Parallelizing I/O Intensive Applications for a Workstation Cluster: a Case Study

Z. Lin and S. Zhou

Computer Systems Research Institute
University of Toronto
Toronto, Ontario, Canada M5S 1A1

Abstract

An I/O intensive application, parallel full text retrieval based on a signature file method, is studied. The text retrieval system is implemented on a cluster of DEC5000 workstations connected by Ethernet. Experiments are performed to evaluate the benefit and cost for running such an application in a workstation cluster. Results show that substantial improvement in speed can be obtained through parallelism in disk accesses, despite the high communication and synchronization overhead that is incurred. Several factors that affect the performance of a parallel I/O application in this type of computing environment are discussed. The advantages of a workstation cluster are its large combined I/O buffer capacity, and possible concurrent accesses to disks local to each workstation. Our study demonstrates that these advantages, when exploited properly, can lead to effective performance improvement without the need for additional hardware.

1 Introduction

Parallel processing in a network of workstations is becoming an attractive approach to high performance computing. This approach makes use of existing resources that are often under-utilized to achieve better performance without the extra cost of adding new hardware. Compared to a multiprocessor or a distributed memory multicomputer system such as BBN Butterfly, nCube, Sequent Symmetry, and Intel Hypercube, workstations are available to a much larger population of users; they are capable of working independently, or cooperating with others to execute a parallel application. Workstations have a stable and friendly programming environment. For those compute intensive sequential applications that run on workstations, parallelizing them holds the promise of performance improvement without the trauma of moving to a different, mostly likely less friendly environment.

The typical architecture of a workstation cluster is a number of uni- or multi-processor hosts interconnected by a serial communication network such as Ethernet or FDDI. Each host may have local disk for paging, swapping and file storage. This architecture is quite different from those in more tightly coupled parallel systems. For example, the communication bandwidth available in a workstation cluster is typically much lower, suggesting that effective parallel computing may only be achievable for those applications with relatively low communication/computation ratio and coarse grain parallelism. However, the availability of local disk and the scalability of workstation clusters present the possibility of supporting certain types of applications better on workstations. We expect that the domain of suitable parallel applications for workstation clusters to *overlap*, rather than being a *subset*, of those for multiprocessor and distributed memory multicomputers. Identifying the domain of suitable applications is an important research problem. Due to the lack of experience in this area, we believe that the most effective approach to this problem is concrete case studies.

A number of works have been reported exploiting the CPU power of a group of workstations to achieve high performance computing for numerical problems [7, 2]. In this paper, we examine the viability of exploiting the combined *file I/O* and *buffer cache* capacity of a group of workstations. Intuitively, a group of workstations offer a larger buffer space than an individual machine, and the local disks attached to the workstations can be accessed simultaneously. On the other hand, a parallelizable application whose subtasks involve intensive disk I/O is inherently of relatively coarse grain, rendering itself less sensitive to the slow interconnection links in a workstation cluster. The potential for such applications to effective utilize resources available in a workstation cluster should be exploited.

We study a full text retrieval application by implementing it on a network of DEC5000 workstations, and measuring its performance. The main issue investigated here is whether or not the application can benefit from parallel processing with a group of fast workstations connected by a relatively slow network (i.e., the Ethernet), both are shared by other users in the system. We also study the important factors affecting the performance of the implementation.

Text databases are traditionally large and archival in nature, with insertions performed regularly but almost no deletions or updates. Examples of such applications can be found in a wide range of areas:

- electronic office filing [1];
- computerized libraries [8];
- automated law and patent offices [10];

All these applications require an efficient full text search method. Like many other database applications, full text search is I/O bound. For example, over 60% of the execution time is devoted to disk accesses when running a signature file-based text search program sequentially on today's workstations[5]. With future faster CPUs, the portion of time for I/O activities will increase further. A key issue in parallelization is how to create opportunities for concurrent accesses to the signature file, which *resides on disks*, without resorting to redundant storage. *Current Frame Signature Files* (CAT), proposed in [6], were shown to be able to solve the problem effectively. In the subsequent sections, we describe an implementation of a full text retrieval system based on CAT on a network of workstations. We present performance results from experiments to demonstrate the effectiveness of running such application on a workstation cluster. Our system is implemented on top of the PVM (Parallel Virtual Machine) software [4], which offers a convenient means for inter-processor communication.

It is worth noting at the outset that for many database applications, a simple query into the database can be processed in a few seconds to a few minutes, as opposed to some numerical programs, which take hours or even months to complete. The question is why it is interesting to speed up such a process by parallelism? The answer is two fold:

- a query can be arbitrarily complex and the query may not always be decomposable into *independent* sub-queries, dictating that the sub-queries can only be processed in a serial fashion. In such a case the only way to speed up processing a query is to speed up processing the simple sub-queries;
- while a query may just take a fraction of a second to complete, the database engine stays on for a long time to serve many users. The throughput of the database engine is also an important performance criterion. High throughput can be achieved by speeding up response time.

In addition to the quest for speed, parallelism provides users a much larger combined storage space. We note that size of a textual database can easily exceed the usable capacity of a single hard disk attached to a workstation. When this happens, spreading data to multiple disks becomes necessary.

Word	Signature
free	1000001000
text	0100100000
doc. signature	1100101000

Figure 1: A word signature is created by setting 2 bits by each word in the bit vector. The document signature is the OR-ing of word signatures.

The paper is organized as following: Section 2 gives a background on the signature file method, CAT and the supporting software, PVM; section 3 describes the design and implementation of the system; section 4 present results from experiments; section 5 concludes our work.

2 Background

In this section we will give a background on full text retrieval with the signature file methods. We will also describe the PVM system we used as a support facility for network communication.

2.1 Signature Files

Generally, a signature file method is a technique that encodes the textual documents by hashing and a superimposed coding method. The code so generated is called a signature. A signature is used to identify a document. A basic form of query to a text database is to find documents that contain certain words. We will consider only boolean query in this paper. If the query is a conjunction of words (logic And) then a set of documents which contain *all* the words in the query are returned. If the query is a disjunction of words (logic Or) then documents which contain *at least one* query word are returned. We note that disjunctive query is usually treated as the union of single word queries for all the words in the disjunct.

There are different ways to create a document signature. Among the alternatives, the superimposed coding method appears most promising. It works as follows: a word signature is created by setting m bit positions in an F bits vector (where m and F are design parameters) using a hash function. A document signature is an OR-ing of all the word signatures in the document, as illustrated in Figure 1. When a query is given, word(s) in the query are first translated into a query signature and then the document signatures are scanned to eliminate the non-qualified ones. A document signature is considered qualified if all the bits set in the query signature are also set in the document signature. Documents corresponding to the few remaining signatures

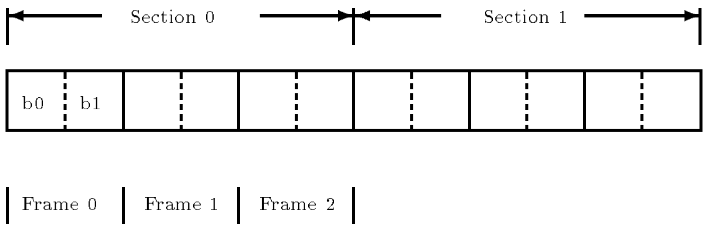

Figure 2: Example of the new signature format: $F = 12, s = 2, n = 2$, and $m = 1$ (one of b_0, b_1 is set to 1).

are examined in their entirety to determine if they are *indeed* qualified documents. Documents which do not satisfy the query but whose signatures somehow qualify are called *false drops*. The percentage of false drops for a query is determined by several design parameters of the signature file, as will be explained later. Using signature files, the time to retrieve documents can be shortened because the overhead of scanning the signatures may be more than covered by reducing number of documents actually read.

Compared to inverted file methods, signature file methods usually incur substantially smaller space overhead and it can handle insertions much more efficiently [6]. However, scanning the entire signature file could be intolerably slow except for a small database. Partitioning the signature file is necessary [5] to reduce the amount of data that need to be processed. Below, we describe a method that

- partitions the signature files effectively, and it
- creates opportunities for concurrent disk access.

2.2 Concurrent Frame Signature File

Concurrent Frame Signature File (**CAT**) was propsoed in [6]. Here is how a **CAT** signature is generated:

Assume the signature vector is F bits long, the vector is divided into k frames of s bits each (i.e. there are $k = F/s$ frames in a signature). Frames are grouped into n sections with k/n frames each. A word signature is created by selecting *one frame from each section* in the signature vector, and setting m (not necessarily distinct) bits in the frames. F, s, n and m are design parameters. Figure 2 illustrates a signature with $F = 12$ bits, $s = 2$ bits, $n = 2$ sections, and $m = 1$ bit. There are 3 frames in each section. Frequently used variables and their meaning are listed in Table 1.

It was shown [6] that the false drop probability is:

$$F_d \simeq [\sum_{t_1=0}^{D} B(D, t_1, n/k) P_{set}(t_1, m)]^n \qquad (1)$$

Name	Meaning
F	Signature length
s	Frame size
k	Number of frames
n	Number of frames selected by a word
m	Bits being set by one word
O_v	Space overhead
F_d	False drop probability
N	Number of documents
P	Number of processors
L	Length of document
D	Number of distinct words in one document
B	Disk block size
p_t	Pointer size
$T_{r,s}$	Response time on single word query
T_i	Response time on insertion
T_{seek}	Avg. disk seek + latency time
T_{trans}	Avg. sequential disk transfer time (1 KB)
T_{scan}	Avg. CPU time for scanning 1 KB data

Table 1: Variables and their meaning

where

$$\begin{aligned} B(X, y, p) &= \binom{X}{y} p^y (1-p)^{X-y} \\ P_{set}(x, m) &= (1 - (1 - 1/s)^{mx})^m \end{aligned}$$

It happens that the false drop probability of **CAT** is almost identical to that of the Frame Sliced Signature Files (**FSSF**) [5], given the same signature size. It will be shown later that the amount of data to retrieve in **CAT** is also the same as FSSF for a query. **FSSF** is a unified framework for several known superimposed coding methods [5, 6] including **SSF**, **BSSF** and **B'SSF**. Because of this **FSSF** is *capable of outperforming any of the special cases by whatever criteria.* The same claim can be make by **CAT** [6].

With **CAT**, signature sections can be distributed to different disks so that they can be accessed in parallel. A simulation studied on the BBN Butterfly TC2000 multiprocessor had demonstrated that the gain in speed can be very substantial [6]. In this paper, we examine the feasibility of such method for a workstation environment. We will describe the desgin of such a system in more detail in next section. The key issue appears to be whether or not the saving in the time for random disk access (due to partitioning and buffering) can amortize the overheads of communication and synchronization, which is much higher in a network of workstations compared to a tightly coupled multiprocessor.

2.3 The PVM System

PVM, *Parallel Virtual Machine*, is a software that allows the utilization of heterogeneous network of parallel and serial computers as a single computational resource [4]. **PVM** provides asynchronous message-passing primitives for inter-process communication and coordination in a network of workstations. **PVM** has a convenient interface to user programs, and it appears to be able to offer "adequate" performance for the kind of computation in our text search system. We would like to clarify, however, that the choice of **PVM** is more out of the consideration of *convenience* rather than *performance*.

3 Implementation

3.1 System Overview

The text retrieval system consists of one **host**, a group of **search engines**, and several **query windows**. We note that all these are functional components in the system. Their mapping to physical processors will become apparent shortly. The host program is started by the system as a server. The host program initiates one search engine on each workstation, which stores parts of the signature file on its local disk. The host can be run either on a dedicated machine, or it may share a machine with a search engine. The host program and the search engines will stay on in the system for a long time as servers. Query windows are opened (and closed) by users. They can be run on any machine in the network. Once a query window is opened, it will establish contact with the host. Multiple query windows can be active at the same time to support multiple users. A query window parses user queries, generates query signatures and forwards them to the host. The host will then distribute the query signature (or a portion of the signature) to the search engines. Results of the query will be returned directly from a search engine to the query window which issued the query. It can be seen that the host program serves as a parallel **task coordinator** after startup. Using a coordinator avoids the possibility of query signatures arriving at different search engines out of sequence. The centralized coordinator could potentially be a performance bottleneck, however, for the current implementation we are not concerned with the problem since our workstation cluster is quite small (i.e., up to ten workstations). Figure 3 illustrates the system architecture.

3.2 Parallel Search Engines

Assume that we have P machines: $M_0, M_1,...,M_{P-1}$, with P disks $D_0, D_1,...D_{P-1}$. M_i's are also referred to as *search engines*.

Storage of the text database: the text database is divided into P *fragments* , Doc_0, Doc_1,..., Doc_{P-1}.

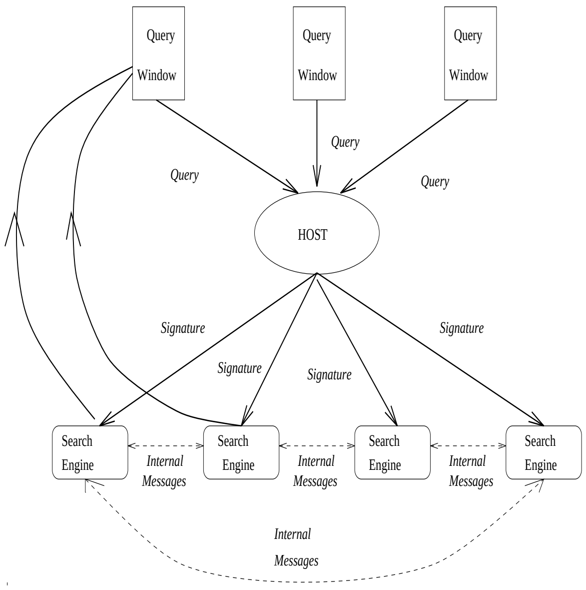

Figure 3: System Architecture.

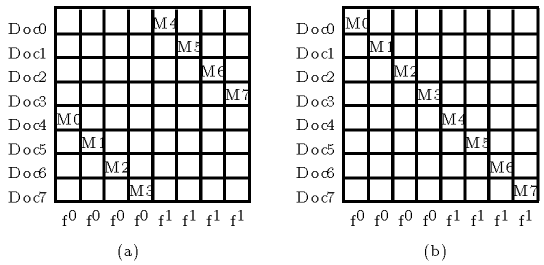

Figure 4: Example of the concurrent pipe-lined execution: processor M_i is processing the segment (row) of the frame (column) that is marked M_i in the figure.

Fragment Doc_i is stored on disk D_i, along with the correspondent portion of the pointer file. In the following, the portion of the signature file corresponding to fragment Doc_i is referred to as signature *segment i*.

Storage of the signature file: mapping the signature file onto disks will be described after we present the retrieval algorithm. At the moment, assume that a complete copy of the signature file is available on every disk.

Retrieval algorithm: first, let us use an example to illustrate the idea of the retrieval algorithm. Figure 4 is an example in which there are 8 machines ($P = 8$), and the number of frames selected by a word is 2 ($n = 2$). In the example, f^0 and f^1 are frames selected by the given query word. Each frame is divided into to P segments (rows). In the figure, each frame is also duplicated P/n times. We note that this is solely for the sake of clarity in visualizing the process of scanning the signature frames in parallel. Here is how some segments of the two frames are processed:

1. Machine M_0 begins with processing the 4th segment of frame f^0; at the same time machine M_4 begins with processing the 0th segment of frame f^1.

2. machine M_0 processes the 0th segment of frame f^0 and at the same time it joins the result from machine M_4, which is available as M_4 is expected to have completed scanning the 0th segment of frame f^1; similarly, M_4 processes the 4th segment of frame f^1 and joins the result from M_0.

After these two steps, machine M_0 has a list of qualified signatures belonging to segment Doc_0; machine M_4 has a list of qualified signatures belonging to segment Doc_4.

Other machines work similarly on other segments of the signature file. Figure 4a is a snapshot of machines at their first stage. A grid is marked with the ID of a machine that is working on the correspondent segment (row label) of the selected frame (column label). Figure 4b is the second and final stage.

A general algorithm can be found in [6], along with a scheme for distributing signature frames onto disks. We note that with the algorithm, signature frame units are *not* duplicated on disks.

4 Experiment Results

Two text databases were used in the experiment. One has 100,000 documents, the other has 200,000 documents. The document size is 1 KB on average. Response time was obtained by measuring the time to process 1,000 unsuccessful queries and then dividing the time by 1,000. Experiments were run with 5 and 10 DEC5000 workstations connected by Ethernet. Each workstation has its own local disk.

As mentioned before, we used the **PVM** library calls for network communication. There are two sets of *send* and *receive* commands provided by **PVM**: they are the *snd(), rcv()*, and *vsnd(), vrcv()* commands. These commands offer the same functionality, but their implementations are different [4]: the former calls involve the **PVM** daemon program, while the latter bypass the daemon. Our experiment shows that the *vsnd()*, and *vrcv()* commands take less than 2/3 the time of *snd(), rcv()* to complete a set of message-passing calls used in our program. Results in the rest of the paper reflect implementation using the *vsnd()* and *vrcv()* calls.

The followings are design parameters for the signature file: $F = 1640$ (i.e., $O_v = 20\%$), $n = 5$, $s = 8$, $m = 2$[1]. A procedure for selecting the optimal parameters can be found in [5]. Expected false drop probability by formula (1) is 7×10^{-5}. For the database with 100,000 documents the average number of false drops is expected to be 7 for a query, for the one with 200,000 documents the number is 14.

CAT-synch	using the concurrent pipelined algorithm in section 3.
CAT-bulk	processing all segments in one section at once. see text for more details.
NAIVE[6]	one processor is assigned to one segment.

Table 2: Options in Signature File Scanning.

4.1 Experiments

Table 2 explains different options of the system we experimented with. These options are used with or without user buffers that cache the entire signature frame to be processed by a processor. We measure the followings in the experiment: response time (physical clock), actual false drop rate, and the time the host program takes to broadcast queries to the search engines. We also obtain the execution time profile of a search engine. All timing measures are taken while the load of the workstations is at a typical working day level unless specified otherwise. There are very few disk i/o activities; the CPU load factor varies but no CPU-intensive program is running. Our program usually adds 20% to the CPU load scale. To approximate the *sequentiality* assumption (section 3) our signature files are created on disk partitions that are near empty.

The **CAT-bulk** algorithm needs more explanation: in the **CAT-synch** algorithm, machines send and receive messages *during* scanning the signature frames. We suspect that this could cause machines "jerking" each other and it may lead to longer execution time. We tried the **CAT-bulk** algorithm. The idea of the algorithm is that, instead of sending and receiving a (partially) processed bit vector (B_l) from another processor, a machine processes all the segments of its signature frame at once and then sends the bit vectors to machines responsible for scanning the false drops for the corresponding database fragments. The bit vectors for a database fragment are merged by the machine assigned to that fragment. With this strategy, communication occurs *only after* signature frames are scanned.

The **NAIVE** method simply divides the database into fragments. Each database fragment, along with the correspondent signature file segment, is assigned to one machine. As explained in [6], the **NAIVE** method cannot reduce the number of *random disk* accesses. How-

[1] These parameters are not *optimal* due to the fact that the current system can only handle signature frames of the size of a multiple of one byte.

	Proc	100M DB		200M DB	
CAT-synch	5	532	3.52	N/A	
	5*	337	5.56	N/A	
	10	330	5.67	586	6.24
	10*	207	9.02	368	9.94
CAT-bulk	5	632	2.97	N/A	
	10	376	4.98	664	5.51
NAIVE	5	674	2.78	N/A	
	10	383	4.89	752	4.86

Table 3: Average Query Response Time (ms) and Speed-up (including scanning false drops). * indicates that the whole signature file is cached.

Words/Query	False Drop Rate
1-word	$7.78 * 10^{-5}$
2-word	$7.72 * 10^{-5}$
3-word	$6.80 * 10^{-5}$

Table 4: False drop rate by experiment.

Proc	Broadcasting Time	Standard Deviation
5	32ms	0.75ms
10	65ms	0.75ms

Table 5: The Cost of Broadcasting. Data are taken when no other processes are active in the system.

ever, the **NAIVE** method allows machines to operate independently.

Table 3 gives the response times and the speed-up factors from running the system with the different options listed at Table 2. Response time is averaged on single word, two word and three word conjunctive queries[2], of five runs each. The standard deviation for all the response times in the table is 29 ms or less. The speed-up factors are calculated against the sequential program (as opposed to the parallel version running on a single machine). The sequential program takes an average of 1.9 seconds on the 100 MB database and 3.6 seconds on the 200 MB database for a single word query. We note that the unusually large speed-up factors from experiments using user buffer cache should not be a surprise because the actual amount of work in the parallel program is less than that in the sequential one as disk I/O is completely eliminated in the parallel program, an advantage made possible by parallelization. Notice also that the difference in response times between the 100 MB database on 5 machines and the 200 MB database on 10 machines (with or without buffers) appears to be a constant (except with the **NAIVE** method which needs to transmit a larger amount of data). This seems to suggest the existence of a non-parallelizable component in the system because the amount of work performed by each search engine is about the same in both cases. We will elaborate on this after presenting the execution profile of the search engines.

The **CAT-synch** method yields speed-up better than the **CAT-bulk** and the **NAIVE** methods. The comparison between **CAT-synch** and **NAIVE** is consistent with the findings in [6]. The cause of difference between **CAT-synch** and **CAT-bulk** will be explained in the next subsection.

Table 4 lists the actual false drop rate observed from experiment. As mentioned before, the expected false drop probability by formula (1) is 7×10^{-5}. The experimental results agree well with the expectation (i.e., the maximal difference is $< 12\%$).

Table 5 gives the overhead of broadcasting the query to search engines by the host program. Results are obtained from 5 consecutive runs, with on other processes active in the system. As can be seen, the cost increases linearly with the number of machines. This is because the current **PVM** implementation treats broadcasting as sending the message to every machine sequentially. Broadcasting is thus a serial operation.

Table 6 is an execution profile of a search engine. The detailed timing statistics for data processing (i.e., CPU time) in each parallel search engine are taken by the Unix profiling facility on the sequential program running with a database of a size equal to one database fragment (20 MB) in the parallel system. With the same amount of data, the amount of work by the sequential program is the same as that by one parallel search engine. Broadcasting costs are taken from table 5. The overhead of communication and synchronization is computed[3] by subtracting the data processing time, disk I/O time, and broadcasting time from the experimental response time. Listed also in the table are the numbers of non-voluntary context switches on each engine while the engine is running in parallel. The number of non-voluntary context switches indicates that the interference from the UNIX scheduler is fairly substantial: there are over 15 such switches in a period of 500 to 600 ms (i.e., the average query response time). We note that the broadcasting cost plus communication/synchronization overhead amounts to over 40% of the total execution time. However, the *only* non-parallelizable bottleneck is broadcasting the query signature, which is a serial operation as shown in table

[2] As mentioned in the previous section, **CAT** is able to achieve the same response time for single word and multiple word queries by always selecting n frames for each query. Experimental results support the expectation. For instance, the average query response times are 540 ms, 521 ms and 534 ms for single word, 2-word and 3-word queries, respectively.

DB Size	100M	100M	200M
Proc	5	10	10
Avg. Rep. Time	532ms	330ms	586ms
Broadcasting	32ms/6%	65ms/20%	65ms/11%
Load Sig. Frame	165ms/31%	95ms/29%	165ms/28%
Scan Sig. Frame	82ms/15%	41ms/12%	82ms/14%
Scan False Drop	75ms/14%	38ms/12%	75ms/13%
Comm./Synch. Overheads	178ms/33%	91ms/28%	199ms/34%
Non-voluntary Context Switch	18 times	10 times	19 times

Table 6: Execution Profile of a Search Engine.

5. With the 100 MB database, the percentage of broadcasting cost increases from 6% to 20% as the number of machines doubles from 5 to 10.

4.2 Discussion

Effect of Context Switch: context switch cannot be avoided given that the application program is a UNIX process. In other words, the processes performing signature scanning are subject to interruption by the UNIX scheduler. The cost of context switch is high (e.g., 18 non-voluntary context switch for one query). However, some of the overhead can be hidden as explained below.

The difference between **CAT-synch** and **CAT-bulk** methods is that, in **CAT-bulk**, scanning signature segments are done all in once and the results from the scanning are sent at the end. It can be shown that the *amount of work* done is the same in the two methods, so is the *number of messages* being sent and received. In an environment where no other process is present, there should be no difference in response time between the two methods[4]. But when some of the machines have other processes, the outcome of the two can be much different due to the fact that the **CAT-synch** method has the *receive* call evenly spread throughout the process. Some of the waiting time of these *receive* calls can *hide* the time in which the search process is interrupted by the system. Figure 5 is an example that illustrates such phenomenon. In the example, we made a simplifying assumption that the receiving process has a fixed delay to retrieve the incoming message. The fixed delay can be considered as the minimum time from issuing the *receive* instruction to actually receiving the message. Recall that **PVM** has a synchronous *receive*, and that a Unix process may be switched out of the CPU at every system call. We further assume that the network latency is the same as the fixed delay. This explanation

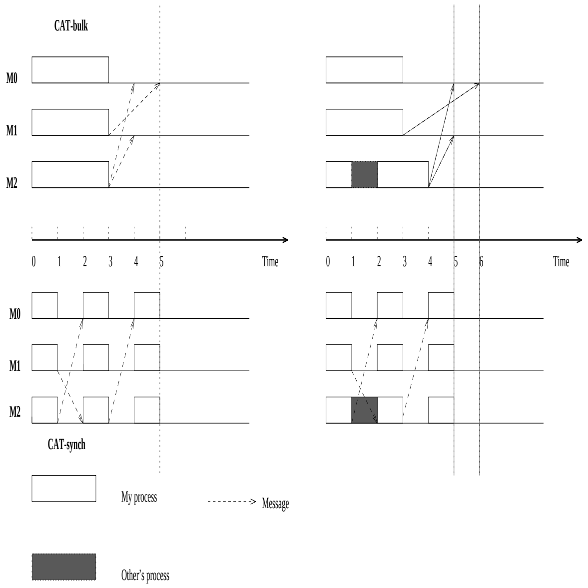

Figure 5: Effect of Interference by other Processes: the Presence of the Other Process on Machine M_3 has an Effect on the Finishing Time with the **CAT-bulk** Method but It has no Effect with the **CAT-synch** Method.

will still be valid should the above assumption be relaxed but the situation can become more complicated. We will call the strategy of arranging communication calls to be evenly distributed throughout a process as *voluntary breaking*. *Voluntary breaking* hides interruptions that prolongs execution.

Problems with the Serial Broadcasting: while the relative cost of broadcasting is not yet very significant in the total response time from our experiments, we expect that the cost will soon become a dominant factor as the number of machines increases. The reason is the following: the cost of broadcasting (as currently implemented in **PVM**) is linear to the number of machines, but actual data processing time on *each* search engine is inversely proportional to the number of machines. Therefore, the relative broadcasting cost (i.e., percentage of the actual data processing time) is at the order of $O(P^2)$ with a database of a fixed size, where P is the number of machines. In other words, the size of the database has to quadruple if the number of machines doubles, in order to retain a constant relative cost of broadcasting. We consider this to be a serious limitation as the rest of our program incurs overheads that are almost a constant fraction of the actual data processing time (see table 6) on each search engine, given that the size of the database increases proportionally to the number of machines in the cluster.

[3] We were not able to measure communication and synchronization overheads directly.

[4] We are able to verify this on machines on which our processes are the only ones that are active.

4.3 Comparison with other Architectures

A performance study of CAT was done by Lin on the Butterfly TC2000 multiprocessor [6]. Disk accesses were emulated in that study as no disks are attached to individual processors. In a tightly-coupled multiprocessor such as the Butterfly TC2000, on which one can execute a program in a dedicated processor cluster, the communication and synchronization overhead is insignificant for the kind of computation granule in full text retrieval. Speed-up of 14 is obtained with 16 processors [6].

Compared to mainframe machines for textual databases: we may assume that there is much more memory available in a mainframe so that a significant portion of the signature file can be stored in the main memory. In such a case the I/O bottleneck can be alleviated. The trade-off is that the cost of a mainframe is substantially higher than a small workstation cluster. Besides, the advantage of having more memory on a mainframe appears to be diminishing as most modern workstations can be configured with a very large amount of cheaper memory should one choose to. A workstation cluster is expected to be able to offer a better performance/cost ratio.

5 Conclusion

We investigated an I/O intensive application, parallel full text search, by implementing it on a network of workstations. Our study shows that substantial improvement in speed can be obtained through parallelism in disk accesses, despite the high communication and synchronization overhead that is incurred in such a loosely-coupled system. Several problems pertinent to parallel I/O applications on this type of computing environment were discussed. Namely, the effect of synchronization delays and and the serial bottleneck in broadcasting. It appears that for an application that is parallelizable, and that it has subtasks requiring intensive disk I/O, a network of workstations with local disks could be an effective platform to achieve better performance. The two major advantages of a workstation cluster are: (1) substantially larger combined I/O cache space, (2) possible concurrent disk accesses. We demonstrated that these advantages can be exploited to achieve high performance without adding extra hardware.

Serial broadcasting has been identified as the sole sequential bottleneck in our implementation. We expect that our system is scalable to workstation clusters of larger size should an efficient broadcasting mechanism become available.

Communication and synchronization overhead is very substantial as shown in our study. However, we believe there is still much to improve in term of execution speed by making the supporting message passing software more efficient.

More challenging problem arises when load balancing becomes an issue. In this case study, we did not encounter such a problem because the application chosen is decomposable into subtasks of nearly equal size. We plan to look into this issue in the future.

References

[1] S. Christodoulakis, F. Ho, and M. Theodoridou, "The Multimedia Object Presentation Manager in MINOS: A Symmetric Approach," *Proc. ACM SIGMOD,* May 1986.

[2] A. L. Cheung and A. P. Reeves, "High Performance Computing on a Cluster of Workstations", *Proceedings of the First International Symposium on High-performance Distributed Computing*, Syracuse, New York, Sept, 1992.

[3] C. Faloutsos, "Signature-Based Text Retrieval Methods: A Survey", *IEEE Data Engineering*, Vol. 13, Mar. 1990. pp. 25-32.

[4] G.A. Geist, "Network Based Concurrent Computing on the PVM System", Technical report TM-11826, Oak Ridge National Lab., 1991.

[5] Z. Lin and C. Faloutsos, "Frame-sliced Signature files", *IEEE Transaction on Data Engineering*, Vol 4, NO. 3, June 1992.

[6] Z. Lin, "Concurrent Frame Signature Files", *Distributed and Parallel Databases: An International Journal*, Vol 1. No. 3. July, 1993

[7] M. Parashar and S. Hariri, "A Requirement Analysis for High Performance Distributed Computing over LAN's", *Proceedings of the First International Symposium on High-performance Distributed Computing*, Syracuse, New York, Sept, 1992.

[8] J. Price, "The Optical Disk Pilot Project at the Library of Congress", *Video-disc and Optical Disk,* vol. 4, no. 6, pp. 424-432, Nov. 1984.

[9] R. Sacks-Davis, A. Kent, and K. Ramamohanarao, "Multikey Access Methods Based on Superimposed Coding Techniques". *ACM Transaction on Database Systems*, Vol. 12, No. 4, Dec. 1987.

[10] C. J. Van-Rijsbergen, *Information Retrieval,* Butterworths, London, England, 1979.