TANDEMCOMPUTERS

Relational Data Base Management for On-Line Transaction Processing

Stewart A. Schuster

Technical Report 81.5
February 1981
PN87604

Relational Data Base Management for On-Line Transaction Processing

Stewart A. Schuster

February 1981

Tandem Technical Report 81.5

Tandem Tr 81.5

Relational Data Base Management For On-Line Transaction Processing

Stewart A. Schuster
Tandem Computers Incorporated
19333 Vallco Parkway, Cupertino, CA. 95014

February 1981

ABSTRACT: This article outlines the requirements of on-line transaction processing and indicates how relational data base technology can be applied and implemented to meet those requirements. A major problem with many experimental or early commercially developed relational systems was their performance in a transaction processing environment. A second purpose of this article, therefore, is to explore organizational and architectural techniques for achieving a functionally complete, high-performance relational data-base system.

TABLE OF CONTENTS

INTRODUCTION

It has been ten years since E.F. Codd's landmark technical paper, "A Relational Model of Data for Large Shared Data Banks"[1] — the first milestone in the development of a formal data base management technology based on the relational model of data. Since that time, the appeal of the relational model has grown steadily, largely because systems implemented around it have met so many of the requirements of large-scale data base management.

Reviewing the progress of relational data base management in 1975, Codd outlined some of the problems that led to its first early developments:

> In 1968, it was possible to observe two mutually incompatible trends in formatted data base systems: On the one hand, the tendency of systems designers to expose users of their systems to more and more complicated types of data structure and, on the other hand, the increasing interest in establishing integrated data bases with a much higher degree of data inter-relatedness and on-line interactive use by non-programmers. At about the same time, it was becoming clear that users were spending too much in manpower and money on recoding and retesting applications programs which had previously worked satisfactorily but which had become logically impaired by data base growth or by changes in the stored data representation for various reasons (the so-called data independence problem) [2].

In recent years, growth and change in data bases has nowhere been greater than in on-line transaction processing systems, which involve very large data bases, large volumes of daily on-line updates, and extensive terminal-handling facilities. These systems have forced designers to adopt the best engineering principles in putting together systems that are simple, reliable, and productive. Since these principles mesh closely with those underlying relational data base processing, it is likely that data processing managers will soon face an increasingly large array of relational or "near-relational" systems and products.

This article outlines the requirements of on-line transaction processing and indicates how relational data base technology can be applied and implemented to meet those requirements. A major problem with many experimental or early commercially developed relational systems was their performance in a transaction processing environment. A second purpose of this article, therefore, is to explore organizational and architectural techniques for achieving a functionally complete, high-performance relational data-base system and to identify systems which are utilizing these techniques.

A SYSTEM TAXONOMY

Of four basic ways in which data processing systems are used, on-line transaction processing is the fastest growing. The other three are batch processing, timesharing, and real-time processing. These four modes of operation are distinguished from one another by the number of users each supports, whether the users are supported simultaneously, whether large data bases are updated on-line, and by the length of time it takes the system to perform an identifiable job on a user's behalf.

Batch Processing

Batch processing views the computation task as data acquisition, then data processing and finally data reporting. Each of these three steps is separated in time and function rather than being integrated. Historically, batch systems were the first to be developed. Because early computing resources were scarce and expensive, batch systems were designed to do as much as possible in the shortest possible time. With batch systems, turnaround time is typically measured in hours. If a job involves access to a data base, that data base is normally loaded into storage for that job, then unloaded when the job is completed.

Timesharing

Timesharing systems grew out of the need to provide programmers and other computer users with more direct access to their machines. The objective of a timesharing system is to support many users interacting with the system simultaneously via terminals, giving each user's program a "time slice" of several milliseconds in which that program has exclusive use of the central processor. The users—who may number anywhere from a few to 100 or more—may be working independently of one another or they may be working on a common task, such as the programming of another system.

The maximum number of timesharing users who can be supported with reasonable response time is usually less than 100, even on the largest systems. The system's response time for an individual user is highly variable, depending as it does on the number of other users, the nature of the work they are doing, and how the system allocates to user programs. Data base sharing is rarely a problem. Timesharing systems are predominantly used for program development or small applications.

Much of the need for timesharing is being eliminated by the decline in hardware costs and the consequent emergence of personal computers.

Real Time

Real-time systems attempt to guarantee an appropriate response to a stimulus or request quickly enough to affect the condition that caused the stimulus. In process-control environments, where real-time systems are most frequently used to monitor and control highly automated chemical or manufacturing processes, the needed response time is measured in milliseconds. To provide this kind of response time, most requests are kept simple, e.g., "turn this switch on," and the operating systems provide only very basic task management and input/output services, with a minimum of system overhead. Most real-time systems lack any data base management facilities.

On-Line Transaction Processing

On-line transaction processing systems are similar to real-time systems insofar as their goal is to provide a predictable response to a predictable request within an acceptable time. Here, the response time need not be as short as it must be for real-time systems, but it must be shorter and less variable than the response time of a timesharing system. The reason: the user's (or "operator's") interaction with the computer is normally part of a larger transaction encompassing another person for whom the operator is performing a service, such as making an airline reservation or a money transfer — or in some cases the user is a customer (e.g., automated bank tellers). Response time and throughput as measures of system performance are the primary targets of optimization in transaction systems. They are sought even at the expense of holding some computer resources in reserve for use during periods of heavy loading or in the event of a system malfunction.

On-line transaction processing applications in such areas as finance, insurance, manufacturing, distribution, and retailing are typically very large by almost any measure: size of data bases, number of users, number of data base updates, complexity and volume of processing, even geographical extent (some systems are distributed over international networks). Emerging applications of transaction processing systems are electronic mail and office automation.

Many of these applications are critical to the ongoing, daily operations of financial and commercial enterprises. They must therefore be capable of high performance (speed, accuracy) as well as be implemented in a way that allows modification and modular growth without interruption of service.

NECESSARY FEATURES OF ON-LINE TRANSACTION PROCESSING SYSTEMS

A number of other characteristics distinguish on-line transaction processing systems and impose on them a broad and stringent set of demands. One is that the users may range from highly skilled data processing professionals to management or administrative people who are relatively unsophisticated in the use of computers but whose demands on the system are nevertheless important and must be met. Another is that the number of users and the number of applications are likely to grow. Taken together, these characteristics dictate that a transaction system support interaction with data bases at several levels of language sophistication and that the system be designed in a way that makes growth and modification easy.

The key components in a transaction processing system are the data base and the data communications facilities. Virtually every transaction involves one or more accesses to the files of a data base, either to retrieve information or to enter new facts into existing files. The availability and reliability of the data base are critical in a transaction system—along with growth, ease of use, and performance, they are the primary determinants of system quality. Also, the support of a large number and variety of terminal types is mandatory. The trend to computer networking and decentralization requires that data base management systems support distributed data bases.

Modular Growth

Experience has shown that once an application goes "on line," the transaction load often increases to fill the capacity of the original system, requiring an upgrade to a higher-performance computer or expansion through distributed processing via a network of systems. It is very difficult to accurately size systems for transaction processing since there are many variables involved. Also, successful systems tend to grow rapidly after initial installation. Hardware and software architectures that facilitate this expansion are required. Often, it must be possible to upgrade and maintain systems without taking them off-line.

Continuous Availability

An on-line system must be continuously available during the hours in which transactions are generated by users. Any failures that bring the system to a halt or compromise data integrity can, and do, result in financial loss or worse. In some applications, such as medicine or air traffic control, such a failure may even result in loss of life.

Data Integrity

Even the most reliable computer hardware eventually fails. When it does, it is essential that it do so in a "soft" manner—that is, without corrupting other components of the system and causing larger, more serious failures. This is particularly true when the system contains large data bases, which must be protected against accidental loss or corruption when a system component fails. When a processor, input/output channel, controller, or disc drive fails during a transaction, the entire transaction must either continue to run to completion or be "backed out," leaving the data base in a consistent state. The data base management system must guarantee such behavior over a wide range of possible hardware and software failure conditions.

Networking

Many organizations have customers distributed around the world or around a nation or state. It is common to decentralize such an organization as semiautonomous but cooperating units which deal with different functions or regions. For similar reasons, it often makes sense to decentralize the orgainzation's computing and transaction processing. Such an approach typically allows the computing system to reflect the organization's structure, places control in the hands of the people with direct responsibility for the function and usually improves both the reliability and responsiveness of the system.

Decentralizing a computer system requires that the computers be able to communicate with one another via telecommunication lines and requires that the programming or management of the system not be prohibitively complex. In particular, one must be able to write a transaction which operates on multiple network nodes and yet has no special code to deal with the distribution of data and terminals. The transaction processing system must provide this facility.

Ease of Use

Since computers were first commercially applied in the 1950's, the cost of computation has plummeted. At the same time, personnel and labor costs have risen steadily. As a result, computers are increasingly being applied to increase the productivity of people. Indeed, this search for greater productivity is one of the major driving forces behind the movement to on-line systems, and the principal reason why computers are being used more often by people who have little or no formal computer training. It is essential that these people be provided with a friendly, nonthreatening interface to the data base and processing functions that are meant to assist them in their day-to-day jobs.

The movement toward on-line applications has been difficult to achieve in practice. On-line programs tend to be more complex than their batch predecessors. Batch programs typically processed transactions from only a single input stream. On-line applications typically must interact with many terminals at one time, and the details of keeping track of multiple outstanding transactions can be very complex. Development and debugging of these applications can be major bottlenecks in the movement to on-line applications. Operating system and data base software for the on-line environment must simplify the job of the applications developer, the goal being to make this process as simple as, or simpler than, that of implementing batch applications.

Performance

To be useful in increasing productivity, on-line systems must provide a high level of performance. The primary measure of an on-line system is throughput: how many transactions it can handle in a given time period? This measure determines the number of simultaneous users who can be supported by a specific hardware configuration for a specific cost. Typically, the system needs to support hundreds of on-line users generating millions of transactions per day.

DESIGN IMPLICATIONS FOR DATA BASE SYSTEMS

From the foregoing discussion, we see that a large on-line transaction processing system requires a "total system" design. Indeed, these systems have become possible only through an increasing synergy between the exploitation of computer hardware and software. Moreover, only in the past fifteen years has the profound importance of data base technology been realized, leading system designers in both the academic and commercial worlds to focus on the theoretical and practical issues surrounding the implementation of data base-oriented transaction processing systems.

As data base systems have grown in size, diversity, and importance, it has become increasingly important to search for ways of accommodating growth in the number and complexity of applications, as well as the migration to cost effective hardware, a greater diversity of users, and multiple levels and types of access to data bases.

The qualities of a data base system that tend to promote its durability and survivability in the face of these demands must reside not only in the organizational structure of the data base itself, but also in the entire hardware/software structure that contributes to the total system. Ideally, all of the major components of the system—the underlying hardware architecture, the operating system, the data communications subsystem, and the data base management facilities—should reinforce and complement one another to produce a system that meets contemporary data base management requirements.

What, then, are the requirements for a complete data base management system for on-line transaction processing? I have indicated above several characteristics that such a system must have. Here I want to stress completeness from an implementation point of view—from the point of view, say, of a corporate director of data processing, who must provide users with a full range of tools.

Six components must be present in a complete transaction-oriented system:

(0) a data model and data manipulation language;
(1) a data dictionary facility, together with a data definition language;
(2) access methods—indexes, alternate keys, etc.;
(3) integrity controls, which include concurrency, crash recovery, and consistency;
(4) query and report-generation facilities; and
(5) terminal and transaction control facilities for screen control, data validation, formatting, transaction initiation, and flow control.

Relational Data Model and Data Manipulation Language

At the heart of a data base management system is the data model—the organizational scheme that determines how a data base is to be logically accessed, how easy or difficult it is to use or modify, and how stable, or "survivable," it is in an evolving and changing system environment. In the world of transaction processing, as we have seen, these qualities are of critical importance.

Of the three primary data models—the hierarchical, the network, and the relational—the relational is the most recent and, consequently, the one that has been developed with the most contemporary needs of users of large-scale data base systems clearly in mind. Data base files in a relational system have three characteristics that distinguish them from files in hierarchical or network-based systems:

(1) all records in a relational file have, from the user's point of view, the same structure,
(2) relationships between records are manifested by comparing common field values,
(3) users need not be concerned with how the data is physically accessed or stored.

A fourth characteristic is often added and is sometimes taken, I believe mistakenly, as the sole distinguishing characteristic of relational systems—that all files must be "flat" two-dimensional tabular representations of data. The flat file is certainly a desirable feature for user-simplicity and supporting powerful query languages; but to enforce the flat-file structure without exception—for example, without allowing an occasional use of a repeating group or a COBOL "occurs" clause—can detract from system performance or applicability in evolving commercial enterprises where existing COBOL programs must be executed.

Relational data bases have the three characteristics above and allow files to be viewed as a set of interrelated tables whose rows are records, columns are the data fields, and whose records are related through fields having matching data.

Although the relational model promotes the use of high-level query languages, the use of such languages should not be confused with the data model. Indeed, one performance enhancement to many current systems is to allow low level, navigational access to relational files.

One primary benefit of a relational data base is that files are logically interdependent, but physically independent. This means that files can grow, that new files can be added, and that files can be redistributed over a network without requiring structural changes to related files. All of these things contribute to the growth and performance of a transaction processing system and minimize the changes that must be made to application programs to access the data.

Another benefit of the relational approach arises from the invisibility of physical file structures and access methods to the user—whether the user is a programmer or ad hoc inquirer. User queries and updates can be formulated in user-oriented languages that need not and do not reflect the manner in which data is actually stored in a data base. Since the user language does not need to reflect storage-level data structures, the query-formulation process and the actual data accesses can be separated into two levels—one logical and one physical. This further enhances a system's capacity for modular growth; it also can be exploited in a distributed network environment, in which the storage-level access processes can be geographically located with the data while the query-formulation processes are located with the users.

Since 1970, nearly thirty relational data base management systems have been reported in the technical literature. Not all of these systems have actually been implemented; some have only been designed, and only a few, for example, Tymshare's MAGNUM, IBM's QUERY BY EXAMPLE, IBM's DS/SQL, Tandem's ENCOMPASS, and Relational Software, Inc.'s Oracle, have been developed for commercial application, the others having been developed as research vehicles to support the exploration of high-level, nonprocedural query/update languages. It is widely believed that the majority of data base management systems of the near future—say, two to three years away—will be relational (CW, July 14, 1980). It has been Tandem's experience that the relational model, integrated into a complete hardware/software architecture, provides the most natural solution to the problems facing designers of on-line transaction processing systems.

Data Dictionary/Data Definition Language

A data definition language (DDL) and a data dictionary are central to any data base management system. The portion of a DDL that describes the records and files in a data base is called a schema; a schema is a list of DDL statements that describe record structures, file types, and file access methods.

A DDL schema in a source language is compiled to produce, among other things, a data dictionary for the data base it describes. The DDL system may also produce a schema report summarizing record structures, index keys, filecreation commands, and data-declaration source code for the host languages supported by the system. The data dictionary is a permanent record of the data base schema, which becomes a system resource providing data base managers with information about how each file is structured and about how all files are related to one another.

In addition, the data dictionary should be augmented by a central repository containing descriptions of other system components such as transaction programs, communication lines and devices, and users.

Access Methods and I/O Architecture

All data base management systems have access methods of some kind. They provide a variety of file types (e.g., sequential, relative, indexed), alternate keys, blocking factors, compression and buffering techniques. They provide efficient access to data stored on discs.

Here, the complementary nature of hardware and software is crucial. The real questions are how efficient the methods are and whether they provide the user with the freedom to view the data independently of how it is stored at the physical level. The most powerful and efficient access methods combine hardware and software strategies to reduce the time required to find data at its physical location on a storage device.

Integrity Controls

A transaction is a series of updates that transforms a data base from one consistent state at time T to a new consistent state at time T'. Once that series of updates begins, the data base is in an inconsistent state until the series is completed. Consistency control mechanisms such as file/record-level locking ensure either that the system goes to state T' and finishes, leaving the data base consistent in the new state, or that it returns to state T, leaving the data base consistent in the old state.

For a distributed environment, the lock owner must be a transaction identifier that can span multiple application processes running on more than one system, which may initiate accesses to a number of files on a number of systems. Once the accesses are opened and the locks imposed, the locks must be held until the transaction ends. This avoids cascading transaction backout due to transaction aborts caused by application, hardware or data communication failures.

In addition, the transaction processing system must have a mechanism to deal with application and system failures. The application program must be able to cause all work done by that transaction to be undone (in order to cancel a transaction). Also, the system must be able to undo incomplete transactions and must be able to redo or reconstruct the data base and the outputs of successful transactions in case of a serious system failure.

Query/Report-Generation Facilities

Query facilities and report-generation facilities, while often referred to as one and the same, are separate components. This is an important issue in discussing the usefulness of data base management systems, and particularly existing relational systems, for commercial applications. Most relational systems have very rudimentary report generators or none at all. This is not due to any shortsightedness on the part of their designers; rather it happened because most of these systems were developed as vehicles for research into higher-level query languages. Report generation was not particularly relevant to those investigations. Nevertheless, report-generation facilities are essential in a commercial environment, and a system lacking them will not be as useful for commercial applications.

Terminal Control Facility

A screen-oriented terminal control facility must be available to capture data and provide operator support in an on-line transaction processing system. This facility must be able to support a large number and variety of terminals.

It must be easy for application programmers to define screens. Once defined, these definitions should work for a large variety of terminals. Application programs should be unaware of (independent of) the terminal type. Translating screen formats into internal system formats, and vice versa is a major problem which must be addressed by on-line transaction processing systems.

In addition, applications will have to sequence through many different screens to assemble and process a transaction. The architecture of such a system can be contrasted with that of earlier batch-processing systems. One of the main advantages of batch systems was that they were very simple. Input typically came in on cards, a data base residing on tape or disc was updated, and reports were generated. The system read one transaction, did what it requested, produced input for a report, and read the next transaction. The system—and, just as importantly, the programmer who created it—had to worry about only one thing at a time.

In designing and implementing a system for on-line transaction processing, one would like to achieve that same simplicity. The problem is that there are a multitude of terminals on one end, and a mass of data on the other, and a variety of processes and processors that need to be accessed in between.

The simplicity of a batch system might be achieved here by writing an application program to deal with each terminal, or with each type of terminal. That application program could then be replicated as many times as necessary to handle all the terminals, so that there might be 100 copies of that application program running, one to a terminal, against the data base. This would make terminal handling relatively easy, but it would consume system resources at an extraordinary rate. If there were only 10 terminals, this organization might be feasible; if there are hundreds, it is not.

An alternative might be to write one program that accepts input from all of the terminals. This program would be multi-threaded: it would deal with all the terminals at once, keeping track of what input came from what terminal, and of which terminals were active and which inactive. This scheme has the appealing quality of centralizing control of access to the data base, but is, unlike the other one, extremely complex and does not lend itself to multiple processor computer architectures or to networking.

Each of these approaches has some advantages, and, given an adequate hardware and operating-system architecture they can be combined. Tandem's distributed architecture led to an approach where a multi-threaded front-end terminal control program can interact with multiple terminals, while many different application programs can operate on the data base. In an order processing system, for example, one application program will generate purchase orders, while another will update customer accounts. The application or "server" programs can be relatively simple if the terminal control programs are designed to deal with the complexity of multi-threaded processing. It is here that the relational data base structure gives application programmers a distinct advantage by permitting them a simple view of data that can readily accommodate growth of the data base and changes in access paths without programmer intervention. Note, however, that a relational data base structure alone is not sufficient: it must be part of a total system approach.

There are advantages to separating the screen-control functions from the data base access functions in a distributed processing system. First, it allows for easier control of load balancing, since one may have many invocations of the data base program that does, for example, order entry. Second, it simplifies maintenance by permitting changes in the way the screens look without requiring changes in how the corresponding business functions are processed.

Language Levels and Language Support

Since relational systems were originally devised at least in part for the purpose of investigating query and retrieval languages, it is important to understand the notion of "level" of language and what it means in a transaction-oriented data base environment.

When we speak of "higher-level" query or update languages, we mean languages that are non-procedural and are best suited to dealing with sets of files and records. When we speak of "lower-level" languages, we mean those that are procedural, navigational, record-oriented—that is, statements manipulate individual records of the data file rather than entire sets. In the early days of relational data base system research, the data management problem being researched was not how to get data into the data base but how to retrieve data from it—and in particular, how to retrieve data from it if you did not know exactly how the data was stored. The focus was, therefore, on higher-level query languages, to the detriment of lower-level data manipulation languages for efficiently traversing data bases. It was partly owing to this neglect that early relational systems acquired a reputation as inefficient.

Over time, and as relational technology has moved into the commercial world, it has become clear that both levels of language are necessary, and that most data base management tasks are suited to one level or the other. A single level of language will not suffice, either for efficiency or for user convenience. Higher-level languages such as IBM's SQL, Tandem's ENFORM, and National CSS's NOMAD perform well for set-oriented query formulation and report generation but are poor vehicles for record-at-a-time updates. Low-level languages, such as Tandem's ENSCRIBE and Tymshare's MAGNUM, perform well for record-at-a-time updates but become complex when used for set-oriented retrieval or update.

Another way of looking at the language issue is to divide transactions into two classes: those that are information-oriented and those that are function-oriented. As examples of information-oriented transactions, we can take the sorts of reports that a manager or planner creates with a decision support system, such as end-of-month financial summaries and "what-if" reports. The transactions involved in producing these reports typically have a few concurrent users accessing large numbers of records. What I call function-oriented transactions, on the other hand, involve a great many concurrent users accessing a small number of records each; examples are order entries and bank-account withdrawals and deposits.

In terms of set-oriented versus record-at-a-time-oriented retrieval, information transactions are clearly of the set-oriented variety, and functional transactions of the record-at-a-time variety. Relational systems handicapped by the lack of an efficient lower-level language capability have not fared well in functional transaction processing. Again, Tandem's success with this type of application strongly suggests that future relational transaction systems will have to provide language support at both the higher and lower levels.

CONCLUSION

A good deal of the recent work that has been done in relational data management has been devoted to devising a theoretical framework of data base design. Also, because of the desirable characteristics of relational data bases and user interfaces, manufacturers are evolving practical means of implementing efficient relational systems. As far as practical measures are concerned, it is widely acknowledged that relational data base management systems must be implemented in highly efficient processing environments if their performance is to match that of hierarchical and network systems. The architecture provided by Tandem's multiprocessor NonStop™ system successfully supports relational data base processing in on-line transaction processing applications. This is because Tandem's hardware and data base operating system are designed specifically to support this requirement.

One still finds both partisans and objectors in the ongoing discussion accompanying the emergence of relational data base management as a practical, powerful discipline for widespread application in the data processing industry. To the extent that there is still a debate, however, it is growing increasingly narrow in the face of successful implementation and use of large-scale systems such as ENCOMPASS. Tandem's purpose in building and offering a relational system was, of course, not to debate but to offer a competitive product in its chosen marketplace. The application of the relational model for their data base offering in the context of large-scale on-line transaction processing is a very important aspect of the development of that product.

TANDEM HARDWARE SUPPORT FOR A RELATIONAL DATA BASE SYSTEM

The Tandem NonStop™ multi-computer system was designed to provide a system for on-line applications that would emphasize high availability and modular expandability [3,4]. Designing a system for this purpose also yielded a system that supports the implementation of fast access paths which are crucial to the performance of relational data base systems.

The hardware structure consists of multiple computer modules interconnected by two interprocessor buses that transfer data at an aggregate rate of 26 megabytes per second. (See Figure 1.) Each processor has its own power supply, with battery backups, up to 2 megabytes of memory, and a block multiplexor i/o channel that can transfer data at rates of 4 megabytes per second. Each i/o controller is redundantly powered and connected to two different i/o channels. As a result, any interprocessor bus failure does not affect the ability of a processor to communicate with any other processor, and the failure of an i/o channel or of a processor does not cause the loss of an i/o device. Likewise, the failure of a module (processor or i/o controller) does not disable any other module or disable any inter-module communication. Finally, i/o devices such as disc drives can be connected to two different i/o controllers, and disc drives can in turn be duplicated (or "mirrored") so that the failure of an i/o controller or disc drive does not result in lost access to data.

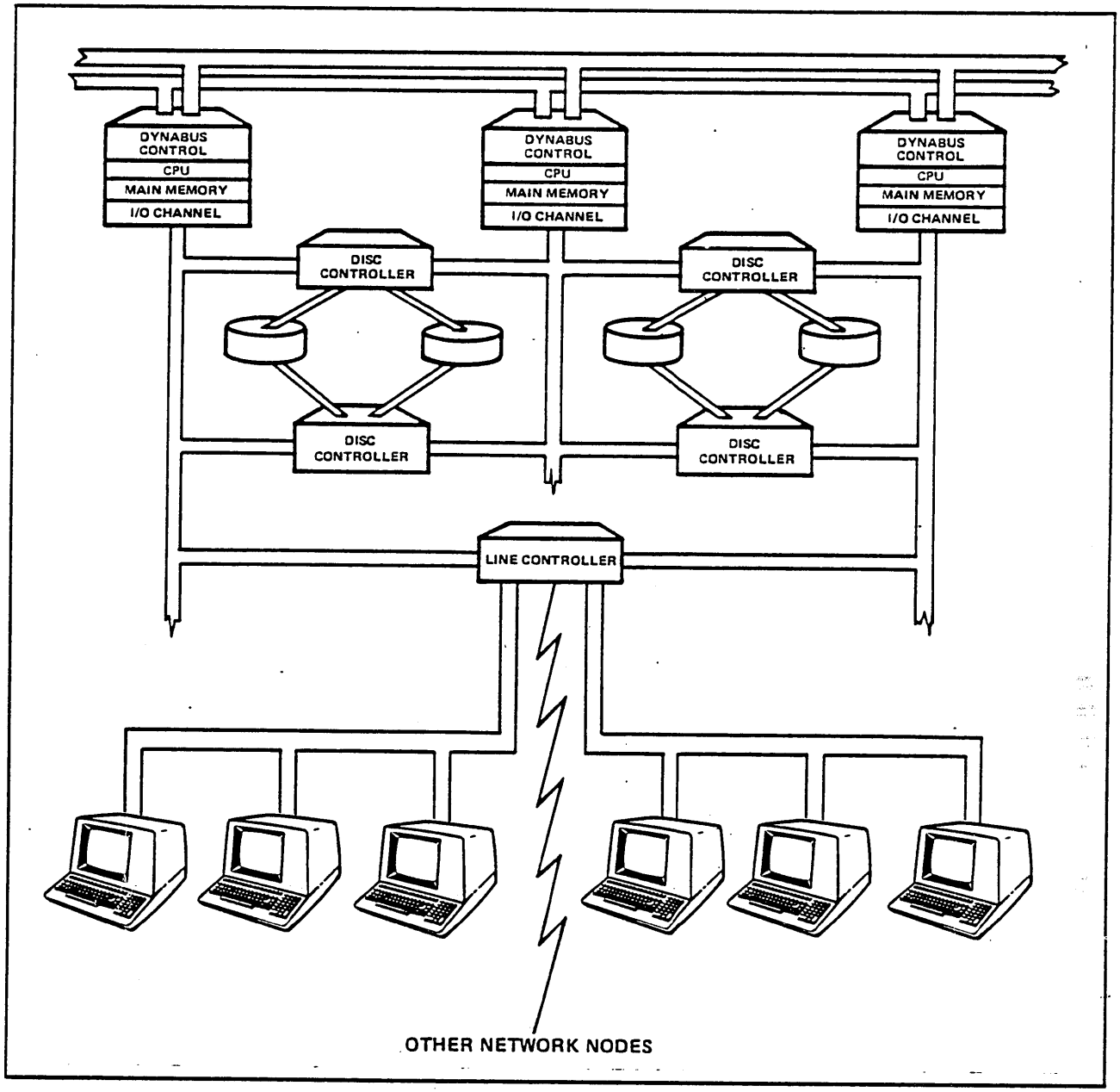

Figure 1. The Hardware Architecture of the Tandem NonStop Systems

The system is more than a simple multiprocessor, rather it is a true multiple computer system. The multiple computer approach is preferable for several reasons. First, since no module is shared by the entire system, it increases the system's availability. Second, a multiple computer system does not require the complex hardware needed to handle multiple access paths to a common memory. In smaller systems, the cost of such a multiported memory is undesirable; and in larger systems, performance suffers because of memory access interference.

The system's modular structure and unique power distribution scheme allows processors, i/o controllers, or buses to be repaired or replaced while the rest of the system continues to operate.

The system structure allows a wide range of system sizes to be supported. As many as 16 processors may be connected into one system. Each processor may also have up to 256 i/o devices connected to it. This provides for tremendous growth of application programs and processing loads without the requirement that the application be reimplemented on a larger system with a different architecture. Also, up to 255 systems can be configured in a single network. Finally, the system is designed to provide a general solution to the problem of providing a failure-tolerant, on-line environment suitable for commercial use. As such, the system supports conventional programming languages and peripherals and is oriented toward supporting large numbers of terminals with access to large data bases.

TANDEM RELATIONAL DATA BASE ACCESS

Tandem's ENCOMPASS distributed relational data base system uses several techniques to take advantage of the architecture of the hardware to provide high performance relational access methods. The access methods are implemented as an integral part of the operating system and as such, are distributed across multiple processors and i/o channels. This optimizes performance by allowing true parallel processing for concurrent users. Some of the methods used by ENCOMPASS for efficient data base access are: distributed multi-key access, buffering of data blocks, the use of mirrored disc volumes, file partitioning, data compression, and distributed locking and transaction monitoring. Also, each CPU has microcode for access method processing.

Multi-Key Access

ENCOMPASS supports three types of file structures: sequential, relative, and indexed (key sequenced). Indexed files are implemented as B*-trees whose index and data block sizes can be varied. Variable length records are supported for both sequential and indexed files. Any file type can have up to 255 alternate key fields which are defined as any contiguous bytes within a record. Alternate keys are implemented as a separate indexed file(s) by concatenating the alternate key value with the primary key value for each record of the file. The data base system automatically maintains alternate key files. Alternated key files may be stored on any disc. If they are placed on a disc that is associated with a different CPU than the primary file, then alternate key files and primary files can be processed with true parallelism for concurrent users.

Data Block Buffering

The purpose of buffering is to keep the most recently accessed data or index blocks in main memory. Thus, index blocks for an indexed file are likely to remain in the buffer because they are heavily referenced. The amount of memory that should be assigned to each buffer is determined by the user installation. When a read request is received, ENCOMPASS automatically first looks in the buffer to see if the block being sought is already there. If it is, then a disc read is avoided.

Another property of Tandem access methods is that the data blocks referenced by the lowest-level index block can be loaded on the same cylinder as the index block. This means that when the disc head is located to read the lowest index block it does not have to move to be in position to read the data block. The only time delay is that required for rotation of the disc—the latency. In the case of a two-level index, which handles most large files, this works out to one seek and two latencies for a random access using a logical key value. The first level index is in buffered memory requiring no seeks or latency; the second level index requires one seek and one latency, and the data block requires only an additional latency.

Mirrored Disc Volumes

Most users of Tandem systems select the option of duplexing (mirroring) their discs. Thus each record is stored on two discs. The primary benefit of discmirroring is that no single disc failure results in a loss of data. Mirroring disc volumes also has a performance advantage in addition to enhancing system availability. For read operations, mirrored volumes permit an option called split seeks. Under this option, each disc is divided into two read areas, an inner area and an outer area. A logical read on a pair of mirrored discs is directed to the inner cylinder area of one disc or the outer cylinder area of the other, dependent on address of the record, thus halving the distance each head has to traverse across the surface of the disc. The system is thus able to do two different reads at once. Furthermore, because the heads are moving over a smaller surface, the seek time is much reduced from what it would otherwise be; those reads happen much faster than they normally would. Of course, both heads must perform a write; but this has a relatively small effect on performance because the discs of a mirrored pair can be written in parallel since they are attached to separate controllers.

Many data base systems use a physical pointer for direct or hashed access to the data record, indicating a particular sector of a particular cylinder of a particular disc. This means that there is one seek and one latency, totally, on the average, 38 milliseconds (30 millisecond average seek and 8 millisecond average latency). In the Tandem system, with a logical pointer, there is one seek and two latencies. In a non-mirrored system, this works out to 46 milliseconds; in a mirrored system, because of the reduction in seek time due to split seeks, this will only take 33 milliseconds. Experiments have verified this analysis. Thus the unique architecture of the Tandem system will not only improve availability but can also be exploited to provide logical access to data faster than many physical addressing schemes.

File Partitioning

Typically, in a transaction environment, a given file may be very heavily accessed, to the point where putting it all on one disc drive will cause a bottleneck. If an application requires service of 10 transactions per second, and each transaction requires reference to 10 data base records, this requires 100 references, or accesses, per second. Since a disc drive is limited to approximately 20 random accesses per second, this suggests spreading the file over five disc volumes. Because of Tandem's multiple-processor architecture, several processors can be involved in driving the discs in parallel. Any file can be partitioned into 16 logical subfiles based on a range of partial primary key values. The allocation, accessing and concurrency control of the single logical file and its alternate keys, which also can be partitioned, is managed transparently by ENCOMPASS.

Partitioning can be used for three other reasons. First, very large files can be created that are larger than a single disc, e.g., up to 4 billion bytes for 300 megabyte formatted discs. Second, partitioning can reduce the number of index levels for large files, thereby reducing access times to random records. And last, partitioning can be used to distribute a file among nodes of the network. A program accessing a partitioned file is unaware that the file is partitioned among many discs on many different network nodes.

Data Compression

ENCOMPASS provides two kinds of data compression to decrease the amount of disc storage required for a data base. One is rear index compression, which is always in effect (that is, it is not optional). In read index compression, only the character that distinguishes one key value from the next is stored in an index block; the entire key does not have to be stored. In most cases, keys can be significantly truncated without entailing a performance penalty.

A second type of compression, front compression, is an option that can be turned on and off. In front compression, which can be applied to index or data blocks or both, the characters that are repeated from one record to the next are coded and stored as a number. If data is very clustered, a file can be significantly compressed. There is a slight CPU overhead associated with front compression because the system cannot go directly to a particular record in a block; it must always begin with first record. This loss is negligible, however, if the compression can reduce the number of index levels of very large files, thereby substantially reducing the average i/o time for accessing a record.

Locking and Transaction Monitoring

ENCOMPASS permits both file and record locking. Locking for concurrency control is completely distributed to the CPUs (transparently across a network if a remote file is accessed) which are controlling the disc that contains the file or partition of the file that a lock request pertains to.

The transaction monitoring facility, built into the operating system, provides for a network-wide identification of a transaction. A transaction is defined as the sequence of inserts, modifies, or deletes that have occurred to any file(s) in single node or multiple nodes in a network between begin-transaction and end-transaction system calls that transforms a data base from one consistent state to the next. The system will keep distributed audit trails of before and after images of records at the sites where the updates have occurred and automatically backout any transaction that fails due to application program, hardware, or data communication failure.

ENCOMPASS assures transaction consistency by requiring all locks acquired during a transaction to be held until the end of a transaction, so that the completion or backout of one transaction will not interfere with a concurrent transaction, and by implementing a two phase commit policy. The first commit phase assures that all the data required to recover a data base, (e.g., backout a transaction even if the transaction involves updates across a network) has been written to mirrored discs on the same nodes where their associated files have been updated. The second commit phase then can proceed to orderly commit the transaction by removing all locks,thus allowing other transactions to access the newly changed records. Because of the NonStop™ architecture of the hardware and operating system, records of the audit trails do not have to be continually written to disc to ensure data base integrity. Disc i/o need only occur at end of transaction. Audit records can be buffered in memory saving many i/os over conventional implementations of transaction logging.

REFERENCES

[1] Codd, E.F., *A Relational Model of Data for Large Shared Data Banks*, CACM, June 1970.

[2] Codd, E.F., *Recent Investigations in Relational Data Base Systems*, in "Data: Its Use, Organization and Management"; Proceedings 1975 ACM Pacific Conference, April 1975, pp. 39-43.

[3] Katzman, J.A., *A Fault-Tolerant Computing System*, Tandem Computers Incorporated, 1977.

[4] Bartlett, J.F., *A NonStop™ Operating System*, Tandem Computers Incorporated, 1977.

Distributed by
TANDEM COMPUTERS
Corporate Information Center
19333 Vallco Parkway MS3-07
Cupertino, CA 95014-2599