MARCH 1986 VOL. 9 NO. 1

a quarterly bulletin
of the IEEE computer society
technical committee
on

Database Engineering

CONTENTS

SPECIAL ISSUE ON HIGH PERFORMANCE TRANSACTION SYSTEMS

Database Engineering Bulletin is a quarterly publication of the IEEE Computer Society Technical Committee on Database Engineering. Its scope of interest includes: data structures and models, access strategies, access control techniques, database architecture, database machines, intelligent front ends, mass storage for very large databases, distributed database systems and techniques, database software design and implementation, database utilities, database security and related areas.

Contribution to the Bulletin is hereby solicited. News items, letters, technical papers, book reviews, meeting previews, summaries, case studies, etc., should be sent to the Editor. All letters to the Editor will be considered for publication unless accompanied by a request to the contrary. Technical papers are unrefereed.

Opinions expressed in contributions are those of the individual author rather than the official position of the TC on Database Engineering, the IEEE Computer Society, or organizations with which the author may be affiliated.

Membership in the Database Engineering Technical Committee is open to individuals who demonstrate willingness to actively participate in the various activities of the TC. A member of the IEEE Computer Society may join the TC as a full member. A non-member of the Computer Society may join as a participating member, with approval from at least one officer of the TC. Both full members and participating members of the TC are entitled to receive the quarterly bulletin of the TC free of charge, until further notice.

Guest editors' introduction to HPTS papers in Database Engineering

The Workshop on High Performance Transaction Systems, sponsored by IEEE and ACM, was held at Asilomar, California on September 23rd to 25th, 1985. The Workshop brought together about 70 developers and users of high volume transaction processing systems to exchange views, discuss experiences, and present methods for constructing large transaction processing systems. The presentations ranged from reports on the use and management of very large, on-line systems to methods for improving performance in specific areas of high performance transaction systems.

Two broad themes emerged as basic issues for high volume transaction systems: the methods used to divide the work when one CPU cannot handle the entire workload, and the management of interference between concurrent transactions in large systems.

When one CPU cannot service the workload, three alternative architectures are available: shared memory multi-processors; shared disk multi-processors; and fully partitioned multi-processor systems. Shared memory systems must communicate to synchronize processor caches. Shared disk systems communicate to synchronize disk buffers in private memories and to serialize transaction access to the shared data. Partitioned systems must communicate to transfer transaction execution to the partitions holding the needed data and to transfer results. The tradeoffs between intense, low level communication and less intense, high level communication were much discussed at the Workshop.

Another theme of the Workshop centered on the problems introduced by concurrent execution of (large) batch transactions and (small) on-line transactions, and the problems caused by frequent updates of summary data (i.e., "hot spots"). Both problems seem to require specialized techniques to allow concurrent access to data without compromising data integrity. Among the techniques discussed were multi-version data bases, breaking up batch transactions, and fuzzy values for "hot spots".

In this issue of Database Engineering, we present four articles from the Workshop on High Performance Transaction Systems. We concentrate on the architectural theme of the Workshop, with articles representing each of the three partitionings of the workload. Mike Stonebraker discusses the tradeoffs among the architectures and advocates the partitioned data architecture. Kurt Shoens discusses shared disk systems with an emphasis on methods used to insure correct synchronization of main memory buffer pools and transaction execution. Phil Bernstein describes the approach taken to synchronize caches in the shared memory Sequoia system. Finally, Livney, Khoshafian, and Boral present a study of data placement and arm scheduling for large systems with many disks. Their simulation results confirm the folklore about spreading data over many arms to even out the request load.

The Workshop on High Performance Transaction Systems, by bringing together expert designers and practitioners, provided a forum for evaluation and discussion of the problems and promises in a increasingly important computing domain. The articles reproduced in this issue of Database Engineering will allow readers to better understand some of the basic issues and to follow the debates over the "best" approach to high performance transaction processing.

Irving L. Traiger
Bruce G. Lindsay
(Guest Editors)

Letter from the Editor-in-Chief

Following the four papers from the Workshop on High Performance Transaction Systems, we have added a fifth, by Madnick, on the INFOPLEX project at MIT. The paper is related to the central theme of this issue; it summarizes an ongoing research effort on fundamental architectural changes to support large-scale information management.

The upcoming June issue of **Database Engineering** is being guest-edited by Sham Navathe. It will consist of summaries of the four panels at the recent Database Directions IV: Information Resource Management Workshop (sponsored by the National Bureau of Standards).

I have received a number of letters about missing or delayed copies of **DBE**, and about difficulties people have had in getting onto our mailing list. My apologies for the problems, some of which are due to a data entry backlog at the IEEE Computer Society headquarters in Washington.

- If you think that an earlier request to join the TC on DBE has not been processed, or you are not a member of the TC, ask the Computer Society for an application form and (re)apply. Their address is: IEEE Computer Society, 1730 Massachusetts Avenue NW, Washington, D.C. 20036-1903. I have spare TC application forms as well (at the address on the inside front cover). Membership is free.
- If you have recently changed your address and have stopped receiving issues, be sure to notify the IEEE Computer Society (at the above address) as well as the IEEE. It probably wouldn't hurt to mention that you're in the TC on Database Engineering, and want to continue to receive **DBE.**
- If you have missed any of the 1985 issues of **DBE,** or would like to have them collected together in a convenient format, the IEEE Computer Society will shortly be publishing them as a softbound volume. This will be **Volume 4** of the collected issues of **DBE.** Volumes 1-3 (1982-84) are also available, as far as I know.
- If all else fails, I have a few spare copies of specific back issues and can send these out until my supplies are gone.

Finally, if you know of anyone who has had subscription problems, please pass along the above suggestions.

David Reiner, Editor-in-Chief
Database Engineering

The Case for Shared Nothing

Michael Stonebraker
University of California
Berkeley, Ca.

ABSTRACT

There are three dominent themes in building high transaction rate multiprocessor systems, namely shared memory (e.g. Synapse, IBM/AP configurations), shared disk (e.g. VAX/cluster, any multi-ported disk system), and shared nothing (e.g. Tandem, Tolerant). This paper argues that shared nothing is the preferred approach.

1. INTRODUCTION

The three most commonly mentioned architectures for multiprocessor high transaction rate systems are:

shared memory (SM), i.e. multiple processors shared a common central memory

shared disk (SD), i.e. multiple processors each with private memory share a common collection of disks

shared nothing (SN), i.e. neither memory nor peripheral storage is shared among processors

There are several commerical examples of each architecture. In this paper we argue that SN is the most cost effective alternative. In Section 2 we present a "back of the envelope" comparison of the alternatives. Then in Sections 3 through 5 we discuss in more detail some of the points of comparison.

2. A SIMPLE ANALYSIS

In Table 1 we compare each of the architectures on a collection of 12 points. Each architecture is rated 1, 2 or 3 to indicate whether it is the best, 2nd best or 3rd best on each point. For certain points of comparison, there are apparent ties. In such situations we give each system the lower rating. Most of the ratings are self-evident, and we discuss only a few of our values.

The first two rows indicate the difficulty of transaction management in each environment. SM requires few modifications to current algorithms and is the easiest environment to support. Hence it receives a "1" for crash recovery. The "2" for concurrency control results from the necessity of dealing with the lock table as a hot spot. SN is more difficult because it requires a distributed deadlock detector and a multi-phase commit protocol. SD presents the most complex transaction management problems because of the necessity of coordinating multiple copies of the same lock table and synchronizing writes to a common log or logs.

This research was sponsored by the U. .S. Air Force Office of Scientific Research Grant 83-0254 and the Naval Electronics Systems Command Contract N39-82-C-0235

System Feature	shared nothing	shared memory	shared disk
difficulty of concurrency control	2	2	3
difficulty of crash recovery	2	1	3
difficulty of data base design	3	2	2
difficulty of load balancing	3	1	2
difficulty of high availability	1	3	2
number of messages	3	1	2
bandwidth required	1	3	2
ability to scale to large number of machines	1	3	2
ability to have large distances between machines	1	3	2
susceptibility to critical sections	1	3	2
number of system images	3	1	3
susceptibility to hot spots	3	3	3

A Comparison of the Architectures

Table 1

The third and fourth points are closely related. Data base design is difficult in current one-machine environments, and becomes much harder in an SN system where the location of all objects must be specified. The other environments do not add extra complexity to the one-machine situation. Balancing the load of an SN system is complex, since processes and/or data must be physically moved. It is obviously much easier in the other environments. The next five points are fairly straightforward, and we skip forward to critical sections. They have been shown to be a thorny problem in one-machine systems [BLAS79], and an SN system does not make the problem any worse. On the other hand, an SM system will be considerably more susceptible to this problem, while an SD system will be in-between. SN and SD systems have one system image per CPU, and system administration will be more complex than an SM system which has only a single system image. Lastly, all architectures are susceptible to hot spots.

Several conclusions are evident from Table 1. First an SM system does not scale to a large number of processors. In my opinion this is a fundamental flaw that makes it less interesting than the other architectures. Second, an SD system excells at nothing, i.e. there are no "1"s in its column. Lastly, one should note the obvious marketplace interest in distributed data base systems. Under the assumption that every vendor will have to implement one, there is little or no extra code required for an SN system. In order to justify implementing something else (e.g. SD) and paying the extra software complexity, one should be sure that SN has some insurrountable flaws. In the next section we discuss the issues of data base design, load balancing and number of messages, which are points of comparison where SN was the poorest choice. In each case we argue that the problems are unlikely to be very significant. Then we discuss hot spots in Section 4, and argue that these are easier to get rid of than to support effectively. Lastly, we discuss concurrency control, and suggest that scaling to larger data bases is unlikely to change the ratings in Table 1. Hence, we will conclude that SN offers the most viable and cost effective architecture.

3. Problems with Shared Nothing

It appears that most data base users find data base design to require substantial wizardry. Moreover, tuning a data base is a subject that data base vendors have clearly demonstrated proficiency relative even to the wisest of their customers. To ordinary mortals tuning is a "black art".

Consequently, I expect many automatic tuning aids will be constructed for most data managers, if for no other reason than to lower the technical support burden. There is no evidence that I am aware of that such tuning aids will be unsuccessful. Similarly, there is no evidence that automatic data base design aids will fail in an SN environment where the data base is partitioned over a collection of systems. Furthermore, balancing the load of an SN data base by repartitioning is a natural extension of such a design aid. Moreover, applications which have a stable or slowly varying access pattern will respond successfully to such treatment and will be termed **tunable**. Only data bases with periodic or unpredictable access patterns will be untunable, and I expect such data bases to be relatively uncommon. Hence, load balancing and data base design should not be serious problems in typical environments.

Consider the number of messages which an SN system must incur in a typical high transaction processing environment. The example consists of a data base with N objects subject to a load consisting entirely of transactions containing exactly k commands, each affecting only one record. (For TP1 the value of k is 4). For any partitioning of the data base, these k commands remain single-site commands. Suppose that there exists a partitioning of the data base into non-overlapping collections of objects such that all transactions are locally sufficient [WONG83]. Such a data base problem will be termed **delightful**. Most data base applications are nearly delightful. For example, the TP1 in [ANON84] has 85% delightful transactions.

Assume further that the cost of processing a single record command is X and the cost of sending and receiving a round-trip message is Y. For convenience, measure both in host CPU instructions, and call T = X/Y the **technology ratio** of a given environment. Measured values of T for high speed networks and relational data bases have varied between 1 and 10 and reflect the relative efficiency of data base and networking software in the various situations. An environment where each is tuned well should result in a T of about 3. We expect the long term value of T to stay considerably greater than 1, because it appears much easier to offload network code than data base code.

As long as T >> 1, network costs will not be the dominent system cost in delightful data bases; rather it will be processing time on the local systems. Moreover, data bases that are nearly delightful will require a modest number of messages. (With a reasonable amount of optimization, it is conceivable to approach 2 messages per transaction for locally sufficient transactions.) Hence, the number of messages should not be a problem for the common case, that of nearly delightful data bases.

4. Hot Spots

Hot spots are a problem in all architectures, and there are at least three techniques to dealing with them.

1) get rid of them
2) divide a hot spot record into N subrecords [ANON84]
3) use some implementation of a reservation system [REUT81]

It has never been clear to me why the branch balance must be a stored field in TP1. In the absence of incredible retrieval volume to this item, it would be better to calculate it on demand. The best way to eliminate problems with hot spots is to eliminate hot spots.

Unfortunately, there are many hot spots which cannot be deleted in this fashion. These include critical section code in the buffer manager and in the lock manager, and "convoys" [BLAS79] results from serial hot spots in DBMS execution code. In addition, the head of the log and any audit trail kept by an application are guaranteed to be hot spots in the data base. In such cases the following tactic can usually be applied.

Decompose the object in question into N subobjects. For example, the log can be replicated N times, and each transaction can write to one of them. Similarly, the buffer pool and lock table can be decomposed into N subtables. Lastly, the branch balance in TP1 can be decomposed into N balances which sum to the correct total balance. In most cases, a transaction requires only one of the N subobjects, and the conflict rate on each subobject is reduced by a factor of N. Of course, the division of subobjects can be hidden from a data base user and applied automatically by the data base designer, whose existence we have speculated in Section 3.

Lastly, when updates are restricted to increment and decrement of a hot spot field, it is possible to use field calls (e.g. IMS Fast Path) or a reservation system [REUT82]. It is clear that this tactic can be applied equally well to any of the proposed architectures; however, it is not clear that it ever dominates the "divide into subrecords" tactic. Consequently, hot spots should be solvable using conventional techniques.

5. Will Concurrency Control Become a Bigger Problem?

Some researchers [REUT85] argue that larger transaction systems will generate a thorny concurrency control problem which may affect the choice of a transaction processing architecture. This section argues that such an event will probably be uncommon.

Consider the observation of [GRAY81] which asserts that deadlocks are rare in current systems and that the probability of a transaction waiting for a lock request is rare (e.g. .001 or .0001). The conclusion to be drawn from such studies is that concurrency control is not a serious issue in well designed systems today. Consider the effect of scaling such a data base application by a factor of 10. Hence, the CPU is replaced by one with 10 times the throughput. Similarly 10 times the number of drives are used to accelerate the I/O system a comparable amount. Suppose 10 times as many terminal operators submit 10 times as many transactions to a data base with 10 times the number of lockable objects. It is evident from queuing theory that the average response time would remain the same (although variance increases) and the probability of waiting will remain the same. The analyis in [GRAY81] can be rerun to produce the identical results. Hence, a factor of 10 scaling does not affect concurrency control issues, and today's solutions will continue to act as in current systems.

Only two considerations cloud this optimistic forcast. First, the conclusion is predicated on the assumption that the number of granules increases by a factor of 10. If the size of a granule remains a constant, then the size of the data base must be linear in transaction volume. We will term such a data base problem **scalable**. Consider the transactions against a credit card data base. The number of transactions per credit card per month is presumably varying slowly. Hence, only a dramatic increase in the number of cards outstanding (and hence data base size) could produce a large increase in transaction rates. This data base problem appears to be scalable. In addition, suppose a fixed number of travel agent transactions are generated per seat sold on a given airline. Consequently, transaction volume is linear in seat volume (assuming that planes are a constant size) and another scalable data base results.

One has to think hard to discover nonscalable data bases. The one which comes to mind is TP1 in an environment where retail stores can debit one's bank account directly as a result of a purchase [ANON84]. Here, the number of transactions per account per month would be expected to rise dramatically resulting in a nonscalable data base. However, increasing the size of a TP1 problem will result in no conflicts for the account record (a client can only be initiating one retail transaction at a time) and no conflict for the teller record (a clerk can only process one customer at a time). Hence, the only situation with increased conflict would be on summary data (e.g. the branch balance). Concurrency control on such "hot spots" should be dealt with using the techniques of the previous section.

The following conclusions can be drawn. In scalable data bases (the normal case) concurrency control will remain a problem exactly as difficult as today. In nonscalable data bases it appears that hot spots are the main concurrency control obstacle to overcome. Hence, larger transaction systems in the best case present no additional difficulties and in the worst case aggravate the hot spot problem.

6. CONCLUSIONS

In scalable, tunable, nearly delightful data bases, SN systems will have no apparent disadvantages compared to the other alternatives. Hence the SN architecture adequately addresses the common case. Since SN is a nearly free side effect of a distributed data base system, it remains for the advocates of other architectures to demonstrate that there are enough non-tunable or non-scalable or non delightful problems to justify the extra implementation complexity of their solutions.

REFERENCES

[ANON84] Anon et. al., "A Measure of Transaction Processing Power", unpublished working paper.

[BLAS79] Blasgen, M. et. al., "The Convoy Phenomenon," Operating Systems Review, April 1979.

[GRAY81]	Gray, J. et. al., "A Straw Man Analysis of Probability of Waiting and deadlock," IBM Research, RJ3066, San Jose, Ca., Feb 1981.
[REUT82]	Reuter, A., "Concurrency on High-Traffic Data Elements," ACM-PODS, March 1982.
[REUT85]	Reuter, A., (private communication).
[WONG83]	Wong, E. and Katz, R., "Distributing a Data Base for Parallelism," ACM-SIGMOD, May 1983.

Data Sharing vs. Partitioning for Capacity and Availability

Kurt Shoens

IBM San Jose Research Laboratory
K55/281
5600 Cottle Road
San Jose, CA 95193

Abstract

This paper describes a transaction system architecture called *data sharing* and compares it with the *partitioned system*. Data sharing is shown to have several operational advantages over partitioned systems.

Introduction

As enterprises expand their use of computers, they demand increasing transaction processing capacity. Often, the transactions must (at least conceptually) run against a large, centralized database. In many cases, the rate of growth has exceeded the corresponding rate of growth in computer instruction execution rates and in disk I/O rates. While the disk throughput limitations can be solved by spreading the database over many independent disk volumes, more complex means are needed to cope with instruction rate limitations.

Another important trend in transaction processing systems has been *availability*. As organizations build larger single system databases and become dependent on them, the cost of service outages increases.

Several approaches have been tried to increase transaction processing capacity and availability. Chief among these are:

1. Get faster computers
2. Partition the data among several computers
3. Share the data among several computers

In order to get a faster computer, one can buy a larger, more expensive model, if one is available. Unfortunately, the instruction rates of large mainframes have not kept pace with transaction processing demands. Promising new technologies that would provide a quantum leap in single machine speeds have not yet proved to be practical. Computer manufacturers have responded by building tightly-coupled multiprocessors, where two or more processors share access to primary memory. Due to memory contention and caching problems, there seems to be a (relatively small) limit on the number of processors that can be effectively connected in a tightly-coupled system.

Having a faster computer does not solve the availability problem alone. Keeping a spare computer around in case the main one breaks is fairly effective, but expensive. Often, the spare computer can be used to run low priority work when it is not needed to substitute for the transaction processing computer.

To date, the most popular method of applying several computers to the management of a single database has been to split the data into two or more *partitions*, each of which is managed by a single computer. Examples of this approach include the Tandem NonStop system [Bartlett 78], the IMS/VS Multiple Systems Coupling Feature [McGee 77], and the Stratus system [Kim 84]. When a transaction enters some system, it is sent to the system that "owns" the data to be referenced. If the data referenced by a transaction spans partitions, most systems automatically generate a strategy to access the remote data.

A partitioned complex has the advantage that when a single system fails, the rest of the systems can continue to operate. In the simple case, requests for data owned by the failed system must wait for correction of the failure. Improved availability can be achieved by providing multiple paths to the data. Then, a surviving system can assume responsibility for data owned by a failed system and make it available quickly.

Recently, the approach of using several systems with direct access to the disks holding the database has been tried. We will show in this paper how this architecture, called *data sharing*, is an attractive alternative to partitioning. In a data sharing system, a transaction can be executed on any of the systems in the complex and reference any part of the data base. Due to the uniform accessibility of data, the failure of a single system does not preclude access by other systems.

Data sharing

Figure 1 on page 3 shows a data sharing architecture. Transactions are entered at terminals connected to *front end* processors and queued for processing on one of the *data base* processors. Each of the data base processors is a large, standard mainframe and runs a standard operating system and somewhat modified data base system. Transactions execute to completion on a single data base processor (barring failures) and return their results to the originating terminal.

Critical parts of a data sharing system can be duplexed to avoid single points of failure. For example, one would need a redundant *multi-way connection mechanism* and redundant paths to the disks. Otherwise, failures of these components could stop the entire complex.

The data sharing approach is used by the Synapse N+1 system [Kim 84], the DEC VAX Cluster system, and the IMS/VS Data Sharing Feature [Strickland 82]. We are prototyping components of an experimental data sharing system in the Amoeba Project [Shoens 85] at the IBM San Jose Research Laboratory.

New approaches to transaction management are needed for data sharing systems. These aspects are discussed in the next section.

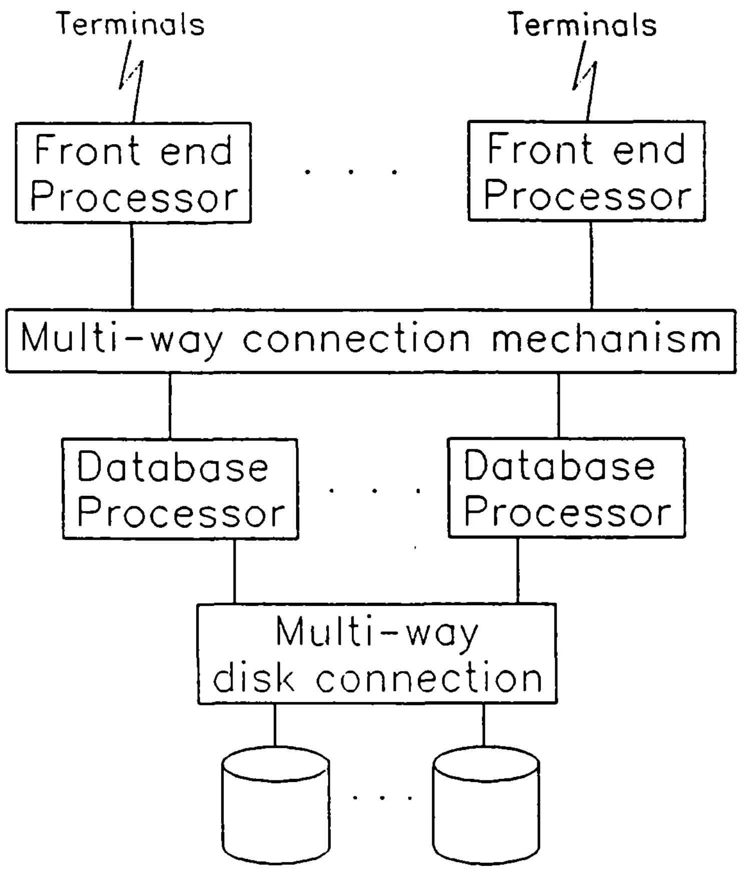


Figure 1. Data sharing architecture

Transaction management

The data sharing architecture requires new approaches to concurrency control, message queue management, and scheduling. This section highlights the differences.

Concurrency Control

Unlike single processor data base systems, a data sharing system has no instruction-accessible memory to hold the data structures needed for locking or other concurrency control schemes. Instead, arbitration must be performed through messages. Because message passing implies a large overhead (in particular, the requirement to *suspend* the requestor until a response is returned) a technique based on locality of reference is used. As succeeding transactions on a database processor reference similar parts of the database, the processor acquires temporary rights to the locks that allow it to acquire and release the locks quickly. Other approaches used combine locking techniques with optimistic concurrency control [Kung 81] to reduce the number of *synchronized* messages (where the sender synchronously waits for a reply) needed to process a transaction.

Message queue manager

Each request to run a transaction entered on a terminal is encoded as a *message* by a front-end processor and put on the queue for the application program that supports the transaction. We currently envision ownership of a transaction type by a single database processor. When a copy of the application is ready to run the transaction, it removes the message from its input queue, executes it, and puts a result on the output queue for the terminal as an atomic action.

The message queue manager is responsible for routing messages to the right destinations and for recovering messages that are lost due to failures such as database processor crashes. Also, when a database processor crashes, the transaction queues that it owns need to be moved to various surviving database processors.

Scheduling

With several equivalent processors available to run transactions, the scheduling component must decide how to split up the workload effectively. Due to buffer management considerations and the design of the concurrency control component, there is an advantage to executing "similar" transactions on the same system. The scheduler's role is to find similar sets of transactions to run together, while keeping the database processors more or less evenly loaded.

Comparison with partitioned systems

Growth aspects

The main advantage that a data sharing system enjoys over a partitioned approach is that the system structure does not change when database processors or disks are added or fail. In contrast, adding resources to a partitioned system may require careful analysis. For example, if a new processor is added, someone must decide what part of the database to take away from existing processors and give to the new processor. In general, this may mean moving data among all elements of the partitioned system and require much down time. A sophisticated partitioned system will work with any partitioning, but the performance will suffer unless care is exercised. Similar care is needed when extra disks are added.

With a data sharing system, the disks and database processors can be considered separately. If more computing power is needed, then additional database processors can be added without changing the arrangement of data on the disks. If more disk space or throughput is needed, extra disks can be added and the database can be rearranged on them, without modifying applications or determining which of the new disk drives should be owned by which processors.

Partitioned systems require careful analysis to split the database effectively plus physical movement of the data between disks. With a good split, most transactions can run completely on a single database processor. In addition, a good split roughly balances the total workload among the database processors. Transactions that are "badly behaved" with respect to the database split will require additional inter-processor communication to run. An effective data partitioning will invariably be a compromise that runs the current mix of transactions reasonably well. A new application may access a significantly different cross section of the database and require that the split be reexamined. Finally, the workload characteristics may vary during the day as the activities of the enterprise change. A partitioned system does not have the flexibility to adapt to changes on a hourly basis.

In contrast, a data sharing system splits up the workload among its set of processors. Changing the workload split simply requires that new work be scheduled on a different processor and can be done while the system is running. Since transactions run to completion on a single database processor, "badly behaved" applications that reference data scattered over the database can be executed efficiently. Bringing a new application online may cause the system to distribute the workload in a different way, but does not require data movement or reorganization. The flexibility in assigning work to database processors also allows the system to adapt to changes in the workload characteristics that occur during the day.

The symmetry of a data sharing system allows the enterprise to allocate some of the database processors to other tasks during lulls in transaction activity. A partitioned system cannot usually tolerate reallocation of its processors.

Recovery

Recovery and availability are simplified in a data sharing system. After a database processor failure, the structure of the system remains unchanged. Any surviving database processor can begin recovery immediately after a failure. When recovery is complete, the scheduler can redistribute the workload among the surviving systems and the complex can resume normal operation. Subsequent failures can be handled in a similar way. After each failure, the additional work given to the surviving processors is a fraction of the workload that the original processor was responsible for.

In contrast, a partitioned approach has less flexibility in its handling of processor crashes. Due to limited accessibility to the disks of a failed processor, only certain processors can assume responsibility for recovering from a failure and executing new work originally destined for the failed system. When a system fails, its work is transferred to one of the surviving systems. The system that takes over this work may now be overloaded, since it must process its own work as well as the new work it has been assigned. Subsequent failures may render parts of the database inaccessible.

Conclusions

Data sharing systems provide an alternative to partitioned approaches. Data sharing systems offer significant advantages in the areas of capacity growth and recovery. To realize these benefits, new approaches are needed in the areas of concurrency control, scheduling, and recovery.

References

[Bartlett 78] Bartlett, J. "A NonStop Operating System," *Eleventh Hawaii Conference on System Sciences*, (January 1978).

[Kim 84] Kim, W., "Highly Available Systems for Database Applications," *ACM Computing Surveys*, 16, 1 (March 1984).

[Kung 81] Kung, H. and Robinson, J., "On Optimistic Methods for Concurrency Control," *ACM Transactions on Database Systems*, 6, 2 (June 1981).

[McGee 77] McGee, W., "The Information Management System IMS/VS: Part I: General Structure and Operation," *IBM Systems Journal*, 16, 2 (1977).

[Shoens 85] Shoens, K., *et al*, "The Amoeba Project," *Proceedings 1985 Spring COMPCON Conference*, (1985).

[Strickland 82] Strickland, J., Uhrowczik, P., and Watts, V., "IMS/VS: An Evolving System," *IBM Systems Journal*, 21, 4 (1982).

Synchronizing Shared Memory in the SEQUOIA Fault-Tolerant Multiprocessor

Philip A. Bernstein *

Sequoia Systems
Boston Park West
Marlborough, MA 01752

1. Introduction

The Sequoia computer uses a tightly-coupled multiprocessor architecture, combined with a hardware approach to fault detection and isolation. The computer is unique in its ability to expand to very large configurations -- up to 48 MIPS of processing, 96 I/O channels, and 256 MB of main memory. The machine is designed to meet the reliability, modular expandability, and high performance requirements of transaction processing. This paper gives an overview of how the hardware architecture and operating system work together to provide these benefits. Other computer architectures for transaction processing are described in [1,2,5].

2. Hardware Overview

A Sequoia computer consists of processor elements (PE's), memory elements (ME's) and I/O elements (IOE's) connected by a system bus (see fig. 1).

Each PE contains dual 10MHz MC68010 microprocessors, operating in lock-step with comparators that test for identical operation on each clock cycle. Each PE also has 1024 128-byte blocks of cache memory and a memory management unit that maps 24-bit virtual addresses into 32-bit physical addresses.

The cache is non-write-through, meaning that updates written to cache by the microprocessor are not immediately written to main memory (ME's). Instead, the operating system (OS) must explicitly ask the PE to flush the contents of its dirty (i.e., updated) data blocks to ME memory. The OS may choose to flush the cache to refresh the main memory copy of data recently updated in cache, or to make room for new data when the cache overflows. Special-purpose PE hardware flushes all dirty cache blocks in one instruction.

* Author's current address: Prof. Philip A. Bernstein, Wang Institute of Graduate Studies, School of Information Technology, Tyng Road, Tyngsboro, MA 01879.

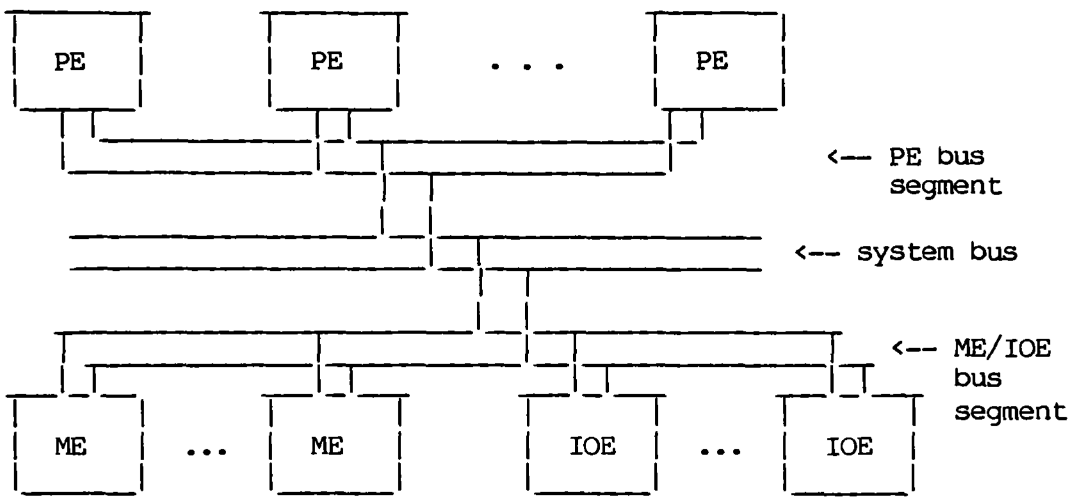


Figure 1. Sequoia Hardware Architecture

Each ME contains 2 MB of RAM and 128 test-and-set locks, which the OS uses to ensure mutually exclusive access to shared memory. The operation test-and-set(X, Y) performs the following sequence atomically: (1) if $X = 0$ (i.e., the lock is not set), then set the lock by storing into X the value Y ($Y > 0$); and (2) return the number stored in X before the test-and-set was executed.

Each IOE has two sections (see fig. 2): a bus adapter (BA), which connects to a memory bus segment; and a multibus adapter (MA), which connects to an IEEE-standard 796 bus (the Multibus*) functioning as the I/O bus. Each BA contains four 4096-byte buffers, and DMA logic for passing information between its associated MA and all of the ME's. Each MA contains dual self-checking 10 MHz MC68010's and 512 KB of RAM, used both for data buffering and program storage. Each Multibus supports up to 14 I/O controllers for tape, disk, and communications.

The bus consists of two 40-bit 10 MHz buses that operate independently, for an aggregate throughput of 80 MB/sec. The buses are made up of three types of segments (see fig. 1): processor bus segments that hold PE's; memory bus segments that hold ME's and IOE's; and system bus segments that connect processor and memory bus segments. Each processor segment holds up to 8 PE's, and connects to the system bus through a master interface (MI). Each memory segment also holds up to 8 elements, and connects to the system bus through a slave interface (SI). The MI arbitrates access to the buses. Both the MI's and SI's are repeaters that electrically isolate each bus segment. Up to 8 processor bus segments and 16 memory bus segments can be connected by

---

* Multibus is a trademark of Intel Corporation

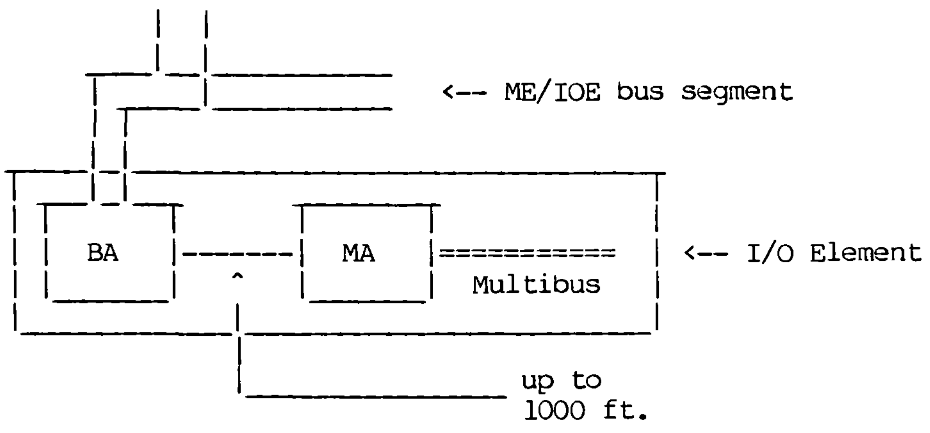


Figure 2. I/O Element

system bus segments in a single system. Thus, a system contains up to 64 PE's and 128 ME's and IOE's (combined total).

Each type of element and the bus is fully covered by fault-detection mechanisms: error-detecting codes, comparison of duplicated operations, and protocol monitoring. Upon detecting a fault, an element instantaneously disables all of its outputs to prevent the malfunction from corrupting other elements. The OS is notified of the fault on the next attempt to access the faulted element. At this point, the OS takes over by initiating the fault recovery activity, which is sketched in Section 4.

3. Operating System Architecture

Sequoia's operating system is compatible with the various UNIX* versions [3]. It is based on a proprietary kernel, which is designed to meet the many special-purpose needs of high performance transaction processing [4,6].

Sequoia's kernel provides the user with fault-tolerance at the level of fail-safe processes and files. As long as the system is configured with at least two of each element (PE, ME, IOE), each process is resilient to any single-point hardware failure. Some performance degradation can result from the failure. However, the larger the configuration, the smaller the marginal loss in computing power that results from the failure of any single element.

Given Sequoia's tightly-coupled architecture, there is only one copy of the kernel and its assoicated tables in ME memory, and that kernel copy is executed on demand by any PE. That is, when a process executing on a PE performs a system call, that system call invokes the kernel on that same PE. Since the kernel may be executing on more than

* UNIX is a trademark of AT&T.

one PE at a time, each PE must synchronize its access to shared kernel data structures.

Each area of memory that can be concurrently accessed by the OS on more than one PE is assigned a test-and-set lock. Accesses to that area are protected by the following protocol:

```
        Get test-and-set lock;
        Invalidate non-dirty cache;

                Critical section code that
                accesses the memory area;

        Flush dirty cache;
        Release test-and-set lock;
```

The lock ensures that only one PE accesses the memory area at a time. The cache invalidation ensures that old values still in cache from the last access to the memory area are not used; this forces the cache hardware to retrieve the most up-to-date values from ME memory. The cache flush ensures that the updated values of the memory area are available to the next PE that access it. This protocol is embedded in a low layer of the kernel that supports the abstraction of shared kernel memory.

There are two principal issues that any locking algorithm must attend to: deadlock and lock contention. Deadlocks are handled by totally ordering all resources on which locks can be obtained and requiring that those resources be locked according to that total order.

When a PE requests an unavailable test-and-set lock, it spins in a loop, periodically testing whether the lock has become available yet. Lock contention may cause PE's to lose a significant fraction of their computing cycles to this loop. To reduce lock contention, all lockable resources are randomly partitioned into smaller granularity units, with a separate lock for each partition. With this strategy, two PE's are less likely to contend for locks on a resource, because they are less likely to require access to the same partition. The partitioning occurs dynamically, in response to increased contention for locks. When the kernel detects high contention for a particular lock, it adds a new partition for the lock.

4. Kernel Support for Fault Recovery

The system may experience a hardware fault at any time. No matter when a fault occurs, it must be possible to recover the process that experienced the fault without losing its process state or losing or duplicating I/O operations.

Suppose a PE faults at a time when it is not flushing its cache. The main memory image of the process that it was executing at that time is the state of that process as of the PE's last cache flush, and therefore is consistent. So the kernel (running on another PE) need only identify the process that was assigned to the faulted PE and return it to the ready queue.

What if the PE was flushing its cache at the time it faulted? A cache flush is not atomic relative to PE faults. So a fault during a cache flush may leave an inconsistent memory image of a process, some memory containing the image of the process before the flush and some after it.

To avoid this problem, writable pages are shadowed in main memory. That is, each writable page is stored in two different ME's. Program pages and read-only data pages are not shadowed, because they are never written and therefore cannot be corrupted by a cache flush.

When a PE flushes its cache, it actually flushes twice. First, it flushes to the primary copy of its pages and, when that is complete, it flushes to the backup. If a PE faults during its primary cache flush, then the backup copy of the pages is consistent as of the previous cache flush. If it faults during its backup cache flush, then the primary copy is consistent as of the current cache flush. In either case, the inconsistent copy is refreshed by reading the consistent copy. After memory is successfully refreshed in this way, the process is returned to the ready queue and continues executing later on another processor.

All writable data pages are shadowed on two ME's. All code and read-only data pages in an ME are backed up on disk. So, if an ME fails, every page that was resident in that ME exists elsewhere in the system and can be recovered.

To initiate an I/O operation, a PE contructs a description of the operation in cache and then flushes that description to a queue in main memory. Suppose a PE fails during a flush that will initiate an I/O. After the failure, the memory state is recovered either to the state before the PE's last cache flush or to the new state. In the former case, the I/O is not issued, because the flush was undone, and the process state is rolled back to a state before it issued the I/O. In the latter case, the I/O is issued, because the flush is completed, and the process state is just past the point that it issued the I/O.

Disk failures are handled in a conventional way, using dual-ported mirrored devices on different IOE's.

5. User Shared Memory

The kernel supports a segmented address space for each process. A segment can be shared among processes and can be writable. Writable shared segments pose the same mutual exclusion problem to processes that shared OS tables pose to the kernel. To help processes synchronize access to shared segments, the kernel supports semaphores, implemented in shared kernel memory. To request or release a semaphore, the kernel sets and releases the test-and-set lock that protects access to the semaphore.

Before accessing a segment, a process requests the semaphore that protects access to the segment. After the access is complete, it releases the semaphore. Since requesting and releasing a semaphore forces the kernel to set and release locks, the appropriate cache flushing and invalidation happen "automatically". Requesting the semaphore entails setting a lock, which causes nondirty cache to be invalidated. Releasing the semaphore entails setting and releasing a lock, which causes a cache flush.

Using shared segments, users can implement other interprocess communication primitives, such as message passing. The user's implementation can be nearly as efficient as the kernel's would be, since the user has direct access to the shared memory containing the messages. This is especially useful in transaction processing, where no one form of message passing is satisfactory for all applications.

6. Final Remark

We believe that the cost of fault tolerance in Sequoia's architecture is relatively low. In the absense of failure, the normal mode of operation, most redundant modules contribute to overall system performance. All nonfailed PE's perform useful work, with no dedicated backups. The dual buses operate independently. And mirrored disks add to the bandwidth of disk reads. Among modules that are duplicated for fault tolerance, only the shadowed memory does not add to system capacity. Although there is cost associated with the electronics needed for fault detection, this function allows the system to automatically self-diagnose its errors, thereby cutting the cost of testing in manufacturing and in the field.

7. Acknowledgments

The architecture of the Sequoia computer was designed by Jack J. Stiffler, founder of Sequoia Systems. As in any product development of this magnitude, the design and implementation was the collective work of many engineers. On behalf of the Company, I thank them all for their technical contributions and hard work in making the architecture a reality.

8. References

1. Borg, A., J. Baumbach, S. Glazer, "A Message System Supporting Fault Tolerance," _Proc. of the Ninth Symp. on Operating System Principles_, 1983, pp. 90 - 99.

2. Bartlett, J. F. "A NonStop Kernel, _Proc. of the Eighth Symp. on Operating Systems Principles_, 1981, pp. 22 - 29.

3. Ritchie, D. M. and K. Thompson, "The UNIX Time-sharing System," _Bell System Technical Journal_, Vol. 57, No. 6, Part 2 (July 1978), pp. 1905 - 30.

4. Stonebraker, M., "Operating System Support for Database Management," CACM, Vol. 24, No. 7 (July 1981), pp. 412 - 418.

5. Stratus/32, VOS Reference Manual, Oct. 1982.

6. Weinberger, P., "Making UNIX Operating Systems Safe for Databases," The Bell System Technical Journal, Vol. 61, No. 9, Part 2 (Nov. 1982), pp. 2407 - 2423.

Multi-Disk Management Algorithms
(Extended Abstract)

Miron Livny

Computer Sciences Department
University of Wisconsin-Madison
1210 W. Dayton St.
Madison, WI 53706

Setrag Khoshafian
Haran Boral

MCC
9430 Research Blvd
Austin, TX 78759

1. Introduction

With current technological trends, fast transaction processing requires efficient solutions to the I/O bottleneck. Presently, DBMS's and transaction processing systems avoid the I/O bottleneck by utilizing only a small portion of the available I/O bandwidth. We believe there are five alternative approaches to solving this problem, some of which are orthogonal:

(1) Main memory resident databases

(2) Alternative, "performance-friendly" mass storage devices

(3) Buffering techniques

(4) Clustering techniques

(5) Utilization of parallelism in a multiple disk I/O system

We are currently exploring these solution types, as well as analyzing their orthogonality and integration. In this paper we study data organizations and scheduling policies for a multiple disk I/O system. We assume the database resides on several disks and explore different methods by which the DBMS can take advantage of the disk multiplicity. We examine two data layout strategies: (i) clustered, in which each relation is stored on contiguous cylinders on a single disk, and (ii) declustered, in which each relation is laid out across all the disks in "bands" (or logical cylinders). Three scheduling algorithms are compared: (i) FIFO, (ii) Shortest Seek First (SSF), and (iii) our own algorithm designed specifically for the declustered data organization.

Our results, obtained form a simulation model, show that except for a highly utilized disk system (greater than 90% utilization) the declustered approach outperforms the clustered approach using a variety of metrics. In spite of the increase in access time due to the declustered data organization, the net impact of this layout is a decrease in the expected queuing time of a request. The concurrent execution of a multi-block request, and the uniformity of service demand imposed on each device overcome delays introduced by the additional seeks. Declustering also yielded a much smaller standard deviation for request response times, and almost a factor of two shorter response time delays for the first block of a multi-block request.

In comparing our own algorithm to FIFO we found that it performed better than FIFO (clustered or declustered) for the shorter request sizes (especially the one track requests). However, the improvements were not as significant as we had hoped. Finally, the performance of SSF with a declustered storage organization, suggested to us that we need to pursue some extensions to pure SSF, in order to make it realistic for general workloads (i.e. avoid starvation).

The remainder of the paper is organized as follows. Multi-disk management is described in Section 2. The disk scheduling algorithms are described in Section 3. In Section 4 we discuss the simulation model and the workload. In Section 5 we describe the experiments and interpret the results. A summary of the paper and future work are outlined in Section 6.

2. Multi-Disk Management

There are two main orthogonal issues for multi-disk management:

(1) storage schemes, and

(2) disk scheduling algorithms

For (1) we consider two types of storage structures: (a) clustered, and (b) declustered. To understand the difference between these two strategies, assume we have a direct storage representation of the relational model, and each relation is clustered on one or more attributes. With a clustered scheme, all the secondary storage blocks of each relation are stored contiguously on as few disk modules as possible.

For the declustered case, the data blocks of the relation are distributed in a round robin fashion across the multiple disks (i.e. block B_i is stored in disk module $D_{i \bmod k}$ where k is the number of disk drives). With the clustered scheme a query accessing b contiguous blocks will retrieve the blocks from adjacent extents of a disk drive. With the declustered scheme the b adjacent blocks will be retrieved with maximum declustered parallelism (i.e. from all k disk drives if $b \geq k$ and from b disk drives otherwise). Contiguous multiblock requests correspond to range queries on clustered attributes, exact match queries on category attributes in statistical databases, and partial match queries on multi-attribute clustered files (such as k-d trees). Another type of request we have analyzed is multi-block non-contiguous requests pertaining to the same relation. These types of requests arise in processing exact match or range queries for a relation through an inverted attribute and in semijoin processing.

For multi-block requests, declustering has a drawback -- it increases seek time. However, by spreading data out over several disks we can take advantage of the resource multiplicity of the multi-disk environment.

For (2) it has been argued (see [Kim85] for example) that since in conventional systems the observed queue lengths are typically short, multi-disk scheduling is unimportant. This contention is true but only because disks are purposely underutilized in most systems (typically less than 50%). That is, by increasing the system cost and underutilizing the available disk resources the I/O bottleneck is avoided. However, with current technology trends (processor speeds increasing at a much higher rate than either increases in disk bandwidth or decreases in disk access time) we see disk scheduling as becoming increasingly important.

3. Disk Scheduling Algorithms

We analyzed the following disk scheduling algorithms:

(1) FIFO - first in first out

(2) SSF - shortest seek first

(3) FFBF - fastest fitting block first

The simplest scheduling algorithm is FIFO. SSF without any modifications is not a reasonable scheduling algorithm, since in real workloads uniformity cannot be guaranteed, and hence some requests will starve. However, the performance of the SSF scheduling strategy provided useful insights while comparing the clustering and declustering storage schemes. In forthcoming studies we will be investigating fixed batched SSF strategies, where the request queues are processed in fixed sized batches and other SSF variants.

Our FFBF scheduling algorithm works as follows: if a multi-block request consisting of b blocks on b different disk modules is delivered to the I/O subsystem, the current status of the queues for each of the disk modules is checked, and an attempt is made to place the parallel block request in the disk queues in such a way that the average response time of the requests approximates the response time for the retrieval of the block whose disk has the longest queue. This strategy implies there will be "holes" in the request queues for some of the disk modules. We shall identify these as NULL requests. The last slot of each queue will also be NULL. A NULL request in the beginning of a request queue is disregarded. Figure 1 contains a brief sketch of the algorithm:

4. The Multi-disk Model and Workload Characterization

Our multi-disk system is modeled by m single-queue, single-server queuing systems and a dispatcher. There are three classes of parameters: (i) those that describe the database, (ii) those that describe the hardware, and (iii) those that describe the workload.

```
Request($B_{i_1}$, · · · $B_{i_p}$) /* the p block request */
    For j in [$i_1$, · · · ,$i_p$] /* j spans the block set indices */
        FIRST[j] := Position of First NULL in Queue j /* for each queue set FIRST to
            the "position" (i.e. the index) of the first NULL request of the queue */
    LI := Max(FIRST) /* LI is the position of the NULL slot furthest away from the head
        of the queue among the queues [$i_1$, · · · $i_p$] */
    For j in [$i_1$, · · · ,$i_p$] /* this loop will place the request in the appropriate slots
        according to the FFBF strategy */
        i := Position of Last NULL in Queue j $\leq$ LI /* i is set to the index of the last
            NULL request in the current queue, such that the NULL request position
            is less or equal to LI. Note that for the queue which yielded the value LI,
            i is set to LI. */
        Put request $B_j$ in slot i
```

Figure 1
The FFBF Algorithm

The database is characterized by the number of relations in it (NR) and the size of each individual relation (the size of relation R_i is SR_i).

The hardware is characterized by the number of disks (m). There are several parameters used to describe the workload. The *offered load* (OL) is the parameter which determines the number of tracks accessed per second. Requests arrive according to a Poisson process. There are K different classes of requests and the probability that a request is of class i is p_i. Each class has a different request size distribution. The number of tracks a request of class i accesses is uniformly distributed on the interval $[l_i,h_i]$. The dispatcher disassembles multi-track requests into a set of single track requests and routes each of them to the disk on which the requested track resides. Consecutive requests to contiguous tracks are chained at the server. We assume that the unit of transfer between main memory and disk is 1 track, that there is no memory access contention, and that buffer space is unlimited. The dispatcher has complete information on both the static layout of data and the current load of each disk drive.

A single request may access (i) a single track, (ii) a sequence of contiguous tracks of a relation, or (iii) a random subset of the relation tracks. We assume that C percent of the multi-track requests are for contiguous tracks.

When the data layout is clustered each relation is stored on a contiguous set of cylinders. In the case of a declustered layout it is assumed that the relation occupies the same set of cylinders on all disks. The random data organization is modeled by a uniform distribution of tracks over the m disks and the cylinders within a disk.

The disk scheduler is located at each server. Depending on the scheduling policy, a waiting request is initiated upon the completion of a previous data transfer. Global information, supplied by the dispatcher, regarding the state of other queues as well as local information is employed by the scheduler in order to select the request to be served next. Access delay for a given request is derived according to the current position of the disk head, and the location of the requested track. We assume "immediate reads" and thus add a uniformly distributed latency delay with a mean equal to a half sector transfer time to the access delay. When reading contiguous tracks from the same cylinder a "head correction" delay is assumed for each head switch within the same cylinder.

The main advantage of using this approach is that we can examine the effect of varying the frequency of each class type for a fixed offered load or we can keep the frequencies constant and examine the effect of increasing the load on the system by varying the value of OL.

A summary of the parameters and their meaning is given in Table 1 below.

Parameter Class	Parameter Name	Explanation
Database	NR	Number of relations
	SR_i	Size of relation R_i
Hardware	m	Number of disks
Workload	OL	Offered Load
	K	Number of request classes
	p_i	Probability of a request being from class i
	$[l_i,h_i]$	Interval of request size
	C	Percent of contiguous-track multi-track requests

Table 1
Summary of Parameters

5. Simulation study

In order to evaluate the different data organizations and scheduling strategies a discrete event simulator of the multi-disk model has been constructed. The simulator is written in DISS [Melm84] and is composed of a source, mass storage, and sink modules. We have used the simulator to predict the performance of different combinations of data organization and scheduling disciplines under a variety of workloads. As described in Section 4, the load imposed on the simulated system by its users is controlled via the offered load, OL, parameter which determines the number of tracks accessed per second. The request arrival rate of the system is derived from the OL parameter and the attributes of the different classes (p_i, l_i, h_i).

For our simulation studies we have assumed $m = 8$ identical RA81 [Dec82] disk modules, where each module possesses its own buffer. In other words there is no bus contention for RAM accesses. Furthermore we have assumed unlimited buffer space, and negligible processing overheads. Although both of these assumptions are unreasonable, we wanted to understand the performance gains of multi-disk I/O subsystems in a completely I/O bound environment.

Figures 2-6 summarize the results of a set of simulation runs with three classes of requests ($K = 3$). The relative frequencies, and the request size distributions for each class are given in Table 2.

Class	No. Of Tracks $l_i - h_i$	Frequency p_i
Class 1	1	70%
Class 2	2 - 5	15%
Class 3	6 - 8	15%

Table 2
Workload Parameter Values Used

We assumed that the disks are fully occupied with relation and that each spans 112 tracks. That is, SR_i has the same value for all i. So, in this initial study the parameter NR is determined by the number of tracks in the disk system and the single relation size. The relation referenced by each request is picked randomly. Further, half of the multi-track requests were assumed to access a contiguous set of tracks ($C = 50\%$). The length of each simulation was 1000 seconds.

In Figure 2 we plot the performance by transaction class for the clustered and declustered organizations for the SSF scheduling policy. The figure shows the expected response time of a request as a function of OL. Each curve in the figure represents a request class. The declustered and clustered layouts are represented by the solid and the dashed lines respectively. Figure 2 provides a clear display of the beneficial impact declustering has on the performance of a multi-disk system. Since the SSF strategy minimizes the access overhead due to declustering, the concurrent scheduling and uniformity of service demands of the declustered system cause a significant reduction in response time. For high values of OL the response time of a multi-track request is longer in the declustered case than in the clustered case. However, declustering leads to a reduction of 30% in the expected response time of single track requests. Declustering has the desired property of giving short service demands preferable service.

The performance of the multi-disk system, for both the clustered and the declustered layout, under the three scheduling algorithms is presented in Figures 3, 4, and 5. Each figure shows the expected response time of a different class of requests for the three scheduling disciplines and the two layouts. For low OL values all three disciplines have a shorter response time in the declustered case then in the clustered layout. However, when the OL is larger than 150 tracks per second the increase in disk utilization due to access time overheads dominates the FIFO and FFBF performance and leads to an exponential increase in response time. (This is the reason for the smaller range of OL values in Figures 3-6.) As anticipated, the FFBF discipline improves the expected response time of single tracks without penalizing multi-track requests. As the OL increases the difference between the performance of the FIFO and FFBF increases. From the point of

Figure 2 : SSF - All Cases

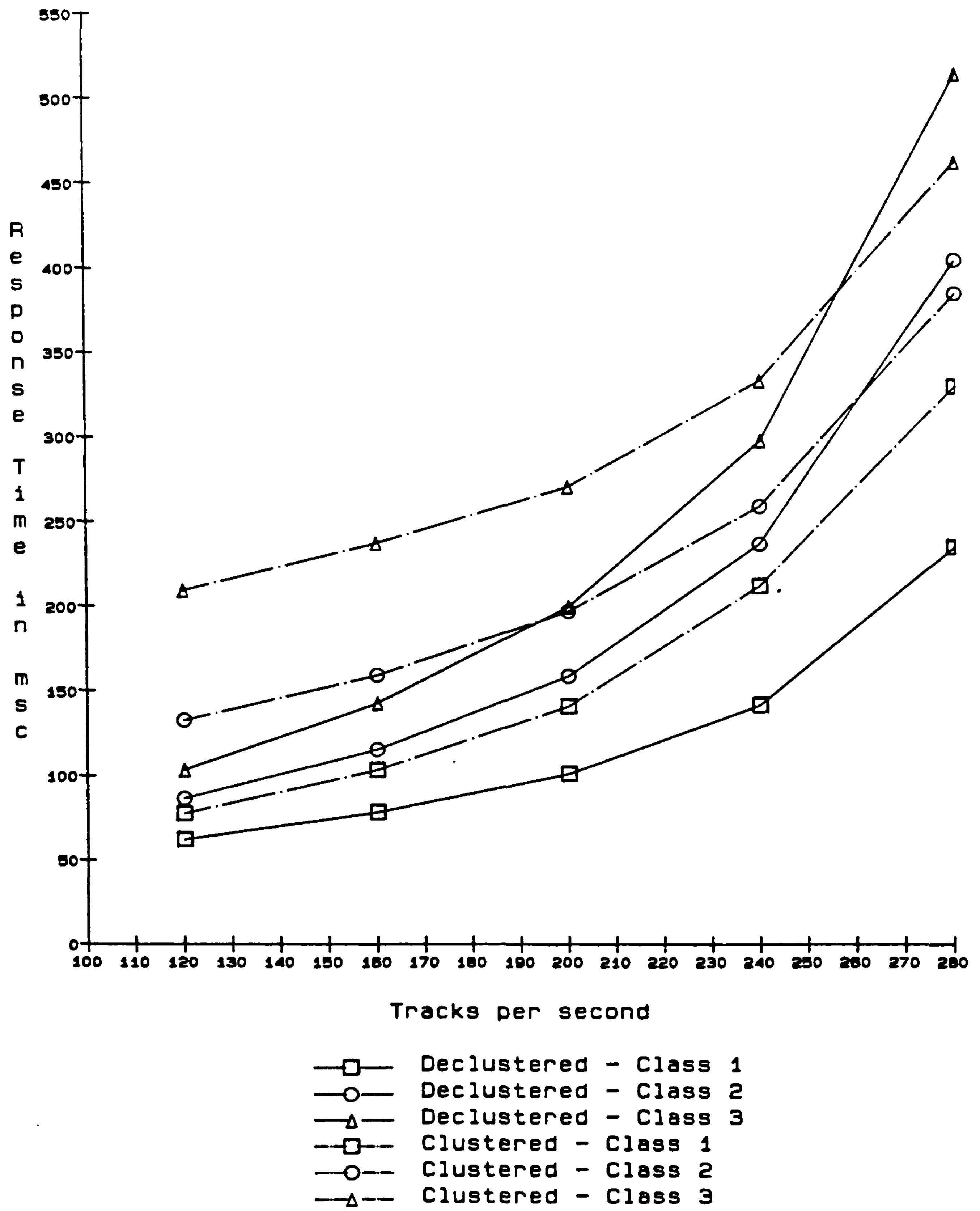


Figure 3 : Class 1 - SSF, FIFO, and FFBF

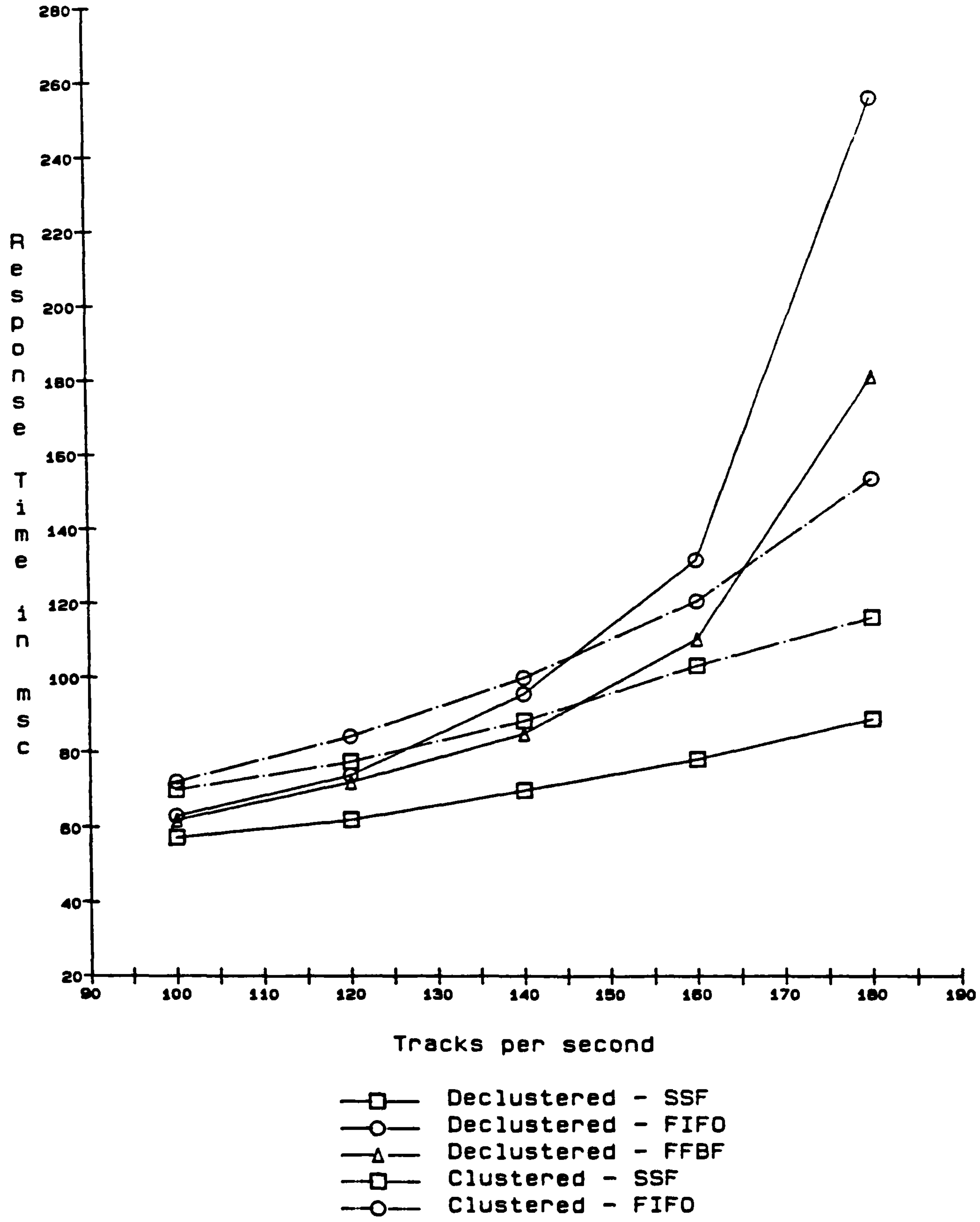


Figure 4 : Class 2 - SSF, FIFO, and FFBF

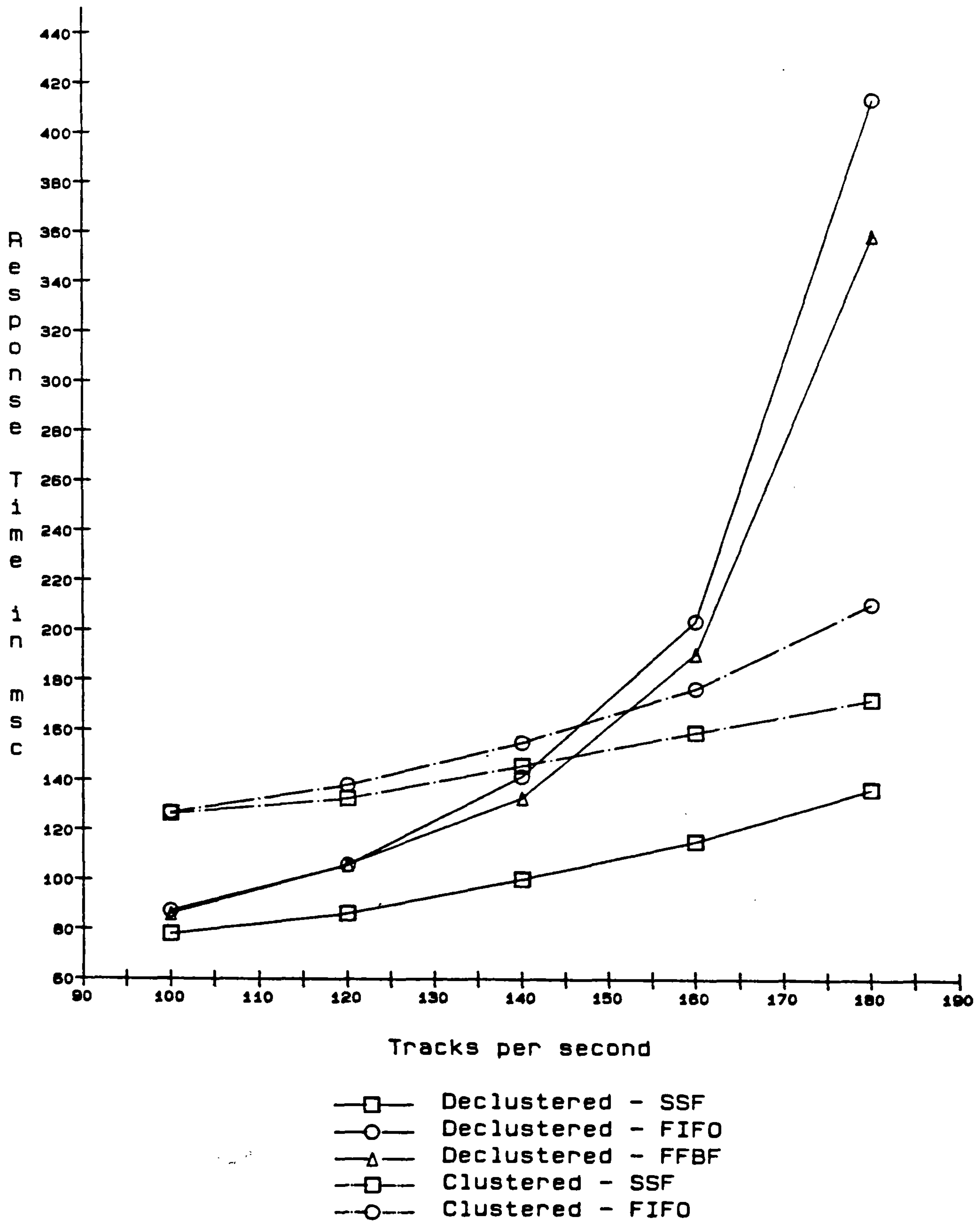
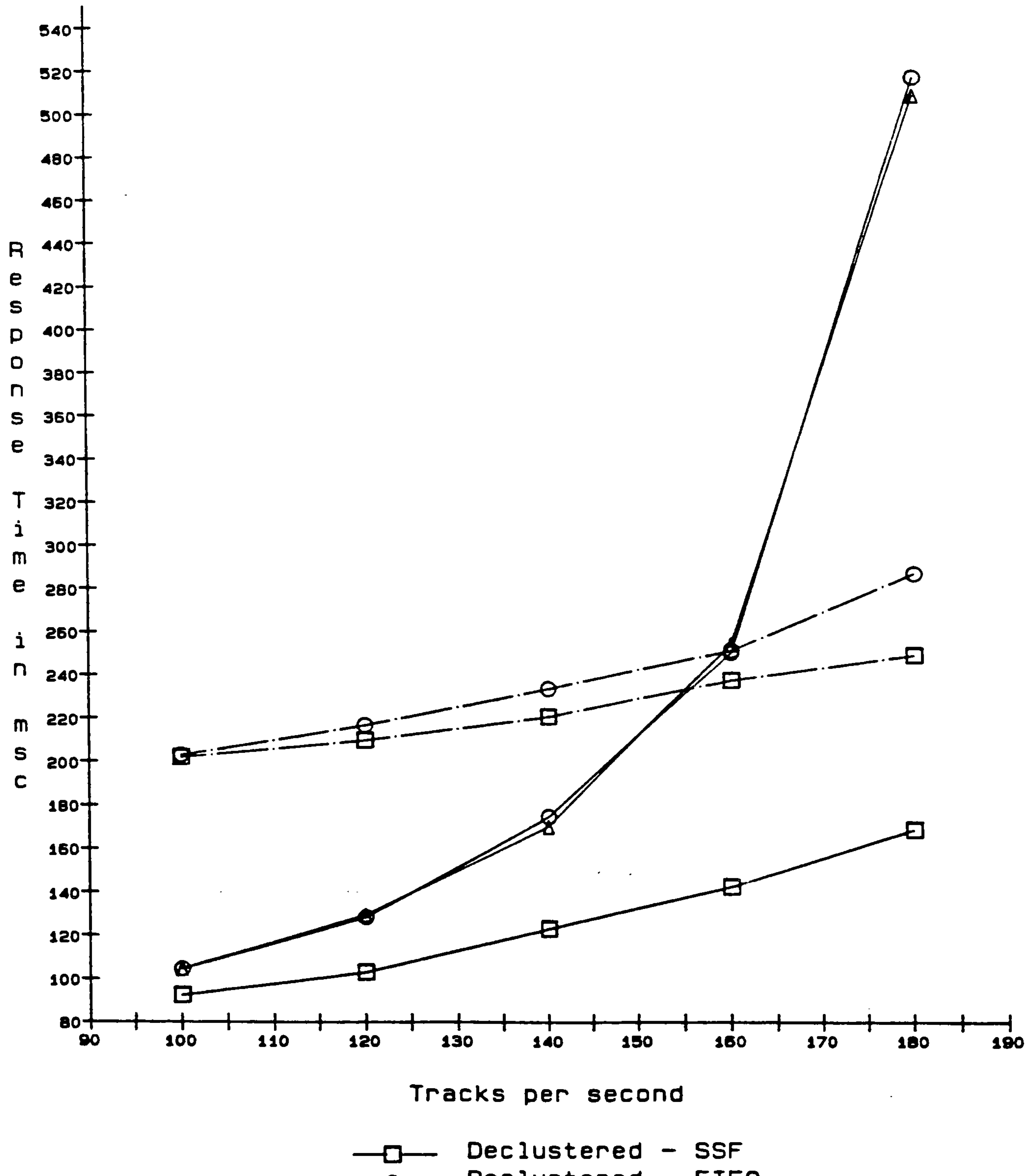
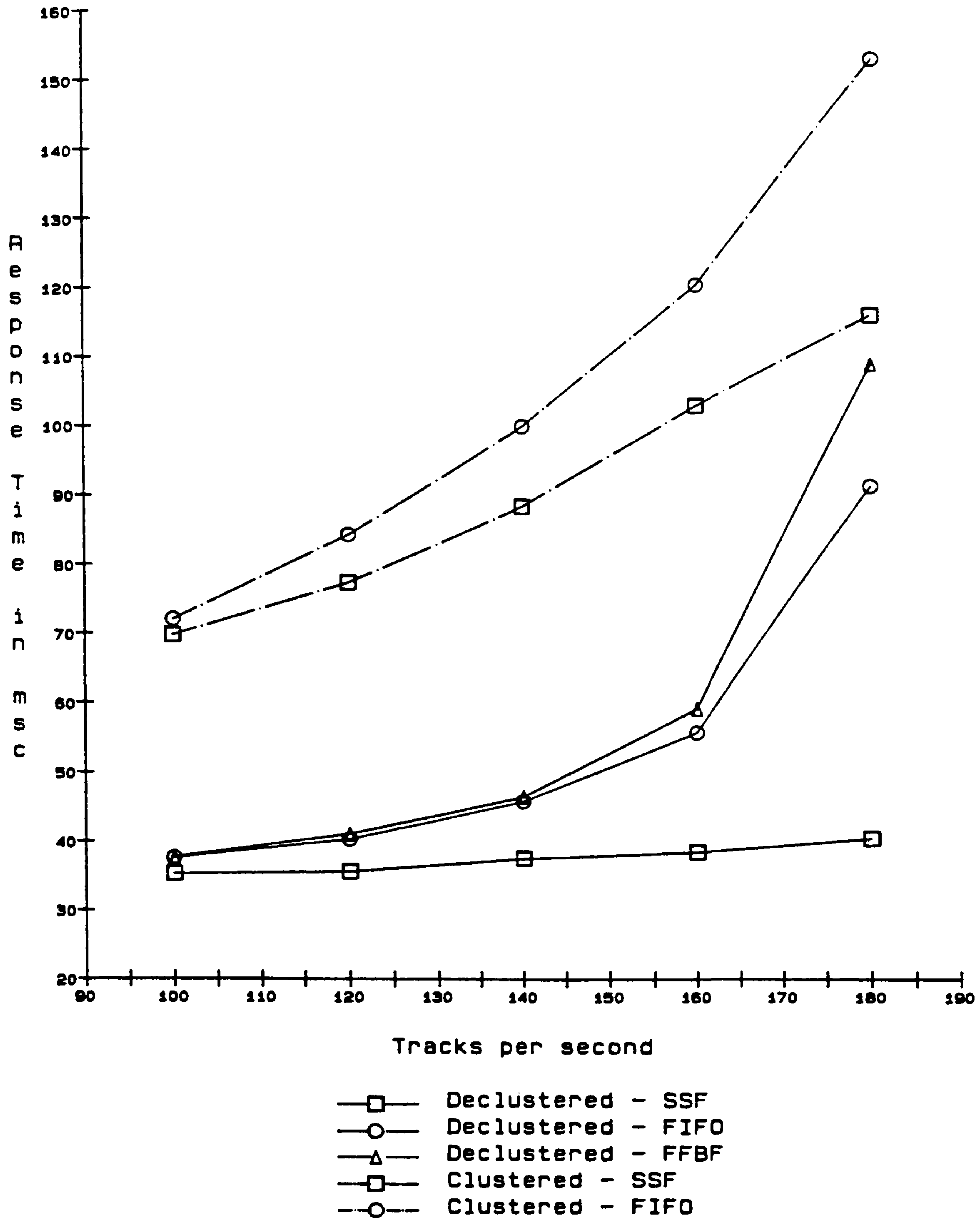


Figure 5 : Class 3 - SSF, FIFO, and FFBF



—□— Declustered - SSF
—○— Declustered - FIFO
—△— Declustered - FFBF
—□-- Clustered - SSF
—○-- Clustered - FIFO

Figure 6 : First Track Response Times - Class 3



view of single track requests FFBF is a winner as long as the OL is under 170 tracks/sec. In cases where a pure SSF strategy cannot be implemented, the underlying principle of the FFBF discipline should be incorporated in order to improve the response time of short requests.

The expected response times for the first track of a class 3 request are given in Figure 6. For all disciplines, the response time of the first page in the declustered case is shorter than in the clustered case. In most cases the difference between the two layouts is about 50%. Observe that the response time is almost independent of the OL for the declustered case if SSF is used. For high OL FFBF causes an increase in the expected response time of first pages due to the preferable treatment it gives to small requests. Note that it is often possible to process tracks as they arrive independently of the order in which the requests were submitted. Thus, the combination of a storage scheme and scheduling algorithm which delivers the first track in a multi-track request "very quickly" is most desirable.

6. Conclusions

The objective of our work is to study storage organizations and disk scheduling for multiple disk systems. Our preliminary results are encouraging. We have shown that despite the disadvantage of longer access time, response time is actually reduced. There are three important benefits in the declustered organization: (i) concurrent scheduling, (ii) reduction in the deviation of service demands, and (iii) faster response time for "first track" in a multi-track request.

The expected time of a request is an increasing function of the coefficient of variation of of the service demands. Thus a decrease in the deviation of the demands reduces waiting time.

It is reasonable to expect that a system which issues a single request for several tracks will be able to process the tracks in any order. Out of order processing can be used in any set oriented operation. For example, in simple selections using inverted files and in processing aggregate functions. As shown by the curves in Figure 6 the response time for the "first track" was significantly lower in the declustered storage scheme than in the clustered one (even for "heavy" load on the system).

We have also seen that the principle of providing faster service to requests that place small demands on the system can be applied to the declustered storage scheme (as shown in the curves for the FFBF disk scheduling algorithm).

We know of two related papers in the literature. Kim [Kim85] studied the effects of declustering by simulating a parallel readout drive using multiple disks. Salem and Garcia-Molina [Sale84] looked at almost the same problem. The main difference is in the unit of layout -- Kim lays the data out on the multiple disks by bytes whereas Salem and Garcia-Molina do it by sectors. Both papers show an improvement in response time for large block transfers. Using a byte parallel data layout scheme (as in [Kim85]) to obtain the parallelism entails keeping the disks synchronized. Sectors seem more appropriate. However, it has been argued quite convincingly in the literature (see [Smit84] for an example) that track transfers make optimal use of the disks. Clearly, the majority of applications will exhibit a variety of access patterns. Thus, optimizing the storage system for a particular type of access (large amounts of data in both papers) does not appear to be wise. As we have shown, declustering is a viable approach even when there are small amounts of data to be accessed.

Our conclusions are rather tentative at the moment. Further experimentation with storage schemes and scheduling algorithms is required. We already ran a variety of tests for the random storage organization and will shortly be looking at a batched version of the SSF scheduling algorithm. We will also be looking at the effect of varying the types of requests for a fixed OL value. More long term work involves a more realistic model involving the CPU.

7. References

[Dec82] "RA81 Disk Drive User Guide," Digital Equipment Corporation, 1982.

[Kim85] Kim M., "Parallel Operation of Magnetic Disk Storage Devices: Synchronized Disk Interleaving," *Proceedings of the Fourth International Workshop on Database Machines,* Springer Verlag, March 1985.

[Melm84] Melman M. and M. Livny, "The DISS Methodology of Distributed Systems Simulation," *Simulation,* April 1984.

[Sale84] Salem K. and H. Garcia-Molina, "Disk Striping," Department of Electrical Engineering and Computer Science, Princeton University, 1984.

[Smit84] Smith A.J., "Disk Cache - Miss Ratio Analysis and Design Considerations," Computer Science Division, Department of Electrical Engineering and Computer Sciences, University of California-Berkeley, 1984.

INFOPLEX:
RESEARCH IN A HIGH-PERFORMANCE DATABASE COMPUTER

Stuart E. Madnick
Massachusetts Institute of Technology

Meichun Hsu
Harvard University

1. Summary

INFOPLEX is an on-going research effort pursuing technologies for large-scale information management for the 1990's. The goal is to achieve orders of magnitude improvement in speed performance, availability and cost-effectiveness for computers used in information management through fundamental architectural changes. This document provides an introduction to the approaches of the INFOPLEX research project, as well as a summary of the recent research results.

2. Motivation

Due to their enormous storage capacity and processing speed, computers have become a dominating tool for managing information in an information society. The notion of *information utility* was proposed in [HM77, Madnick75b, Madnick77] which described a community in which personal computers and large scale computers are connected to a complex of *information nodes* that provides information services to the community (Figure 1). This view of the future is becoming increasingly plausible, particularly in light of the revolution of the computer industry in the recent past.

To provide information utility, large shared databases are inevitable for a variety of economic and technical reasons [Madnick79]. A computer serving as an information node must satisfy the requirements of high performance, high capacity and high reliability. In [Madnick79] it was envisioned that information nodes in 1990's would be required to have the capability of processing tens or even hundreds of thousands of requests per second (versus around a thousand requests per second on the largest computers today [Scrutchin85]), handling in excess of one hundred billion bytes of on-line structured data, and having the appearance of being operational continuously round the clock.

It has been argued that it would become increasingly costly to improve the speed of the conventional single-processor computer system. One avenue, therefore, to realize the information processing requirements of the 1990's is to seek for changes in computer architecture. Research work in this direction has been categorized as adopting one of the following four approaches [HM77]: (1) new instructions through microprogramming, (2) intelligent controllers, (3) dedicated minicomputers for database operations ("backend computers"), and (4) specialized database computers. In the recent past, commercial database machines, such as Britton Lee's IDM-500 and Teradata, have begun to appear.

3. The INFOPLEX Approach

INFOPLEX is a computer system with special hardware and software architecture dedicated to large-scale information management. The hardware architecture being pursued is a structured multi-processor complex, comprising of up to thousands of micro-processors. The software architecture of the system emphasizes functional decomposition and intelligent synchronization and coordination algorithms to achieve the highest degree of parallelism and robustness.

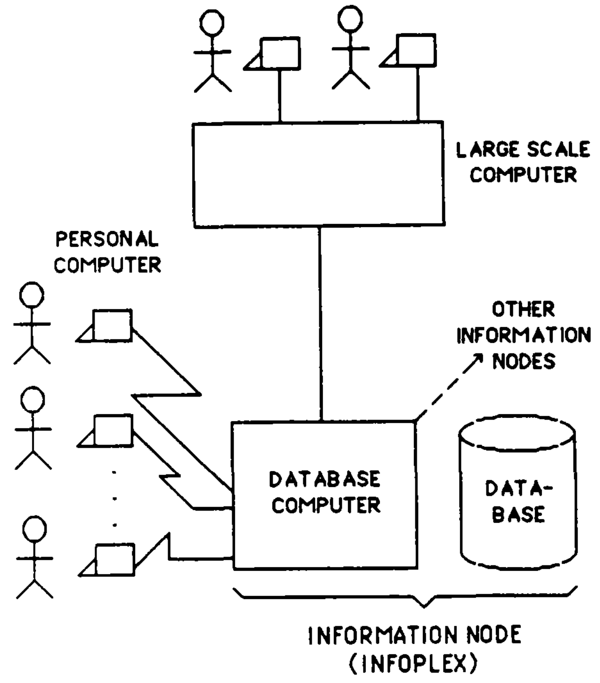


Figure 1.
A Future Trend of Computing: The Information Utility

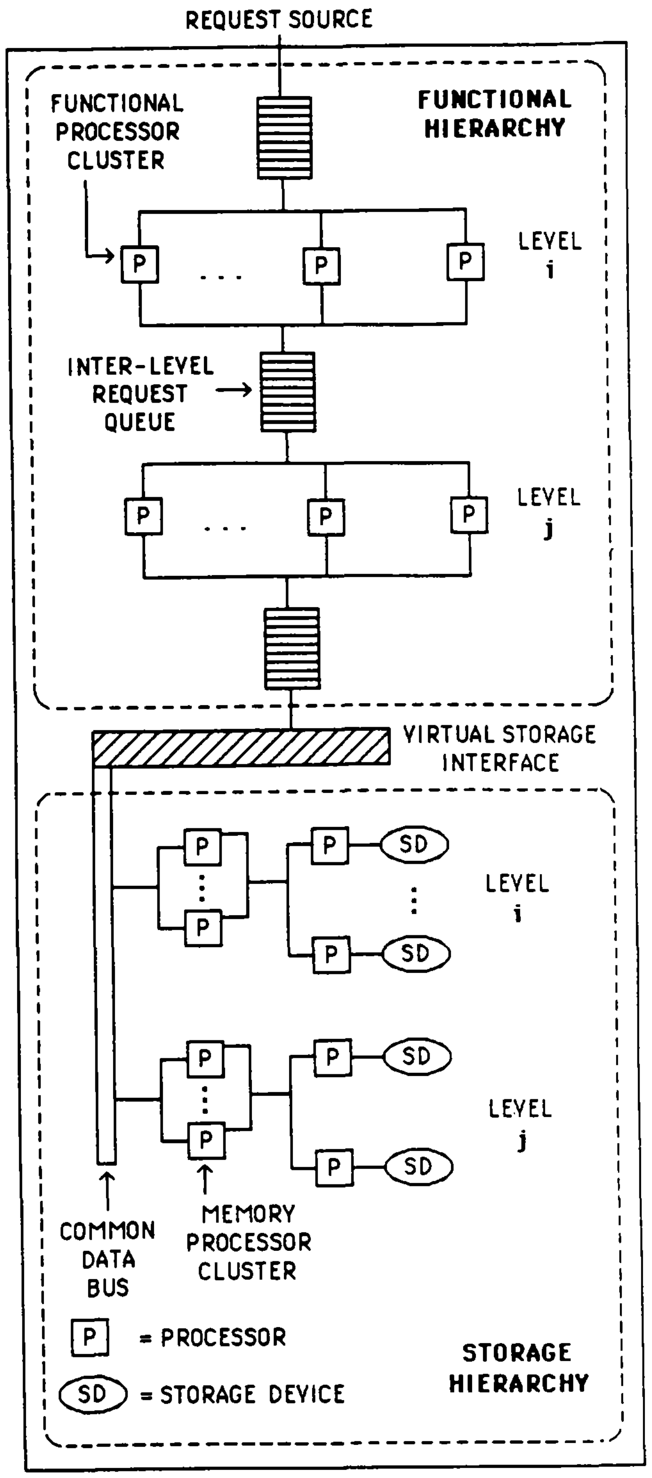


Figure 2.
INFOPLEX Database Computer Conceptual Organization

3.1. The Architecture of INFOPLEX

The conceptual organization of the INFOPLEX architecture is shown in Figure 2. The architecture design goal of INFOPLEX is to offer large processing capacity economically, reliably and modularly. The trend in hardware technology motivates the use of a large number of microprocessors to take advantage of the cost-effectiveness of the general-purpose microprocessors. In INFOPLEX, these processors are connected via a two-level *bus hierarchy*. The bus architecture is chosen for its simplicity. However, to alleviate the limitation on the number of processors that a single bus can handle, multiple single-bus processor clusters are connected together via a second-level bus, the *global bus*. With such a bus hierarchy and a special high-throughput bus protocol, this architecture is expected to be able to connect a large number of processors together to achieve the required processing capacity.

Central to the coordination of INFOPLEX is the notion of *distributed control*. To attain high performance and high availability, one of the principles followed in the design of INFOPLEX is to have activities of the system coordinated through *distributed control algorithms*. Distributed control algorithms allow components of the system to perform relatively independently of each other by executing predefined protocols without relying on a central coordinator. The use of a central coordinator is often the source of performance and reliability bottleneck problems in a system. Distributed control algorithms are needed to eliminate such a bottleneck. This principle for design is manifested in both of the two major components in INFOPLEX: the *functional hierarchy* and the *storage hierarchy*.

INFOPLEX provides a very large virtual storage that exploits reference locality, is dynamically managed, and does away with centralized memory mangement control. The storage devices are organized into a *storage hierarchy*. The higher levels of the storage hierarchy utilizes fast but more expensive storage devices, while the lower levels slower but less expensive devices. The objective is to employ intelligent memory management algorithms for migrating data between levels of the storage hierarchy, achieving the goal that a very large percentage of the data referenced can be found in the higher levels of the storage hierarchy. To perform memory management functions such as memory map, storage allocation and data migration reliably and in a distributed manner, each storage level is also controlled by multiple microprocessors that implement intelligent memory management functions.

The INFOPLEX storage hierarchy is specifically designed to be able to handle any type of storage devices. Thus unlike some other database computer designs which may be specialized to a particular type of storage devices, INFOPLEX can adapt to the changing application needs as well as take advantage of new technological advances in storage devices.

The functional processor clusters are collected into a *functional hierarchy*, each level of the functional hierarchy corresponds to a single-bus processor cluster. The storage hierarchy therefore appears to the functional hierarchy as a very large and intelligent virtual storage. The functional hierarchy therefore dedicates its effort to performing information management tasks, such as message processing, security checking, query decomposition, and internal organization of the data in the database.

The first step towards achieving distributed control in the functional hierarchy is *functional decomposition*. The complex information management function is broken into small tasks. These tasks are arranged in a pipeline manner where each task, representing a stage in the pipeline, may be performed concurrently with other stages. Each stage is assigned to a cluster of processors which comprises a *level* of the functional hierarchy. Within a level, processors perform just one task, with multiple instances of the task being processed in parallel by all processors in that level, thereby gaining further concurrency. Since processors within the same level perform the same set of tasks, intra-level communication/synchronization is more economically achieved through shared memory, while high-throughput inter-level buses are provided for communication among tasks.

In summary, the INFOPLEX architecture is motivated by the need to provide for large processing and storage capacity reliably and economically. The consequence of adopting a structured multiprocessor complex as the basic architecture is the need for these processing and storage components to be organized intelligently and coordinated through distributed algorithms to eliminate potential performance bottleneck.

4. INFOPLEX Recent Research Results

4.1. Research Objectives

Within the above architectural framework, research in INFOPLEX is conducted with the following research objectives:

(1) Discover and define efficient distributed control algorithms and their formal properties;

(2) Study and verify the nature of locality of references in databases for measuring effectiveness of storage hierarchy;

(3) Identify and construct relevant performance evaluation methodologies and apply them to predict INFOPLEX performance and to uncover potential bottlenecks; and

(4) Experiment through software and hardware test vehicles.

4.2. Distributed Algorithms

We describe two particular research efforts aiming at discovering algorithms that are suitable for controlling activities in INFOPLEX: the first effort studies algorithms for controlling migration of data in the storage hierarchy; the second aims at increasing parallelism in database concurrency control and reducing overhead.

In [Madnick75a and LM79], algorithms for data migration in a generalized storage hierarchy are identified. Specifically, data migration algorithms can be classified along two dimensions, both concerning the problem of the maintenance of the LRU (Least Recently Used) stack: the first addresses the treatment of references to the virtual storage from functional processors, the second addresses the treatment of references to levels of the storage hierarchy due to overflow placement.

Along the first dimension, two alternatives are possible: global LRU, which considers every reference to the virtual storage from the functional processors a reference to all levels of the storage hierarchy as far as the maintenance of the LRU stacks at each level is concerned; and local LRU, which only updates the LRU stack at levels of the storage hierarchy that are needed to satisfy the virtual storage reference from the functional processors. Along the second dimension, there are also two alternatives: static overflow placement (SOP) and dynamic overflow placement (DOP). Under SOP, overflow of a page from a higher level i of the storage hierarchy (i.e., closer to the functional hierarchy) to a lower level j of the storage hierarchy is not considered a reference to that page at level j, unless the overflown page is not resident at level j at the time of overflow. Under DOP, however, the overflown page is always considered as being referenced at the lower level, causing an update to the LRU stack at level j.

Two desirable properties significant to the data migration algorithms are also identified: multi-level inclusion (MLI) and multi-level overflow inclusion (MLOI). A storage hierarchy satisfies the MLI property if every page resident at a level i is also resident at the next lower level $i+1$. On the other hand, MLOI is satisfied if every page *overflown* from level i is also found to be resident at level $i+1$ at the time of overflow. These two properties enable the overflow handling algorithms to be simplified considerably and have important implications on data availability. It is proven in [LM79] that only global LRU algorithms can achieve MLI and MLOI. In addition, depending on whether dynamic or static overflow placement algorithm is used, in order for these properties to hold, there is a specific constraint on the size of the lower level of the storage hierarchy in comparison with that of the higher level. The precise nature of the size requirements is also shown in [LM79].

Using the formal results described above, the design of the INFOPLEX data storage hierarchy is described in [LM79a, LM79b, LM79c, Abraham79, and Madnick80].

In [HM83], a database concurrency control algorithm is identified which aims at reducing synchronization needs among data partitions. In a large database, the data can often be found to be organized in an information hierarchy, with some transactions updating raw data, some transforming the raw data and updating the partition that contains the derived data, and some transforming the first-level derived data and updating the second-level derived data. When the database is partitioned accordingly and distributed to multiple processors, it is desirable that inter-partition synchronization be minimized.

The hierarchical concurrency control algorithm described in [HM83] takes advantage of such a hierarchical structure of the database and leads to protocols which allow the transactions updating one partition to proceed without interfering with transactions updating another partition, thereby minimizing inter-partition synchronization. The protocols described and proven in [HM83] were further extended in [Hsu83] to allow certain degree of cyclic accesses in hierarchically partitioned database.

The above hierarchical algorithms are timestamp based concurrency control algorithms. However, the underlying principles of the algorithm can be abstracted to apply to two-phase locking based algorithms. In [HC85], the adaptation of the hierarchical timestamp algorithm to partitioned two-phase locking is described. This generalization broadens the applicability of the theoretical results.

4.3. Database Locality

The INFOPLEX data storage hierarchy - as well as many file servers, database machines, and DBMS packages - employ dynamic buffering techniques that rely upon pragmatically plausible, but often unproven, database locality for success. The study of database locality provides a theoretical framework for measuring load placed on a memory system by a sequence of requests for access to data. A consistent measure of load at the logical, internal and storage stages of database processing was needed to facilitate the study and comparison of alternative DBMS designs and database structures.

In [Madnick73], notions of temporal and spatial localities are defined. Temporal locality describes the clustering of references along the time dimension: if a data element is referenced at time t, then it is likely to be referenced again shortly after t. Spatial locality, on the other hand, refers to the clustering of references along the space dimension: if a data element$d$ is referenced at time t, then it is likely that some data element d' in the *vicinity* of d is referenced at time $t+1$.

This first attempt to approach locality formally was followed by [McCabe78, Robidoux79] in which reference strings obtained from an existing application system was analyzed for purpose of identifying reference locality. These empirical works further illuminated the need for a theoretical framework for the study of database locality.

In [Moulton86], a theoretical foundation for database locality has been developed using models that are close analog of program locality working set models, but applicable at all stages of database processing - logical, internal, and storage. A two dimensional model that incorporates both temporal and spatial locality effects, with separately adjustable spatial and temporal parameters, is defined. The model reduces to the pure temporal model in the limiting spatial case. This model enables one to measure the locality of a given reference string. Work is currently being performed in applying the above model to measure locality of existing database systems and examining the theoretical properties of the model.

4.4. Performance Evaluation

In this part we describe research efforts directed towards evaluating the speed performance of INFOPLEX. Early performance modeling through simulation for the INFOPLEX storage hierarchy was reported in [LM79d, WM81]. Due to the high level of concurrency being simulated, these early simulation efforts were very costly and it became clear that it was necessary to build analytical evaluation tools that are suitable for a distributed architecture. However, the very nature of the system having ***unbalanced asynchronously-spawned (UAP) parallel tasks*** violates the flow balance requirement of classical queueing theory.

In [WM84a], an analytical performance evaluation methodology for ***flow-unbalanced*** networks is described. The method enables a flow-unbalanced queueing network to be transformed into an "equivalent" flow-balanced queueing network through decomposition. For flow-unbalanced ***open*** queueing networks, this transformation enables the key performance measures to be computed by directly applying the classical queueing theory. For flow-unbalanced ***closed*** queueing networks, however, the transformation is feasible only when conditions of ***network stability*** are satisfied. These conditions are formally derived and readily computable given the parameters of a flow-unbalanced closed queueing network. In addition, an efficient iterative procedure for estimating the key performance measures of a flow-unbalanced closed queueing network is developed. The procedure is based on Buzen's convolution algorithm which efficiently computes the normalization constant in the product form solution of a classical closed queueing network.

The solution methodology for flow-unbalanced queueing networks is applied to the INFOPLEX storage hierarchy [WM84b and WM86]. The necessary and sufficient conditions for the INFOPLEX storage hierarchy to be stable when modeled as a closed queueing network are identified. Furthermore, an algorithm has also been devised to test whether a design alternative of the INFOPLEX storage hierarchy will be stable.

4.5. Experimental Test Vehicles

In this section, the approach taken to build a test vehicle for INFOPLEX is described. It consists of a multi-microprocessor hardware test vehicle and a simulated software test vehicle.

4.5.1. Software Test Vehicle

The purpose of the software test vehicle (STV) project is to test out the preliminary designs of the functional hierarchy [Hsu80] and the storage hierarchy [LM79a] in software emulation on contemporary hardware before committing them to hardware prototypes. The first approach was to emulate the parallelism of the target architecture on a conventional single-processor computer. The STV project was carried out on an IBM 370 mainframe and was implemented in PL/1. The project consisted of three parts: a functional hierarchy STV [BM81, Hsu82a, Lee82, Lu82], a storage hierarchy STV, and a hardware emulator, called Shell [To82].

The preliminary design implemented in STV adopts a software paradigm composed of modules. The modules in STV are distributed to functional levels in the emulated functional hierarchy based on the nature of the task performed by the modules. The inter-level module invocation is performed through message passing, emulated by "Shell", and no argument passing through shared memory variables is allowed. From the STV experience, it appears that there is a need to identify a software paradigm, or a functional decomposition methodology, that is more sensitive to the distributed nature of the target hardware, and subject the design of the functional modules to the paradigm. Work is currently being performed to identify such a methodology.

The cost of software emulation of a concurrent system architecture employing high degrees of parallelism and pipeline processing on a conventional mainframe renders it impractical to conduct extensive experiments. A more effective method for experimenting with the concurrent functional software is to adopt parallel computers, such as off-the-shelf micro-computers interconnected through a high bandwidth network. Work is currently being pursued to set up such a more advanced test bed environment for further software experiments.

4.5.2. Hardware Test Vehicle

For the hardware test vehicle, the purpose is to demonstrate the feasibility of the INFOPLEX multi-level multiprocessor hardware architecture. The first task is to build a one-level shared-memory multiprocessor system. Several issues are investigated. The first issue is the choice of the processor. To this end, off-the-shelf microprocessors are evaluated [TG81, GT83a, GT83b, GT83c] and trends in evolution monitored closely [GT85b]. Methodologies for evaluating microprocessors are reported in [GAT81, GT82]. In particular, in [GT82], a hierarchical approach to evaluating microprocessors is also proposed.

The issue of multiprocessor interconnection scheme is studied in [GT80, GST80, Gupta82] In particular, the *pended bus architecture* utilizing the split transaction protocol is shown to be especially effective in minimizing potential contention on the bus and therefore able to support a large number of processors on the bus. The split transaction bus protocol allows a processor to relinquish control of the bus when its memory request is being serviced at a memory module, enabling another processor or memory module to obtain control of the bus in the mean time to transfer another request or data over the bus. The protocol therefore allows the effective bus throughput to be increased. Performance of the pended bus architecture is studied in detail to resolve relevant design decisions at the implementation level. These results are reported in [Gupta82, GT82, GT84, GT85a], and are used in designing the bus interface unit (BIU) that connects processors and memory modules to the intra-level bus of INFOPLEX.

Another shared memory multiprocessor issue is the cache consistency problem. In [AM81], this problem is studied in detail and simulation conducted to evaluate the performance of various schemes. In the prototype multiprocessor system, the cache consistency problem is avoided by not allowing data segments to be cached.

The prototype one-level shared-memory multiprocessor system is completed, together with its kernel software, i.e., the *local operating system* [TAL86]. Work is being performed in replicating the one-level prototype and connecting them through a global bus (i.e., the inter-level bus). The same BIU design will be utilized in connecting a level to the global bus. The local operating system will also be extended to handle inter-level communication protocols.

5. Conclusion

The purpose of the INFOPLEX research project is to advance the technologies for information management through both basic and experimental research within the framework of a structured multi-processor architecture. In this document, a summary of the recent research results of the project is reported. The research falls in the following categories: (1) distributed algorithms, (2) database locality, (3) performance evaluation methodology, and (4) experimental test vehicles. These works lay the foundation for future research and implementation efforts, while additional work is needed to refine and apply the theoretical results and resolve detailed design and implementation decisions.

6. Technical Reports and References

[Abraham79]: Abraham, M. 'Properties of reference algorithms for multi-level storage hierarchies,' Master's Thesis, M.I.T. Sloan School of Management, 1979.

[AM81]: Abdel-Hamid T.K. and Madnick, S.E. 'A study of the multicache-consistency problem in multi-processor computer systems,' *Proceedings of the Sixth Workshop on Computer Architecture for Non-Numeric Processing*, June 1981.

[BM81]: Blumberg, B. and Madnick, S.E., 'INFOSAM: A sample database management system,' NTIS No. AD-A116-593, December 1981.

[GAT81]: Gupta, A., Abdel-Hamid, T. and H.D. Toong, 'A Comparison of analytic and simulation models,' M.I.T. Sloan School of Management, December 1980.

[GST80]: Gupta, A., Strommen, S.O. and H.D. Toong, 'Evaluation of multimicroprocessor bus architecture,' M.I.T. Sloan School of Management, May 1980.

[GT80]: Gupta, A., and H.D. Toong, 'Interactive multimicroprocssor performance systems,' Sloan School of Management, April 1980.

[GT82]: Gupta, A., and H.D. Toong, 'Enhanced concurrency in m-n multiprocessor systems,' *Proceedings of the IEEE Third International Conference on Distributed Computing Systems*, October 1982.

[GT83a]: Gupta, A., and H.D. Toong, (eds.) *Advanced Microprocessors*, IEEE Press Selected Reprint Series, IEEE Computer Society, IEEE Press, New York, NY, 1983.

[GT83b]: Gupta, A., and H.D. Toong, 'An Architectural comparison of 32-bit microprocessors,' *IEEE Micro*, Vol. 3, No. 1, February 1983. (republished in *Microprocessors and Microcomputers*, IEEE Press: New York, NY, 1984)

[GT83c]: Gupta, A., and H.D. Toong, 'Microprocessors - the first twelve years,' *Proceedings of the IEEE*, Vol. 71, No. 11, November 1983.

[GT84]: Gupta, A., and H.D. Toong, 'Microcomputers in industrial control applications,' *IEEE Transactions on Industrial Electronics*, Vol. IE-31, No. 2, May 1984.

[GT85a]: Gupta, A., and H.D. Toong, 'Increasing throughput of multiprocessor configurations,' *IEEE Transactions on Industrial Electronics*, August 1985.

[GT85b]: Gupta, A., and H.D. Toong, 'Trends in Microcomputers,' in *International Handbook of Information Technology and Office Systems*, A.E. Cawkell (ed.) North-Holland Publishing Company: Amsterdam, Netherlands, 1985.

[Gupta82]: Gupta, A., 'Performance modeling of multimicroprocessor systems,' *Proceedings of the South-East Asia Regional Computer Conference*, Kuala Lampur, Malaysia, September 1982.

[HC85]: Hsu, M. and Chan, A. 'Partitioned two-phase locking,' *Proceedings of the First International Workshop on High Performance Transaction Systems*, September 1985.

[HM77]: Hsiao, D.K. and Madnick, S.E., 'Database machine architecture in the context of information technology evolution,' *Proceedings of the Third International Conference on VLDB*, October 1977.

[HM83]: Hsu, M. and Madnick, S.E. 'Hierarchical database decomposition: A technique for database concurrency control,' *Proceedings of 2nd ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, March 1983.

[Hsu80]: Hsu, M. 'A preliminary architectural design for the functional hierarchy of the INFOPLEX database computer,' NTIS No. AD-A102-924, November 1980.

[Hsu82]: Hsu, M. 'FSTV: The software test vehicle for the functional hierarchy of the INFOPLEX database computer,' NTIS No. AD-A116-591, January 1982.

[Hsu83]: Hsu, M. 'The hierarchical decomposition approach to database concurrency control,'TR M010-8312-16, M.I.T. Sloan School of Managemenet, December 1983.

[Krakauer80]: Krakauer, L. 'Virtual information in the INFOPLEX database computer,' Master's Thesis, M.I.T. Sloan School of Management, 1980.

[LM79]: Lam, C.Y. and Madnick, S.E., 'Properties of storage hierarchy systems with multiple page sizes and redundant data,' *ACM Transactions on Database Systems*, Vol 4, No. 3, September 1979.

[LM79a]: Lam, C.Y. and Madnick, S.E., 'Intelligent memory system architectures - research directions,' NTIS No. AD-A073-485, June 1979

[LM79b]: Lam, C.Y. and Madnick, S.E., 'The IMS data storage hierarchy - DSH-I,' NTIS No. AD-A073-375, June 1979.

[LM79c]: Lam, C.Y. and Madnick, S.E., 'The IMS data storage hierarchy - DSH-II,' NTIS No. AD-A073-376, June 1979.

[LM79d]: Lam, C.Y. and Madnick, S.E., 'Simulation studies of the DSH-II data storage hierarchy system,' NTIS No. AD-A074-503, June 1979.

[Lee82]: Lee, J. 'Virtual information facility of the INFOPLEX software test vehicle (Part I),' NTIS No. AD-A116-503, May 1982.

[Liu82]: Liu, D. 'N-ary level design of the INFOPLEX software test vehicle,' Bachelor's Thesis, M.I.T., 1982.

[Lu82]: Lu, P. 'Virtual information facility of the INFOPLEX software test vehicle (Part II),' NTIS No. AD-A116-502, May 1982.

[Madnick73]: Madnick, S.E. 'Storage hierarchy systems,' TR-105, Project MAC, M.I.T., 1973

[Madnick75a]: Madnick, S.E., 'INFOPLEX - Hierarchical decomposition of a large information management system using a microprocessor complex,' *Proceedings AFIPS 1975 International Computer Conference*, Vol. 44, May 1975.

[Madnick75b]: Madnick, S.E., 'Design of a general hierarchical storage system,' *Proceedings of the International Convention and Exposition of the Institute of Electrical Engineers*, April 1975.

[Madnick77]: Madnick, S.E., 'Trends in computers and computing: the information utility,' *Science*, Vol. 185, March 1977.

[Madnick79]: Madnick, S.E., 'The INFOPLEX database computer: concepts and directions,' *Proceedings IEEE Computer Conference*, February 1979.

[Madnick80]: Madnick, S.E., 'Recent research results on the INFOPLEX Intelligent Memory System Project,' *Proceedings of the International congress on Applied Systems Research and Cybernetics*, December 1980.

[McCabe78]: McCabe, E., 'Locality in logical database systems: a framework for analysis,' Master's Thesis, M.I.T. Sloan School of Management, 1978.

[Moulton86]: Moulton, A., 'The foundation of database locality,' in preparation.

[Robidoux79]: Robidoux, S., 'A closer look at database access patterns,' Master's Thesis, M.I.T. Sloan School of Management, 1979.

[Scrutchin85**]: Scrutchin, T. 'TPF: Performance, capacity and availability,' *Proceedings of the First International Workshop on High Performance Transaction Systems*, September 1985.

[Stortz83]: Stortz, H., Jr., 'A classification scheme for database machine architectures,' Bachelor's Thesis, M.I.T., 1983

[TAL86]: Toong, H.D., Abraham, M. and Linsky, M. 'The design and implementation of a fault-tolerant multiprocessor for the INFOPLEX database computer,' in preparation.

[TG81]: Toong, H.D. and Gupta, A. 'An architectural comparison of contemporary 16-bit microprocessors,' *IEEE Micro*, Vol.2, No. 2, May 1981. (reprinted as a chapter in *Microcomputer Networks*, IEEE Press: New York, NY, 1981)

[TG82]: Toong, H.D. and Gupta, A. 'Evaluation kernels for microprocessor analysis,' *Performance Evaluation*, North-Holland Publishing Company: Amsterdam, Vol., 2, No. 1, 1982.

[TG84]: Toong, H.D. and Gupta, A. 'Hardware feasibility of INFOPLEX design strategy,' Sloan School of Management, May 1984.

[To82]: To, T., 'SHELL: a simulator for the software test vehicle for the INFOPLEX database computer,' NTIS No. AD-A116-592, August 1982.

[WM81]: Wang, Y.R. and Madnick, S.E., 'Performance evaluation of the INFOPLEX database computer using operational analysis,' TR M010-8109-13, M.I.T. Sloan School of Management, September 1981.

[WM84a]: Wang, Y.R. and Madnick, S.E., 'Queueing network systems with unbalanced flows and their applications to highly parallel distributed information systems,' TR M010-8408-14, M.I.T. Sloan School of Management, August 1984.

[WM84b]: Wang, Y.R. and Madnick, S.E., 'Performance evaluation of distributed systems with unbalanced flows: an analysis of the INFOPLEX data storage hierarchy,' TR M010-8109-15, M.I.T. Sloan School of Management, July 1984.

[WM86]: Wang, Y.R. and Madnick, S.E., 'Modeling multiprocessor computer systems with unbalanced flows,' to be published in *Proceedings of the Joint Conference on Computer Performance Modelling, Measurement, and Evaluation*, 1986.

*Note:

Documents distributed through the National Technical Information System (NTIS) can be obtained by writing to: NTIS, 5282 Port Royal Rd., Springfield, Virginia 22161

Acknowledgement: The authors would like to thank the following current and past participants in the INFOPLEX Project for their valuable contribution to the research progress and results reported in this paper:

Current Investigators:

Hoo-Min D. Toong, M.I.T.   Amar Gupta, M.I.T.
Richard Ying Yu Wang, University of Arizona

Current Participants:

Mike Abraham   Allen Moulton
B.E. Prasad (on a Grant from Indian Government)

Previous Investigators and Participants:

Peter P.S. Chen	David K. Hsiao
Tarek K. Abdel-Hamid	Zeev Bulka
Bruce Blumberg	Benjamin Chou
Larry Krakauer	Earl Goodrich
Chat Lam	Jameson Lee
Peter Lu	David Lui
Edward McCabe	John-Francis Mergen
Susanne Robidoux	Charles J. Smith
Herb Stortz	Svein Ove Strommen
Tak To	Anthony Wang

The INFOPLEX project has been mainly supported by the Space and Naval Warfare Systems Command (formerly the Naval Electronic Systems Command) under contract numbers N00039-78-G-0160, N00039-80-K-0573, N00039-80-K-0498, N00039-81-C-0663, N00039-83-C-0463, N00039-85-C-0571. Some earlier work was supported by a grant from National Science Foundation under contract number MCS 77-20829. The work on the evaluation and implementation of the multiprocessors has been partly sponsored by a grant from I.B.M. The views expressed in this document are those of the authors and should not be interpreted as representative of the official policies, either expressed or implied, of any sponsoring agencies.

CALL FOR PAPERS

DATA ENGINEERING

The Third International Conference on Data Engineering

Pacifica Hotel
Los Angeles, California, USA
February 2-6, 1987

Sponsored by the IEEE Computer Society

Committee

Steering Committee Chairman:
C. V. Ramamoorthy
University of California, Berkeley, CA 94720

Honorary Chairman:
P. Bruce Berra
Syracuse University, Syracuse, NY 13210

General Chairman:
Gio Wiederhold
Dept. of Computer Science
Stanford University, Stanford, CA 94305
(415) 723-0685 wiederhold@sumex.stanford.edu

Program Chairman:
Benjamin W. Wah
Coordinated Science Laboratory
University of Illinois, Urbana, IL 61801
(217) 333-5216 wah%uicsld@uiuc.arpa

Program Co-Chairpersons:
John Carlis, *Univ. of Minnesota, Minneapolis, MN 55455*
Iris Kameny, *SDC, Santa Monica, CA 90406*
Peter Ng, *Univ. of Missouri-Columbus, Columbia, MO 65211*
Winston Royce, *Lockheed, Austin, TX 78744*
Joseph Urban, *Univ. of SW Louisiana, Lafayette, LA 70504*

International Coordination:
Tadao Ichikawa, *Hiroshima University, Higashi-Hiroshima 724, Japan*
G. Schlageter, *Fern Universitat, D 5800 Hagen, FR. Germany*

Tutorials:
James A. Larson, *Honeywell Computer Sciences Center*
1000 Boone Avenue North, Golden Valley, MN 55427
(612) 541-6836 jalarson@hi-multics.arpa

Awards:
K.H. Kim, *University of South Florida, Tampa, FL 33620*

Treasurers:
Aldo Castillo, *TRW, Redondo Beach, CA 90278*

Local Arrangements:
Walter Bond, *Cal State University, Dominquez Hills, CA 90747*
(213) 516-3580/3398 bond@calstate.bitnet
Mary C. Graham, *Hughes, P. O. Box 902,*
El Segundo, CA 90245 (213) 619-2499

Publicity:
Dick Shuey, *2338 Rosendale Rd., Schenectady, NY 12309*
shuey@ge-crd.arpa

Committee Members (Tentative)

Jacob Abraham
Adarsh K. Arora
J.L. Baer
Faroh B. Bastani
Don Batory

Bharat Bhargava
Joseph Boykin
Richard Braegger
Alfonso Cardenas
C.R. Carlson
Nick Cercone
Peter P. Chen
Bernie Chern
Roger Cheung
David Choy
Wesley W. Chu

J. DeJong
David J. DeWitt
Ramez ElMasri
Robert Epstein
Michael Evangelist
Domenico Ferrari
Hector Garcia-Molina
Georges Gardarin
Sakti P. Ghosh
Arnold Goldfein
Georg Gottlob
Laura Haas
Lee Hollaar
Yang-Chang Hong
David K. Hsiao
H. Ishikawa
Sushil Jajodia
Jie-Yong Juang
Arthur M. Keller
Larry Kerschberg
Won Kim
Roger King
Dan Kogan
Robert R. Korfhage
Tosiyasu L. Kunii
Winfried Lamersdorf
Matt LaSaine
W.-H. Francis Leung
Victor Li
Yao-Nan Lien
Witold Litwin
Jane W.S. Liu
Ming T. (Mike) Liu
Raymond Liuzzi
Vincent Lum
Yuen-Wah Eva Ma
Mamoru Maekawa
Gordon McCalla
Toshimi Minoura
N.M. Morfuni
Jack Mostow
Jaime Murow
Sham Navathe
P M. Neches
Erich Newhold
G.M. Nijssen
Ole Oren
G. Ozsoyoglu
Z. Meral Ozsoyoglu
C. Parent
J F. Paris
D.S. Parker
Peter Rathmann
Lakshmi Rebbapragada
David Reiner
Gruia-Catalin Roman
Domenico Sacc à
Giovanni Maria Sacco
Sharon Salveter
Edgar Sibley
David Spooner
John F. S owa
Peter M. Stocker
M. Stonebraker
Stanley Su
Denji Tajima
Marjorie Templeton
A.M. Tjoa
Yosihisa Udagawa
Susan Urban
P. Valduriez
R.P. VanDeRiet
Yann Viemont
Neil Walker
Helen Wood
S. Bing Yao

For further information write to:
Third International Conference on Data Engineering
c/o IEEE Computer Society
1730 Massachusetts Ave., N.W.
Washington, D.C. 20036-1903
(202) 371-0101

SCOPE

Data Engineering is concerned with the role of data and knowledge about data in the design, development, management, and utilization of information systems. As such, it encompasses traditional aspects of databases, knowledge bases, and data management in general. The purpose of this conference is to continue to provide a forum for the sharing of experience, practice, and theory of automated data and knowledge management from an engineering point-of-view. The effectiveness and productivity of future information systems will depend critically on improvements in their design, organization, and management.

We are actively soliciting industrial contributions. We believe that it is critically important to share practical experience. We look forward to reports of experiments, evaluation, and problems in achieving the objectives of information systems. Papers which are identified as such will be processed, scheduled, and published in a distinct track.

TOPICS OF INTEREST

- Logical and physical database design
- Data management methodologies
- Distribution of data and information
- Performance evaluation
- Expert systems
- Data security
- Design of knowledge-based systems
- Architectures for data- and knowledge-based systems
- Data engineering tools
- Applications

The days preceding and after the conference will be devoted to tutorials. Additional mini-tutorials will be presented during the last evening of the conference. A special DBMS vendor day will include short DBMS-specific tutorials to acquaint attendees with current commercially available products. Those interested in presenting tutorials should contact the tutorial chairman by May 15, 1986.

Awards, Student Papers, and Subsequent Publication:
An award will be given for the best paper at the conference. The best student paper will receive the K.S. Fu award, honoring one of the early supporters of the conference. Up to three awards of $500 each to help defray travel costs will be given for outstanding papers authored by students. Outstanding papers will be considered for publication in the IEEE Computer Society *Computer* Magazine, the *IEEE Expert* Magazine, the *IEEE Software*, and the *IEEE Transactions on Software Engineering*. For more information, contact the General Chairman.

Paper Submission:
Four copies of papers should be mailed before June 15, 1986 to:
Third Data Engineering Conference
IEEE Computer Society
1730 Massachusetts Ave., N.W.
Washington, DC 20036-1903
(202) 371-0101

Conference Timetable:

Tutorial proposals due:	May 15, 1986
Manuscripts due:	June 15, 1986
Acceptance letters sent:	September 15, 1986
Camera-ready copy due:	November 11, 1986
Tutorials:	February 2, 6, 1987
Conference:	February 3-5, 1987

Epilog
The correct design and implementation of data systems requires attention to principles from databases, knowledge bases, software engineering, and system evaluation. We hope you will participate.

IEEE COMPUTER SOCIETY
IEEE THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

CALL FOR PAPERS

International Workshop on Object-Oriented Database Systems (OODBS)

September 23-26, 1986 **Asilomar Conference Center, Pacific Grove, California**

Sponsored by:	Association for Computing Machinery — SIGMOD IEEE Computer Society — TC on Database Engineering
In cooperation with:	Gesellschaft fur Informatik, Germany FZI at University of Karlsruhe, Germany IIMAS, Mexico

Purpose:

To bring together researchers actively interested in specific concepts for database systems that can directly handle objects of arbitrary structure and complexity. Application environments for which such characteristics are required include CAD, software engineering, office automation, cartography and knowledge representation. Important issues include data/information models, transaction mechanisms, integrity/consistency control, exception handling, distribution, protection, object-oriented languages, architectural issues, storage structures, buffer management, and efficient implementation.

Format: Limited attendance workshop. Participation is by invitation only.

Everybody wishing to participate must submit a full paper that will be reviewed by the program committee. Description of work in progress is encouraged and modifications to the submitted paper can be made immediately after the workshop and prior to publication in order to reflect the progress made during the time between submission and publication and the insights gained from the workshop.

Participants will be invited by the program committee based upon the relevance of their interests/contributions. There will be ample discussion time with presentations and special discussion sessions. Proposals for discussion topics are invited.

Program committee:

K. Dittrich (FZI Germany) — chairman	U. Dayal (CCA) — co-chairman
D. Batory (Univ. of Texas)	M. Haynie (Amdahl)
A. Buchmann (Univ. of Mexico)	D. McLeod (USC)

Conference Treasurer: D. McLeod

Local arrangments: M. Haynie

Publication:

All participants will be sent copies of the accepted papers prior to the meeting. A book containing revised papers and recorded discussions (as far as justified by quality) may be published after the workshop.

Important dates:

Submission of manuscripts:	April 25, 1986
Notification of acceptance:	June 15, 1986
(early notification via electronic mail)	June 3, 1986
Submission of papers for preconference distribution:	July 10, 1986

Mode of submission: Please mail 7 copies of manuscript to:

Umeshwar Dayal
CCA
Four Cambridge Center
Cambridge, MA 02142
USA

dayal@cca-unix.arpa
Phone: +1 617/492-8860

or

Klaus Dittrich
FZI
Haid-und-Neu-Strasse 10-14
D-7500 Karlsruhe 1
Germany

dittrich@Germany.arpa
Phone: +49 0721/69 06-0

Remember to include your electronic mail address for early notification.