. There was a permanent thread for each disk, acting as a disk driver. New IOP requests arrived as inter-processor interrupts. The interrupt handler started a new thread, and copied the rest of the interrupt message onto the stack of the new thread, before returning from the interrupt. Thus, each request had its own thread to shepherd the request through the process of checking the cache and waiting for the disk. Thread-creation overhead in STARFISH was quite low, but not insignificant. If the requests were for 8 bytes, as they were in some patterns, thread creation overhead could drag down performance.

The alternate structure had each IOP pre-allocate a pool of threads. The interrupt handler would not create a new thread for each arriving IOP requests, but instead enqueue the request in a queue. Each thread would repeatedly service requests from the queue. The IOPs allocated as many threads as there were buffers in the IOP cache. Thus, there were likely many more threads allocated at one time than there were in the previous structure, which required a lot of memory, but the startup latency for each request was reduced.

We call this structure "queue requests" rather than "thread requests."

4 Experiments

We ran a series of experiments, for each of the above improvements or alternatives, to measure the performance (throughput) of the system under all of the access patterns we used in the other

Table 1: Parameters for simulator.

MIMD, distributed-memory	32 processors
Compute processors (CPs)	16
I/O processors (IOPs)	16
CPU speed, type	50 MHz, RISC
Disks	16
Disk type	HP 97560
Disk capacity	1.3 GB
Disk transfer rate	2.11 MB/s, for multi-track transfers
File-system block size	8 KB
I/O buses (one per IOP)	16
I/O bus type	SCSI
I/O bus peak bandwidth	10 MB/s
Interconnect topology	6×6 torus
Interconnect bandwidth	200×10^6 bytes/s bidirectional
Interconnect latency	20 ns per router
Routing	wormhole
Memput call (IOP)	46-56 cycles
Memput handler (CP)	91 cycles + 1 cycle/word
Memput return (IOP)	72 cycles + thread wakeup
Memget call (IOP)	51-66 cycles
Memget handler (CP)	103 cycles + background DMA
Memget return (IOP)	58 cycles + 1 cycle/word + thread wakeup

disk-directed I/O papers. The parameters for each set of experiments are shown in Table 1. These are the same parameters used in the other disk-directed I/O papers.

4.1 Disk scheduling

We compared FCFS to the new Cyclic-Scan disk-scheduling algorithm, for all of our access patterns, both 8-byte and 8192-byte records, on both contiguous and random disk layouts, for the traditional parallel file system. Again, disk-directed I/O is independent of the choice of disk-scheduling algorithm.

The results for the contiguous disk layout are shown in Table 2. The new disk-scheduling algorithm was nearly always faster; it was 9% slower in the 8-byte `wc` pattern. The most dramatic improvements came in patterns like `rb`, where each CP was working in a different region of the file. With FCFS disk scheduling, the disk head was constantly jumping from one region of the disk to another, defeating the disk's own caching and prefetching, and adding seek and rotational latency to every access. With the new disk-scheduling algorithm, these requests were reordered to allow one a group of requests from one CP, in one region, to be processed before jumping to a new region. The disk schedule was still not optimal, but it was much better. Table 3 compares the number of disk movements (including rotational delays) in the two cases.

The results for the random disk layout are shown in Table 4. The new disk-scheduling algorithm made less difference here, because the random disk layout forced a fairly long seek on every access, and the disk's own cache was useless in either case. Only the 8-byte `wc` pattern was substantially slower, losing 16% of its throughput. Table 5 compares the number of disk movements (including rotational delays) in the two cases; both require a disk movement for each disk access because of the random-access pattern.

The 8-byte `wc` pattern involved the CPs making 8-byte IOP requests, in an interleaved pattern, so that the CPs were working through the file in approximately the same place. Each IOP was slowly building one block at a time, 8 bytes at a time, as each message arrived. As each block became ready, it was stuck on the disk queue. The message-passing and cache processing for the next block, however, took longer than the disk took to finish writing the block! So the disk queue was empty and the disk was idle long before the next block was available to write. Thus, the disk-scheduling policy did not matter: every disk write had substantial seek and/or rotational latency. The throughput appears to be lower due to the increased computational overhead of the new disk-scheduling algorithm.

In any case, we chose to use the new disk-scheduling algorithm in all of the other experiments in this paper, and in our revised disk-directed I/O experiments [Kot96].

Table 2: Throughput of the traditional parallel file system under all access patterns, with both 8- and 8192-byte records, and with both the FCFS and the Cyclic Scan (CS) disk-scheduling algorithm, on the **contiguous** file layout. The throughput of `ra` seems high because it broadcasts the same data to all CPs. The ratio of the throughput of CS to that of FCFS is greater than 1.0 if CS was faster than FCFS.

	Throughput in MB/s					
	8-byte records			8192-byte records		
Pattern	FCFS	CS	CS/FCFS	FCFS	CS	CS/FCFS
`ra`	-	-	-	474.9	500.7	1.05
`rn`	-	-	-	31.4	31.4	1.00
`rb`	7.1	31.1	4.40	7.1	31.1	4.40
`rc`	2.9	2.9	1.00	31.4	31.4	1.00
`rnb`	16.5	16.5	1.00	31.4	31.4	1.00
`rbb`	6.3	6.4	1.02	8.3	14.8	1.78
`rcb`	26.9	26.9	1.00	31.4	31.4	1.00
`rbc`	2.1	2.1	0.99	7.3	7.4	1.02
`rcc`	2.7	3.0	1.10	9.4	14.6	1.55
`rcn`	31.4	31.4	1.00	15.5	31.4	2.02
`wn`	-	-	-	31.1	31.2	1.00
`wb`	9.0	28.8	3.19	9.0	28.8	3.19
`wc`	1.8	1.6	0.91	31.4	31.4	1.00
`wnb`	31.3	31.2	1.00	10.1	30.5	3.02
`wbb`	9.2	27.8	3.01	8.9	29.8	3.33
`wcb`	31.4	31.4	1.00	9.7	30.6	3.14
`wbc`	0.9	0.9	0.99	8.0	29.5	3.69
`wcc`	1.6	1.6	1.01	9.9	29.9	3.01
`wcn`	31.4	31.4	1.00	11.5	29.2	2.53

Table 3: Number of disk movements used by the traditional parallel file system under all access patterns, with both 8- and 8192-byte records, and with both the FCFS and the Cyclic Scan (CS) disk-scheduling algorithm, on the **contiguous** file layout. The ratio of the number of disk movements of CS to that of FCFS is *less than* 1.0 if CS was better than FCFS.

	Number of disk movements					
	8-byte records			8192-byte records		
Pattern	FCFS	CS	CS/FCFS	FCFS	CS	CS/FCFS
`ra`	-	-	-	148	144	0.97
`rn`	-	-	-	144	144	1.00
`rb`	1280	144	0.11	1280	144	0.11
`rc`	1264	1264	1.00	144	144	1.00
`rnb`	37	37	1.00	136	136	1.00
`rbb`	1280	1280	1.00	939	577	0.61
`rcb`	51	51	1.00	144	144	1.00
`rbc`	1274	1248	0.98	1280	1280	1.00
`rcc`	1263	1262	1.00	682	543	0.80
`rcn`	144	144	1.00	563	144	0.26
`wn`	-	-	-	16	16	1.00
`wb`	1176	46	0.04	1176	46	0.04
`wc`	1283	1286	1.00	16	16	1.00
`wnb`	16	16	1.00	885	28	0.03
`wbb`	1280	74	0.06	1218	39	0.03
`wcb`	16	16	1.00	1168	39	0.03
`wbc`	1280	1280	1.00	1280	42	0.03
`wcc`	1288	1290	1.00	1258	42	0.03
`wcn`	16	16	1.00	987	39	0.04

Table 4: Throughput of the traditional parallel file system under all access patterns, with both 8- and 8192-byte records, and with both the FCFS and the Cyclic Scan (CS) disk-scheduling algorithm, on the **random** file layout. The throughput of `ra` seems high because it broadcasts the same data to all CPs. The ratio of the throughput of CS to that of FCFS is greater than 1.0 if CS was faster than FCFS.

	Throughput in MB/s					
	8-byte records			8192-byte records		
Pattern	FCFS	CS	CS/FCFS	FCFS	CS	CS/FCFS
`ra`	-	-	-	72.3	72.3	1.00
`rn`	-	-	-	4.5	4.5	1.00
`rb`	4.3	4.5	1.04	4.3	4.5	1.04
`rc`	2.2	2.2	0.99	4.5	4.5	1.00
`rnb`	3.3	3.3	1.00	4.5	4.5	1.00
`rbb`	3.7	3.8	1.03	4.5	4.4	0.97
`rcb`	4.1	4.1	1.00	4.5	4.5	1.00
`rbc`	2.0	2.0	1.03	4.3	4.4	1.01
`rcc`	2.5	2.4	0.97	4.2	4.3	1.01
`rcn`	4.5	4.5	1.00	4.4	4.5	1.02
`wn`	-	-	-	4.9	4.9	1.00
`wb`	4.9	4.9	0.99	4.9	4.9	0.99
`wc`	1.5	1.3	0.84	4.9	4.9	1.00
`wnb`	4.9	4.9	1.00	4.9	4.9	1.00
`wbb`	5.1	4.9	0.97	5.0	4.9	0.98
`wcb`	4.9	4.9	1.00	5.0	4.9	0.97
`wbc`	0.8	0.8	1.06	4.9	4.9	1.00
`wcc`	1.6	1.6	1.00	4.9	4.9	1.00
`wcn`	4.9	4.9	1.00	5.0	4.9	0.99

Table 5: Number of disk movements used by the traditional parallel file system under all access patterns, with both 8- and 8192-byte records, and with both the FCFS and the Cyclic Scan (CS) disk-scheduling algorithm, on the **random** file layout. The ratio of the number of disk movements of CS to that of FCFS is *less than* 1.0 if CS was better than FCFS.

	Number of disk movements					
	8-byte records			8192-byte records		
Pattern	FCFS	CS	CS/FCFS	FCFS	CS	CS/FCFS
`ra`	-	-	-	1280	1280	1.00
`rn`	-	-	-	1280	1280	1.00
`rb`	1280	1280	1.00	1280	1280	1.00
`rc`	1280	1280	1.00	1280	1280	1.00
`rnb`	1280	1280	1.00	1280	1280	1.00
`rbb`	1279	1280	1.00	1280	1280	1.00
`rcb`	1280	1280	1.00	1280	1280	1.00
`rbc`	1280	1280	1.00	1280	1280	1.00
`rcc`	1280	1280	1.00	1280	1280	1.00
`rcn`	1280	1280	1.00	1280	1280	1.00
`wn`	-	-	-	1280	1280	1.00
`wb`	1280	1280	1.00	1280	1280	1.00
`wc`	1280	1280	1.00	1280	1280	1.00
`wnb`	1280	1280	1.00	1280	1280	1.00
`wbb`	1280	1280	1.00	1280	1280	1.00
`wcb`	1280	1280	1.00	1280	1280	1.00
`wbc`	1280	1280	1.00	1280	1280	1.00
`wcc`	1280	1280	1.00	1280	1280	1.00
`wcn`	1280	1280	1.00	1280	1280	1.00

4.2 Increased number of outstanding requests

This section affects only the traditional parallel file system.

In the original STARFISH, we only allowed one outstanding request from each CP to each disk. In the new STARFISH, we can allow a larger number of requests. In this section we compare various values for parameter K, the maximum number of outstanding requests. Note that the cache size increased as this parameter increased, although the effect of this larger cache is minimal: in these patterns the main benefit comes from a better disk schedule. As mentioned above, we used the new disk-scheduling algorithm in these experiments. We only used access patterns whose chunk size was larger than one block, because other patterns would be limited to one outstanding request per CP, regardless of the new policy.

Figures 2 and 3 show the results for the contiguous disk layout, on two separate graphs for clarity. The larger number of outstanding requests was primarily useful for patterns that had the CPs working in separate regions of the disk, such as `rb`. Each CP could send several requests to each disk, which then better filled the disk queue, and gave the disk scheduler an opportunity to serve more than one or two requests for a given CP before seeking to another region for another CP. The `rbb` pattern never improved because its chunk size was 8, that is, each file-system request was for eight blocks, so there was no room in the request for more than one outstanding request per disk. The `rbb` throughput does improve (not shown) on larger files, where it has larger chunks. Four outstanding requests, or fewer, was sufficient for all patterns to achieve their best performance.

Figures 4 and 5 show the results for the random disk layout, on two separate graphs for clarity. The read access patterns in Figure 4 were helped by allowing more outstanding requests, although in this case `rbb` needed a larger value to obtain its best performance. Figure 4 seems dramatic, but the y-axis scale shows that the variations here are largely in the noise.

Overall, four outstanding requests seemed to be a reasonable compromise. We used four outstanding requests in all of other experiments in this paper, and in our revised disk-directed I/O experiments [Kot96].

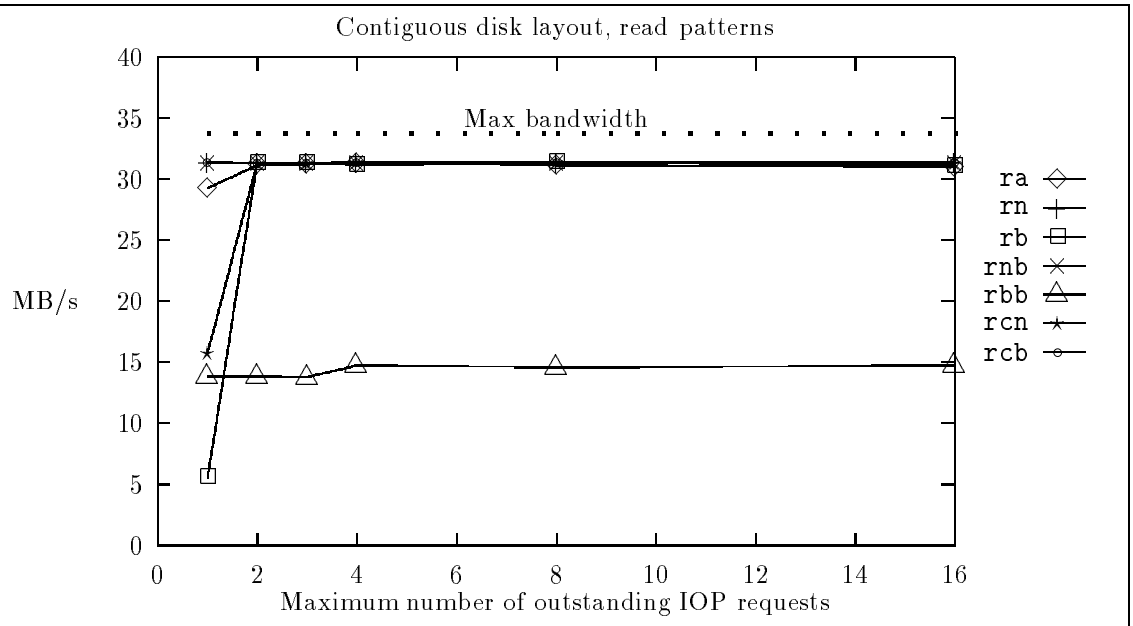

Figure 2: Throughput of the traditional parallel file system under most **read access patterns**, with 8192-byte records, with a varying maximum number of outstanding requests, on the **contiguous** file layout. The throughput of `ra` has been normalized by the number of CPs.

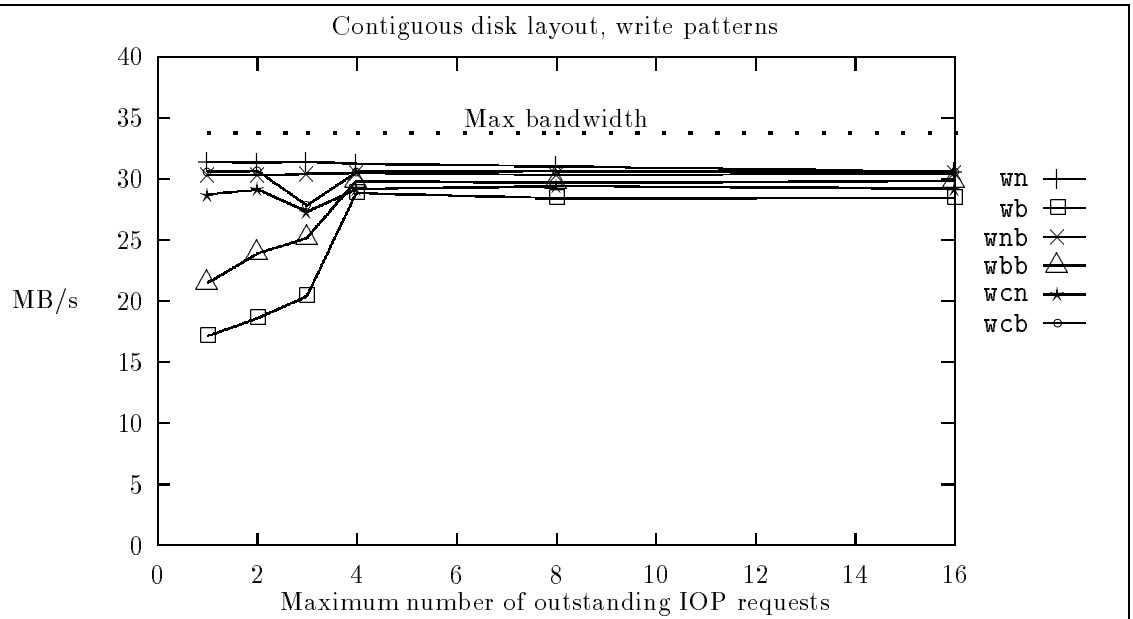

Figure 3: Throughput of the traditional parallel file system under most **write access patterns**, with 8192-byte records, with a varying maximum number of outstanding requests, on the **contiguous** file layout. The throughput of `ra` has been normalized by the number of CPs.

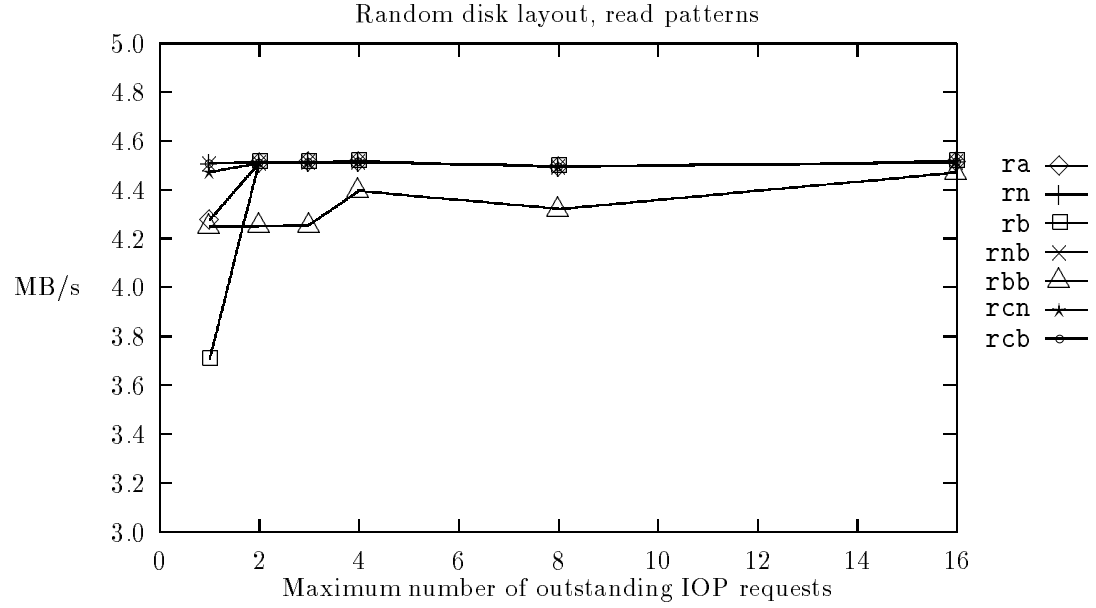

Figure 4: Throughput of the traditional parallel file system under all **read access patterns**, with 8192-byte records, with varying maximum numbers of outstanding requests, on the **random** file layout. The throughput of `ra` has been normalized by the number of CPs. Note that the y-axis is not based at zero.

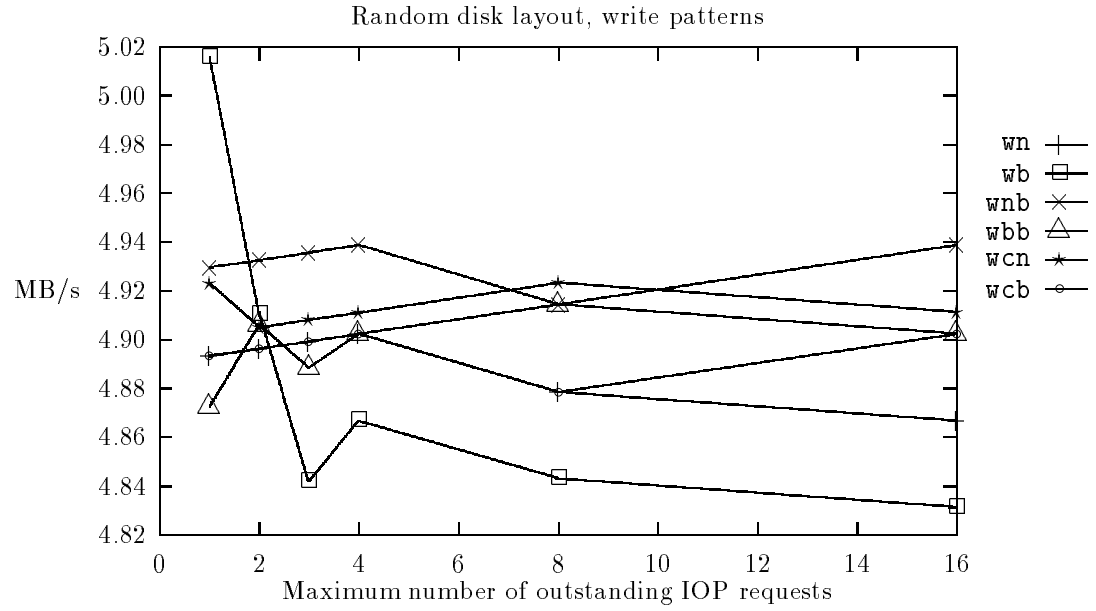

Figure 5: Throughput of the traditional parallel file system under all **write access patterns**, with 8192-byte records, with varying maximum numbers of outstanding requests, on the **random** file layout. The throughput of `ra` has been normalized by the number of CPs. Note that the y-axis is not based at zero, and that its scale is different from Figure 4.

4.3 Queued Memput and Memget

Memput and Memget are only used in disk-directed I/O and in two-phase I/O (we used Memgets and Memputs to implement the permutation phase). So we compare the performance of each, with and without Queued Memputs/Memgets.

Table 6 presents the results for disk-directed I/O and the contiguous disk layout. The throughput was mostly unaffected; the throughput of `ra` was 21% slower, and the throughputs of some patterns with 8-byte chunks were 10–24% faster.

Tables 7 and 8 present the results for the random disk layout, with and without the pre-sorting option of disk-directed I/O. Queued Memputs and Memgets made no difference here, because the data-distribution overhead was completely hidden by the slow disk performance.

Table 9 presents the results for two-phase I/O and the contiguous disk layout. Some 8192-byte patterns were slower, but most 8-byte patterns were faster, by as much as 145%! Three 8-byte patterns were 2–5% slower.

Table 10 presents the results for two-phase I/O and the random disk layout. The 8192-byte patterns were largely unaffected, except for `ra`, but many 8-byte patterns were faster, by as much as 40%.

Clearly, queued Memput and Memget were often, but not always, a good idea. So we chose to use queued Memput and Memget in all 8-byte patterns, but not in any 8192-byte patterns, in all of the other disk-directed I/O experiments in this paper, and in our revised disk-directed I/O experiments [Kot96]. Note that this choice only adversely affected three 8-byte patterns in two-phase I/O, which were slower by 2–5%.

Table 6: Throughput of **disk-directed I/O** under all access patterns, with both 8- and 8192-byte records, with both the original (Or) and queued Memputs/Memgets (QM), on the **contiguous** file layout. The throughput of `ra` seems high because it broadcasts the same data to all CPs. The ratio of the throughput of QM to that of Or is greater than 1.0 if Queued Memputs/Memgets were faster than the original Memputs/Memgets. Ratios in *italics* do not represent a statistically significant difference at the 95% confidence level; all others do.

	Throughput in MB/s					
	8-byte records			8192-byte records		
Pattern	Or	QM	QM/Or	Or	QM	QM/Or
`ra`	-	-	-	500.0	393.8	0.79
`rn`	-	-	-	31.3	31.3	*1.00*
`rb`	31.3	31.2	1.00	31.3	31.2	1.00
`rc`	17.3	20.4	1.18	31.3	31.3	1.00
`rnb`	31.4	31.3	*1.00*	31.3	31.3	1.00
`rbb`	31.4	31.2	1.00	31.3	31.3	1.00
`rcb`	31.4	31.3	1.00	31.3	31.3	1.00
`rbc`	13.5	15.2	1.13	31.3	31.3	1.00
`rcc`	13.7	15.4	1.13	31.3	31.3	1.00
`rcn`	31.3	31.3	1.00	31.3	31.2	1.00
`wn`	-	-	-	31.4	31.4	1.00
`wb`	31.4	31.4	1.00	31.4	31.4	1.00
`wc`	16.5	20.5	1.24	31.4	31.4	*1.00*
`wnb`	31.5	31.4	*1.00*	31.4	31.4	*1.00*
`wbb`	31.5	31.4	*1.00*	31.4	31.4	*1.00*
`wcb`	31.5	31.4	*1.00*	31.4	31.4	*1.00*
`wbc`	14.0	15.4	1.10	31.4	31.4	*1.00*
`wcc`	13.1	15.7	1.20	31.4	31.4	*1.00*
`wcn`	31.4	31.4	*1.00*	31.4	31.4	1.00

Table 7: Throughput of **disk-directed I/O, without sorting** under all access patterns, with both 8- and 8192-byte records, with both the original (Or) and queued Memputs/Memgets (QM), on the **random** file layout. The throughput of `ra` seems high because it broadcasts the same data to all CPs. The ratio of the throughput of QM to that of Or is greater than 1.0 if Queued Memputs/Memgets were faster than the original Memputs/Memgets. Ratios in *italics* do not represent a statistically significant difference at the 95% confidence level; all others do.

	Throughput in MB/s					
	8-byte records			8192-byte records		
Pattern	Or	QM	QM/Or	Or	QM	QM/Or
`ra`	-	-	-	71.7	71.5	1.00
`rn`	-	-	-	4.5	4.5	1.00
`rb`	4.5	4.5	1.00	4.5	4.5	1.00
`rc`	4.5	4.5	1.00	4.5	4.5	1.00
`rnb`	4.5	4.5	1.00	4.5	4.5	1.00
`rbb`	4.5	4.5	1.00	4.5	4.5	1.00
`rcb`	4.5	4.5	1.00	4.5	4.5	1.00
`rbc`	4.5	4.5	1.00	4.5	4.5	1.00
`rcc`	4.5	4.5	1.00	4.5	4.5	1.00
`rcn`	4.5	4.5	1.00	4.5	4.5	1.00
`wn`	-	-	-	4.9	4.9	1.00
`wb`	4.9	4.9	1.00	4.9	4.9	1.00
`wc`	4.9	4.9	*1.00*	4.9	4.9	*1.00*
`wnb`	4.9	4.9	1.00	4.9	4.9	*1.00*
`wbb`	4.9	4.9	1.00	4.9	4.9	1.00
`wcb`	4.9	4.9	1.00	4.9	4.9	1.00
`wbc`	4.9	4.9	*1.00*	4.9	4.9	1.00
`wcc`	4.9	4.9	*1.00*	4.9	4.9	1.00
`wcn`	4.9	4.9	*1.00*	4.9	4.9	*1.00*

Table 8: Throughput of **disk-directed I/O, *with* sorting** under all access patterns, with both 8- and 8192-byte records, with both the original (Or) and queued Memputs/Memgets (QM), on the **random** file layout. The throughput of `ra` seems high because it broadcasts the same data to all CPs. The ratio of the throughput of QM to that of Or is greater than 1.0 if Queued Memputs/Memgets were faster than the original Memputs/Memgets. Ratios in *italics* do not represent a statistically significant difference at the 95% confidence level; all others do.

	Throughput in MB/s					
	8-byte records			8192-byte records		
Pattern	Or	QM	QM/Or	Or	QM	QM/Or
`ra`	-	-	-	100.9	100.6	1.00
`rn`	-	-	-	6.3	6.3	1.00
`rb`	6.3	6.3	1.00	6.3	6.3	1.00
`rc`	6.3	6.3	1.00	6.3	6.3	1.00
`rnb`	6.3	6.3	1.00	6.3	6.3	1.00
`rbb`	6.3	6.3	1.00	6.3	6.3	1.00
`rcb`	6.3	6.3	1.00	6.3	6.3	1.00
`rbc`	6.3	6.3	1.00	6.3	6.3	1.00
`rcc`	6.3	6.3	1.00	6.3	6.3	1.00
`rcn`	6.3	6.3	1.00	6.3	6.3	1.00
`wn`	-	-	-	7.3	7.3	1.00
`wb`	7.3	7.3	1.00	7.3	7.3	1.00
`wc`	7.3	7.3	*1.00*	7.3	7.3	*1.00*
`wnb`	7.3	7.3	*1.00*	7.3	7.3	*1.00*
`wbb`	7.3	7.3	1.00	7.3	7.3	*1.00*
`wcb`	7.3	7.3	*1.00*	7.3	7.3	1.00
`wbc`	7.2	7.2	*1.00*	7.3	7.3	1.00
`wcc`	7.2	7.2	*1.00*	7.3	7.3	*1.00*
`wcn`	7.3	7.3	*1.00*	7.3	7.3	*1.00*

Table 9: Throughput of **two-phase I/O** under all access patterns, with both 8- and 8192-byte records, with both the original (Or) and queued Memputs/Memgets (QM), on the **contiguous** file layout. The throughput of `ra` seems high because it broadcasts the same data to all CPs. The ratio of the throughput of QM to that of Or is greater than 1.0 if Queued Memputs/Memgets were faster than the original Memputs/Memgets. Ratios in *italics* do not represent a statistically significant difference at the 95% confidence level; all others do.

	Throughput in MB/s					
	8-byte records			8192-byte records		
Pattern	Or	QM	QM/Or	Or	QM	QM/Or
`ra`	-	-	-	220.0	169.8	0.77
`rn`	-	-	-	26.4	25.3	0.96
`rb`	31.3	31.3	*1.00*	31.3	31.3	*1.00*
`rc`	5.6	12.9	2.30	28.8	28.2	0.98
`rnb`	27.8	27.3	0.98	26.9	26.3	0.98
`rbb`	29.6	28.0	0.95	27.1	26.2	0.97
`rcb`	29.1	28.0	0.96	26.8	25.9	0.97
`rbc`	6.7	16.5	2.45	27.1	26.2	0.97
`rcc`	5.6	12.8	2.28	26.8	25.9	0.97
`rcn`	28.8	28.2	0.98	26.7	25.6	0.96
`wn`	-	-	-	24.1	15.1	0.63
`wb`	28.7	28.7	*1.00*	28.7	28.7	*1.00*
`wc`	5.9	13.0	2.21	25.7	25.6	0.99
`wnb`	25.0	25.4	1.02	26.4	26.3	*0.99*
`wbb`	27.1	25.7	0.95	26.3	26.3	*1.00*
`wcb`	25.9	25.6	0.99	26.5	26.3	*0.99*
`wbc`	6.8	15.1	2.24	26.3	26.3	*1.00*
`wcc`	5.7	13.0	2.29	26.5	26.3	*0.99*
`wcn`	25.8	25.7	1.00	25.9	25.7	*1.00*

Table 10: Throughput of **two-phase I/O** under all access patterns, with both 8- and 8192-byte records, with both the original (Or) and queued Memputs/Memgets (QM), on the **random** file layout. The throughput of `ra` seems high because it broadcasts the same data to all CPs. The ratio of the throughput of QM to that of Or is greater than 1.0 if Queued Memputs/Memgets were faster than the original Memputs/Memgets. Ratios in *italics* do not represent a statistically significant difference at the 95% confidence level; all others do.

	Throughput in MB/s					
	8-byte records			8192-byte records		
Pattern	Or	QM	QM/Or	Or	QM	QM/Or
`ra`	-	-	-	60.5	56.0	0.93
`rn`	-	-	-	4.4	4.3	0.99
`rb`	4.5	4.5	*1.00*	4.5	4.5	*1.00*
`rc`	2.7	3.7	1.37	4.4	4.4	1.00
`rnb`	4.4	4.4	1.00	4.4	4.4	1.00
`rbb`	4.4	4.4	0.99	4.4	4.4	0.99
`rcb`	4.4	4.4	0.99	4.4	4.4	0.99
`rbc`	2.9	4.0	1.35	4.4	4.4	0.99
`rcc`	2.7	3.7	1.37	4.4	4.4	0.99
`rcn`	4.4	4.4	1.00	4.4	4.4	0.99
`wn`	-	-	-	4.7	4.2	0.90
`wb`	4.9	4.9	*1.00*	4.9	4.9	*1.00*
`wc`	2.9	4.1	1.38	4.8	4.8	*1.00*
`wnb`	4.8	4.8	*1.00*	4.8	4.8	*1.00*
`wbb`	4.8	4.8	0.99	4.8	4.8	*1.00*
`wcb`	4.8	4.8	*1.00*	4.8	4.8	*1.00*
`wbc`	3.1	4.2	1.35	4.8	4.8	*1.00*
`wcc`	2.9	4.1	1.40	4.8	4.8	*1.00*
`wcn`	4.8	4.8	*1.00*	4.8	4.8	*1.00*

4.4 Memget writes

We compared our original scheme, in which CPs wanting to write data sent the data as part of the IOP requests, to the "Memget writes" scheme, in which CPs sent only the request, and waited for the IOP to use Memget to fetch the data.

Table 11 presents the results on the contiguous disk layout. Although Memget writes were slightly (1–3%) faster in a few cases, they were often slower than the current plan, especially for patterns with 8-byte chunks.

Table 12 presents the results on the random disk layout, and is no more optimistic.

As a result, we chose to retain the original scheme, rather than using Memget writes, in the other experiments in this paper and in our revised disk-directed I/O experiments [Kot96].

Table 11: Throughput of the traditional parallel file system under all access patterns, with both 8- and 8192-byte records, and with both the original (Or) and the "Memget writes" (MW), on the **contiguous** file layout. The throughput of `ra` seems high because it broadcasts the same data to all CPs. The ratio of the throughput of MW to that of Or is greater than 1.0 if Memget writes were faster than the original scheme. Ratios in *italics* do not represent a statistically significant difference at the 95% confidence level; all others do.

	Throughput in MB/s					
	8-byte records			8192-byte records		
Pattern	Or	MW	MW/Or	Or	MW	MW/Or
`ra`	-	-	-	499.7	499.3	*1.00*
`rn`	-	-	-	31.4	31.4	*1.00*
`rb`	31.3	31.3	*1.00*	31.3	31.3	*1.00*
`rc`	2.8	2.7	*0.96*	31.4	31.3	*1.00*
`rnb`	16.5	16.4	*1.00*	31.4	31.3	*1.00*
`rbb`	6.4	6.4	*1.00*	14.7	14.5	*0.99*
`rcb`	26.6	26.6	*1.00*	31.4	31.4	*1.00*
`rbc`	2.1	2.1	*1.00*	7.4	7.4	*1.00*
`rcc`	3.0	2.8	0.95	14.5	14.5	*1.00*
`rcn`	31.4	31.3	*1.00*	31.3	31.3	*1.00*
`wn`	-	-	-	31.2	31.3	*1.00*
`wb`	28.7	29.6	1.03	28.7	29.6	1.03
`wc`	1.9	0.3	0.19	31.4	31.4	*1.00*
`wnb`	31.3	17.7	0.56	30.4	30.5	*1.00*
`wbb`	27.9	22.2	0.80	29.8	30.1	1.01
`wcb`	31.4	31.4	*1.00*	30.4	30.6	*1.01*
`wbc`	0.8	0.3	0.38	29.6	29.6	*1.00*
`wcc`	1.6	1.3	0.83	29.9	30.1	*1.01*
`wcn`	31.4	31.4	*1.00*	29.1	30.0	1.03

Table 12: Throughput of the traditional parallel file system under all access patterns, with both 8- and 8192-byte records, and with both the original (Or) and the "Memget writes" (MW), on the **random** file layout. The throughput of `ra` seems high because it broadcasts the same data to all CPs. The ratio of the throughput of MW to that of Or is greater than 1.0 if Memget writes were faster than the original scheme. Ratios in *italics* do not represent a statistically significant difference at the 95% confidence level; all others do.

	Throughput in MB/s					
	8-byte records			8192-byte records		
Pattern	Or	MW	MW/Or	Or	MW	MW/Or
`ra`	-	-	-	71.6	71.6	1.00
`rn`	-	-	-	4.5	4.5	1.00
`rb`	4.5	4.5	1.00	4.5	4.5	1.00
`rc`	2.2	2.2	*0.99*	4.5	4.5	1.00
`rnb`	3.3	3.3	1.00	4.5	4.5	1.00
`rbb`	3.8	3.8	1.00	4.5	4.5	1.00
`rcb`	4.1	4.1	1.00	4.5	4.5	1.00
`rbc`	2.0	2.1	1.03	4.3	4.3	1.00
`rcc`	2.5	2.5	*1.01*	4.2	4.2	1.00
`rcn`	4.5	4.5	1.00	4.5	4.5	1.00
`wn`	-	-	-	4.9	4.9	*1.00*
`wb`	4.9	4.9	*1.00*	4.9	4.9	*1.00*
`wc`	1.6	0.3	0.23	4.9	4.9	1.00
`wnb`	4.9	4.9	*1.00*	4.9	4.9	1.00
`wbb`	4.9	4.9	*1.00*	4.9	4.9	*1.00*
`wcb`	4.9	4.9	1.00	4.9	4.9	*1.00*
`wbc`	0.8	0.3	0.39	4.9	4.9	*1.00*
`wcc`	1.6	1.3	0.83	5.0	4.9	*1.00*
`wcn`	4.9	4.9	1.00	4.9	4.9	*1.01*

4.5 Queued requests *vs.* thread requests

We compared the original "thread requests," in which each incoming IOP request was allocated a new thread, with "queued requests," in which each each incoming IOP request was matched with an existing thread from a pool of pre-allocated, reusable threads.

Table 13 shows the results for the contiguous disk layout. As expected, this change only affects 8-byte access patterns, indeed, only the patterns with 8-byte chunks (`rc`, `rbc`, `rcc`, `wc`, `wbc`, `wcc`), because only they have a tremendous number of tiny requests. The cost of thread creation is dominant in those patterns, and the queued-request system works better there, with no adverse affects in the other cases.

Table 13 shows the results for the random disk layout. There are some larger improvements (36%), but one case (`wc` with 8-byte records) is 7% slower.

As a result we chose to use queued requests in all of the other experiments in this paper, and for our revised disk-directed I/O experiments [Kot96].

Table 13: Throughput of the traditional parallel file system under all access patterns, with both 8- and 8192-byte records, and with both Thread Requests (TR) and Queued Requests (QR) on the **contiguous** file layout. The throughput of `ra` seems high because it broadcasts the same data to all CPs. The ratio of the throughput of QR to that of TR is greater than 1.0 if Queued Requests were faster than the Thread Requests.

	Throughput in MB/s					
	8-byte records			8192-byte records		
Pattern	TR	QR	QR/TR	TR	QR	QR/TR
`ra`	-	-	-	498.0	500.7	1.01
`rn`	-	-	-	31.4	31.4	1.00
`rb`	31.0	31.1	1.00	31.0	31.1	1.00
`rc`	2.8	2.9	1.04	31.4	31.4	1.00
`rnb`	16.5	16.5	1.00	31.3	31.4	1.00
`rbb`	6.4	6.4	1.00	14.8	14.8	1.00
`rcb`	26.9	26.9	1.00	31.4	31.4	1.00
`rbc`	1.8	2.1	1.14	7.4	7.4	1.00
`rcc`	2.1	3.0	1.39	14.5	14.6	1.00
`rcn`	31.4	31.4	1.00	31.2	31.4	1.01
`wn`	-	-	-	31.2	31.2	1.00
`wb`	28.8	28.8	1.00	28.8	28.8	1.00
`wc`	1.5	1.6	1.07	31.4	31.4	1.00
`wnb`	31.2	31.2	1.00	30.5	30.5	1.00
`wbb`	27.6	27.8	1.00	29.8	29.8	1.00
`wcb`	31.5	31.4	1.00	30.5	30.6	1.00
`wbc`	0.6	0.9	1.40	29.5	29.5	1.00
`wcc`	1.5	1.6	1.06	29.8	29.9	1.00
`wcn`	31.4	31.4	1.00	29.1	29.2	1.00

Table 14: Throughput of the traditional parallel file system under all access patterns, with both 8- and 8192-byte records, and with both Thread Requests (TR) and Queued Requests (QR) on the **random** file layout. The throughput of `ra` seems high because it broadcasts the same data to all CPs. The ratio of the throughput of QR to that of TR is greater than 1.0 if Queued Requests were faster than the Thread Requests.

	Throughput in MB/s					
	8-byte records			8192-byte records		
Pattern	TR	QR	QR/TR	TR	QR	QR/TR
`ra`	-	-	-	72.2	72.3	1.00
`rn`	-	-	-	4.5	4.5	1.00
`rb`	4.5	4.5	1.00	4.5	4.5	1.00
`rc`	2.1	2.2	1.04	4.5	4.5	1.00
`rnb`	3.3	3.3	1.00	4.5	4.5	1.00
`rbb`	3.8	3.8	1.00	4.4	4.4	1.00
`rcb`	4.1	4.1	1.00	4.5	4.5	1.00
`rbc`	1.7	2.0	1.21	4.4	4.4	1.00
`rcc`	2.3	2.4	1.07	4.3	4.3	1.00
`rcn`	4.5	4.5	1.00	4.5	4.5	1.00
`wn`	-	-	-	4.9	4.9	1.00
`wb`	4.9	4.9	1.00	4.9	4.9	1.00
`wc`	1.4	1.3	0.93	4.9	4.9	1.00
`wnb`	4.9	4.9	1.00	4.9	4.9	1.00
`wbb`	4.9	4.9	1.00	4.9	4.9	1.00
`wcb`	4.9	4.9	1.00	4.9	4.9	1.00
`wbc`	0.6	0.8	1.36	4.9	4.9	1.00
`wcc`	1.5	1.6	1.06	4.9	4.9	1.00
`wcn`	4.9	4.9	1.00	4.9	4.9	1.00

5 Conclusions

We experimentally examined five major features in the STARFISH 3.0 parallel file-system simulator:

Disk scheduling: It was clear that the new Cyclic Scan disk-scheduling algorithm was much better than our old FCFS algorithm. We used that algorithm for all other experiments in this paper and in our revised disk-directed I/O experiments [Kot96].

Increased number of outstanding requests: Allowing more outstanding requests increased the throughput of the traditional parallel file system on many access patterns with large chunks, because the resulting deep disk queues permitted better disk scheduling. Since two-phase I/O uses `rb`, and `rb` benefited a lot from this change, two-phase I/O would also benefit greatly from this change. Four outstanding requests appeared to be a good maximum; few patterns improved beyond this point. We used four outstanding requests for other disk-directed I/O experiments in this paper and in our revised disk-directed I/O experiments [Kot96].

Queued Memput and Memget: The queued Memput and Memget functions were helpful in both two-phase I/O and disk-directed I/O, but only for 8-byte, not 8192-byte, access patterns. Thus, we chose to use queued Memput and Memget for 8-byte access patterns only, and we did so in all other experiments in this paper and in our revised disk-directed I/O experiments [Kot96].

Memget writes: Memget writes rarely increased throughput, the increases were small, and the decreases were often dramatic. We chose not to use them in any other experiments.

Queued requests *vs.* thread requests: The overhead of thread creation was clearly a drag on performance for those patterns that made a lot of IOP requests, i.e., those with 8-byte chunks. The alternate implementation, queued requests, which kept a pool of ready and reusable threads, was much faster for those patterns, and no slower for the other patterns. We chose to use queued requests in all other experiments in this paper and in our revised disk-directed I/O experiments [Kot96].

5.1 Comparing file systems

Based on the above conclusions, we revised our earlier experiments, in which we compare the traditional parallel file system, two-phase I/O, and disk-directed I/O. All of these experiments use the parameters from Table 1, and the features described in the conclusions above. The results are graphed and examined in detail in the full paper [Kot96], but the raw data is presented below in Tables 15–21.

Table 15: A comparison of the throughput of disk-directed I/O (**DDIO**) and the traditional parallel file system (**TPFS**), on a **contiguous** disk layout. `ra` throughput has been normalized by the number of CPs. Each point represents the average of five trials of an access pattern on both methods (maximum cv is 0.23, though most are less than 0.090). A ratio greater than 1.00 means that disk-directed I/O was faster than the traditional parallel file system in that case. Ratios in *italics* do not represent a statistically significant difference at the 95% confidence level; all others do. Note that the peak disk throughput was 33.8 MB/s.

	Throughput in MB/s					
	8-byte records			8192-byte records		
Pattern	TPFS	DDIO	DDIO/TPFS	TPFS	DDIO	DDIO/TPFS
`ra`	-	-	-	499.7	500.0	*1.00*
`rn`	-	-	-	31.4	31.3	*1.00*
`rb`	31.3	31.2	*1.00*	31.3	31.3	*1.00*
`rc`	2.8	20.4	7.28	31.4	31.3	*1.00*
`rnb`	16.5	31.3	1.90	31.4	31.3	*1.00*
`rbb`	6.4	31.2	4.89	14.7	31.3	2.14
`rcb`	26.6	31.3	1.18	31.4	31.3	*1.00*
`rbc`	2.1	15.2	7.39	7.4	31.3	4.23
`rcc`	3.0	15.4	5.14	14.5	31.3	2.17
`rcn`	31.4	31.3	*1.00*	31.3	31.3	*1.00*
`wn`	-	-	-	31.2	31.4	1.01
`wb`	28.7	31.4	1.09	28.7	31.4	1.09
`wc`	1.9	20.5	11.53	31.4	31.4	*1.00*
`wnb`	31.3	31.4	1.00	30.4	31.4	1.03
`wbb`	27.9	31.4	1.13	29.8	31.4	1.05
`wcb`	31.4	31.4	*1.00*	30.4	31.4	1.03
`wbc`	0.8	15.4	18.11	29.6	31.4	1.06
`wcc`	1.6	15.7	9.96	29.9	31.4	1.05
`wcn`	31.4	31.4	*1.00*	29.1	31.4	1.08

Table 16: A comparison of the throughput of disk-directed I/O (**DDIO**) to that of the traditional parallel file system (**TPFS**), on a **random-blocks** disk layout. DDIOs represents disk-directed I/O with the block-list presort. `ra` throughput has been normalized by the number of CPs. Each point represents the average of five trials of an access pattern on both methods (maximum coefficient of variation (cv) is 0.25, although most were less than 0.042). The column r is the ratio of DDIO (or DDIOs) to TPFS. If $r > 1$, disk-directed I/O was faster than the traditional parallel file system in that case. Ratios in *italics* do not represent a statistically significant difference at the 95% confidence level; all others do.

	Throughput in MB/s									
	8-byte records					8192-byte records				
Pattern	TPFS	DDIO	r	DDIOs	r	TPFS	DDIO	r	DDIOs	r
`ra`	-	-	-	-	-	71.6	71.7	*1.00*	100.9	1.41
`rn`	-	-	-	-	-	4.5	4.5	*1.00*	6.3	1.41
`rb`	4.5	4.5	*1.00*	6.3	1.41	4.5	4.5	*1.00*	6.3	1.41
`rc`	2.2	4.5	2.01	6.3	2.82	4.5	4.5	*1.01*	6.3	1.42
`rnb`	3.3	4.5	1.37	6.3	1.93	4.5	4.5	*1.01*	6.3	1.42
`rbb`	3.8	4.5	1.18	6.3	1.66	4.5	4.5	*1.00*	6.3	1.41
`rcb`	4.1	4.5	1.10	6.3	1.55	4.5	4.5	*1.00*	6.3	1.41
`rbc`	2.0	4.5	2.23	6.3	3.14	4.3	4.5	1.04	6.3	1.46
`rcc`	2.5	4.5	1.79	6.3	2.52	4.2	4.5	1.06	6.3	1.49
`rcn`	4.5	4.5	*1.01*	6.3	1.42	4.5	4.5	*1.00*	6.3	1.41
`wn`	-	-	-	-	-	4.9	4.9	*1.00*	7.3	1.49
`wb`	4.9	4.9	*1.00*	7.3	1.49	4.9	4.9	*1.00*	7.3	1.49
`wc`	1.6	4.9	3.29	7.3	4.85	4.9	4.9	*1.00*	7.3	1.49
`wnb`	4.9	4.9	*1.00*	7.3	1.49	4.9	4.9	*1.00*	7.3	1.48
`wbb`	4.9	4.9	*1.00*	7.3	1.49	4.9	4.9	*1.00*	7.3	1.48
`wcb`	4.9	4.9	*1.00*	7.3	1.49	4.9	4.9	*1.00*	7.3	1.49
`wbc`	0.8	4.9	5.88	7.2	8.70	4.9	4.9	*1.00*	7.3	1.49
`wcc`	1.6	4.9	3.12	7.2	4.62	5.0	4.9	0.99	7.3	1.48
`wcn`	4.9	4.9	*1.00*	7.3	1.49	4.9	4.9	*1.00*	7.3	1.49

Table 17: A comparison of the throughput of two-phase I/O (**2PIO**) and disk-directed I/O (**DDIO**), on a **contiguous** disk layout. `ra` throughput has been normalized by the number of CPs. Each point represents the average of five trials of an access pattern on both methods (maximum cv is 0.012). A ratio greater than 1.00 means that disk-directed I/O was faster than two-phase I/O in that case. Ratios in *italics* do not represent a statistically significant difference at the 95% confidence level; all others do. Note that the peak disk throughput was 33.8 MB/s.

	Throughput in MB/s					
	8-byte records			8192-byte records		
Pattern	2PIO	DDIO	DDIO/2PIO	2PIO	DDIO	DDIO/2PIO
`ra`	-	-	-	220.0	500.0	2.27
`rn`	-	-	-	26.4	31.3	1.19
`rb`	31.3	31.2	*1.00*	31.3	31.3	*1.00*
`rc`	12.9	20.4	1.57	28.8	31.3	1.09
`rnb`	27.3	31.3	1.15	26.9	31.3	1.16
`rbb`	28.0	31.2	1.11	27.1	31.3	1.16
`rcb`	28.0	31.3	1.12	26.8	31.3	1.17
`rbc`	16.5	15.2	0.92	27.1	31.3	1.16
`rcc`	12.8	15.4	1.20	26.8	31.3	1.17
`rcn`	28.2	31.3	1.11	26.7	31.3	1.17
`wn`	-	-	-	24.1	31.4	1.31
`wb`	28.7	31.4	1.09	28.7	31.4	1.09
`wc`	13.0	20.5	1.58	25.7	31.4	1.22
`wnb`	25.4	31.4	1.24	26.4	31.4	1.19
`wbb`	25.7	31.4	1.22	26.3	31.4	1.20
`wcb`	25.6	31.4	1.23	26.5	31.4	1.19
`wbc`	15.1	15.4	1.02	26.3	31.4	1.20
`wcc`	13.0	15.7	1.21	26.5	31.4	1.19
`wcn`	25.7	31.4	1.23	25.9	31.4	1.22

Table 18: A comparison of the throughput of two-phase I/O (**2PIO**) and disk-directed I/O *without* the block-list presort (**DDIO**), on a **random** disk layout. `ra` throughput has been normalized by the number of CPs. Each point represents the average of five trials of an access pattern on both methods (maximum *cv* is 0.010). A ratio greater than 1.00 means that disk-directed I/O was faster than two-phase I/O in that case. Ratios in *italics* do not represent a statistically significant difference at the 95% confidence level; all others do.

	Throughput in MB/s					
	8-byte records			8192-byte records		
Pattern	2PIO	DDIO	DDIO/2PIO	2PIO	DDIO	DDIO/2PIO
`ra`	-	-	-	60.5	71.7	1.19
`rn`	-	-	-	4.4	4.5	1.03
`rb`	4.5	4.5	*1.00*	4.5	4.5	*1.00*
`rc`	3.7	4.5	1.20	4.4	4.5	1.01
`rnb`	4.4	4.5	1.02	4.4	4.5	1.02
`rbb`	4.4	4.5	1.02	4.4	4.5	1.02
`rcb`	4.4	4.5	1.02	4.4	4.5	1.02
`rbc`	4.0	4.5	1.13	4.4	4.5	1.02
`rcc`	3.7	4.5	1.20	4.4	4.5	1.02
`rcn`	4.4	4.5	1.02	4.4	4.5	1.02
`wn`	-	-	-	4.7	4.9	1.04
`wb`	4.9	4.9	*1.00*	4.9	4.9	*1.00*
`wc`	4.1	4.9	1.21	4.8	4.9	1.03
`wnb`	4.8	4.9	1.03	4.8	4.9	1.02
`wbb`	4.8	4.9	1.03	4.8	4.9	1.02
`wcb`	4.8	4.9	1.03	4.8	4.9	1.02
`wbc`	4.2	4.9	1.15	4.8	4.9	1.02
`wcc`	4.1	4.9	1.21	4.8	4.9	1.02
`wcn`	4.8	4.9	1.03	4.8	4.9	1.02

Table 19: A comparison of the throughput of two-phase I/O (**2PIO**) and disk-directed I/O *with* the block-list presort (**DDIOs**), on a **random** disk layout. `ra` throughput has been normalized by the number of CPs. Each point represents the average of five trials of an access pattern on both methods (maximum *cv* is 0.031). A ratio greater than 1.00 means that disk-directed I/O was faster than two-phase I/O in that case. Ratios in *italics* do not represent a statistically significant difference at the 95% confidence level; all others do.

	Throughput in MB/s					
	8-byte records			8192-byte records		
Pattern	2PIO	DDIOs	DDIOs/2PIO	2PIO	DDIOs	DDIOs/2PIO
`ra`	-	-	-	60.5	100.9	1.67
`rn`	-	-	-	4.4	6.3	1.45
`rb`	4.5	6.3	1.41	4.5	6.3	1.41
`rc`	3.7	6.3	1.69	4.4	6.3	1.43
`rnb`	4.4	6.3	1.44	4.4	6.3	1.43
`rbb`	4.4	6.3	1.43	4.4	6.3	1.43
`rcb`	4.4	6.3	1.43	4.4	6.3	1.43
`rbc`	4.0	6.3	1.58	4.4	6.3	1.43
`rcc`	3.7	6.3	1.69	4.4	6.3	1.43
`rcn`	4.4	6.3	1.43	4.4	6.3	1.43
`wn`	-	-	-	4.7	7.3	1.55
`wb`	4.9	7.3	1.49	4.9	7.3	1.49
`wc`	4.1	7.3	1.79	4.8	7.3	1.53
`wnb`	4.8	7.3	1.53	4.8	7.3	1.52
`wbb`	4.8	7.3	1.53	4.8	7.3	1.52
`wcb`	4.8	7.3	1.53	4.8	7.3	1.51
`wbc`	4.2	7.2	1.71	4.8	7.3	1.52
`wcc`	4.1	7.2	1.79	4.8	7.3	1.51
`wcn`	4.8	7.3	1.53	4.8	7.3	1.52

Table 20: A comparison of the throughput of the traditional parallel file system (**TPFS**) and two-phase I/O (**2PIO**), on a **contiguous** disk layout. `ra` throughput has been normalized by the number of CPs. Each point represents the average of five trials of an access pattern on both methods (maximum cv is 0.24, although most are less than 0.090). A ratio greater than 1.00 means that two-phase I/O was faster than the traditional parallel file system in that case. Ratios in *italics* do not represent a statistically significant difference at the 95% confidence level; all others do.

	Throughput in MB/s					
	8-byte records			8192-byte records		
Pattern	TPFS	2PIO	2PIO/TPFS	TPFS	2PIO	2PIO/TPFS
`ra`	-	-	-	499.7	220.0	0.44
`rn`	-	-	-	31.4	26.4	0.84
`rb`	31.3	31.3	*1.00*	31.3	31.3	*1.00*
`rc`	2.8	12.9	4.62	31.4	28.8	0.92
`rnb`	16.5	27.3	1.65	31.4	26.9	0.86
`rbb`	6.4	28.0	4.39	14.7	27.1	1.85
`rcb`	26.6	28.0	1.05	31.4	26.8	0.86
`rbc`	2.1	16.5	8.00	7.4	27.1	3.66
`rcc`	3.0	12.8	4.27	14.5	26.8	1.86
`rcn`	31.4	28.2	0.90	31.3	26.7	0.85
`wn`	-	-	-	31.2	24.1	0.77
`wb`	28.7	28.7	*1.00*	28.7	28.7	*1.00*
`wc`	1.9	13.0	7.32	31.4	25.7	0.82
`wnb`	31.3	25.4	0.81	30.4	26.4	0.87
`wbb`	27.9	25.7	0.92	29.8	26.3	0.88
`wcb`	31.4	25.6	0.81	30.4	26.5	0.87
`wbc`	0.8	15.1	17.84	29.6	26.3	0.89
`wcc`	1.6	13.0	8.24	29.9	26.5	0.89
`wcn`	31.4	25.7	0.82	29.1	25.9	0.89

Table 21: A comparison of the throughput of the traditional parallel file system (**TPFS**) and two-phase I/O (**2PIO**), on a **random** disk layout. `ra` throughput has been normalized by the number of CPs. Each point represents the average of five trials of an access pattern on both methods (maximum *cv* is 0.25, although most were less than 0.042). A ratio greater than 1.00 means that two-phase I/O was faster than the traditional parallel file system in that case. Ratios in *italics* do not represent a statistically significant difference at the 95% confidence level; all others do.

	Throughput in MB/s					
	8-byte records			8192-byte records		
Pattern	TPFS	2PIO	2PIO/TPFS	TPFS	2PIO	2PIO/TPFS
`ra`	-	-	-	71.6	60.5	0.84
`rn`	-	-	-	4.5	4.4	0.97
`rb`	4.5	4.5	*1.00*	4.5	4.5	*1.00*
`rc`	2.2	3.7	1.67	4.5	4.4	*0.99*
`rnb`	3.3	4.4	1.34	4.5	4.4	*0.99*
`rbb`	3.8	4.4	1.16	4.5	4.4	*0.98*
`rcb`	4.1	4.4	1.08	4.5	4.4	0.98
`rbc`	2.0	4.0	1.98	4.3	4.4	1.02
`rcc`	2.5	3.7	1.49	4.2	4.4	1.04
`rcn`	4.5	4.4	0.99	4.5	4.4	0.98
`wn`	-	-	-	4.9	4.7	0.96
`wb`	4.9	4.9	*1.00*	4.9	4.9	*1.00*
`wc`	1.6	4.1	2.71	4.9	4.8	0.97
`wnb`	4.9	4.8	0.97	4.9	4.8	0.98
`wbb`	4.9	4.8	0.97	4.9	4.8	0.98
`wcb`	4.9	4.8	0.97	4.9	4.8	0.98
`wbc`	0.8	4.2	5.09	4.9	4.8	0.98
`wcc`	1.6	4.1	2.58	5.0	4.8	0.98
`wcn`	4.9	4.8	0.97	4.9	4.8	0.98

References

[BDCW91] Eric A. Brewer, Chrysanthos N. Dellarocas, Adrian Colbrook, and William E. Weihl. Proteus: A high-performance parallel-architecture simulator. Technical Report MIT/LCS/TR–516, MIT, September 1991.

[KC95] David Kotz and Ting Cai. Exploring the use of I/O nodes for computation in a MIMD multiprocessor. In *IPPS '95 Workshop on Input/Output in Parallel and Distributed Systems*, pages 78–89, April 1995.

[Kot94a] David Kotz. Disk-directed I/O for MIMD multiprocessors. In *Proceedings of the 1994 Symposium on Operating Systems Design and Implementation*, pages 61–74, November 1994. Updated as Dartmouth TR PCS-TR94-226 on November 8, 1994.

[Kot94b] David Kotz. Disk-directed I/O for MIMD multiprocessors. Technical Report PCS-TR94-226, Dept. of Computer Science, Dartmouth College, July 1994. Revised November 8, 1994.

[Kot95a] David Kotz. Disk-directed I/O for an out-of-core computation. In *Proceedings of the Fourth IEEE International Symposium on High Performance Distributed Computing*, pages 159–166, August 1995.

[Kot95b] David Kotz. Expanding the potential for disk-directed I/O. In *Proceedings of the 1995 IEEE Symposium on Parallel and Distributed Processing*, pages 490–495, October 1995.

[Kot95c] David Kotz. Interfaces for disk-directed I/O. Technical Report PCS-TR95-270, Dept. of Computer Science, Dartmouth College, September 1995.

[Kot96] David Kotz. Disk-directed I/O for MIMD multiprocessors. Submitted to TOCS, October 1996.

[PEK96] Apratim Purakayastha, Carla Schlatter Ellis, and David Kotz. ENWRICH: a compute-processor write caching scheme for parallel file systems. In *Fourth Workshop on Input/Output in Parallel and Distributed Systems*, pages 55–68, May 1996.

[SCO90] Margo Seltzer, Peter Chen, and John Ousterhout. Disk scheduling revisited. In *Proceedings of the 1990 Winter USENIX Conference*, pages 313–324, 1990.