An Evaluation of Multiple-Disk I/O Systems

A. L. NARASIMHA REDDY AND PRITHVIRAJ BANERJEE, MEMBER, IEEE

Abstract—**With increasing processor speeds and multiprocessor organizations, the demands on the I/O subsystem have been steadily growing. To obtain speedups that scale with the processing speeds, it is necessary to improve the I/O performance. In this paper, we consider alternatives to building an I/O subsystem with multiple disks to improve the I/O performance. There exist several alternative ways to configure a number of disks to achieve higher performance. Specifically, we consider disk synchronization, data declustering/disk striping, and a combination of both these approaches. We evaluate many different organizations that have not been considered before. The effects of block size and other parameters of the system are evaluated. Two different workloads are considered for evaluation: 1) file/transaction system workload, 2) scientific applications workload. Through simulations, it is shown that synchronized organizations perform better than other organizations at very low request rates. It is shown that there is a tradeoff in the amount of declustering/synchronization to be used in a system. It is shown that systems with higher parallelism in reading a file perform better in a scientific workload. We conclude the paper with a number of questions for future research into improving the I/O performance.**

Index Terms—**File systems, I/O systems, mutiple-disk systems, multiprocessors, performance evaluation, scientific applications.**

I. Introduction

In THE last few years, the performance of processors has been growing steadily, and with multiprocessor organizations, the processing power of a computer system has been increasing at a rapid pace. However, the I/O performance has not kept pace with the gains in processing power. Currently, there exists a huge disparity between the I/O subsystem's performance and the processing power. With such differences in processing and I/O capacities, eventually the problem solving speed will be determined by how fast the I/O can be done. If the I/O is done serially, even if we solve the problem considerably faster, the speed of the solution will depend on the serial I/O bottleneck. So it is essential that some kind of improvements be made in I/O performance to avoid these bottlenecks. The importance of balancing the I/O bandwidth and the computational power has been pointed out by Kung [1], where it is shown that for some of the applications, the I/O problem cannot be alleviated by adding more memory at the processors. For such applications, it is essential that the I/O power grows as fast as the processing power of the system. Unless the I/O can also be speeded up by the same factor as that of the processing, we cannot truly obtain linear speedups. Hence, to improve the overall performance of a system, it is essential that the I/O subsystem performance also has to be improved. The intrinsic I/O requirements of some problems are dealt with in [2] and [3].

The data rates of disks are limited by the speeds of physical motion of heads and it is unlikely that these speeds will improve dramatically in the future. Hence, there have been a number of proposals to organize a number of disks together to form a powerful disk system. Some of the previous studies in improving I/O performance include [4]–[7]. We will briefly describe these approaches in Section II.

Studies by Livny *et al.* [7] and Salem *et al.* [4] report that a system with data declustering is better than a traditional system. Livny *et al.* showed that when the block of I/O transfer is a track, a declustered system outperforms a traditional system. They did not consider the effects of the block size on the performance of the system. Similarly, Kim [5] shows that disk synchronization leads to efficient transfer of large blocks of data when compared to a traditional system. Kim only considered a system with specific utilizations of various disks and did not consider the effect of different workloads.

The above studies have not considered the effects of all the parameters of the system, viz., size of the basic block of transfer, different workloads, etc. Moreover, there are many other possible disk organizations, as will be shown in Section II, that need to be evaluated. A comparison of relative performances also remains to be done to see which one of the several alternatives is a better choice. In this paper, we present extensive results from simulations to show the effects of different parameters on the performance of the system.

The rest of the paper is organized as follows. Section II describes the various alternatives available for organizing disks. Section III presents the details of workloads considered in the simulations. Section IV contains the results from simulations of the file/transaction workload and Section V reports the results of the scientific workload. Section VI discusses several issues regarding building multiple disk systems in the light of the results obtained from the simulations. Section VII makes conclusions about the obtained data and presents the future work that needs to be done.

II. Available Alternatives

With each disk I/O request, there are three main components of service time, namely, seek time, latency time, and read time. Seek time is the time spent in accessing the right

Manuscript received May 15, 1989; revised July 10, 1989. This work was supported in part by an IBM Graduate Fellowship and in part by a National Science Foundation Presidential Young Investigator Award under Contract NSF MIP 86-57563 PYI.

The authors are with the Department of Electrical and Computer Engineering and the Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, Urbana, IL 61801.

IEEE Log Number 8931172.

track of the disk and latency time is the time taken for the required block to come under the read/write head. Read time is the time taken for reading the data from the disk to a buffer. Latency and seek times are overheads incurred in reading a block of data. The response time of a request is the sum of its service time and the waiting time in the queue.

Disk synchronization [5] is useful when file read time is the dominant part of the I/O service time. In this organization, a file is interleaved bytewise among the disks. The disks are synchronized such that each disk head is positioned at the same place. All the disks together function as a single large disk with m times the transfer rate and m times the capacity of a single disk, where the seek time and latency time remain the same. When the transfer time dominates the I/O service time, this organization yields high performance. Since all the disks are coupled together, each disk sees the same request rate and hence a balanced load on all the disks. The utilization of each disk is higher since each request has to be processed by all the disks. Hence, this organization is useful when large files need to be transferred. A system that employs this organization is defined to be a *synchronized system* in this paper. Some of the existing disk systems have this organization [8], [9].

We define the *degree of synchronization* (ds) to be the number of disks synchronized together to form a single unit. For example, a system with 16 disks and $ds = 2$ consists of 8 units of 2 disks synchronized together. If the disk system consists of m disks, then the degree of synchronization can be varied anywhere from 1 to m. Kim only reports degree m synchronization [5]. We consider different degrees of synchronization in this paper.

A system where all the disks are organized as independent units and where each file is located only on one disk is defined to be a *traditional system* in this paper. The traditional system is simple to implement and has a reasonably good performance on single block requests. If the files are not uniformly distributed, a large number of requests may arrive at a single disk. Hence, there is a possibility that a traditional system under unbalanced conditions could yield poor performance. This organization does not reduce the time of any component (seek, latency, or read time) of the service time. In a traditional system, the response time of a request is improved by decreasing the waiting time in the queue when the number of disks in the system is increased. The service time remains the same. We consider this system as a basis of comparison.

Disk striping [4] or data declustering [7] calls for blockwise interleaving of the file such that a number of blocks of a file can be read or written in parallel on different disks. One way of interleaving the file is such that if block i is on disk j, then $i + 1$ is on $j + 1$ and so on. If the I/O request is for a single block, then the request is queued in the disk containing that block. If a multiple-block request arrives at the disk system, then that request is broken up into a number of single-block requests and it is queued at the corresponding disks. This results in concurrent access to the blocks. This organization also enables serving multiple single-block requests concurrently. By interleaving the data, we can keep the disk system balanced with respect to request rates. In this organization, the seek and latency penalties are paid for each block of transfer and hence the servie time has effectively increased. The study in [7] indicates that the declustering approach trades off service time effectively to reduce the variance in the queueing times of a traditional system to achieve high performance. A system that breaks up a file into a number of blocks on different disks is defined to be a *declustered system* in this paper.

We define the *degree of declustering* (dd) to be equal to the number of concurrent accesses that can be made to a file. If $dd = 1$, then only one block of a file can be accessed at any given time. When $dd = m$, m blocks of the file can be read concurrently. The study in [7] only considered systems with $dd = m$, where m is the number of disks in the system. We call such a system a *fully declustered* system. Again, we consider different degrees of declustering in this paper.

We also consider systems based on a combination of the above two approaches, declustered-synchronized system. A *declustered-synchronized* system is a synchronized system where each file is declustered among the different synchronized units. Synchronizing more than a certain number of disks may not be feasible. In such a case, declustering the file among the different synchronized units increases the concurrency of access to a file.

Although data declustering and disk synchronization are two ways of achieving the same goal, there are some fundamental differences in their approaches. The synchronized system keeps the file together on one unit and hence does not allow concurrent access (reads/writes) to different blocks of a single file. The request response time in both the systems may be better than in a traditional system due to differences in waiting time and concurrency in serving a given request. A synchronized system makes use of the locality of access to a file, i.e., a large block of transfer needs to pay the seek and latency penalty only once. A declustered system takes away the locality of file access in the disk system by distributing the access to a large block of data by many accesses to smaller blocks of data. A fully synchronized disk system corresponds to an SIMD approach and a fully declustered system corresponds to an MIMD approach.

The alternatives we consider for evaluation are 1) synchronized system with different degrees of interleaving, 2) declustered system with different degrees of declustering, and 3) declustered-synchronized system with different degrees of synchronization and declustering. These alternatives include a fully declustered system, fully synchronized system, and a traditional system. Some of these alternatives have been considered in [6]. Some of the disk organizations are shown in Fig. 1 where $f.m$ indicates the mth block of the file f, and f by itself stands for the whole file. In the rest of the paper, we follow the notation shown in Fig. 1, i.e., we characterize the disk organization by three variables (dd, ds) and m. Using such a notation, a traditional system is denoted by (1, 1), a fully declustered system by (m, 1), and a fully synchronized system by (1, m), where m is the number of disks in the system. Any organization can then be characterized by the three numbers dd, ds, and m. It is to be noted that $dd \leq m/ds$, i.e., degree of declustering is limited by the number of individual units. We assume that the number of links between the individual units and the I/O buffers is limited by the total

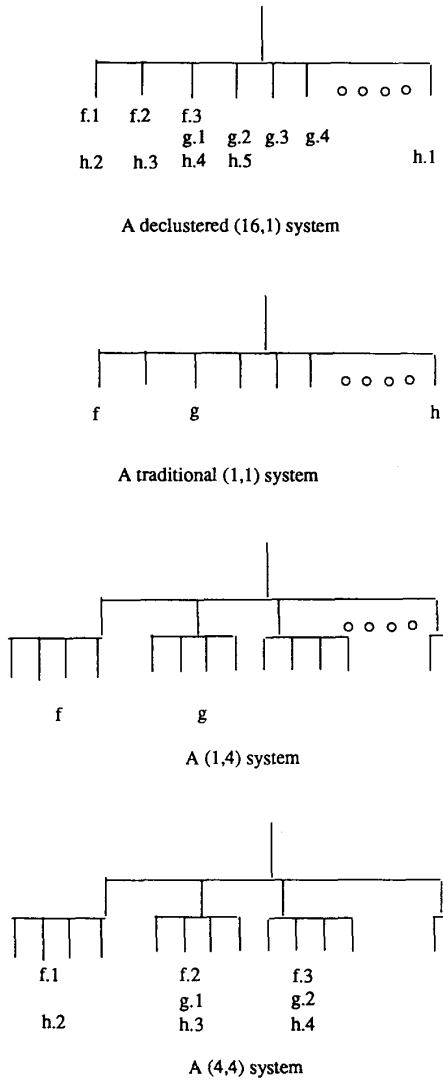

Fig. 1. Some of the disk organizations.

number of disks. To make fair comparisons, a synchronized unit with ds disks together is assumed to have ds links to the I/O buffer so that all the different organizations utilize equal hardware. In [10], the importance of increasing the link bandwidth for this configuration is pointed out.

III. Simulation Model

The simulated I/O system consists of 16 disks. Each disk is assumed to have characteristics similar to an RA81 disk [11]. The characteristics of interest to us are listed in Table I. Each disk is modeled by a queue with either of two scheduling policies, FIFO or SCAN. FIFO is a "first come first serve" policy. In SCAN, the disk arm moves unidirectionally across the disk toward the inner track. When there are no more requests for service ahead of the arm it jumps back to service the request nearest the outer track and proceeds inwards again. This policy guarantees that no request gets indefinitely postponed and at the same time does some seek optimization (on its scan inwards).

TABLE I
Parameters of the System and Workload

	Parameter	Value
System	Max. latency time	16.6 ms
	Seek time/track	0.04ms
	Tracks per disk	1258
	# of heads	14
	# of sectors/track	52
	# of bytes/sector	512
	# of Disks	16
Workload	Read time/block	2.6,4,8,16,32 ms
	Multi-block frequency	10%, 30%, 50%, 70%, 90%
	Deg. of Declustering	1,2,4,8,16
	Deg. of Synchronization	1,2,4,8,16
	Ave. inter-arrival time	10, 20, 40, 80 ms
	Max. # of blocks/request	4,8,16

We use two different workloads for our simulations. The first workload characterizes a multiuser file system usage and a transaction-based system usage. The second workload considers single-user operation of a multiprocessor for solving scientific problems. These two workloads capture the needs of two different sets of users and represent two extremes in the data access patterns. The first environment typically consists of many small requests and the second environment consists of few requests of large files. These two workloads capture most of the I/O traffic patterns.

A. File/Transaction System Workload Characteristics

The characteristics of the first workload are based on the measurements carried out on UNIX file systems [12], [13]. A similar workload is used in measuring the performance of a system in transaction workload [7]. In the first workload, the I/O request arrival is assumed to be a Poisson process. The request rate is specified by the mean time between arrivals. The actual number of blocks requested can be easily obtained from the interrequest time as follows: if n is the average number of blocks per request then the average rate of requested blocks is given by n/t, where t is the average interarrival time of the requests. For example, when $t = 20$ ms, and $n = 8$, the average rate of blocks requested $= 8/0.02 = 400/\text{s}$. The request rate is varied until one of the systems under study gets saturated. Each request is either for a single block or a multiple number of blocks of data. The block size is a variable and is varied such that its read time varies from 2.6 to 32 ms. Read time is the time taken by a single disk to read a block of data. In most of the simulations, the multiblock request sizes are assumed to be uniformly distributed over (2, number of disks). The effect of varying the maximum number of blocks in a request is also studied. The percentage of multiblock requests is a variable and we varied it between 10, 30, 50, 70, and 90 percent. Each request specifies a file number and a block number within the file. We used two different distributions for the file numbers of a request: uniform distribution

and normal distribution. In the first, all the files on the system are assumed to be equally likely to be accessed. In the second distribution, 30 percent of the files receive 70 percent of the accesses. The blocks within a file are assumed to be accessed uniformly.

The file system is assumed to have 800 active files, each 4 Mbytes in size. In a multiblock request, the blocks are assumed to be contiguous blocks in a given file. In declustered systems, consecutive blocks of a file reside on different units and in a nondeclustered system, the file is assumed to be stored on contiguous tracks of the disk. Different blocks of a file that map onto the same disk in a declustered system are assumed to lie on contiguous locations of that disk.

The communication links for data transfer between disk and the I/O buffers are assumed to have a sustained throughput of 2 Mbytes. The simulations were carried out using CSIM, a simulation language based on C [14]. A summary of system and workload parameters is given in Table I. Each time, serving 50 000 requests at the disk system was simulated, each request being either a single or a multiple block request.

B. Scientific Workload Characteristics

The second workload considered the I/O request patterns of some of the basic matrix arithmetic algorithms. We considered matrix–matrix multiplication, matrix–vector multiplication, and the fast Fourier transform (FFT). These three applications represent different levels of I/O requirements. The ratio of computations performed to the total number of data elements for these three applications is $O(\sqrt{n})$ for matrix–matrix multiplication, $O(\log n)$ for FFT, and $O(1)$ for matrix–vector multiplication, where n is the data size. These three basic algorithms are used in evaluating the disk architectures in the second workload. We simulated the solution of these different problems in an MIMD environment. Specifically, we considered a shared-memory machine, but the I/O traffic patterns will not be considerably different for these applications in other machines. All the processors in the system (in our case, 16) are used for solving a single problem at a time. Each processor obtains the data it needs, carries out the arithmetic, communicates with other processors if necessary, and once the computation is complete writes the results back to the disk system. It is assumed that there is a single process running on each processor and that the process is blocked waiting for I/O data, but never swapped out. This represents a typical single-user usage of a large machine.

C. System Characteristics

The simulation used two system variables, degree of declustering (dd), degree of synchronization (ds). We considered the values of 1, 2, 4, 8, and 16 for dd and 1, 2, 4, 8, and 16 for ds. It is to be noted that $dd \leq m/ds$, since the degree of declustering cannot be more than the number of independent units in the system. The seek time is based on the location of the current file with respect to the location of the last file served. The latency times are assumed to be uniformly distributed between 0, Max-latency time (16.6 ms).

In nondeclustered systems, multiblock requests are treated as a single request for larger amount of data and hence the seek and latency penalties are paid only once. In declustered systems, the multiblock requests are treated as follows: if the number of blocks requested is less than the degree of declustering, each unit receives a request for one block; if the number of blocks requested is more than the degree of declustering, the disk units receive requests for appropriately larger amount of data. For example, if 13 blocks (each block = m bytes) of data are required in a (4, 4) system, 3 disk units would get requests for $3m$ bytes of data and one disk unit would receive a request for $4m$ bytes of data. As mentioned earlier, different blocks of a file that reside on the same disk unit are assumed to lie on contiguous locations on a disk track. Hence, this would result in paying seek and latency penalties only once if multiple blocks are requested at once from a disk unit. This assumption makes the results a little optimistic. It was observed that the request scheduling policy had a great influence on the outcome of the results.

We use two different characteristics to measure the performance of the system in the two workloads considered. In the first workload, we are primarily interested in the average response time of a request. The comparisons of different systems in this workload is based on the average response times of requests. In the second workload, we are concerned mainly about the speed at which a problem can be solved. Hence, we use the time-to-completion as a measure of performance in that workload.

IV. Results of File/Transaction System Workload

In this section, we present the results obtained in the file system workload. FIFO scheduling policy yielded very poor results compared to SCAN and hence we just looked at SCAN in the rest of the simulations.

A. Request Interarrival Time

Fig. 2 shows the average response times of various organizations as a function of the request interarrival times when the block size is 4 kbytes (reading time of approximately 2.6 ms with the disk characteristics considered). The maximum number of blocks in a given request was fixed at 16 and the multiblock request frequency was fixed at 30 percent. The 95 percent confidence intervals have been calculated. The intervals are found to be very small and hence not marked on the figure. The following points are noted from Fig. 2.

1) Systems with $dd \leq m$, (2, 1), (4, 1), and (8, 1) perform worse than (16, 1) system at the low request rates. At low request rates, restricting the amount of parallelism available to serve a given request increases the response time. Specifically, if a multiblock request asks for more than dd blocks, requests for some of the blocks have to be queued. At low request rates, there is not much probability for other requests to arrive before the current request is finished service and hence some of the disks remain idle and do not as such help in reducing the response time of the request. But at higher request rates, we observe that (16, 1) system performs worse than the other three systems. This is mainly due to the higher utilization of the disks in the (16, 1) system. Each block of request pays latency and seek penalties in the (16, 1) organization

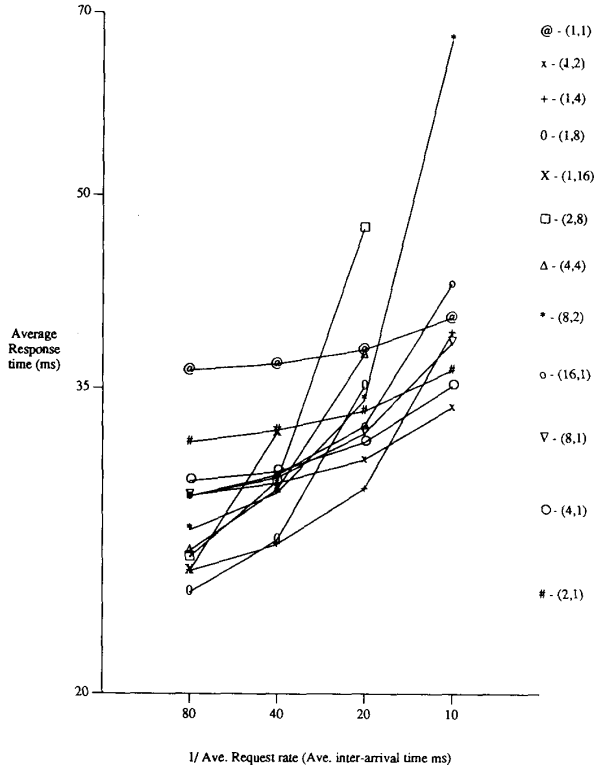

Fig. 2. Response time of various organizations.

and this results in higher utilizations. In the other systems, the seek and latency penalties are limited by the amount of declustering. For example, when a request for 7 blocks has to be served by different systems, in the (16, 1) system, the seek and latency penalties are paid at 7 different disks and in the (4, 1) system only at 4 disks. Limiting the seek and latency penalties results in lower utilizations, but also limits the amount of parallelism used for serving a request. As seen from the figure, (4, 1) seems to offer a good compromise for the workload considered. At low request rates, the parallelism dominates since the utilizations (and hence the waiting times) are low. But at higher request rates, the higher disk utilizations result in higher waiting times and thus resulting in relatively poor performance of (16, 1).

2) Synchronized systems (1, 16), (1, 8), and (1, 4) have better performances than other systems at very low request rates. When interarrival time (IATM) is less than 20 ms, the systems are so heavily loaded that the response time is far worse than other systems. The points missing from the figure (such as 20, 10 ms points for (1, 16) organization) are beyond the ordinate scale. In these organizations, *ds* number of disks are needed to serve any given request and hence higher utilization of disks. This higher utilization results in early saturation and hence poor performance at higher request rates. Each request, whether a single-block request or multiple-block request, can utilize the parallelism available in (1, 16) and (1, 8) organizations unlike in the declustered organizations where only the multiblock requests can avail of the available parallelism in serving a request. Hence, the difference in performance at

TABLE II
COMPONENTS OF RESPONSE TIME

System	Times (ms)			
	Response	Service	Wait	Transfer
(1,16)	32.28	20.16	11.68	0.44
(1,8)	26.49	21.07	4.50	0.92
(1,4)	26.28	22.28	2.22	1.78
(1,2)	29.41	24.62	1.22	3.57
(1,1)	36.66	28.53	0.83	7.29
(16,1)	29.85	22.28	1.72	5.84
(8,1)	29.70	22.95	1.35	5.40
(4,1)	30.03	24.57	1.07	4.39
(2,1)	32.43	26.80	1.03	4.60
(8,2)	28.87	21.63	2.68	4.56
(4,4)	29.06	21.33	4.35	3.39
(2,8)	29.51	20.93	6.16	2.42

lower request rates. It is also observed that the (1, 8) organization performs better than the (1, 16) organization. This is also a consequence of the tradeoff between the parallelism and the overheads associated with it.

3) The organizations that use both synchronization and declustering (2, 8), (4, 4), and (8, 2) perform worse than their synchronized counterparts (1, 8), (1, 4), and (1, 2) and better than their declustered counterparts (2, 1), (4, 1), and (8, 1) at low request rates (IATM's > 40 ms).

4) Declustered systems (16, 1), (8, 1), (4, 1), and (2, 1) do not perform as well as the synchronized systems (1, 16), (1, 8), (1, 4), and (1, 2) at lower workloads. However, the declustered systems are more resilient to higher workloads than the synchronized systems.

5) The transfer time (time to transfer the block of data to a buffer) is considerably higher in nonsynchronized systems. This difference will be much higher with larger size blocks. It is to be remembered that each synchronized unit is assumed to have as many links as the number of disks synchronized. Table II shows the contributions of various components to the response time of a request at 40 ms interarrival time, 30 percent multiblock frequency. In synchronized systems, the data are transferred in parallel to the buffer and hence a lower transfer time component in the request response time. It was noticed that this resulted in considerable difference in the response times between say (1, 16) and (16, 1) systems. This difference in transfer time is found to be higher with larger block sizes. In the fully declustered systems, each disk is assumed to have a single dedicated link to transfer the data to the buffer and hence the data transfer time is not reduced. Had we assumed that the synchronized units also use a single link to transfer the data, the results and hence the conclusions could have been considerably different. Our assumption of being able to use multiple links was based on the fact of giving equal hardware to all organizations to make a fair comparison.

6) Overheads may be incurred in using synchronized systems and declustered systems. Some of these overheads are: possible speed penalty due to synchronization, processing time to find out where a given block lies in a declustered system,

TABLE III
AVERAGE UTILIZATIONS AT DIFFERENT REQUEST RATES

System	Average Utilization at Inter-arrival times (ms)			
	80	40	20	10
(1,16)	25.4	50.2	99.4	Sat
(1,8)	13.2	26.5	51.4	95.0
(1,4)	6.9	13.9	28.0	54.5
(1,2)	3.8	7.6	15.2	29.8
(1,1)	2.2	4.5	8.9	17.6
(16,1)	5.9	11.8	23.7	47.2
(8,1)	4.7	9.5	19.2	38.5
(4,1)	3.5	6.9	14.1	27.9
(2,1)	2.8	5.5	11.1	21.9
(8,2)	9.1	18.2	36.7	73.2
(4,4)	12.2	24.6	48.6	97.5
(2,8)	17.4	34.8	66.7	Sat

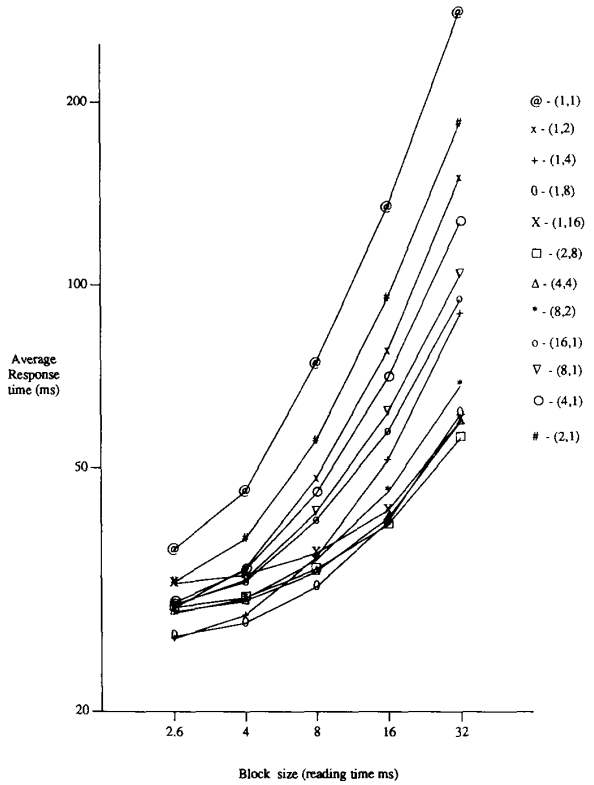

Fig. 3. Effect of varying block size.

etc. We did not include these penalties in our evaluation since we do not know how disk synchronization may affect the performance. To make a fair comparison we did not include the processing penalty in the declustered system as well.

Table III shows the average utilizations of the disks in each of the organizations at different interarrival times (IATM). The traditional system (1, 1) has the lowest utilization at all IATM's. Even when the (1, 1) system has less than 50 percent utilization, some of the other systems are saturated. We restrict our attention to the lower spectrum of utilizations (IATM $\geq$ 20 ms) since the response times are not satisfactory at higher request rates. The much higher utilizations at higher request rates point to a need for using more disks to achieve the required I/O performance at such high request rates.

B. Block Size

Fig. 3 shows the effects of varying the size of the block of a request at an interarrival time of 40 ms and 30 percent multiblock frequency. The declustering considered in this paper can be termed block declustering, i.e., the data are declustered at a block level, where the size of the block is varied such that its reading time on a single disk varies from 2.6 to 32 ms. The basic unit of transfer and I/O requests is a block. We also consider the situation where the unit of declustering is a fraction of the basic block of request, the results of which are shown later. The figure shows that (1, 8) and (1, 4) organizations outperform other organizations at lower block sizes at that request rate. Many organizations (2, 8), (1, 16), (4, 4), and (1, 8) perform well at the larger block sizes. As the block size is increased, all the organizations outperform (1, 1) organization. At higher block sizes, a higher fraction of the response time is contributed by read time. Since all the organizations reduce the read time, they perform better at the higher block sizes. The relative performance of the other organizations is a function of the block size. For example, at a block size of 2.6 ms, (1, 4) organization performs better than (1, 16) system and vice versa at a block size of 32 ms. At larger block sizes, the declustered-synchronized systems perform better than their synchronized counterparts and declustered counterparts.

C. Multiblock Frequency

Until now we have considered a multiblock request frequency of 30 percent. To see if this frequency made any difference in the performance trends, we varied the multiblock request frequency from 10 to 90 percent. Fig. 4 shows the effect of varying the multiblock request percentage of the total requests with a block reading time of 2.6 ms and 40 ms interarrival time. From Fig. 4, the (1, 8), (1, 4), and (1, 16) organizations outperform other alternatives at higher multiblock frequencies. The (1, 16) organization is relatively unaffected by the multiblock frequency. The read time is a small fraction of the response time of this organization. As multiblock frequency is increased, this small component is increased keeping the relatively large overhead of seek and latency penalties constant. This results in relatively flat characteristics. When read-ahead and delayed-write are used, the multiblock frequencies will be higher. In such environments, these organizations will perform better. All other organizations have similar characteristics as multiblock frequency is varied.

D. Number of Blocks in a Request

Fig. 5 shows the effect of changing the maximum number of blocks requested by a multiblock request when the block reading time is 2.6 ms, the request interarrival time is 40 ms and the multiblock frequency is 30 percent. A lower limit on the maximum number of blocks requested in a multi-

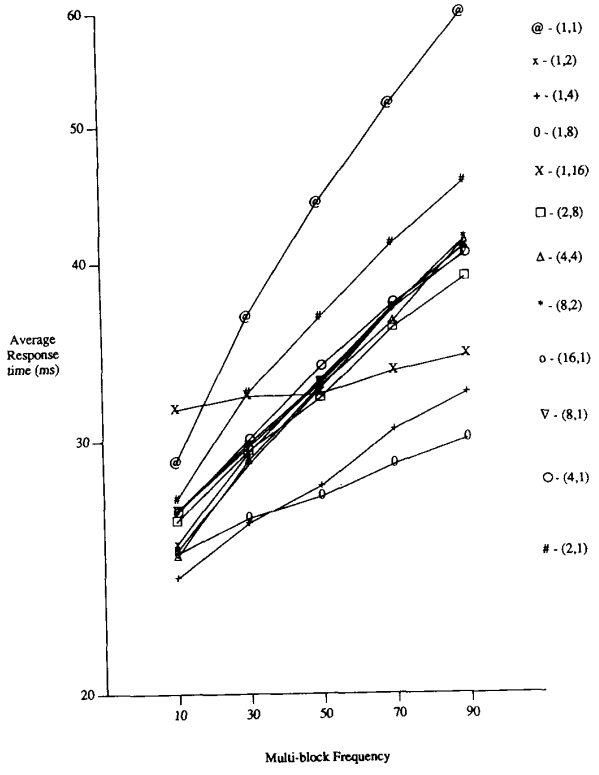

Fig. 4. Effect of multiblock request frequency.

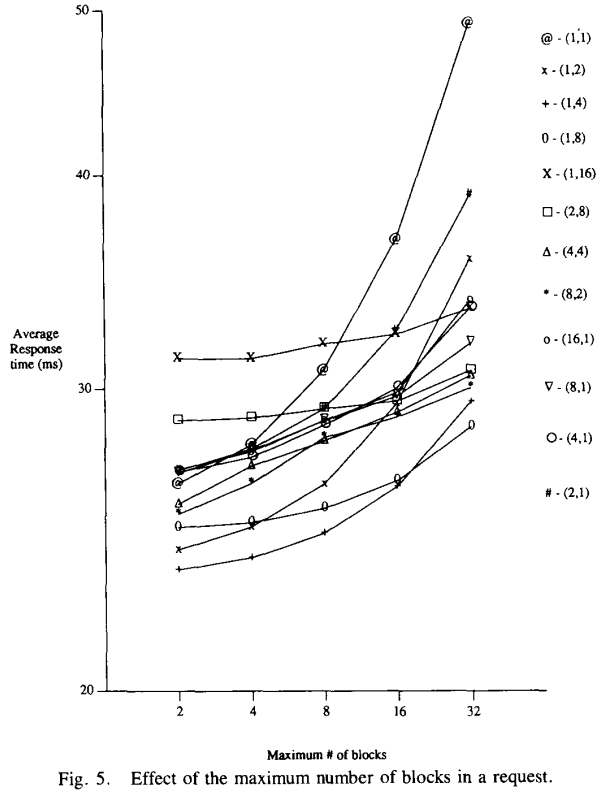

Fig. 5. Effect of the maximum number of blocks in a request.

block request, keeping the block read time and the request rate constant, means a lower workload and hence a better performance. From Fig. 5, we see that (1, 8) and (1, 4) organizations outperform other organizations at the higher number of blocks. The relatively flat characteristics of (1, 16) organization is due to the same reason as explained earlier. The relative performance of (1, 2), (1, 4), and (1, 8) organizations is seen to be a strong function of the number of blocks in a request. All the organizations perform much better than (1, 1) organization at higher number of blocks.

E. Nonuniform File Access

Fig. 6 shows the performance of the different systems when the files are accessed at nonuniform frequencies with 2.6 ms block read time, 30 percent multiblock frequency, and a maximum of 16 blocks per request. The files are assumed to be located randomly on the disks. Since the results would depend on the file location, we carried out 5 simulations for each point in the figure. The file requests are generated using a normal distribution $N(0, 0.3)$. With this distribution, 30 percent of the files in the system are accessed 70 percent of the time. The results are quite similar to the case of uniform file access. The (1, 16) and (1, 8) organizations perform well at low request rates and get saturated earlier than the other organizations. It is noted that the (1, 1) organization tends to perform better at higher request rates relative to other organizations. Here again, we obsere that the (4, 1) organization performs better than the other declustered organizations (16, 1), (8, 1), and (2, 1) at higher request rates.

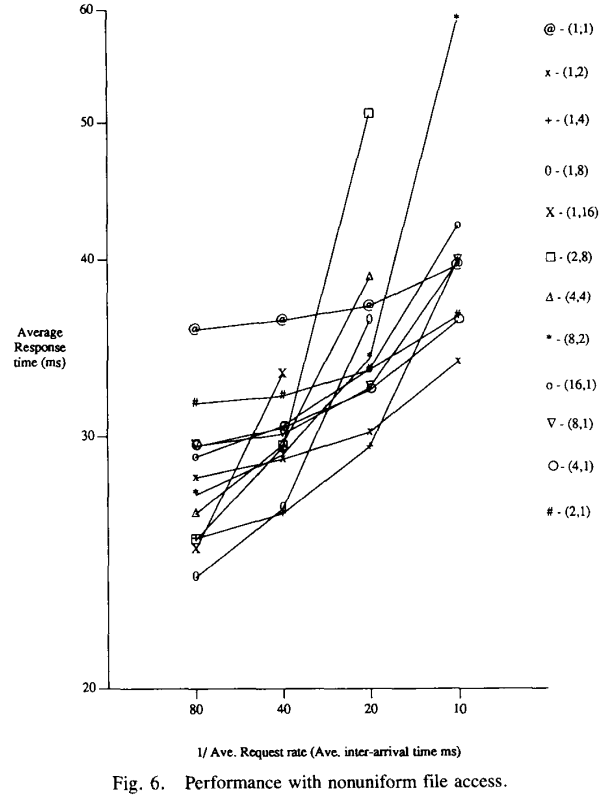

Fig. 6. Performance with nonuniform file access.

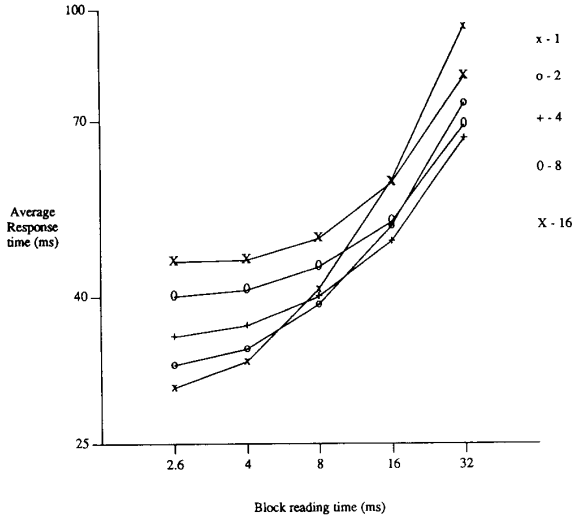

Fig. 7. Effect of subblock declustering on (16, 1).

F. Subblock Declustering

When the unit of declustering is smaller than the size of the basic block of a request, the system is said to employ subblock declustering. We considered the effects of subblock declustering on the (16, 1) system. Fig. 7 shows the effects of subblock declustering as a function of number of subblocks per a block at various block sizes. When there are 8 subblocks per a block with a read time of 16 ms, the block of declustering has a read time of $16/8 = 2$ ms. The figure shows that subblock declustering is useful at higher block sizes. The results incidate that the unit of declustering should be such that its read time is about 4 ms.

The general conclusions from the experiments in this section are the following. 1) The synchronized systems are best suited in environments where few requests for large pieces of data are common. 2) If the main objective of building a multiple disk system is to support more users with each user requesting small pieces of data, then there seem to be several competing organizations such as (1, 2), (4, 1), and (2, 1). The (1, 1) organization performs relatively better at higher request rates. If the overheads in building the parallel organizations are excessive, then the (1, 1) organization may be a better choice. The overheads are not taken into account in this study. 3) There seems to be tradeoff in the amount of declustering/synchronization to be used. The larger the declustering, the larger the parallelism, but also the larger overheads. It was found that (4, 1) organization offered a good compromise for the considered workload. Similarly (1, 8) was found to perform better among the synchronized organizations. 4) The type of system to be built depends heavily on the basic block of transfer and the workload.

V. Results of Scientific Workload

In this section, we present the results obtained in the second workload. Since the scientific computations are structured and regular, it is possible to alleviate some of the I/O needs of a problem through prefetching. However, if the data processing speeds and data reading (from disk) speeds are considerably different, the amount of data that needs to be prefetched could be considerably larger than what the buffers could hold. Hence, even with buffering there is a need to improve the basic data reading time from the disks.

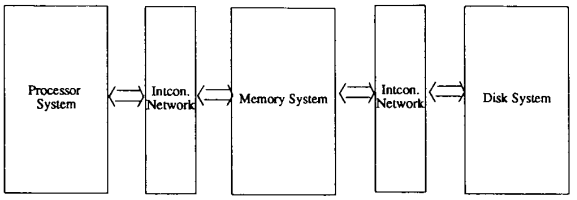

Fig. 8. Multiprocessor system for scientific workload.

We consider three different matrix problems, matrix–matrix multiplication, matrix–vector multiplication, and the FFT, for our evaluation. These three problems have different I/O requirements. The ratio of total number of computations to the total number of I/O operations is $O(\sqrt{n})$, $O(\log n)$, $O(1)$ for matrix–matrix multiplication, the FFT, and matrix–vector multiplication, respectively. It is expected that these three applications give a wide range of I/O requirements such that observations can be made regarding the performance of the different organizations. We use a shared-memory multiprocessor model to simulate parallel processing of the data. The I/O patterns obtained in such a model are very sensitive to the assumptions made about the relative sizes of memory and the problem. Here we assume that, for each problem, the size of all the available memory space is large enough to hold the required data for solving the problem. This assumption leads us to obtaining I/O traffic patterns of the following form: every processor requests for a piece of data, solves the problem in a cooperative way, and then writes the results back to the I/O system. This I/O traffic pattern is near-synchronous. Since we did not have actual I/O traces, we had to restrict our attention to such a model. We are currently in the process of obtaining I/O traces to make stronger conclusions about the relative merits of the different organizations. An evaluation of IBM RP3 architecture regarding the I/O issues, for seismic applications, can be found in [15].

A shared-memory multiprocessor with different disk organizations is modeled. The considered system is shown in Fig. 8. The disk system is assumed to be connected directly to the bus connecting the different main memory modules. the shared memory bus was assumed to be fast enough to support the peak data transfer from the disk system. Different disk organizations as discussed earlier were considered for evaluation. The simulations were carried out with the same multiprocessor system but different I/O subsystems. Since the algorithms are assumed to be implemented exploiting all the features of the system, any differences in performance are mainly due to the differences in the I/O subsystem. However, the data are assumed to be transferred over multiple links (where available) from the disk to I/O buffers. The shared memory is assumed to be large enough to hold the data required for solving the problem. The time taken to solve the given problem was taken to be the performance measure. We simulated the problem with different processor speeds. Start-

ing with a base speed of 1, the processor speed was varied from 0.1 to 10 000 relative to the base speed. The base speed of 1 corresponds roughly to 0.5 MFLOPS. The problem size was fixed through simulations. Since the data size was assumed to be small enough to fit into the memory of the system, we did not have to make further assumptions about increases in memory size as the processor performance is increased. All the processors are assumed to obtain the required data before beginning the processing phase. Blocked storage is used for declustered systems. Since the files are stored as single entities in nondeclustered systems, row-major form of storage is assumed in such systems. When the files are written back, it is required that they be written back in the same order as the input data are organized. The data files are assumed to be split among the different disks equally. The block of declustering is assumed to be equal to the amount of data to be transferred from each disk unit, i.e., for example, if each disk has to transfer 200 kbytes, then the block of declustering is assumed to be 200 kbytes. It is assumed here that the data declustering is done by the user to obtain maximum benefit out of the disk system.

Three different applications were simulated on this system with different disk organizations. The applications were assumed to exploit the data organizations on the disk. For example, when the data are stored in the form of rectangular blocks on the disks, the applications are assumed to use blocked versions of the algorithms. When needed, the data are reordered into other organizations to suit the application. The data reordering cost is taken into account in the time-to-completion.

In application 1, the multiplication of two 1024×1024 matrices was simulated. The matrices A and B are stored in the shared memory. The load of generating the matrix C is equally shared by the processors. To generate matrix $C = AB$, each processor multiplies a part of A with B. Hence, the different elements of matrix B are used by several processors during the problem execution. We did not, however, model the problem of memory contention. Memory contention would effect the results, but the effect would be fairly uniform among systems with different disk organizations and hence the conclusions are not affected. Only matrix C is written back to the disk system. Fig. 9 shows the performance of various machines executing matrix multiplication.

In application 2, a two-dimensional FFT algorithm on 1024×1024 points was simulated. Each processor computes a one-dimensional FFT of each of the rows assigned to it. The matrix is then transposed to get a columnwise partitioning for the second phase. Each processor computes a one-dimensional FFT of each of its columns to obtain the two-dimensional FFT of the data. The data are reorganized into blocked form or row-major form and written back to the disks. Fig. 10 shows the performance of different organizations executing FFT.

In application 3, multiplication of a 1024×1024 matrix with a 1024 vector was simulated. Each processor generates part of the resultant vector. Each processor computes its part of the matrix-vector product and writes back the resultant vector to the disks. Fig. 11 shows the performance of different organizations executing matrix-vector multiplication.

Figs. 9, 10, and 11 show the performance of various ma-

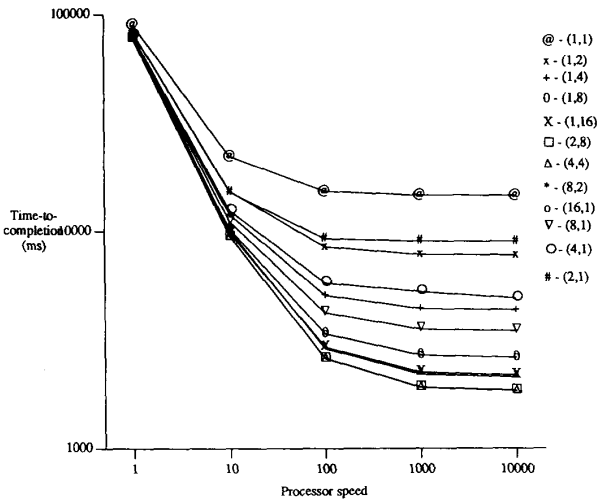

Fig. 9. Performance of different organizations with matrix multiplication.

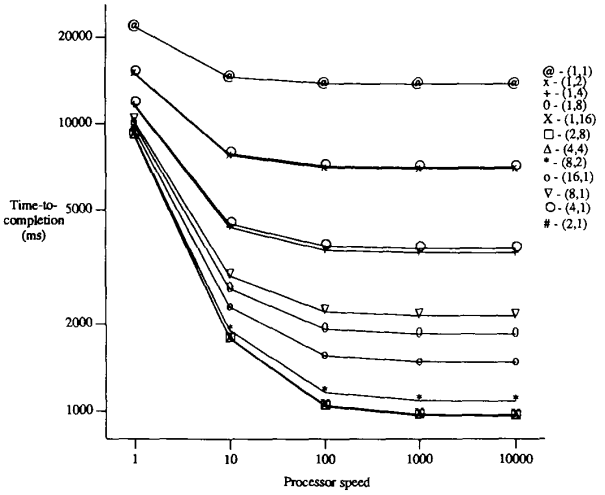

Fig. 10. Performance of FFT.

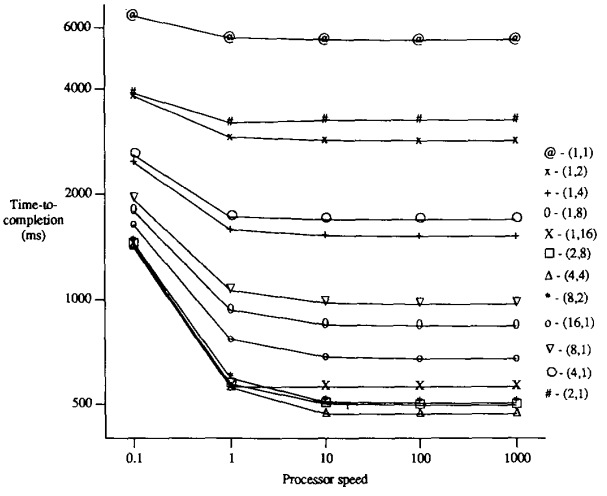

Fig. 11. Performance of matrix-vector multiplication.

chines as a function of processor speed for the three problems: matrix multiplication, FFT, and matrix–vector multiplication, respectively. The following observations can be made from the figures.

1) Matrix multiplication is compute bound at processor speeds up to 1000, FFT up to 100, and matrix–vector multiplication up to 1. As expected, these three problems exhibit different balances between computations and I/O. Beyond these points, the problems are I/O bound. The differences in the I/O performance can be noted when the problem is I/O bound. The differences in performance are due to the differences in I/O subsystem since the multiprocessor system is unchanged in all the organizations. The organizations that offer higher parallelism for reading data from a single file, i.e., with higher $ds \times dd$, perform better than organizations with lower parallelism. Even when the problem is compute bound, though relatively minor, the differences in performance can be noticed.

2) The traditional (1, 1) organization is seen to perform very poorly compared to other organizations. This shows that putting together a number of disks without increasing the concurrency of access to a file is not a good choice.

The differences in performance of the organizations with $dd \times ds = 16$ do not differ drastically to conclude about their relative performances. Moreover, the data obtained about these organizations in scientific environments are also not large enough to draw any meaningful conclusions about their relative performance. However, the simulations clearly show the need for parallelism in I/O. We are in the process of obtaining I/O traces for a number of scientific applications to evaluate the organizations with maximum parallelism ($dd \times ds = n$) available for reading a single file.

VI. Scalability and Other Issues

In this section, we discuss several issues regarding building a larger system of disks than was considered earlier. Consider building a disk system with larger than 16 disks. How well do the different organizations scale to larger sizes? Disk synchronization is not possible beyond probably 8 or 16 disks. Hence, when a disk system has to be built with a larger number of disks, a fully synchronized system is no longer an option. Hence, we have to consider some kind of declustering beyond a certain number of disks to achieve the required concurrency in reading a file. Then, should we build a fully declustered system or a declustered–synchronized system?

First, we note that the performance is not always improved when we use declustering over synchronized units. For example, it is observed that the (2, 8) system performs worse than a (1, 8) system in the file system workload when the block size is such that its read time is smaller than 16 ms. The relative performances of different organizations is sensitive to the block size, maximum number of blocks in a request, etc. These observations point out that though the I/O performance can be improved by declustering among different synchronized units, the performance is very sensitive to the different parameters of the workload. However, it is observed that at higher block sizes (read time $>$ 16 ms), declustered–synchronized systems perform better than their declustered and synchronized counterparts. If the aim of building a multiple-disk system is to support more users with each user requesting smaller amounts of data, the composite systems do not necessarily yield higher performance.

However, in a scientific workload, the I/O performance seems to be directly proportional to the available parallelism in reading a file as noted from Figs. 9, 10, and 11. This indicates that some form of declustering is necessary when we build a disk system with large number of disks since synchronizing more than 8–16 disks may not be feasible. Whether we should build a fully declustered system or a declustered–synchronized system is an open question. As noted earlier, the synchronized systems have the advantage of being able to serve every request, whether single-block or multiple-block requests, with concurrency. Hence, the different disk organizations have to be evaluated with a larger number of scientific applications to make such a decision.

When a file is distributed among many disks, fault tolerance becomes an important issue and such issues are being investigated by other researchers [16].

In our simulations of scientific applications, we assumed that the processors fetch the data completely before starting to process it. However, it is possible to overlap these two activities to some extent. Effects of such overlapping also remain to be evaluated. Compiler techniques could be used to reduce the I/O requirements of a program by suitable transformations such as loop blocking [17], etc. How well these techniques can be utilized and what other transformations could be useful for reducing I/O remains to be investigated.

Declustering by block number will not be a feasible solution in general. Consider a situation where a block of data is written in the middle of an existing file. If the data were to be declustered among the different disks purely by the block location in the file, the later part of the file has to be moved from one disk to the next one and so on. This would result in excessive overheads. To avoid these overheads, a hash function or some table lookup mechanism has to be used to locate blocks on different disks. Such a file system has been recently proposed [18]. Once such a mechanism is used, determining where a particular block of data resides is no longer a simple matter. Consider designing a name server in such a system. The name server has to be efficient such that it does not become a bottleneck. In large systems, the name server itself has to be distributed or replicated. There are several issues to be resolved here such as how to maintain a consistent view among the different name servers. Designing such an efficient name server is an open area of research. Many of the concepts from distributed systems can be applied here with suitable modifications to suit the different characteristics of the systems.

VII. Conclusions

This paper looked at various alternative organizations available for a multiple-disk system. We presented a method of characterizing the different organizations and also showed that there exist organizations that have not been considered earlier (systems with $dd \neq m$ and $ds \neq m$). An extensive evaluation of the different organizations of disks is pre-

sented. It was observed that different organizations performed well under different characteristics of the workload. The synchronized systems were found to outperform the other organizations at low request rates in the file system workload. The effects of different parameters of the model were studied. The overheads involved in availing the parallelism in a declustered system somewhat offset its advantages compared to the synchronized organizations. Tradeoffs were observed in the amount of declustering/synchronization to be used. The relative performance of different organizations was found to be a strong function of the basic block of I/O transfer. The declustered–synchronized systems were found to perform better than their synchronized counterparts and declustered counterparts at large block sizes.

In the scientific workload, the organizations with higher concurrency ($dd \times ds$) in reading a file were observed to perform better. The data transfer speeds also have to increase if the I/O performance has to be increased further. The data are not enough to conclude about the relative merits of the organizations with the same concurrency ($dd \times ds = 16$).

The different organizations considered in the paper were mainly aimed at reducing the read time of the I/O service. Other organizations that reduce other components of service (latency time or seek time) remain to be investigated for environments where a large number of requests for small pieces of data have to be supported.

Several issues concerning building disk systems are discussed. It was argued that from scalability point of view, some form of declustering is needed. A number of issues such as overlapping data reading and processing, relative merits of different organizations, in scientific applications, with the same amount of parallelism in reading a file, remain to be investigated.

Several of the software/OS issues remain to be investigated. Designing an efficient name server is one of the identified problems. Efficient declustering methods, when the files can be modified, remain to be investigated.

Acknowledgment

Referees' comments have greatly improved the quality and presentation of the paper.

References

[1] H. T. Kung, "Memory requirements for balanced computer architectures," in *Proc. 13th Annu. Int. Symp. Comput. Architecture*, 1986, pp. 49–54.

[2] J. W. Hong and H. T. Kung, "I/O complexity: The red-blue pebble game," in *Proc. 13th Annu. ACM Symp. Theory Comput.*, 1981, pp. 326–333.

[3] A. Aggarwal and J. S. Vitter, "The input/output complexity of sorting and related problems," *Commun. ACM*, vol. 31, no. 9, pp. 1116–1127, Sept. 1988.

[4] K. Salem and H. Garcia-Molina, "Disk striping," in *Proc. Int. Conf. Data Eng.*, 1986, pp. 336–342.

[5] M. Y. Kim, "Synchronized disk interleaving," *IEEE Trans. Comput.*, vol. C-35, no. 11, pp. 978–988, Nov. 1986.

[6] D. A. Patterson, G. Gibson, and R. H. Katz, "A case for redundant arrays of inexpensive disks (RAID)," in *Proc. ACM SIGMOD Conf.*, June 1988.

[7] M. Livny, S. Khoshafian, and H. Boral, "Multi-disk management algorithms," in *Proc. ACM SIGMETRICS*, May 1987, pp. 69–77.

[8] Fujitsu America, *M2360A Parallel Transfer Disk Engineering Specifications*, vol. B03P-4905-0001A, 3055 Orchard Drive, San Jose, CA 95134-2017, 1986.

[9] Cray Research, *Cray X-MP and Cray-1 Computer Systems: Disk Systems Hardware Reference Manual*, vol. H0077, 1440 Northland Drive, Mendota Heights, MN 55120, 1985.

[10] S. Ng, D. Lang, and R. Selinger, "Trade-offs between devices and paths in achieving disk interleaving," in *Proc. 15th Annu. Int. Symp. Comput. Architecture*, May 1988, pp. 196–201.

[11] *RA81 Disk Drive User Guide*, Digital Equipment Corp., 1982.

[12] M. V. S. Devarakonda, "File usage analysis and resource usage prediction: A measurement-based study," in Tech. Rep. CSG-79, Univ. of Illinois, Urbana-Champaign, Dec. 1987.

[13] J. K. Ousterhout *et al.*, "A trace-driven analysis of the UNIX 4.2 BSD file system," in *Proc. 10th Symp. Oper. Syst. Principles*, Dec. 1985, pp. 15–24.

[14] H. D. Schwetman, "CSIM: A C-based, process-oriented simulation language," Tech. Rep. PP-080-85, Microelectronics and Computer Technology Corp., Austin, TX.

[15] S. Sivaramakrishnan, "Evaluation of logical external memory architectures for multiprocessor systems," in Tech. Rep. TR-88-32, Dep. Comput. Sci., Univ. of Texas, Austin, Sept. 1988.

[16] G. Gibson *et al.*, "Failure correction techniques for large disk arrays," in *Proc. 3rd Int. Conf. Architectural Support Programming Languages Oper. Syst.*, Apr. 1989.

[17] D. Gannon, W. Jalby, and K. Gallivan, "Strategies for cache and local memory management by global program transformation," CSRD Rep. 698, Univ. of Illinois, Urbana, Feb. 1988.

[18] A. Barak, B. A. Galler, and Y. Farber, "A holographic file system for a multicomputer with many disk nodes," Tech. Rep., Univ. of Michigan, Ann Arbor, vol. CSE-TR-01-88.

A. L. Narasimha Reddy received the B.Tech. degree in electronics and electrical communication engineering from the Indian Institute of Technology, Kharagpur, in 1985 and the M.S. degree in electrical and computer engineering from the University of Illinois, Urbana–Champaign, in 1987.

Currently he is pursuing the Ph.D. degree in Electrical and Computer Engineering at the University of Ilinois, Urbana–Champaign. He is the recipient of an IBM Graduate Fellowship. His research interests are in parallel processing, computer architecture, and fault tolerance.

Prithviraj Banerjee (S'82–M'84) received the B.Tech. degree in electronics and electrical engineering from the Indian Institute of Technology, Kharagpur, in August 1981, and the M.S. and Ph.D. degrees in electrical engineering from the University of Illinois at Urbana–Champaign in December 1982 and December 1984, respectively.

He is currently Associate Professor of Electrical and Computer Engineering and the Coordinated Science Laboratory at the University of Illinois at Urbana–Champaign. His research interests are in multiprocessor architectures with emphasis on fault tolerance, and parallel algorithms for VLSI design automation, and he is the author of over 50 papers in these areas.

Dr. Banerjee was the recipient of the President of India Gold Medal from the Indian Institute of Technology, Kharagpur, in 1981, the IBM Young Faculty Development Award in 1986, and the National Science Foundation's Presidential Young Investigators' Award in 1987. He has served on the Program and Organizing Committees of the Fault Tolerant Computing Symposia in 1988 and 1989, and as the General Chairman of the International Workshop on Hardware Fault Tolerance in Multiprocessors, 1989. He is also a consultant to Westinghouse Corporation and the Jet Propulsion Laboratory.