Copyright © 1978, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

A DISTRIBUTED DATA BASE MACHINE

by

M. Stonebraker

Memorandum No. UCB/ERL M78/23

23 May 1978

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

A DISTRIBUTED DATA BASE MACHINE

by

Michael Stonebraker
Electronics Research Laboratory
University of California, Berkeley

ABSTRACT

In this paper we propose the use of a distributed data base machine to augment the performance of a data base system. As such, the paper has points in common with proposals for "back end data base machines" such as [CANA74]. However, our organization is fundamentally oriented toward multiprocessing of data base commands rather than simply moving their execution from an expensive host to a cheaper back end. Hence, our design bears a close resemblance to previously proposed distributed data base systems such as distributed INGRES [STON77] and SDD-1 [ROTH77]. However, unlike such systems, we propose to exploit the notion of a central controller ("GOD") to simplify the flow of control and augment performance.

I INTRODUCTION

There are at least two possible motives for proposing distributed computing in a data base management environment. The first is to exploit the parallelism possible with multiprocessor systems in order to increase the throughput of data base management transactions. The second is to allow distribution of the control of a data base and its contents (geographically and/or logically).

Systems such as SDD-1 [ROTH77] and distributed INGRES [STON77] were designed to exploit the advantages that distribution of data and control may accrue. They are

---

Research sponsored by National Science Foundation Grant #MCS75-03839-A01 and Army Research Office Grant #DAAG29-76-G-0245.

multiprocessor systems in which there is no notion of a central controller (or "GOD") and details concerning the location of data are hidden from the end user. Issues such as resiliency to crashes and distributed concurrency control are stressed. The primary motivation behind systems such as RAP [OZKA75,OZKA77], CASSM [COPE73], and RARES [LIN 76] is parallelism.

These may be described as "associative disks", and the general idea is to duplicate the search logic for the data base on each track of a fixed head disk (or CCD device) so as to search a device in parallel.

While the focus of this paper is on the parallelism made possible by distributed computing, the proposed design bears a close resemblance to SDD-1 or distributed INGRES. In fact, the main difference is that this design specifically includes a central controller whose existence simplifies the control flow and augments performance. There is no resemblance to associative disks.

The design of the distributed data base machine is motivated by the following deficiencies that we see in associative disks.

1) Associative disks do not substantially outperform conventional access methods for simple transactions.

For example, an associative disk can find the salary of Stonebraker by a parallel search of an associative device. However, a conventional access method can store the employee records hashed on employee name. As such it can execute the above request by directly addressing the correct bucket. For a given secondary storage device (fixed head disk, conventional disk, or CCD), both methods will perform comparably.

In many environments a large majority of the transactions are quite simple. In such cases an associative disk (which will cost far more than a conventional one) will not be particularly cost effective.

2) Services such as crash recovery and concurrency control must be an integral part of any data base system. Associative disks have tended to overlook these issues.

3) Associative disks have exploited the parallelism possible within the execution of one query. However, they have ignored the potential parallelism possible in the execution of different concurrent transactions.

4) Associative disks do not appear to be cost effective on large data bases.

Associative disk proposals appear to perform well when the whole data base fits on the device. If this is not possible, relevant data must be loaded a portion at a time from a larger capacity backing store (typically a moving head disk). In such cases the device will perform approximately at the speed of the backing store. Hence, it will not outperform a system which directly manipulates the same backing store.

Consequently, an associative disk will only be cost effective for data bases that are small (say 10**7bytes) or that have a small subset which almost all transactions exclusively access. In addition, an associative disk performs best for complex transactions (i.e. ones with several boolean conditions in the qualification). For simple transactions or large data bases without locality a different architecture is needed.

In the next section we discuss an architecture which avoids all deficiencies mentioned above. Then in Section 3 we discuss six significant design decisions and our current thinking on each. Lastly, Section 4 summarizes our approach and discusses an implementation plan.

One feature of our proposal is that it is designed to fit nicely into the current structure of the existing INGRES uniprocessor data base system [HELD75, STON76]. In fact, our proposal amounts to offloading portions of the existing code onto auxiliary processors. Consequently, we plan to minimize implementation difficulties by initially insisting that all processors be PDP-11's.

Another recent proposal, DIRECT [DEWI78] is embedded in the basic INGRES software architecture. It is a multiprocessor system which overcomes deficiency 3 (and perhaps 2) but does not address issues 1 and 4. It has points in common with our proposal.

II SYSTEM ARCHITECTURE

The conceptual architecture which we have in mind is shown in Figure 1. Here, we hypothesize a host CPU (say an 11/70 or VAX) with its own collection of peripheral equipment together with a group of satellite CPU's each with its own secondary storage system. In our implementation these will be 11/34's or 11/40's, but in principle they could be anything which could support the needed functions. The general idea is that each satellite CPU will perform some data base management functions. The host will provide the remainder of the functions. It is expected that the satellites will have no responsibilities unrelated to data base management.

Each satellite will contain one (or perhaps more) moving head disk devices and perhaps a cache. (The utility of a cache is discussed in Section 3). In addition, the satellite CPU's will use conventional access methods to limit the amount of data examined. Lastly, the satellites will operate asynchronously. Hence, different satellites may or may not be executing the same command.

The general advantages of such an architecture are the following.

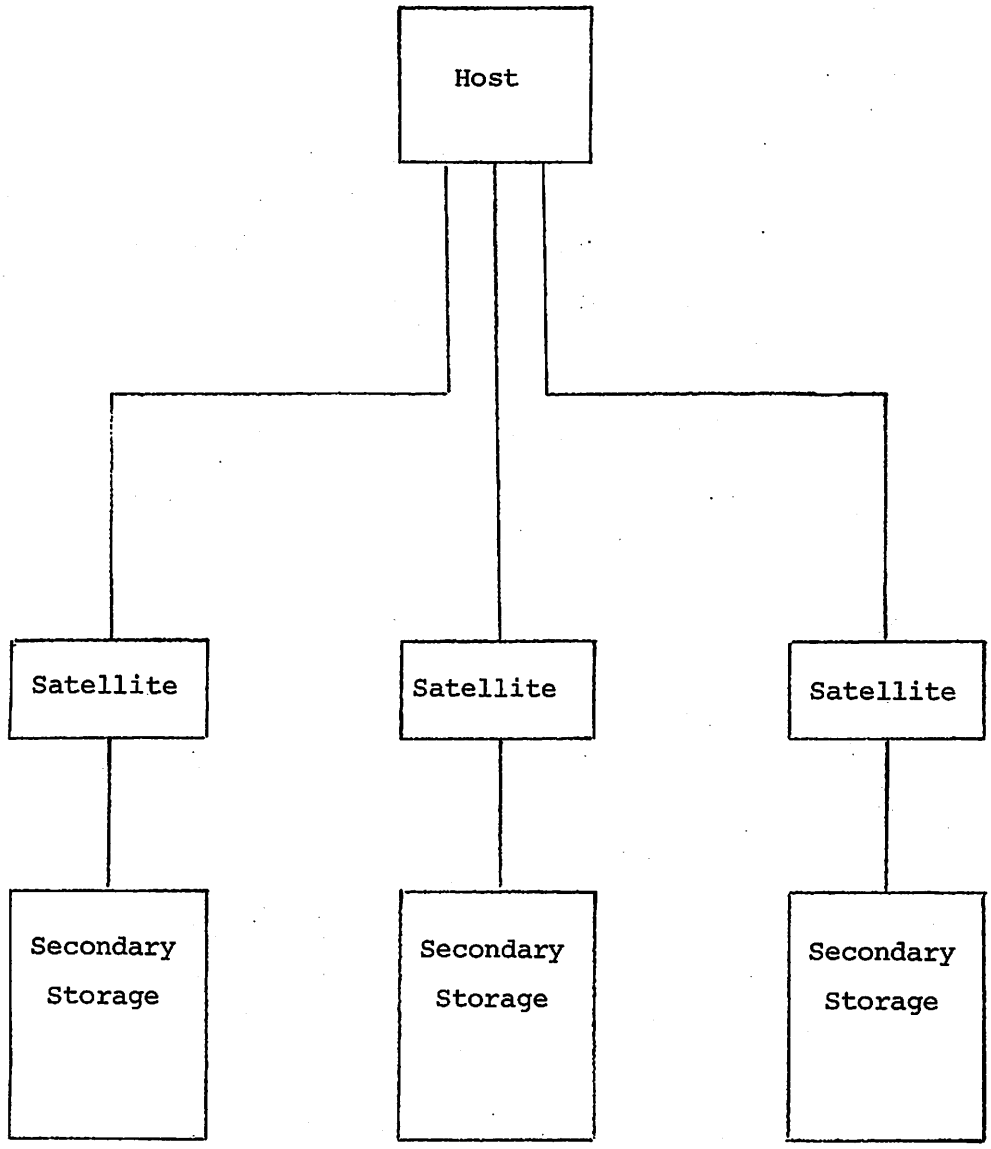
* Large data bases can be accommodated.

* Conventional access methods avoid exhaustive searches for simple transactions.

* Parallelism is achieved whenever multiple satellites can be coordinated to execute a command. This will happen if desired data exists on multiple devices. Moreover, parallel execution of transactions which reference data on different satellites is possible. Hence both inter-query and intra-query parallelism [DEWI78] is possible.

* Concurrency control and crash recovery can be provided (either by the host or by the host in conjunction with the satellites).

* The satellite CPU may be cheaper than a host because it may be slower and have less functions.



The Proposed Architecture

Figure 1

* The software in the satellite may be small and simple (and hence reliable and fast).

* The satellite CPU can be microprogrammed to execute DBMS functions very rapidly.

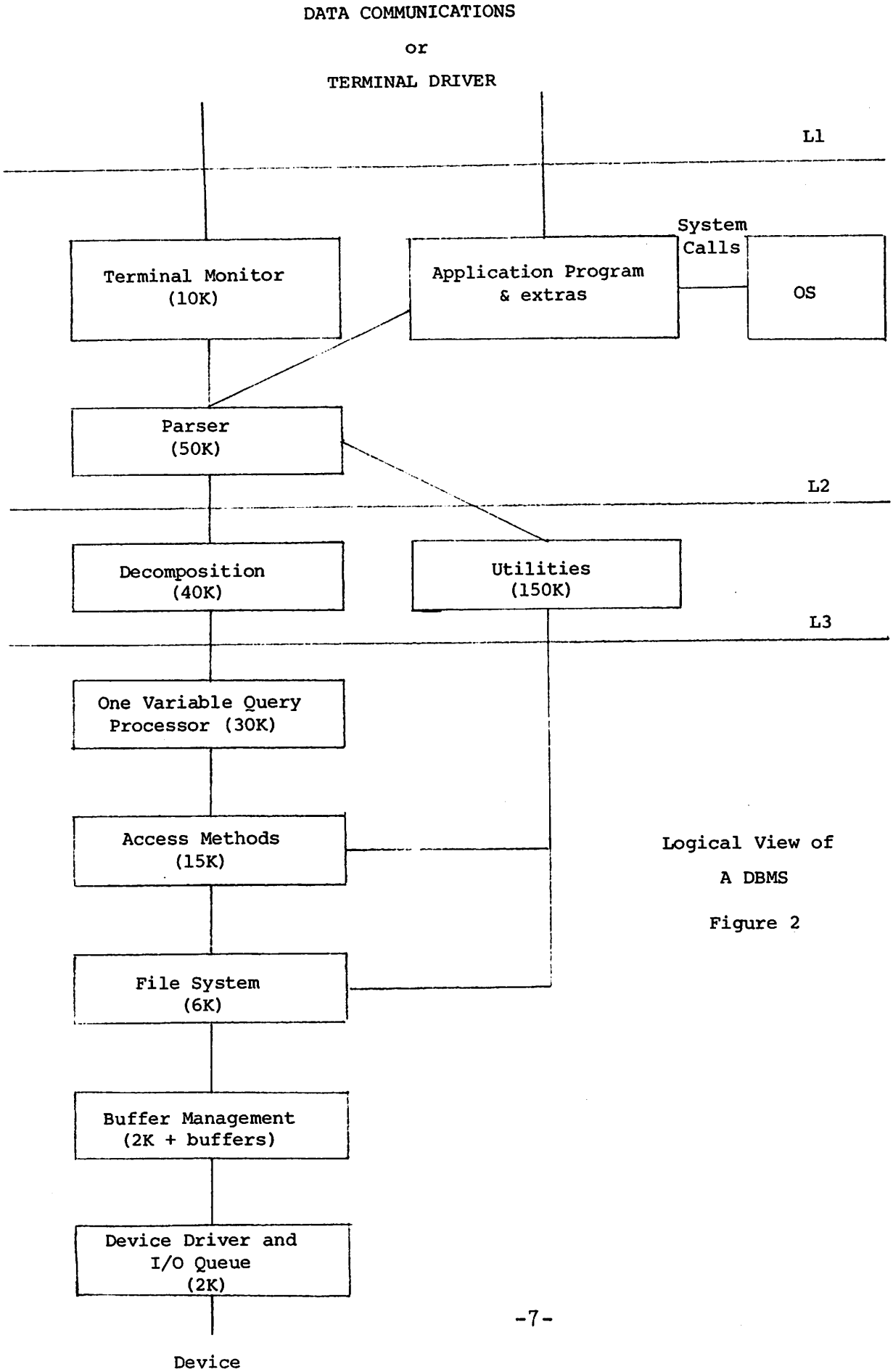
* The satellites need never communicate with each other. Hence, interface hardware and software is economized.

Although it is possible that the satellite and the host CPU's will be the same, it is felt that such a choice will not be particularly cost effective.

We turn now to the functions which should be performed by the satellites. Of course, our proposal depends on the structure of the INGRES data base system which is diagramed in Figure 2.

The following interpretation of Figure 2 is intended. At the top of the figure one notes that input from humans comes from data communications or terminal support software in the operating system. This input either goes to a terminal monitor (for humans interactively running INGRES) or to an application program. Such an application program may make requests on the data base system as well as system calls for other operating system services. The part of the operating system not explicitly shown elsewhere is present to the right of the applications program. Either the applications program or the terminal monitor makes data base requests. These requests are either data manipulation commands (RETRIEVE, APPEND, DELETE, REPLACE) or utility commands (CREATE, DESTROY, HELP, SAVE, BULKLOAD, MODIFY). The utility commands are, in general, ones which interact with the system catalogs (data dictionary) or affect the physical organization of the data base. These utility commands are processed by calls on the access methods or the file system.

The data manipulation commands may span several relations. Hence, they must be decomposed into simpler commands only involving one relation. Then a one relation command can be processed by making calls on the access methods. The lower three levels are easily identifiable as steps taken within the operating system (UNIX) in executing an access method request.



Logical View of
A DBMS

Figure 2

An estimate of the size of most modules is given in bytes for the current version of INGRES. It should also be noted that the data base system can perform the functions of some of the boxes at compile time for application programs. Naturally, for ad hoc interactions submitted through the terminal monitor, all actions must take place at run time. In the current INGRES environment, all actions take place only at run time. If application programs had functions performed at compile time, there would be only a slight impact on this proposal.

Actions to be performed at run time can be done either in the host or in one of the satellites. There are alternatives within this division of labor, labeled L1 through L3. In each case, everything below the line runs in the satellite; everything above the line in the host.

Basically L1 suggests running the entire data base system plus the application program in the satellite. On the other hand, L2 suggests running the data base system in the satellite, while L3 has the "inner loop" of the data base system in the satellite.

It should be noted that no lower level interface between the satellite and the host makes sense. The general reason is that it will take 2000-4000 host instructions to set up a call from the host to the satellite and process the return. This overhead includes a system call from the application program, some validity checking, writing the command over the interface to the satellite, doing a task switch to run another process, accepting a return from the satellite, awakening the user process and communicating the return to it. With comparable overhead the host can also directly process a file system call or an access method call. Hence, the host might as well do the call directly as pass it to a satellite. The bottom line is that a satellite architecture will win only if the time to process a satellite command is large compared to 2000-4000 instructions.

It should be noted that both L1 and L2 would offer the general architecture proposed for distributed INGRES and SDD-1. We feel, however, that L3 is an attractive interface when performance enhancement is the only goal. Hence, in the remainder of this section we explore some advantages and

disadvantages of L3 compared to a conventional uniprocessor data base system. In the next section we briefly return to the advantages which L3 might have over L2 or L1.

2.1 Advantages of L3

1) Data need never be transferred between the host and the satellite unless the user wishes it displayed.

For example, a user can request that the data base system give a ten percent raise to employee Zilch. Only the command needs to be transmitted from host to satellite and a status condition returned; Zilch's record does not need to be moved. As a result, the information sent across the interface is economized.

2) Natural parallelism can be exploited.

It is clear that independent commands from different host processes can be simultaneously executed on different satellites. However, decomposition also allows innate parallelism in processing commands which span multiple relations [WONG76, STON76]. Such parallelism can be easily exploited in this architecture.

3) The access methods and file system can be combined, which results in performance improvement.

The satellite software need not support any notion of a UNIX file. Instead, only custom software directly supporting the access method need be present. Such software can do buffer management more cost effectively than is possible in a general purpose operating system. For example, the access method often knows exactly which physical storage block will be examined next. Hence, a "perfect" prefetch buffer management policy can be implemented. Also, should indexed sequential access be desired, a multilevel tree structure will be constructed. This tree can be integrated with the information concerning physical placement of blocks. Doing both tasks at once instead of separately results in performance improvement.

4) Low level code is economized.

It is possible that no protection and concurrency control features will be needed in the satellite. Consequently, code to support them need not be present. More on this subject later.

5) The satellite can be efficiently microcoded.

The only task of the satellite is to run 60-80 Kbytes of code efficiently. Hence, the instruction set can be chosen appropriately. For example, a substantial portion of the one variable command processor is code to evaluate a QUEL qualification for a given record. This expression evaluator can be microcoded. Searching directory pages in an indexed sequential collection of records can be similarly committed to microcode. Other microprogrammable functions are data compression and computation of hash functions.

2.2 Disadvantages of L3

1) The one variable command processor is essentially confined to being an interpreter.

There are cases where a substantial gain is possible from compilation [LORI77]. However, it is felt that a well chosen instruction set for the interpreter can be as efficient as the code which might be compiled for a general purpose host. Hence, it is expected that disadvantage 1 is alleviated by advantage 5. Also, the one variable command processor can optionally compile on the fly code for the evaluation of an individual command. This strategy will also eliminate the potential disadvantage.

2) A mechanism must be found to convert between storage structures.

It is clearly essential to convert between different implementations of the access methods (say indexed sequential to hash) and to create secondary indices on the fly. These utilities must be run in the satellite otherwise the host-satellite interface must be enlarged so that these functions can be performed under the direction of the host. To do so would require that at least the access method calls of GET and INSERT be visible to the host.

My bias is to run this portion of the utilities in the satellite. Since they are run infrequently, the code can overlay the normally resident code.

3) Performance may be improved by showing the access method to the host as well as the one variable command processor.

In particular, the data dictionary may be stored in one (or more) satellites. Access to the data dictionary usually involves returning exactly ONE record. This can be more efficiently processed if expressed as an access method call rather than a query.

The disadvantage of showing the access method to the host is that it will be conceptually less elegant.

4) Decomposition may require records to be returned to the host one at a time.

If decomposition includes the notion of tuple substitution as a tactic, then the host must be able to iterate through a collection of records one at a time. Again, the obvious way to allow this is to support the access method interface as well as the one variable command interface.

5) The notion of a file may disappear.

It is not expected that the satellite will support anything resembling a UNIX file. Hence, unless the host has a conventional file system for one of its peripherals, the notion of a file will not be supported. For example, text editors would have to store and access data using the one variable command interface (or maybe the access methods). Although this would be appropriate for the text editor, it is less than ideal for all subsystems.

III SIGNIFICANT DESIGN DECISIONS

In this section we discuss how to handle the following issues:

1) multiprogramming of the satellite
2) CCD devices
3) placement of system catalogs

4) whether a relation can span multiple satellites
5) concurrency control
6) crash recovery

We treat each issue in turn.

1) multiprogramming

It is my bias that the satellite NOT be multiprogrammed. Instead, it will simply execute commands in a first-in-first-out manner. The bottom line is that satellite code will be simpler; there will be no scheduler, no I/O queue, no processes, etc. Also, concurrency control may be simplified. Lastly, it is expected that performance, comparable to that of multiprogramming, can be achieved.

The satellite can easily employ "look ahead" to prefetch the next data block for those commands which reference more than one block. In this way, overlap between I/O and CPU execution in a satellite is obtained. Moreover, if a satellite is "hanging" on a secondary storage transfer, it can perform initial processing on the next command in the interim.

Two possible drawbacks of this approach should be addressed.

a) A uniprogramming system cannot take advantage of disk seek optimization tactics such as SSTF, SCAN, FSCAN, etc. Rather it performs secondary storage accesses on a FIFO basis. My belief is that this is not a significant disadvantage. In general, each command will have 0 or 1 outstanding secondary storage request. Hence, the number of entries in the I/O queue is bounded above by the number of active commands if multiprogramming is supported. It is hard to imagine that this is greater than three or four. Hence, little scan optimization is possible.

Also, it is likely that a collection of I/O requests from one command will be localized. This is certainly the case for an indexed sequential access method during directory searches. If so, the best disk scheduling strategy may well be not to move the disk arm

for a while in the hope that another request for the same cylinder will be forthcoming from the process which issued the last request. Again, elaborate optimization of scanning may be inappropriate.

b) Uniprogramming cannot exploit the parallelism possible if multiple disk drives are connected to one satellite.

This is a correct disadvantage. My opinion is that only one drive will typically be connected to each satellite and both pieces would reside in one physical package. To efficiently service two drives, a satellite CPU would have to be twice as fast and contain more complex software. My preference is for two slower simpler CPU's.

2) CCD Devices

The satellite CPU can be a general purpose (fast) microprocessor with 60-80 Kbytes of primary memory. Here, we discuss the nature of the secondary store.

We argue that a three level system ("core", CCD and disk) is not viable. Hence, the system will either be "speedy" ("core" and CCD) or "pokey" ("core" and disk). (Regardless, for those situations in which the following analysis is inappropriate and a CCD cache is advantageous, we would propose handling it in the same fashion that a register cache for main storage is handled.)

In order for a CCD cache for a disk to be viable, one must get a high percentage of "hits" in the cache. We envision that "core" memory in the satellite will contain a buffer pool with space for:

1 page	(the current block)
3 pages	(indirect blocks, i.e. file control blocks)
3 pages	(index blocks for indexed sequential access)
3 pages	(secondary index blocks)

When processing a single call, the command processor will have NO locality other than what is on these ten pages. The only advantage to making the buffer pool larger than 10 blocks is to employ "look ahead". In any case it should be

clearly noted that a very limited amount of main memory will hold all desirable information.

In order to exploit a CCD cache (which is substantial compared to 10 blocks), one must obtain locality between commands, i.e. the data accessed by one command must be relevant to a subsequent command. Clearly, this possibility is highly application dependent. However, the following simple analysis "ballparks" application requirements.

Assume that something resembling the 80-20 law holds (i.e. that eighty percent of the references are to twenty percent of the objects). Hence, let R be the hit ratio and X the size of the cache (measured as a percentage of the whole data base). We suppose an R versus X curve of the form noted in Figure 3.

Basically Figure 3 says that Q percent of the references are to P percent of the data base. Below a cache size of P, the hit ratio is quadratic in X; above P the hit ratio is linear.

Further suppose that a CCD device is fifty times faster than a disk. Hence, the average delay to access a block from such a device (assuming the CCD has access time of one) is:

```
delay = R*1 + (1-R)*50
```

The cost of the device (assuming that the disk has unit cost and the CCD is Z times as expensive) is:

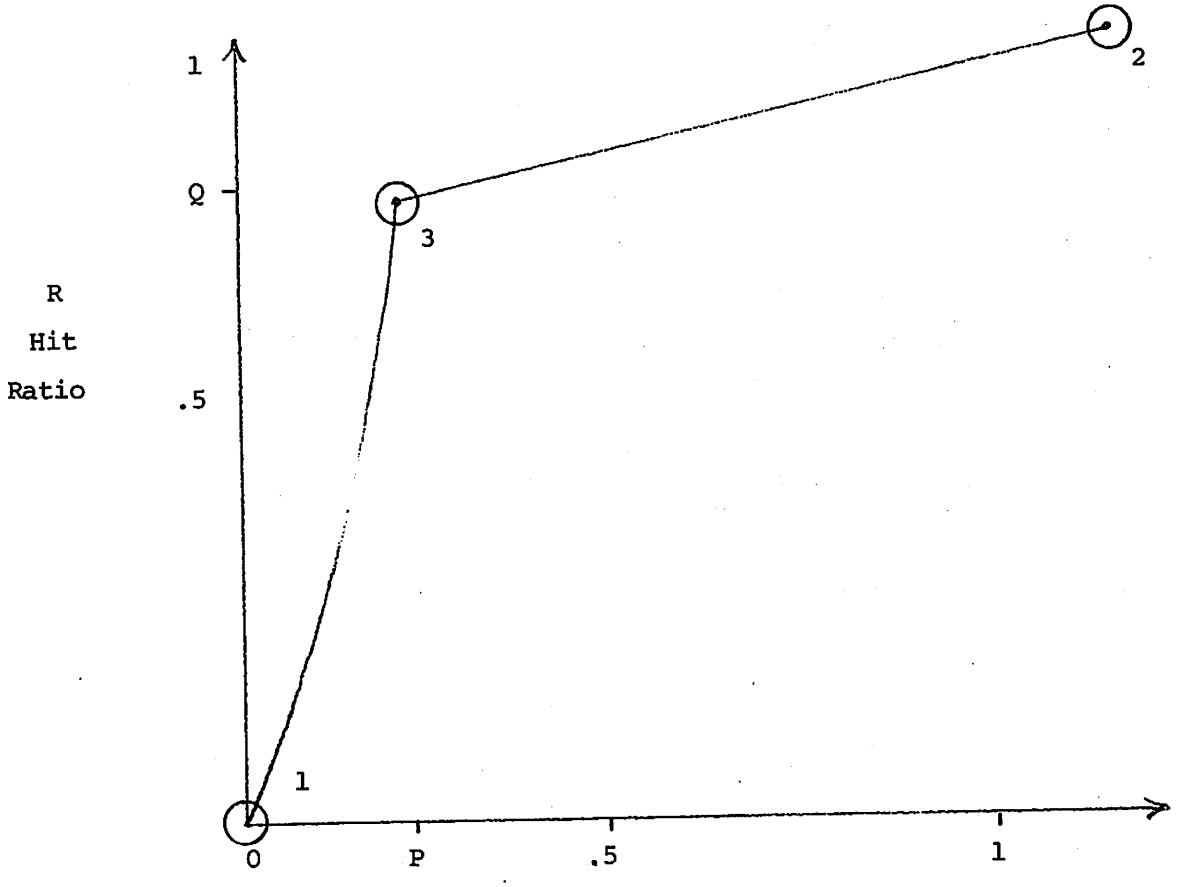
```
cost = 1 + X*Z
```

We now wish to choose a cache size, X, in order to minimize the following function:

```
function = cost*(delay)**Y
```

for some given Y.

In summary, Q and P are characteristics of the application, Z is a characteristic of CCD economics and Y is a characteristic of how important device speed is. The following results can be obtained.



$$R = \left(\frac{Q}{P^2}\right) X^2 \quad \text{for } X \leq P$$

$$R = \left(\frac{1-Q}{1-P}\right) X + \left(\frac{Q-P}{1-P}\right) \quad \text{for } X > P$$

Hit Ratio Curves

Figure 3

1) For P = .2 and Q = .8 (the 80-20 law) one should operate at point one or two in Figure 3 and never at point three for all choices of Y and Z. Hence, a CCD cache is never a winner.

2) For P = .1 and Q = .9 (90 percent of the accesses to 10 percent of the data base), there are choices of Y and Z which suggest operating at point three.

If Y = .5 one should operate at point three only if 3 < Z < 20. For Y = 1 the region is 15 < Z, < 70 and if Y = 2 the region is empty. The best gain possible is about 30 percent for the CCD cache.

In general, it does not look like a winner to have a CCD cache; at least not big enough to justify its cost and complexity.

3) Can a relation span multiple satellites?

It seems clear that graceful expansion requires that this be true. Hence, a decomposition algorithm along the lines suggested in [EPST78] is required. Moreover, system catalogs must contain the distribution criteria which controls how a relation is divided over multiple satellites. This subject is further discussed in [RIES78].

4) Placement of system catalogs

It appears possible to have the system catalogs for all satellites stored on ONE satellite. In addition, some catalog information must be stored at each site.

All accesses to the system catalogs are controlled through the host. Hence, it can directly access the "special" satellite. Moreover, the parser runs in the host so it can access the special satellite in a similar fashion. All relations which are created or destroyed are done by utilities in the host or by decomposition in the host. The host can therefore alert the special satellite of all such actions. Note that code in the satellites need NEVER access system catalogs at the "special" site.

5) Concurrency control

The standard issues concerning concurrency control are present here. Only three points make this architecture special.

a) It is possible (but not necessarily desirable) to totally resolve concurrency control in the host machine and have the satellites free of such code.

b) One variable commands can be run with no controls since they are physically serialized.

c) Any code necessary to resolve deadlock (if it is allowed to occur) can be run easily in the host. Hence, distributed deadlock detection is not required.

6) Crash recovery

Safety from crashes where data is irretrievably lost can be obtained in two ways.

a) Relations can be duplicated on more than one satellite. Decomposition can easily coordinate keeping all copies correctly updated. If a satellite crashes, then the system can continue if all data on that satellite is duplicated elsewhere. The only hitch is that the whole system dies if the host crashes.

b) a checkpoint/logging procedure can be used. The log can be written by decomposition and can reside on one of the satellites.

"Soft crashes" offer the routine headaches in this architecture and can be handled in the routine ways.

We conclude this section with three reasons why L3 may be a more attractive interface than L2 or L1.

a) It is reasonably justifiable to have L3 support a uniprogramming system in the satellite. However, L2 commands might take a VERY long time and better response could be obtained from preemptive scheduling rather than FIFO scheduling. This would only be possible if the satellite were multiprogrammed.

b) If L2 is used then messages must be sent from satellite to satellite (for example, to do logging). In addition, the system catalogs cannot easily be centralized on one satellite. As a result the overall system would become more complex.

c) L1 (and to a lesser extent L2) require a substantial operating system in the satellite. The notion of processes, a scheduler, interprocess communication, etc. must be present.

IV CONCLUSIONS

Our architecture is essentially a very intelligent channel, probably a microprocessor with 60-80 Kbytes of main memory. It would execute a very primitive operating system and the command processor for one variable interactions.

In general terms we expect that a system composed of a host and five satellites would be two to five times as fast as a host alone depending on the transaction traffic. Moreover, it would cost much less than twice as much. The limit on the number of satellites which could be connected would occur when the host CPU saturates. It is expected that a VAX host might support up to ten 11/34 satellites.

To get a better feel for the cost effectiveness of our scheme, we plan to build a prototype. The initial prototype will have a PDP-11/70 (or possibly a VAX) as a host and two PDP-11/34's (or 11/40's) as satellites. The initial plan will be to run a conventional UNIX in each satellite together with the one variable processor code. This appears to require little (if any) changes to the current software. Hence, the only tasks will be to modify the parser and decomposition. Once we have hardware in place, it should be only a few months worth of work to get the initial plan operational.

If the initial plan appears sound, we will then proceed in designing a "mini-mini-UNIX kernel" containing just the functions that we need and perhaps a customized access/method file system. Also, we would hope to investigate replacing the 11/34's by a microprocessor so we could optimize the

instruction set and discover exactly how economical our satellite can be.

REFERENCES

[CANA74] Canaday, R. H., et. al. "A Back End Computer for Data Base Management," CACM 17, 10, October, 1974.

[COPE73] Copeland, G. P. et. al. "The Architecture of CASSM: A Cellular System for Non-numeric Processing," Proc. First Annual Workshop on Computer Architecture, 1973.

[DEWI78] Dewitt, D. J., "DIRECT - A Multiprocessor Organization for Supporting Relational Data Base management Systems," Proc. Fifth Annual Symposium on Computer Architecture, 1978.

[EPST78] Epstein, R. S., et. al., "Distributed Query Processing in a Relational Data Base System," Proc. 1978 ACM-SIGMOD Conference on Management of Data, Austin, Texas, May, 1978.

[HELD75] Held, G. D. et. al., "INGRES - A Relational Data Base System," Proc. 1975 National Computer Conference, Anaheim, Ca., May 1975.

[LIN 76] Lin, C. S. et. al., "The Design of a Rotational Associative Memory for a Relational Data Base Management Application," TODS 1, 1, March 1976.

[LORI77] Lorie, R. A. and Wade, B. W., "The Compilation of a Very High Level Data Language," IBM Research, San Jose, Ca., RJ2008, May 1977.

[OZKA75] Ozkarahan, E. A. et. al., "RAP - An Associative Processor for Relational Data Bases," Proc. 1975 National Computer Conference, Anaheim, Ca., May 1975.

[OZKA77] Ozkarahan, E. A. and Sevcik, K. C., "Analysis of

Architectural Features for Enhancing the Performance of a Data Base Machine," TODS 2, 4, December 1977.

[RIES78] Ries, D. and Epstein, R., "Evaluation of Distribution Criteria for Distributed Data Base Systems," Electronics Research Laboratory, University of California, Berkeley, Ca., Memo #M78-22.

[ROTH77] Rothnie, J. B. and Goodman, N., "An Overview of the Preliminary Design of SDD-1: A System for Distributed Data Bases," Proc. 2nd Berkeley Workshop on Distributed Data Bases and Computer Networks, Berkeley, Ca., May 1977.

[STON76] Stonebraker, M. et. al., "The Design and Implementation of INGRES," TODS 2, 3, September 1976.

[STON77] Stonebraker, M. and Neuhold, E., "A Distributed Data Base Version of INGRES," Proc. 2nd Berkeley Workshop on Distributed Data Bases and Computer Networks, Berkeley, Ca., May 1976.

[WONG76] Wong, E. and Youssefi, K., "Decomposition - A Strategy for Query Processing," TODS 2, 3, September 1976.