Cost-effective ways of improving database computer performance

by DAVID K. HSIAO

Naval Postgraduate School
Monterey, California

ABSTRACT

In this paper the hardware features that characterize the performance bottlenecks of conventional database computers are identified. Motivations for and proposals of new architectures for overcoming the bottlenecks for future database computers are given.
BACKGROUND

According to the 1974 publication on its database computer as a backend of the mainframe host computer for database management, XDMS was aimed to provide the following:

1. Cost-saving and performance gain through dedicated software and backend hardware
2. Shared databases
3. Centralized protection
4. Ease in software development on a standalone backend and for the backend

The last three aims had largely been achieved. For example, XDMS was able to communicate not only with the original host, a Univac 1108, but also with a new host, an IBM 360. In other words, XDMS allowed two host computers (more specifically, their respective application programs) to access the same database, even though the application programs running on the one host cannot be run on the other host. Obviously, XDMS was provided with multitasking and multirequest support and update locks. By allowing only the backend to control, manage, and access the database store, namely the disks, XDMS achieved the aim of centralized protection. For the first time, the disks of the database could be physically protected from the disks of the mainframe. The only possible way of compromising the protected database was by way of the “front door,” through the host-backend communications. Data security employing encryption could ease such “frontal” attack. Finally, it was obvious for the developer that software development could always be accomplished readily if the machine for which the software was aimed was present and if a new machine could be used for software development. What was not clear at the time was the success of achieving the first aim—the cost saving and performance gain through dedicated software and hardware.

Cost saving was arguable. The software development and purchase cost had been shown to be, at worst, comparable to the software development and purchase cost of the database management system (DBMS) for the mainframe. The hardware cost of the backend had not been higher than the hardware cost of mainframe upgrade for the DBMS. For example, the original backend of XDMS was a 16-bit minicomputer known as Meta-4, which cost $60K in 1974. Since the cost of minis is rapidly coming down, the same minicomputer should cost less than $30K now. In fact, it would be cheaper than a disk controller. Disks for the database were needed whether the backend was utilized or not. On the other hand, if the backend were not employed, the DBMS acutely needed the addition of main memory, processing power, and channel capacity for its support in the mainframe. Thus, the cost saving by way of the database management backend was indeed arguable.

The major disappointment was in performance gain. Performance may be measured in terms of the following: (a) the communication and transmission times between the backend and its host, and (b) the transaction execution time. XDMS was reported to have very low degradation due to communication and transmission. It was also reported to have good performance for complex queries. The latter report was difficult to justify in view of the small size (i.e., 1,500 records) of the database involved. Most database system designers knew that the time complexity of certain queries (e.g., relational joins) was proportional to the size of the database involved (e.g., cardinality of the relations involved). The time complexity of a query was a determining factor of the query execution time. Thus, the larger the database there was, the longer the transaction time would be. Perhaps the argument on performance gain was made in terms of the relative performance of the transaction execution time; that is, given the same set of transactions, the transaction set was first executed in a host without a backend and then executed in the backend. By comparing their response times (i.e. the communication and transmission time, plus the transaction execution time) relative to each other, we could then conclude one's gain over the other. Subsequent experiments with XDMS-like backends yielded no appreciable performance gain over the conventional mainframe-oriented ones. This lack of performance gain was due to two factors: First, it was difficult, if not impossible, to come up with software design that ran appreciably faster in the backend but not in the mainframe. Second, the backend, as a general-purpose minicomputer, was inherently slower in speed and meager in resources than the well-endowed host. Thus, database management software did not necessarily run faster on the backend.

Others attempted to introduce, instead of a dedicated mini, specially configured hardware to speed up the transaction execution time. These new backends, such as Britten-Lee's IDM 500 and Intel's iDBP are mostly microprocessor-based hardware with a generous use of random-access memories. However, even with these attempts, the performance gain may still be elusive. Let us examine the architecture of current database computers and identify their performance bottlenecks.

WHERE ARE THE PERFORMANCE BOTTLENECKS?

Consider the architecture of a backend computer depicted in Figure 1, where conventional disks are used for the database
It is important to note that everything is hung on the high-speed bus. In other words, for the data coming off the disk and going into the random-access memories, coming off the memories and going toward the hosts or terminals, or being accessed by the major database processor and other minor database processors, the high-speed bus is the only throughway for the data movement and access. Consequently, the performance of the backend cannot exceed the capacity of the high-speed bus. Presently, a typical high-speed bus has a transfer rate of 20 to 320 megabits/sec.

For a database computer with four disk controllers, each of which has four disk drives, assuming that the disk is of medium capacity, 300 megabytes/drive, we have the following calculations:

\[
4 \times (4 \text{ drives}) \times 300 \times 10^6 \text{ bytes/drive} = 48 \times 10^9 \text{ bytes of the database}
\]

\[
= 384 \times 10^8 \text{ bits,}
\]

\[
384 \times 10^8 / (20 \times 10^6) = 1,920 \text{ seconds to read out the entire database}
\]

\[
= 32 \text{ minutes.}
\]

\[
384 \times 10^8 / (320 \times 10^6) = 120 \text{ seconds} = 2 \text{ minutes.}
\]

This indicates to us that for text search and retrieval, an important application of database management, it takes at least 2 to 32 minutes to search and retrieve the entire textual database. Even if the search and retrieval is restricted to a fraction—say a quarter—of the database, it will require at least one-half minute to 8 minutes. Consequently, conventional database computers are not suitable for text search and retrieval.

For formatted database management, where indices are used and accesses to the database are more selective, we do not use disk drives as the basis for calculation. Instead, we consider that the physical records correspond to disk tracks, which are the units of data access and transfer. Assume that a track is of 24 Kbytes. We then have the following:

\[
(20 \times 10^6 / 8) / 10^3 = 2,500 \text{ byte/msec (the bus capacity),}
\]

\[
24K / 2.5K = 10 \text{ msec}
\]

to place a track of data in the main memories. Similarly, at the higher rate of 320 × 10^6 bits/sec, we need .6 msec to place a track of data in the main memories. We assume the optimal situation, that only one track of data is needed from each disk drive and that all the track seeks have been overlapped. Thus, all 16 disk tracks coming from 16 separate drives can be read back-to-back at the maximal bus rate. We then need

\[
16 \times 10 = 160 \text{ msec and } 16 \times .6 = 9.6 \text{ msec.}
\]

Since, for formatted databases, it uses indices and auxiliary information to select data tracks, the database computer must access the indices and auxiliary information that are also stored on the disks. Thus we need another 9.6 to 160 msec. Altogether, then, we need a minimum of 9.6 to 160 msec to get a unit of data into main memories for processing. Assuming that the CPUs, with cycles ranging from 200 nsec to 1 μsec, of most of the 16-bit microprocessors and minicomputers can keep up with the incoming data by executing short system programs, we quickly see that to improve the performance we must improve the transfer rate of the high-speed bus, that is reduce the time needed to transfer a unit of data.

Although a higher speed (say, beyond 320 megabits/sec) of the bus is attainable at higher cost, the gain in bus speed is nevertheless offset by the higher capacity of the disk (say, of 1.25 gigabytes/drive, which would make a database of 20 gigabytes a reality). Larger-capacity disks imply larger databases; in turn, these imply wider distributions of related data on the disks, which in turn imply more data pages needed in the main memories. Consequently, the higher-speed bus is used for disks of ever larger capacity. With these new figures and assumptions, we may go through the same calculation again. We then discover that a backend is required to pay a fixed "cost" (of approximately 9.6 to 160 msec) for a unit of data management and data transfer, despite the large (i.e., gigabyte) capacity of the disks and the high cost of the faster bus used for the backend. In fact, the use of such an expensive bus with large disks will not show any performance gain over the use of a less-expensive bus with medium disks. By now, we learn that our earlier conclusion is not valid. The cost-effective way of improving the database computer performance does not lie in the expensive higher-speed bus, since larger-capacity disks for very large databases will offset any possible reduction of the fixed overhead in data management and transfer. The question is therefore whether or not we can improve the database computer performance, despite the presence of the fixed overhead and the lack of any prospect of taking advantage of the advancement of bus and disk technology.

ARCHITECTURAL SOLUTIONS FOR IMPROVING THE PERFORMANCE

There are two proposals. Although they are different in their architectural approaches and technological choices, these pro-
Proposals use the same principle. The principle is sharing the same fixed cost (overhead) by using a multiplicity of identical hardware and by processing the data on the identical hardware concurrently.

**Database Stores with Built-in Parallel Processing Logic**

Consider our first proposal, which is depicted in Figure 2. In this proposal, the moving-head disks are modified so that they can perform parallel read-out and write-in operations. Such technology was reported as early as 1978. The disk controller is also modified so that there is a processing unit for each data stream coming from the track. Therefore, there are as many processing units in the controller as there are tracks in a cylinder. Assuming 20 tracks per cylinder, we need only 20 processing units in a controller. Each one of the 20 processing units has an identical microprocessor-based architecture with considerable use of random-access or shift-register-like memories. Each unit is essentially a major database processor of the sort depicted in Figure 1. The differences between Figure 1 and Figure 2 are that

1. The same major database processor is multiplied 20 times in the new disk controllers.
2. Different processing units process their own data streams coming from different tracks.

The first difference has been to some extent overcome, since the manufacturer of the controllers has already incorporated processing logic for defect detecting and error decoding into the controller, as well as logic for executing the software online I/O routines (known as access methods). There is no reason why the disk manufacturers cannot make the controller even more intelligent. The second difference can be overcome since we can achieve a microprocessor-based architecture where single-instruction-and-multiple-data-stream (SIMD) and multiple-instruction-and-multiple-data-stream (MIMD) modes of database management become a reality. Thus, all the data streams “share” the same fixed overhead; meanwhile, 20 times as much data access, transfer and processing may be accomplished.

This approach to reducing overhead and improving performance is cost-effective because the disk technology for parallel read-out and write-in is here, the controller technology for built-in logic is also here, and the addition of identical hardware is proportional to the performance gains. We observe that in multiplying the hardware in the controller we are asking not for ever-faster buses, but for multiplication of the existing bus and processor structure. This proposal has been thoroughly analyzed and studied.

**Software Multibackends**

Consider our second proposal, which is depicted in Figure 3. This proposal is aimed at addressing the following question. Is it possible to use a multiplicity of minicomputers and their disk systems, unconventional hardware configuration, and innovative software design for achieving improvements in throughput and response-time for large and growing databases over what conventional and single-backend database management can provide?

---

From the collection of the Computer History Museum (www.computerhistory.org)
Measures of a good multibackend database system are that
1. The throughput improvement is proportional to the multiplicity of the backends.
2. The response time is inversely proportional to the multiplicity of backends.
3. The system is extensible for capacity growth and/or performance improvement.

The cost-effective ways of extending the software and hardware of the multibackend system with one controller (i.e., master) and several backends (i.e., slaves) are to
1. Allow the addition of more backends of the same type, instead of the replacement of the present backends with more powerful models
2. Require identical software in each of the backends and replicate the existing software on new backends
3. Minimize the role of the controller of the backends

We note that the addition of the same type of minis and disk systems, our first way, will incur few system interruptions. Intuitively, for the same database we double the number of minis if we want to double the performance gain. For a growing database, say, that will double the current database size, we would double the hardware to maintain the present performance. Our second way of extending the system, software replication, can be easily accomplished on the new hardware by doing a system-generation (i.e., SYSGEN); furthermore, the increase of software replication does not imply the increase of software complexity, since all the software replicated in the slaves is identical. Finally, the master should not become the bottleneck of the new and existing slaves, else there will be little extension of the system capacity and little improvement in system performance. To keep the master from becoming a potential bottleneck, we require that the controller (i.e., master) perform minimal (yet necessary) work. Presently, a software multibackend prototype is being implemented. Projected performance gain (or loss) will soon be validated (or invalidated).

CONCLUDING REMARKS

In this short paper, we attempt to show that cost-effective ways of improving database computer performance do not lie in the single backend system. Instead, the ways may be found either in multibackend systems or in disk systems with parallel read/write and processing capability. Either way is within the state of the art of hardware and software technologies. What we must do is try these cost-effective ways.

ACKNOWLEDGMENTS

The work reported herein is supported by the Office of Naval Research under Contract N00014-75-C-0573 and, in part, by an equipment grant from the Digital Equipment Corporation.

REFERENCES