

Design decisions for the intelligent database machine

by ROBERT EPSTEIN and PAULA HAWTHORN

Britton-Lee Inc.
 Albany, California

1. INTRODUCTION

The Intelligent Database Machine (IDM), manufactured by Britton-Lee Inc., is a back-end processor and storage system that contains a complete data management system. It includes specialized hardware to perform data management functions. It is our contention that the designers of a database machine must choose a focus for their machine which strongly influences other design decisions. The IDM is low cost, high performance machine designed to support "mid-range" users. This paper presents the reasons for this choice, and the resultant design issues.

The discussion is divided into four sections. In section two we describe our conception of the population of database users. We show that a single machine cannot solve all classes of user performance and cost problems. We identify the class of applications for which we focus the IDM. In section three we discuss the design trade-offs for that class of applications. The last section, section 4, is the conclusion.

2. TARGET USER POPULATION

There are four user attributes which affect the design of a database machine. These are:

(1) Required transaction rate

If an extremely high transaction rate is required, a high degree of parallelism and/or very fast storage (semiconductor memory, fixed head disks), must be incorporated into the design of the database machine.

(2) Storage requirements

Very large storage requirements preclude the use of expensive storage as the total storage required, and mandate a multi-level storage system.

(3) Wealth

The cost of the database machine is the major constraint in its design.

(4) Predictability of access to data

Is the data base generally accessed in a predictable manner? Certain applications naturally reference data by one or more keyed values, for example, part numbers, employee numbers, employee names, etc. In some applications, there may be many possible access patterns, for example, census data and other statistical

applications. Database machines can be designed toward the expected usage pattern.

Using these four attributes we shall attempt to define general user populations. Let us define four general user categories: the small business system, the scientific and medium business system, the large business system, and the special purpose system. Figure 1 shows the transaction rates and storage requirements for these categories of users. The transaction rate is on the horizontal axis, and the storage requirements on the vertical axis. A small business system has relatively few users and correspondingly a low transaction rate. The storage requirements for the small business system are also relatively small.

The transaction rate and storage requirements increase proportionately for the medium-scale and scientific systems, the large business systems, and, finally, the special purpose system, where the database is huge and/or the necessary transaction rate very high. Of course, there will be exceptions to this conceptual graph: there are systems that are small and require very fast access, or large and have relatively low transaction rates. However, most systems appear to fall in the general categories marked on the graph.

The cost of a system must be related to the performance needed by the end user. Those who require high-performance, large storage systems can usually pay premium prices for them; the small to medium scale users are usually the

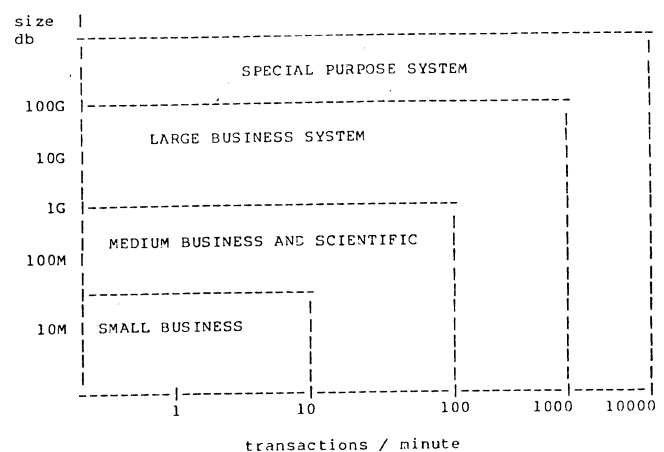


Figure 1—User classification.

ones who do not need, and do not wish to pay for, extremely high performance.

Unlike attributes (1) through (3), predictability of access cannot be correlated with the size of the database. It is an attribute of many general purpose business database applications. A typical example is a business system, where there is an employee file. If the file contains employee name, address, phone number, employee number, etc., it is very likely that it will be accessed by specifying a name or employee number, and very unlikely that it will be accessed by specifying a phone number. This means it is reasonable to expect a request such as "What is Smith's phone number?" and not "Who has phone number 527-7646?"

We shall now show that the choice of target user population dictates the design of the machine.

2.1. Possible decisions

If a database machine is designed for users who need extremely high transaction rates it must include either very fast storage (ram or fixed head disks), as in DIRECT [DEWI79], or it must include a mechanism to parallel search the disk, as in CASSM [LIPO78]. On the other hand, if the focus of the machine is toward the user of very small databases, under 1M on our graph, and a low transaction rate, a stand-alone microprocessor system might suffice. The focus of the machine dictates the design of the machine because each increase in performance (either storage or transaction rate) results in an increase in price. As the price goes up, the potential market changes.

2.2. The IDM decision

It was decided to focus the IDM toward the mid-scale user. By "mid-scale" we mean users who require transaction rates in the range of 100 to 1000 per minute (possibly higher in certain applications). We further expect mid-scale users to have applications where the majority of transactions have predictable access patterns. This decision was made because large and special users are well served by other database machines being developed, such as DBC [BAUM76], MUFFIN [STON79] and DIRECT [DEWI79]. Small to medium scale systems are currently not served well at all: data management systems, because of their necessary complexity, are expensive and do not perform well on small to medium scale computer systems. As a result, general data management systems are often not used by this user group, and special purpose in-house systems are often developed. A family of database machine products can be developed which provide mid-scale performance and can be trimmed down toward the small user or expanded upwards toward the larger user.

In choosing this market, it is essential to have a good price/performance ratio and be as host independent as possible, since there are many different types of host machines in use by mid-scale users and even more diversity among the small business systems users. The decision to focus the design

toward the mid-scale user dictates various design choices, which are discussed in the following section. The IDM can accommodate moderately large or small users. It is designed to store databases up to 32 Gigabytes. It should achieve transaction rates up to 2000/minute in certain applications but also be capable of being scaled down for users who only need 100 transactions/minute.

3. DESIGN TRADE-OFFS

Having chosen to build a database machine emphasizing a low cost/performance ratio, there are a number of fundamental design trade-offs which must be considered. We will explain each one and discuss how the choice of user application influences the decision.

3.1. Functionality of the back-end machine

The first decision is what functionality to provide in the database machine. The degree of host independence is closely tied to the level of functionality provided in the back-end machine. The performance improvement derived from using a database machine is related to how much work can be off-loaded to the back-end. Figure 2 shows possible levels of service that a back-end database machine can provide. These are:

- 1) User-programmable, cached controller.
Allows the user a programming environment in the disk controller where (s)he can control the on-disk processing. Caching can be used to read a track at a time and other such techniques can be used to reduce the apparent access times.
- 2) Search unstructured files for values requested by the front-end.
This is a simple search engine, and is most suited to applications with data that cannot be structured (such as text processing).
- 3) A record management system.
This would provide an interface at the record level such as insert record, get next record, etc.
- 4) Provide a basic relational data-management system,

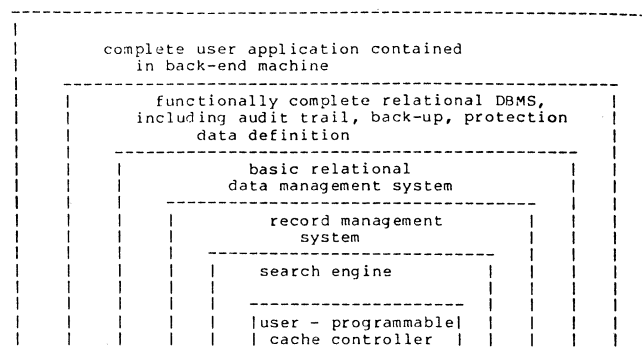


Figure 2—Evolution of back-end intelligence.

with search and update capabilities, interfaced through a high level language.

Host communication time and software support is minimized by such an approach. The database machine can be used most effectively by fully containing a basic database system within it.

- 5) Provide audit trails, back-up and recovery facilities. If a data management system is to be contained in the back-end, it should maintain its own back-up and recovery system; otherwise, the host will have communication difficulties knowing what the system is doing, and when.
- 6) Provide protection and data definition facilities. Logically, this can be done in the host or in the database machine; doing it in the database machine provides a mechanism that the database machine can use to optimize performance.
- 7) Full end user support. If the database machine is to provide a complete data management system, why not make it directly control the terminals and run the application programs, thus eliminating the need for a host?

Figure 2 represents the evolution of increasing back-end intelligence from a disk controller to a back-end machine. Each of the steps makes sense but the crucial issue is which are cost effective and how much do they affect the host system. Until one reaches a level where the back-end is functionally independent from the host, many portions of the host are closely tied to the back-end. Another consideration is that a database machine is a piece of hardware replacing what has traditionally been software. It is not readily subject to user modification. Issues which are closely tied to the end user should be solved in the host since they are likely to require local customization.

A final consideration, and one that cannot be over-emphasized, is that the amount of work which the back-end offloads from the host should be significantly greater than the amount of work needed to communicate with the back-end.

The amount of work done by the back-end in 1, 2, and 3 is not large compared with the overhead of interacting with the back-end. Also, at such a low level, the host processor system is tightly coupled to the low level implementation decisions of the back-end. For these reasons we feel the first significant performance improvement occurs when the host deals with the back-end at a high level (case 4). Case 4 is a database management system with a high level interface. The decision to make the system relational was never questioned.

Providing the additional functionality of 5 and 6 increases the complexity of the database machine but greatly simplifies the tasks done in the host.

The final step, doing everything in the back-end, is counter productive. It casts in concrete parts of the end user interface and furthermore, the back-end processor has no performance advantage for running application programs over any number of very good host processors available today. The database machine's hardware performs extremely well for database

management but offers little to the general programming environment. Running application programs requires a different architecture than running a dedicated database management system.

3.2. Storage medium

In order to keep the cost/performance ratio low, providing on line storage in the range of 8 Megabyte to 32 Gigabyte requires the use of moving head disks. They currently are the only non-volatile storage medium with a price/performance suitable for the target market place. The end user costs for standard, moving head disks are \$60 to \$120 per megabyte for large systems (over 100M) and \$100 to \$200 per megabyte for small systems. These costs have been dropping and research has shown that they will continue to drop for some time to come.

3.3. Search mechanism

A fundamental part of database management involves searching through objects in the database. The choices for search mechanism can be broken into two categories: complete scanning of the object, or indexing/ hashing techniques (which we shall call access methods for the purpose of this discussion). Access methods limit a search based on certain predefined keys. Some commonly used access methods in modern database management systems are B-trees [BAYE70], ISAM [IBM66], and hashing [KNUT73]. In contrast, a complete scan takes any search keys and linearly searches. On traditional systems such a search will take an inordinate amount of time and is impractical except in very small databases. Special database machines, however, can be designed to perform multiple linear searches in parallel. For moving head disks, this requires having multiple heads active at the same time. Each head needs search logic sufficient to search at the transfer rate of the disk. A number of database machines, such as RAP [OZKA78], have been proposed whose speed depends on having many search elements.

The basic trade-off is to compare access methods against multiple search paths, based on expected access patterns and cost. With current technology, the cost of one search mechanism which can perform at disk speeds is substantially less than the cost of multiple search paths employing multiple active heads. Multiple search paths may work for the upper end of the cost/performance curve but they are unaffordable at the other end. Access methods are clearly cheaper; the next step is to determine how they can be expected to perform compared to multiple search paths. Studies have shown that access methods out-perform multiple search path systems on transactions which have a predictable access pattern [HAWT79].

One final trade-off to consider is the impact on the end user. Multiple search paths relieve the Database Administrator from having to determine the physical structure of the database objects, that is, what access methods should be used on what fields. In some cases this is a simple task (for

example employee numbers and employee name will be commonly accessed), but there are other cases when the optimal choice of access methods is extremely difficult to determine. We have opted to leave this burden with the end user in exchange for a system which will be much lower in cost and will have a potentially high performance.

In summary, the use of access methods fits in well with databases which have predictable access patterns. To the degree that this is true, an access method database machine will have a much better cost/performance ratio than a multiple search path machine.

3.4. Low cost processors

Having decided to use standard moving head disks and access methods, we need processing elements which have comparable speeds. Low cost implies the use of microprocessors but no existing microprocessors have a speed which is satisfactorily matched to the database application. Existing database management systems are frequently execution bound even on fast mainframes. Moving a DBMS onto one of today's 16 bit microprocessors makes economic sense but does not provide reasonable performance for the class of users we have identified.

The trade-off we pursued was to achieve processing rates at least one order magnitude greater than those of microprocessors but at only a modest increase in price. The folklore in processing costs, Grosch's Law [GROS75, CALE79], dictates that this is impossible in the general case. However, if one chooses to do a specific task, not a general one, some surprising results come out. It is possible to structure a DBMS such that an enormous percentage of its time is spent in a small (under 4K) amount of its code. This code is simple in function and specific in nature due to the fact that it deals only with the issues of one task, database management.

By building a 10 MIPS, pipeline machine from standard Shottky TTL logic and microcoding a well defined collection of subroutines, we are able to meet a high performance at a modest increase in price. The special purpose "Database Accelerator" board costs approximately two times the cost of a microprocessor board but in our case has a 30 times increase in performance for the set of code it is intended to perform. Such trade-offs are inherent in special purpose architectures such as database machines.

The high speed of the Database Accelerator enables the IDM to process data as it is coming off the disk. For operations which require disk access, the Database Accelerator will appear to process data with zero real time cost. The operation is limited by the speed of the disk. When only a small transaction rate is required, the Database Accelerator can be removed, reducing the cost of the machine.

3.5. Use of cache memory

If we want a machine that can run faster than disk speed, it is possible to implement a disk cache using random access memory. This will improve performance only if a disk page

is referenced more than once in a reasonably small period of time. The trade-off is the cost of the cache versus the relative performance improvement. The performance is completely dependent on the amount of "rereferencing." This in turn is highly dependent on the relative ratio of the cache size to the data storage size.

In general most database applications show very little rereferencing. There are, however, certain exceptions [HAWT79]. These include references to the upper levels of index pages, references to system catalog pages (audit trail, locking, data dictionary, etc.) and rereferencing does occur in more complicated user queries. The optimal amount of cache is therefore strongly application dependent. The low end IDM provides a minimum amount of cache (approximately 32K). Additional cache can be included as a user option. Those applications which have a very high transaction rate on a modest size database can incorporate sufficient cache to speed the database machine to near main memory speeds. This ability is consistent with the desire to serve a large range of speed requirements.

3.6. Single/multi-thread

Both the host independence and the performance of the database machine are affected by the decision of whether to multi- or single-thread the database transactions. A database machine has the choice of either sequentially executing one transaction at a time or accepting multiple transactions and scheduling their execution in a manner similar to what is done in time sharing systems.

To provide an intuitive example, we examine an analogy found in traditional disk controllers. Ignoring the issue of overlapped seeks, a controller takes one operation (read a sector) and does not accept another until the current operation is complete. To enhance performance an operating system will schedule the next disk request from a queue of requests by some strategy which tends to minimize overall seek time. If the selection strategy were moved into the disk controller, it would then appear to be multi-threaded. Disk requests could be made by the host at any time. When an operation was complete, the disk controller would need to tell the host processor not only that an operation was complete, but what operation was completed. The trade-off for doing scheduling in the disk controller is minimal. It would move a small amount of overhead out of the host. It would also allow global scheduling when multiple hosts are connected to one controller.

For a database machine, the trade-offs between single and multi-threading are much more significant. To begin with, how would an operating system decide how to schedule a database transaction? It would require an enormous amount of information about what the transaction does. This amount of information is contrary to the one of the goals of back-end processing, e.g. independence between devices.

The range of times it takes to process a database transaction varies enormously. A typical transaction may require only a few 10's of milliseconds, but other transactions can take minutes or hours.

To allow a mixture of different transactions of differing amounts of work, it is necessary to be multi-threaded and also allow preemption of a transaction. The cost to do this includes adding scheduling code, and room for swapping different transactions in and out of execution. This requires a significant amount of logic in the database machine. Many of the proposed database machines are single-threaded or depend on a cooperating program in the host to coordinate rescheduling.

The decision for the IDM is to provide a multi-threaded environment. This gives a very high degree of independence from the host operating system. Additionally, the problem of where to store transactions which are waiting for resources is solved by "stealing" memory from the cache. At any point in time a percentage of the cache memory is dedicated to transactions and to caching disk activity. The percentage is allowed to change dynamically. For example, at a particular point in time there may be only one transaction running and it may have nearly all the cache memory to use for processing. Similarly, if there are many, short transactions, the cache will be allocated mostly for transaction swapping. A heuristic algorithm is used to determine how best to utilize the cache resource. This fine degree of control is best done in the database machine. The host operating system has very little to do except pass program requests to the IDM and pass results back to the program. In DMA interfaces, this overhead is trivial, making the driver program in the host operating system very simple.

4. CONCLUSIONS

In this paper we have examined a number of the fundamental design decisions in designing database machines. We began by identifying the class of applications which the database machine was to address, namely, the mid-range user who has applications with predictable access patterns. To build a machine with a good cost/performance range over a wide range of storage requirements requires moving head

disks and access method search techniques. To achieve high execution speed, special purpose hardware (the Database Accelerator) is used. There are specific applications where a variable size cache can greatly improve performance. Finally we have illustrated why a database machine must be multi-threaded and the rationale behind making the database machine support a functionally complete, relational database management system.

BIBLIOGRAPHY

- [BAUM76] Baum, Richard I., Hsiao, David K., and Kannan, Krishnatmuti, "The Architecture of a Database Computer - Part I: Concepts and Capabilities," Technical Report OSU-CISRC-TR-76-1, Computer and Information Research Center, The Ohio State University, Columbus, Ohio (National Technical Information Service Number AD-A034 154).
- [BAYE70] Bayer, R. and McCreight, E., "Organization and Maintenance of Large Ordered Indices," *Proc. 1970 ACM-SIGFIDET Workshop on Data Description, Access, and Control*, Houston, Texas, Nov. 1970.
- [CALE79] Cale, E. G., et al., "Price/Performance Patterns of U.S. Computer Systems," *CACM*, Volume 22, Number 4, April 1979.
- [DEWI79] Dewitt, D. J., "Query Execution in DIRECT," *Proceedings, SIGMOD International Conference on the Management of Data*, 1979, pp. 13-22.
- [GROS75] Grosch, H. A., "Grosch's law revisited," *Computerworld* 8, 16, April 16, 1975.
- [HAWT79] Hawthorn, Paula, "Evaluation and Enhancement of the Performance of Relational Database Management Systems," Electronics Research Laboratory, University of California at Berkeley, Berkeley, Ca., Memo Number M79-70, December, 1979.
- [IBM66] IBM Corporation, "OS ISAM Logic," IBM, White Plains, N.Y., 1966, GY28-6618.
- [KNUT73] Knuth, D. *The Art of Computer Programming*, Vol. 3, Addison-Wesley, Reading, Mass., 1973.
- [LIPO78] Lipovski, G. J., "Architectural Features of CASSM: a Context Addressed Segment Sequential Memory," *Proceedings, Fifth Annual IEEE Symposium on Computer Architecture*, April, 1978.
- [OZKA78] Ozarahan, E. A., Schuster, S. A., and Sevcik, K. C., "Performance Evaluation of a Relational Associative Processor," *AFIPS Conference Proceedings*, Vol. 44, 1975, pp. 379-388.
- [STON79] Stonebraker, Michael, "MUFFIN: A Distributed Database Machine," Electronics Research Laboratory, University of California at Berkeley, Berkeley, Ca., Memo Number M79-28, May 1, 1979.