# Highly Available Systems for Database Applications

WON KIM

*IBM Research Laboratory, San Jose, California 95193*

As users entrust more and more of their applications to computer systems, the need for systems that are continuously operational (24 hours per day) has become even greater. This paper presents a survey and analysis of representative architectures and techniques that have been developed for constructing highly available systems for database applications. It then proposes a design of a distributed software subsystem that can serve as a unified framework for constructing database application systems that meet various requirements for high availability.

Categories and Subject Descriptors: A.1 **[General Literature]**: Introductory and Survey; C.1.2 **[Processor Architectures]**: Multiple Data Stream Architectures (Multiprocessors)—*interconnection architectures*; C.4 **[Computer Systems Organization]**: Performance of Systems—*reliability, availability, and serviceability*; H.2.4 **[Database Management]**: Systems—*distributed systems, transaction processing*

General Terms: Reliability

Additional Key Words and Phrases: Database concurrency control and recovery, relational database

## INTRODUCTION

In this paper we examine major hardware and software aspects of highly available systems. Its scope is limited to those systems designed for database applications. Database applications require multiple paths from the processor to the disks, which gives rise to some difficult issues of system architecture and engineering. Further, they involve the software issues of concurrency control, recovery from crashes, and transaction management.

In a typical business data processing environment, a user message from a terminal invokes an application program. The application program interacts with a transaction manager to initiate and terminate (commit or abort) a transaction. Once a transaction has been initiated, the application program repeatedly interacts with a database manager to retrieve and update records in the database.

A transaction is a collection of reads and writes against a database that is treated as a unit [Gray 1978]. If a transaction completes, its effect becomes permanently recorded in the database; otherwise, no trace of its effect remains in the database. To support the notion of a transaction, undo log of data before updates and redo log of data after updates are used to allow a transaction to be undone or redone after crashes. The Write Ahead Log protocol often is used to ensure that the log is flushed to the disk before the updated database records are written to the disk.

The most fundamental requirement in constructing a highly available system for database applications is that each major hardware and software component must at least be duplicated. At minimum, the sys-

Author's current address: Microelectronics and Computer Technology Corporation, 9430 Research Boulevard, Austin, Texas 78759.

CONTENTS

INTRODUCTION
1. MACHINE AND STORAGE ORGANIZATION TAXONOMY
2. INTUITIVE CRITERIA FOR HIGH AVAILABILITY
3. LOOSELY COUPLED SYSTEMS WITH A PARTITIONED DATABASE
   3.1 Tandem NonStop System
   3.2 AT&T's Stored Program Controlled Network
   3 3 Bank of America's Distributive Computing Facility
   3.4 Stratus/32 Continuous Processing System
   3.5 Auragen System 4000
   3 6 System D Prototype
4. TIGHTLY COUPLED SYSTEMS WITH A SHARED DATABASE
5. LOOSELY COUPLED MULTIPROCESSOR SYSTEM WITH A SHARED DATABASE
   5.1 GE MARK III
   5.2 Computer Consoles' Power System
6. REDUNDANT COMPUTATION SYSTEMS: SYNTREX'S GEMINI FILE SERVER
7. A FRAMEWORK FOR THE MANAGEMENT OF SYSTEM CONFIGURATION
8. CONCLUDING REMARKS
ACKNOWLEDGMENTS
REFERENCES

tem requires two processors. There may have to be two paths connecting the processors, and it is desirable to have at least two paths from the processors to the database, that is, two I/O subsystems consisting of a channel (I/O processor), controller, and disk drives. The disk controllers must be multiported, so that they may be connected to more than one processor.

On the software side, the system needs five essential ingredients: (1) a network communication subsystem, (2) a data communication subsystem, (3) a database manager (or a file system), (4) a transaction manager, and (5) the operating system.

The network communication subsystem must support interprocess(or) communication within a cluster of locally distributed processors. If the highly available system is a node on a geographically distributed system, the communication subsystem must also support internode communication.

The data communication subsystem is the terminal handler that receives user requests, invokes application programs, and delivers the results that it receives from the database/transaction manager.

The database manager must support the two fundamental capabilities of concurrency control and recovery. That is, it must guarantee that the database remains consistent despite interleaved reads and writes to the database by multiple concurrent transactions. Techniques such as locking and time stamping have been developed and extensively studied for this purpose [Bernstein and Goodman 1981; Kohler 1981].

The transaction/database manager must ensure that the database consistency is not compromised by system failures or transaction failures caused by software errors or deadlocks. In particular, it must be able to recover from transaction failures and soft crashes, which corrupt only the contents of the main memory, as well as from hard crashes, which destroy the contents of the disks. For recovery from soft crashes, the undo and redo logs of transactions are used [Gray et al. 1981; Haerder and Reuter 1983]. To recover from hard crashes, systems rely on periodic dumping of the database into archival storage.

The operating system is needed not only to run the other software components, but also to detect most of the common, low-level software/hardware errors that occur. Most computer systems rely on program checks (interrupts) and machine checks to detect such errors. Program checks are used to detect exceptions and events that occur during execution of the program [IBM 1980]. Exceptions include the improper specification or use of instructions and data, for example, arithmetic overflow or underflow, and addressing or protection violation. Machine checks are used to report machine malfunctions, such as memory parity errors, I/O errors, and missing interrupts. The hardware provides information that assists the operating system in determining the location of the malfunction and extent of the damage caused by it.

Most systems have been designed to survive the failure of a single software/hard-

ware component. If a system is to be truly continuously operational, however, it must guarantee availability during multiple concurrent failures of software/hardware components, during on-line changes of such components, and during on-line physical reconfiguration of the database itself. To support the full range of high-availability requirements, the operating system must be supplemented with a software subsystem that can manage all software and hardware components of a system. Such a software subsystem will receive failure reports from the system components and reconfiguration requests from the system operator. It will analyze the status of all the resources it manages, and compute the optimal configuration of the system both in response to multiple concurrent failures of components and requests for load balancing. Further, it will initiate and monitor system reconfiguration, effecting mid-course correction of a reconfiguration that does not succeed. Finally, it will diagnose a class of failures that other components fail to recognize.

The systems introduced in the following paragraphs are often considered highly available, and are used as examples of various architectures and stratagems throughout the remainder of this paper, particularly in Sections 3 through 6. A certain number of these systems do not, in my opinion, meet the criteria for classification as highly available; these are discussed in Section 2.

Tandem Computers has been successful for several years in marketing fault-tolerant computer systems, which shield the users from various types of failures of the software and hardware components. Other companies have entered this market, including August Systems, Auragen Systems, Computer Consoles, Stratus Computer, Synapse Computer, Syntrex, Sequoia, Tolerant Systems, and others [Electronic Business 1981; IEEE 1983]. Of these, Auragen, Computer Consoles, Stratus, and Synapse have focused on transaction processing; August Systems aims at industrial and commercial process control.

A number of other systems address the issue of reliability and availability: Syntrex is marketing a local network file server called GEMINI; the MARK III Cluster File System, developed by General Electric, provides time-sharing services for its telephone-switching network users; and the Distributive Computing Facility developed by Bank of America automates the teller functions for accounts.

In addition, the late 1970s SRI SIFT project for aircraft control evoked the August Systems' products; Bolt Beranek and Newman (BBN) developed the PLURIBUS system for use as a highly reliable communications processor on ARPANET; and during the 1960s AT&T developed No. 1 ESS and No. 2 ESS (Electronic Switching System) for telephone switching services.

System D, a distributed transaction-processing system, was created as a prototype at IBM Research in San Jose with availability and modular growth as its major objectives, and there are other research projects currently under way at IBM Research to investigate availability and performance issues under various software/hardware structures.

The remainder of this paper is organized as follows. In Section 1, a taxonomy of system structures that has been used to construct highly available systems is developed, and a discussion is provided of the advantages and disadvantages of four possible structures. An intuitive set of criteria for highly available systems is given in Section 2. Systems belonging to each of the four system structures are surveyed and critiqued, where possible, in Sections 3 through 6. The discussions in these sections focus on various philosophies of system structure and transaction processing. In Section 7 the functions and structure of a software subsystem that provides a framework for high availability are described.

In view of the fact that such issues as concurrency control, recovery, transaction model, and network communications have been extensively addressed elsewhere, these aspects of highly available systems will not be given detailed treatment. Further, it is generally recognized that such mundane sources as downed telephone lines, careless computer operators, lack of defensive coding, and the way in which the operating

system reacts to failures that it detects can seriously limit the availability of a system. Although important, such aspects are not within the scope of this paper.

## 1. MACHINE AND STORAGE ORGANIZATION TAXONOMY

Two fundamental decisions in constructing a multiple-processor system are the choice of machine organization and physical storage organization. The discussion in this section of the advantages and disadvantages of typical machine and storage organizations significantly benefits from Traiger [1983]. Two conventional techniques for organizing multiple processors are loosely coupled and tightly coupled multiprocessor organizations. In tightly coupled systems, two or more processors share main memory and disks, typically through an interconnection switch, and execute one copy of the operating system residing in the shared main memory. A local cache memory is usually associated with each processor to enhance access speed. In loosely coupled systems, each processor has not only a local cache memory but also its own main memory, and may or may not share disks with other processors. Each processor executes its own copy of the operating system from its own main memory. There are various ways to loosely couple the processors, including shared bus structures, cross-point switches, point-to-point links such as channel-to-channel adapters, and globally shared memories, as in Cm* [Swan et al. 1977].

Tightly coupled multiprocessor systems, such as the Synapse N+1 System, and BBN's PLURIBUS, offer important potential advantages. First, they naturally present a single-system image, since multiple processors execute one copy of the operating system and a common job queue. Second, the processors do not need to communicate via interprocessor messages, with their inherent overhead. However, this performance advantage may be offset by certain problems imposed by this architecture. First, there is contention among processors for the use of shared memory and other shared resources. This must somehow be reduced, especially if the cost/performance of the system is to keep up with its expansion as extra processors are added. Second, potentially complex techniques must be supported to ensure that the contents of each processor's cache memory are up-to-date.

In addition to these performance concerns, there is a potential availability problem with tightly coupled multiprocessors. All processors run the same operating system from shared main memory, and thus, when the operating system is corrupted or the shared memory system fails, the entire system must be restarted. Therefore application systems designed to run on a tightly coupled multiprocessor system must be able to restart very quickly in order to guarantee high overall availability.

Just as there are two techniques for organizing multiple processors, there are two ways to organize disks, and thus the database. One is to assign a set of disks to one processor and allow access to it only through that processor; the other is to have all of the processors share all the disks. The two techniques of organizing multiple processors and the two techniques of organizing disks are combined to give rise to four distinct system structures. The tightly coupled multiprocessor organization and the shared database organization results in a system structure that is called a tightly coupled system with a shared database. The loosely coupled multiprocessor organization gives rise to three other system structures. When a database is split into $N$ partitions and each partition is stored in one set of disks assigned to one of $N$ processors, the resulting system structure is called a loosely coupled multiprocessor with a partitioned database. When each of $N$ processors can directly access the entire database, stored in one set of disks, the system structure is called a loosely coupled multiprocessor with a shared database. When an entire database is replicated in each set of disk volumes attached to each of $N$ processors, and each processor computes the same user request in parallel, the resulting structure is called a loosely coupled multiprocessor with redundant computation.

The Tandem NonStop System, the Auragen System 4000, the Stratus Continuous Processing System, IBM's System D pro-

totype (and its sequel, the Highly Available Systems project), Bank of America's Distributive Computing Facility, and AT&T's Stored Program Controlled Network are loosely coupled multiprocessors with partitioned databases. In this architecture, a database manager residing in each processor owns and manages the partition of the database assigned to that processor. Any user request (transaction) that requires access to more than one database partition is satisfied by message communication among the database managers that own the necessary partitions. Communication overhead is the single most significant disadvantage of the loosely coupled system with a partitioned database. There are two aspects to this interprocess(or) communication overhead: One is the messages sending requests to servers and receiving results from servers; another aspect is the messages and processing involved in the distributed commit protocol that ensures that the database, which is distributed across processors, is left in a globally consistent state when the transaction completes or aborts.

Some variation of the two-phase commit protocol described by Gray is used in committing or aborting a distributed transaction [Gray 1978]. One of the participating transaction managers is designated as the commit coordinator. During phase 1, the commit coordinator sends a "prepare to commit" message to all other participants. The participants reply with "yes" or "no" messages to the coordinator and enter phase 2. If the coordinator receives "yes" votes from all participants, it sends a "commit" message to all participants. If any participant replied with a "no" vote, the coordinator sends an "abort" message to all the participants. During phase 1 all participants retain the right to unilaterally abort the transaction. However, once they enter phase 2, they no longer can unilaterally abort the transaction; they must obey the decision of the commit coordinator.

Loosely coupled multiprocessor systems with shared database architecture, such as GE's MARK III Cluster File System, Computer Consoles' Power System, and IBM's AMOEBA research project [Traiger 1983], offer a potentially enhanced availability over a tightly coupled multiprocessor system, since the operating system is not shared among the processors. One disadvantage, however, is the lack of a single-system image; that is, system operators and system programmers must contend with multiple copies of the operating system. One important advantage of this architecture over a loosely coupled multiprocessor system with a partitioned database is that it avoids the difficult problem of deciding which partition of the database should be stored in which processors' disks. Processors may be added to the system without having to repartition the database, and new disk drives may be added without having to worry about which processors should own them.

However, contention on the shared disks is a potential problem, with each processor moving the disk arms to random positions. Further, algorithms for coordinating the global locking and logging of database updates must be carefully designed. A global locking technique in which all database managers must acquire and release locks through a single global lock manager will cause excessive communication overhead and create a bottleneck for performance and availability.

Loosely coupled multiprocessor systems with redundant computation, such as Syntrex's GEMINI file server, and SRI's SIFT (as well as its offspring, the Basic Controller of August Systems, Inc.), achieve fault tolerance by having more than one task perform the same computation and then comparing the results of the computation. In such applications as spacecraft control and process control in a nuclear power plant, correct results of computations are more critical than is the case in typical database applications. Further, the computations are well defined, and the results are often known in advance. For such applications, it makes sense to have multiple processors perform the same computations in order to detect and (even correct) conflicting results. But for office word-processing or transaction-processing applications, this approach may not be desirable.

For applications that require time-consuming computations and/or disk accesses, there tend to be ample opportunities for

program checks and machine checks to detect low-level failures, and for time-outs or defensive coding to detect high-level failures. Unless the need for error detection and correction is highly critical, the redundant-computation approach appears to waste the processing power of the system. Further, the exchange of data and status information among the replicated tasks for each input and output is a considerable performance overhead.

## 2. INTUITIVE CRITERIA FOR HIGH AVAILABILITY

Now we must address the problem of what is meant by availability. The overall availability of a system may intuitively be defined as the ratio between the time when the end user and applications actually have access to all the database and the time when the end user and applications require access to the database. For example, if users require the system to be up for 8 hours a day and the system is actually up for 6 hours during the 8 hours, the availability of the system is 6/8 = .75 during the 8-hour period.

It is more difficult to precisely define a single measure of availability. In the first place, it is not clear how to define the "mission duration" for the system. In spacecraft control applications, the mission duration is clearly defined: While the spacecraft is in orbit, the system must be available 24 hours per day. In business data processing applications, however, the mission duration can be several years. At what point in the life of a system, and for how long, should we measure availability? During one arbitrary month, a year after the installation of the system?

In the second place, should the entire database be available for access by authorized users during the entire duration? For example, when a single database partition becomes inaccessible in a loosely coupled multiprocessor architecture with a partitioned database, one may take the view that the system is no longer available. In fact, this is my view for the purposes of this paper. However, one may equally well take the more charitable view that the system is still largely available, since users may access other database partitions and perform useful work [Good 1983].

Similarly, in a geographically distributed environment, it is not entirely clear where to draw the line on availability, for example, when a node of the system cannot communicate with another node and consequently cannot access data owned and managed by the other node.

Very few vendors of the systems that we are considering have provided availability figures for public review. Suffice it to say here that, however one may define it, often a highly available system is expected to provide higher than 99 percent overall availability.

We use the following criteria to classify a system for database applications as highly available. The first three are hard criteria and provide the rationale for including and excluding detailed discussion of various systems in this paper. The last two are soft criteria, satisfied by very few existing systems and mainly included for future considerations.

(1) The system must support transaction-processing or file server capabilities, specifically concurrency control and recovery techniques, to maintain database consistency. A distributed database system must support a distributed commit protocol to ensure global consistency of the database.

BBN's PLURIBUS [Katsuki et al. 1978], SRI's SIFT design [Wensley et al. 1978], the Basic Controller system now being marketed by August Systems Inc. [Kinnucan 1981], and AT&T's No. 1 ESS and No. 2 ESS [Spencer and Vigilante 1969] do not satisfy this criterion, and will not be discussed in detail in the system survey portion of this paper.

(2) The system must support automatic takeover of full workload by a backup process when a primary process fails. This criterion excludes systems that rely on manual replacement of failed processors to survive a single failure.

The problem with the manual replacement approach is that (1) the responsibility of detecting failure often falls on users or the operators, and (2) the new processor is aware of either the database or the termi-

nals and hence applications, and the system must be cold-started.

Japan National Railways' MARS train seat reservation system, a loosely coupled multiprocessor system with a partitioned database, does not support the concept of backup processes at the present time [Tsukigi and Hasegawa 1983]. Hence, when the processor that manages one partition of the database crashes, no user can access the database partition until the database manager that owns it can be restarted. This system is not given detailed treatment here.

IBM's Information Management System (IMS) with the data-sharing feature [Strickland et al. 1982] is a loosely coupled multiprocessor system with a shared database, with an IMS/VS system in two different processors, each able to access the database in shared disks. When one IMS/VS crashes, its terminals lose access to the database until it can be restarted, and the surviving IMS/VS is not aware of the transactions in process on the system that failed, and hence cannot abort or complete them. We do not consider this system highly available.

The Stratus system does not currently support the concept of primary–backup processes; however, in view of the great extent to which the system incorporates hardware fault tolerance and capabilities for on-line system reconfiguration, it is classified as a highly available system.

(3) The system must survive at least a single failure of such major components as processor, I/O channel, I/O controller, disk drives, and interprocessor communication medium. In particular, a single failure should not make any part of the database inaccessible to the users for beyond a reasonable recovery duration. A reasonable recovery duration may be 1 minute for a mini- and microprocessor system and perhaps 10 minutes for a mainframe, because of the larger number of terminals and applications dealt with by a mainframe system.

This criterion does not imply that a single point of failure is unacceptable, rather that when a single point of failure exists, it must not cause overall availability to suffer.

For example, the Synapse N+1 System has a single point of failure in its shared main memory, but is designed to recover quickly and provides high overall availability. System D has a single point of failure in the electronic switch that connects a pair of processors with the shared disks and terminals, but the probability of failure of such a switch is very low and hence does not severely compromise overall availability.

(4) The system should support on-line integration of repaired or new hardware/software components. Further, in the case of a partitioned database, the system should support on-line migration of the database from one disk system to another.

(5) Additional features aimed at making the component failures transparent to the users may be useful. One is the ability to automatically restart transactions in progress when system crash occurs, which may require the data communication subsystem to log the transaction request on the disk. Another is for the interprocessor communication subsystem to reroute messages originally targeted to a failed process to its backup. In view of the fact that transactions in typical business data processing are short-lived, that is, they complete within a few seconds, it does not appear that important to burden a system with these additional capabilities.

## 3. LOOSELY COUPLED SYSTEMS WITH A PARTITIONED DATABASE

This section provides overviews of architectures and transaction-processing strategies as employed in the Tandem NonStop System, Auragen System 4000, Stratus/32 Continuous Processing System, AT&T's Stored Program Controlled Network, Bank of America's Distributive Computing Facility, and System D.

It is noted that the Auragen, Stratus, AT&T, and Bank of America systems are actually loosely coupled clusters of processors, in which each cluster consists of two or more processors and manages one partition of the database. The cluster itself is not necessarily a loosely coupled multiprocessor architecture. Further, the Auragen, AT&T, and Bank of America systems currently do not support distributed on-line transaction processing: User requests are completely processed within one cluster,

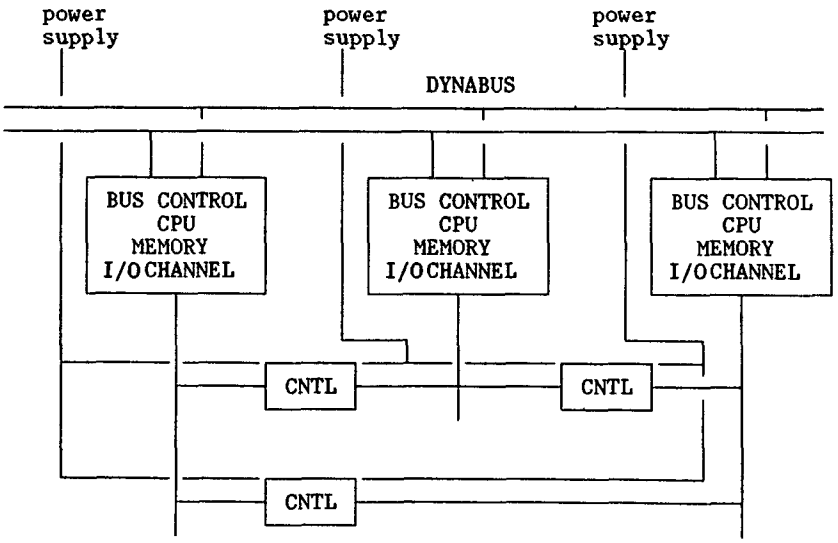


**Figure 1.** Tandem NonStop system architecture.

without requiring the participation of other clusters.

### 3.1 Tandem NonStop System

The NonStop System, developed by Tandem Computers about 1976, has made important contributions to the area of high availability [Bartlett 1978; Katzman 1977, 1978]. As shown in Figure 1, each processor module consists of a central processing unit (CPU), memory, interface to an interprocessor bus system called Dynabus, and an I/O channel. Each of the I/O controllers is connected to two processors via its dual-port arrangement, and each processor is connected to all other processors via a dual Dynabus. Further, as shown in Figure 2, each processor is connected to a pair of disk controllers, which in turn maintain a string of up to four pairs of (optionally) mirrored disk drives. Mirroring is supported in the I/O supervisor, which issues two disk writes for each page of data to be written to the disk. Thus it is clear that the system provides many paths to data, and hence the data are available to the user regardless of any single failure of a disk drive, disk controller, I/O channel, or processor.

The Tandem system was designed to continue operation through any single component failure, and also to allow the failure to be repaired without affecting the availability of the rest of the system. Each processor module has a separate power supply, which can be shut off to replace the failed module without affecting the rest of the system. Similarly, each I/O controller is powered by two power supplies associated with the two processors to which it is attached, and can be powered down by a corresponding switch, without affecting the rest of the system. Thus the I/O controllers survive a single power failure, and each I/O controller and processor module can be repaired without shutting down the rest of the system.

In order to detect a processor failure in the Tandem system, each processor broadcasts an "I-AM-ALIVE" message every 1 second and checks for an "I-AM-ALIVE" message from every other processor every 2 seconds [Bartlett 1978]. If a processor decides that another processor has failed to send the "I-AM-ALIVE" message, it initiates recovery actions, as described later. Although there is a possibility that different processors may reach different decisions as to which processor has crashed, the single-failure assumption precludes consideration (and prevention) of such a possibility.

This "active" failure-detection approach of the Tandem system helps to detect a processor failure soon after it occurs. However, it is not very useful to say that a

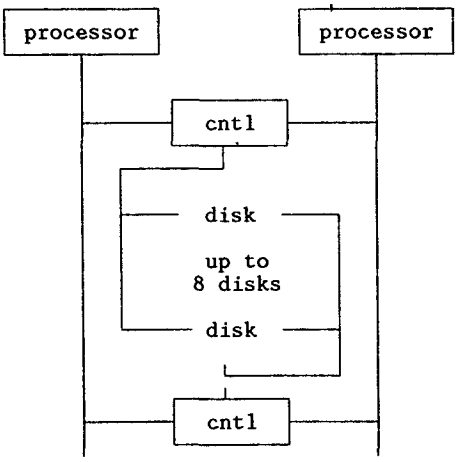


**Figure 2.** Tandem NonStop disk subsystem organization.

processor is "alive" simply because it can send the "I-AM-ALIVE" message, when tasks running in it may have crashed. Software failures must be detected by message time-outs.

The operating system that runs on the Tandem NonStop System is called Guardian. It is constructed of processes that communicate by using messages. Guardian provides high availability of processes by maintaining a primary–backup pair of processes, each in a different processor. The primary process periodically sends checkpoint information to its paired backup process, so that the backup will stand ready to take over as soon as the primary process fails. The checkpoint data from a primary I/O process contain information about the files that are opened and closed. The backup I/O process opens and closes the checkpointed files while the primary is still active, so that in the event of failure of the primary, the backup recovers and proceeds with normal processing without the time overhead of opening the files.

An "ownership" bit is associated with each of the two ports of an I/O controller, which indicates to each port whether it is the primary or backup. Only one port is active for the primary I/O process; the other is used only in the event of a path failure to the primary port, and any attempt to access data through the backup port is rejected. Upon detecting or being notified of the failure of the primary process, the backup process instructs Guardian to issue a TAKE OWNERSHIP command to the backup port. This command causes the I/O controller to swap its two ownership bits and do a controller reset.

The database/transaction manager that runs on the Tandem system is called ENCOMPASS [Borr 1981]. ENCOMPASS consists of four functional components: a database manager, a terminal manager, a transaction manager, and a distributed transaction manager. The ENCOMPASS database manager is implemented as a primary/backup I/O process (called the DISCPROCESS) pair per disk volume. In other words, a primary database manager in one processor and its backup in another processor own the partition of the database stored in the disk volume to which the two processors are connected. The primary database manager checkpoints to the backup, so that if the primary fails before completing a transaction, the backup may take over the disk volume and redo committed transactions and abort incomplete ones.

To run transactions against the database, the user provides two sets of programs: the Screen COBOL program and application server programs. The Screen COBOL program performs screen formatting and sequencing, data mapping, and field validation, and sends transaction requests to application server programs. The application server programs perform application functions against the database by invoking the database manager, the DISCPROCESS.

The Screen COBOL program is interpreted by the terminal management component of ENCOMPASS, called Terminal Control Process (TCP). TCP is also configured as a process pair; the primary TCP checkpoints the backup TCP with data extracted by the Screen COBOL program from input screens.

The transaction management component of ENCOMPASS, which implements the conventional model of transactions, is called Transaction Monitoring Facility (TMF). TMF consists of a lock manager, a log manager (called the AUDITPROCESS), and a recovery manager (called the BACKOUTPROCESS) to provide concurrency control and recovery of interleaved

execution of concurrent transactions. The user's Screen COBOL program interfaces with TMF to indicate the beginning and end of a transaction. The Screen COBOL program receives a transaction identification from TMF at the beginning of a transaction, and attaches the transaction identification to all transaction request messages that it sends to the application server programs. When the Screen COBOL program notifies end of transaction, TMF initiates a transaction commit protocol to complete the transaction and make the effect of the transaction permanent. TMF does not support a global deadlock detection mechanism; deadlocks are detected by time-out, where the time limit is specified as part of lock requests.

### 3.2 AT&T's Stored Program Controlled Network

AT&T's Stored Program Controlled (SPC) Network [Cohen et al. 1983] consists of two key components: the Network Control Point (NCP) and the Action Point (ACP), as shown in Figure 3. The NCP is a database system which manages a database of customer records that is geographically distributed over a network of computers interconnected by the Common Channel Interoffice Signaling (CCIS) network. The ACP is a telephone call processing system, and is a highly reliable No. 4 ESS.

When a call is made to a customer of the expanded 800 services or Direct Services Dialing Capability (DSDC) services, the call is routed to an ACP. The ACP transmits the request to an NCP, which maintains the customer record. The NCP returns the response to the ACP, which in turn routes the call according to the response from the NCP. An administrative system called the User Support System (USS) is used to insert new customer records and update existing ones. The USS sends records to be inserted or updated through the Operations Support Network (OSN).

This system may require several NCPs, according to NCP capacity, performance requirements and market forecasts. The database is partitioned and each partition is assigned to two NCPs, one primary and

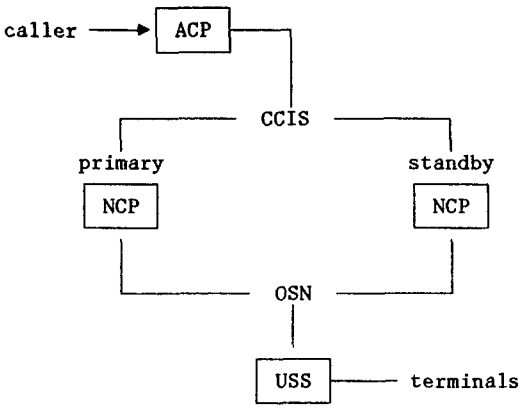


**Figure 3.** AT&T's Stored Program Controlled Network.

one backup, for call processing. A database partition is replicated in an NCP and its backup. One NCP may be a backup to another NCP with respect to one database partition, and a primary to that NCP with respect to another partition of the database. When a primary NCP fails and there are insufficient data at the site to restore its operation, its backup takes over while continuing to function as the primary for another database partition.

Each NCP is constructed using a 3B-20D processor [Mitze et al. 1983]. A 3B-20D consists of two identical processors: One component processor is active and the other is a standby at any given time. Each has its own main memory and control unit. Further, both processors share all the disks, and each processor has indirect access to the main memory of the other.

During normal operation, the active processor updates the main memory of the standby processor, which is ready at all times to take over if the primary should fail. At each NCP, the I/O channel, disk controller, and links to its standby NCP are duplicated. Further, the database partition managed by an NCP is quadruplicated, and stored in four separate sets of disk drives connected to the 3B-20D. The standby NCP in turn keeps four copies of the database partition.

For a customer record to be updated under normal conditions, a transaction is sent to the primary NCP with which the record is associated. The primary NCP checks the

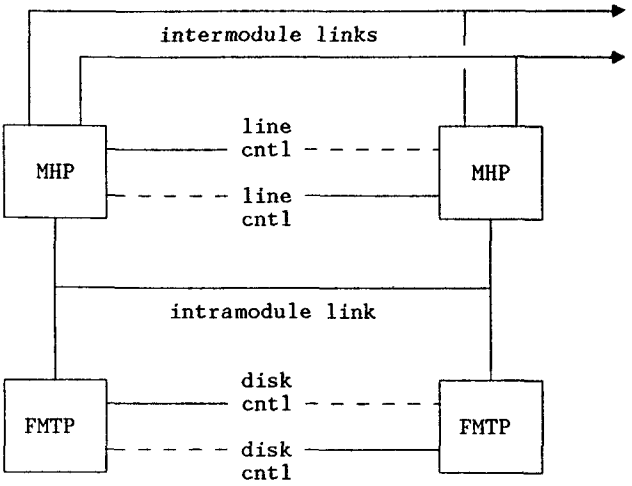


**Figure 4.** A module of Bank of America's DCF.

transaction for consistency and authorization; if it is valid, the primary NCP logs the transaction on the disk and sends an acknowledgment to the user. The primary NCP makes changes to its database and sends the update to its backup. The backup NCP applies the update to its database and acknowledges the primary, at which point the primary inserts a record in the transaction log and finishes the transaction. In the event of a network partition, when a primary NCP is disconnected from its backup NCP due to failure of the communication line, the primary updates its database without requiring agreement from its backup, but maintains a special history log of database changes, which it sends to its backup when communication is restored.

### 3.3 Bank of America's Distributive Computing Facility

Bank of America developed the Distributive Computing Facility (DCF) around 1978 to automate teller functions for customer checking and savings accounts [Good 1983]. By leased lines, branch offices access a customer accounts database in two data centers in Los Angeles and San Francisco. Each data center houses a DCF cluster, which consists of eight DCF modules interconnected by a local-area network. Each DCF module is a local-area network of four GA16/440 minicomputers, which manages one partition of the customer accounts database. Two of the four processors in a DCF module are communications front ends called Message Handling Processors (MHP). The other two are database back ends called File Management Transaction Processors (FMTP). The four processors communicate via a bus called the intramodule link.

As shown in Figure 4, each module is configured such that, under normal operation, each MHP is paired with one FMTP to operate on half the module's lines and database. When one MHP fails, the other takes control of all the lines. When one FMTP fails, the other takes control of the entire database of the module.

The DCF uses a simple scheme for detecting processor failures. Each processor has a watchdog timer, which it periodically resets. If the timer is not reset on schedule, the DCF module shuts the processor down and notifies the peer processor to assume full work load. When an MHP times out on a transaction request to an FMTP, it assumes that the FMTP is dead and starts sending subsequent transactions to the other FMTP. The transaction messages are not logged, and so any transaction in progress on a failed MHP or FMTP is lost.

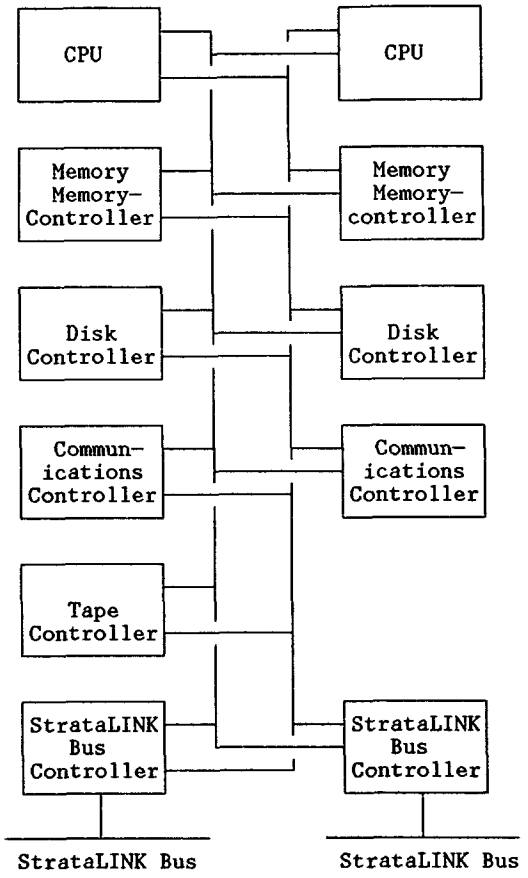


**Figure 5.** A Processing Module of Stratus/32.

### 3.4 Stratus/32 Continuous Processing System

The Stratus/32 Continuous Processing System [Kastner 1983; Stratus 1982] consists of 1–32 Processing Modules, where each Processing Module consists of duplicated CPU, memory, controller, and I/O as shown in Figure 5. The memory may be configured to be redundant or nonredundant, as the two memory subsystems are not paired with the two CPUs. In a redundant configuration, the CPUs read from and write to both memory subsystems simultaneously; in a nonredundant configuration, each memory subsystem becomes an independent unit and the memory capacity is doubled. Each Processing Module has duplicated power supplies. The Processing Modules are connected through a dual-bus system called the StrataLINK.

The duplicated components (CPU and controllers) of a Processing Module each perform the same computation in parallel. Each component (board), in turn, consists of two identical sets of hardware components on the same board. As shown in Figure 6 for a disk controller, a hardware logic compares the results of the computation by the duplicated boards. If the results are identical, they are sent to the bus or device. Otherwise, the results are not sent, the board is automatically disabled, and an interrupt signal is sent to the Stratus VOS operating system. However, processing continues with the duplexed board of the Processing Module.

All detected hardware malfunctions are reported to a maintenance software, which determines the cause and nature of the malfunction. The board is automatically restarted if the malfunction was caused by

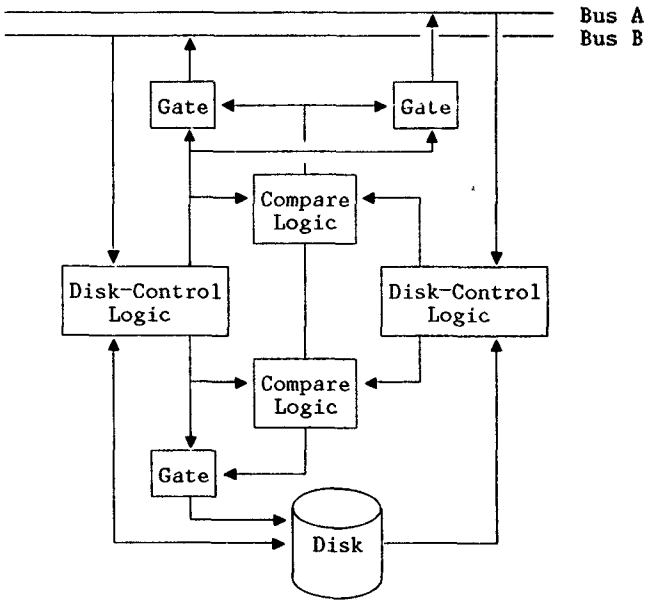


**Figure 6.** Self-checking disk controller in a Stratus/32.

a transient error, whereas permanent errors result in the board remaining out of service and a report being sent to an operator terminal.

The Stratus system supports optional mirroring of disk volumes. It also allows on-line removal and replacement of all duplexed boards and associated peripheral devices; in particular, it allows on-line integration of a new Processing Module. When a duplexed component is replaced, the new component is automatically brought to the same state as its partner. For example, the second disk in a dual-disk system is brought up-to-date while the first disk is used for normal processing. The VOS operating system accomplishes this by writing new blocks of pages to both disks and copying blocks from the first disk to the second concurrently with normal processing.

Further, the Stratus system allows the memory subsystems to be dynamically reconfigured to redundant or nonredundant mode, without taking the Processing Module off line.

The Stratus system supports transaction processing by providing a Transaction Processing Facility (TPF), VOS File System, the StrataNET network communications subsystem, and a Forms Management Facility. The system does not currently support a database management system; TPF invokes the File System to manipulate the database. TPF supports a two-phase commit protocol for on-line distributed transaction processing.

The Stratus system's continuous comparison of the results of computation from duplicated hardware components on the same board significantly reduces the probability of a hardware-induced error from propagating and corrupting the system and data integrity, provided that the hardware that compares the results does not malfunction. Further, the Stratus hardware and operating system provide protection against some system crashes induced by software errors, such as attempts by one user's program to cross another user's address space, to read or write into the operating system, to write into executable code, or to execute data.

The Stratus system does not currently support the concept of a backup subsystem, and hence applications running on Stratus may not survive software-induced crashes. The Stratus fault-tolerance philosophy is based on the view that the hardware can detect errors and automatically shut down a malfunctioning component while an iden-

tical component operating in parallel continues to function, thus ensuring database integrity and providing continuous processing of user requests. In my opinion, this approach does not safeguard the system against crashes induced by a class of errors that even the most sophisticated operating systems cannot cope with. For instance, IBM's System/370 and its Multiple Virtual System (MVS) operating system [IBM 1979] provide extensive measures to detect hardware and software failures and to repair and recover from them. Yet, applications running on MVS, and MVS itself, do occasionally crash, usually as a result of software failures.

### 3.5 Auragen System 4000

The Auragen System 4000, developed at Auragen Systems Corp., New Jersey, consists of 2–32 clusters of tightly coupled multimicroprocessors interconnected by a dual-bus system [Gostanian 1983]. Each cluster consists of three MC68000s, its own local memory, several types of I/O controllers, power supply, and battery backup. One of the 68000s is used exclusively to execute the operating system, whereas the other two are used to execute user tasks. Each cluster periodically broadcasts an "I-AM-ALIVE" message to detect failure of other clusters; the time-out mechanism is used to detect process failures. All peripheral devices are dual ported and are attached to two different clusters. The Auragen system allows disks to be configured in mirrored pairs, and mirroring may be specified on a file basis.

The Auragen database system, called AURELATE, is based on the ORACLE relational database system [Weiss 1980]. As in the Tandem system, Auragen 4000 supports a primary–backup pair of processes, implemented within the AUROS operating system, which is an enhanced version of UNIX III. To reduce the number of checkpoint messages, the AUROS operating system sends a collection of transaction messages to both the primary and the backup. After the primary has processed a predetermined number of these transaction messages, the backup is notified to process the checkpoint message. Once the backup finishes processing the checkpoint message, it is removed from the message queue.

The Auragen system's current recovery technique is based on the roll-forward approach. When the backup takes over for the primary, it begins execution at the last point of synchronization with the primary. That is, it begins with the last checkpoint message in its message queue. A technique has been proposed that does not allow the backup to redo work that already may have been done by the primary before the crash [Gostanian 1983].

The recovery technique requires an undo log to allow transaction abort. However, a redo log is optional and is used for recovery from a single disk crash, when disk mirroring is not used. Since a backup process will redo in-progress transactions, Auragen's proposal for commit processing does not call for the Write-Ahead Log protocol. However, this leaves the system unprotected from simultaneous failures of both the primary and backup processes.

### 3.6 System D Prototype

System D is a distributed transaction-processing system designed and prototyped at IBM Research, San Jose, as a vehicle for research into availability and incremental growth of a locally distributed network of computers [Andler et al. 1982]. The system was implemented on a network of Series/1 minicomputers interconnected with an insertion ring, and was the predecessor to the Highly Available Systems project currently under way at IBM Research, San Jose [Aghili et al. 1983; Kim 1982]. Although System D is not itself a highly available system, its rather novel transaction-processing and failure-diagnosis strategies warrant discussion here.

The System D transaction-processing software consists of three distinct types of modules: application, data manager, and storage manager modules. A module is a function that exists in a node and may consist of one or more processes called agents. The application module, called A, provides user interfaces for interactive users or application programmers. The data manager module, called D, transforms the record-level requests from the A module to

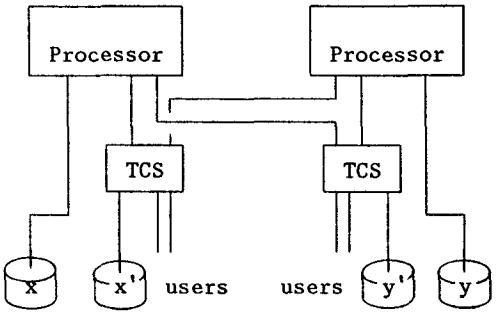


**Figure 7.** Processor pair in System D.

page-level requests for the storage module. All the changes made by an application are kept locally in the data manager module, which sends them to the storage manager only when the application commits the data changes to stable storage. The storage manager module, called S, supports multiple concurrent transactions against the physical database and database recovery from failures.

As shown in Figure 7, terminals and shared disks are connected to a pair of processors through an electronic switch, called a Two-Channel Switch (TCS). Only one of the processors has access to the shared disks and terminals at any given time. The processor periodically resets the timer in the TCS; if this does not occur on schedule, the TCS switches the shared disks and terminals over to the other processor. Thus, rather than requiring the processors to communicate with each other, System D gives the TCS the task of detecting the failure of a processor. Moreover, the TCS prevents a processor that is declared dead from writing to the disk.

System D provides a software subsystem called the Resource Manager (RM), which is responsible for diagnosing and taking appropriate actions to recover from the failures of modules and agents. The basic premise of its design is that the time-out mechanism detects all failures, including deadlock, agent or module crash, communication medium failure, or processor failure. Failures are detected only when service requests are sent.

Rather than the Tandem-like notion of primary and backup processes, System D supports multiple agents of a module running in the same processor. The Resource Manager attempts to bring down and restart failed agents, while normal service requests are handled by other agents. In the event of a processor failure, agents are brought up in the backup processor and all transactions in progress are aborted. The initial program load (IPL) of a low-end processor normally takes under 1 minute, which was considered a reasonable recovery duration, and System D was designed to avoid the overhead of maintaining synchronized pairs of primary and backup processes.

The failure-diagnosis logic described below was necessary because of the inadequate failure-detection capabilities of the operating system on which System D ran. The RM attempts to diagnose the problem by first attempting to establish communication with the node in which the agent is running. If the response is positive, the RM issues an "ABORT_TRANSACTION" command to the agent. The rationale here is that the service-request message may have experienced a data- or timing-dependent error that may not recur if the transaction is aborted and resubmitted. Successful processing of the "ABORT_TRANSACTION" command also indicates that the agent involved probably has not crashed.

If the transaction cannot be aborted, the RM attempts to bring down and restart the agent, since the code or the control structure of the agent may have been destroyed. If the agent cannot be brought down, the RM attempts to shut down and restart the module itself, since the control structures

shared by the agents of the module may have crashed. If the RM fails to shut down and restart the module, probably the remote RM or the operating system has crashed, in which case an IPL must be executed remotely for the node in question.

Now, if the RM fails to establish communication with the node in which the resource resides or if the remote IPL was not successful, it will try to communicate with the RM in the backup node of the resource, since the TCS has probably switched. If the TCS has not switched over, the RM attempts to remotely execute an IPL for the node in which the resource resides. The reason is that initially it may have failed to communicate with the RM in that node because the RM, the CSS, or the operating system in that node may have crashed.

The standard two-phase commit protocol allows each node to unilaterally abort the transaction as long as the commit protocol has not entered the second phase. The design of System D recognizes that this privilege, often called site autonomy, is not so important in a locally distributed environment, and implements a different commit protocol [Andler et al. 1982]. In a sense, the System D protocol requires just a single phase and as soon as the commit coordinator decides to commit; no other node may abort the transaction.

In System D, actual updates to the database are not made until the transaction commits. At transaction commit, the transaction's log, maintained by the D module, is sent to the S module that is designated the commit coordinator, the first S module that receives page request from the D module. The commit coordinator writes the log to the disk and acknowledges the D module. The D module then sends "commit" messages to the other participating S modules. Each participant writes its log to the disk and then makes database changes.

The recovery procedure for this commit protocol is as follows. Upon restart, an S module re-DOes the changes in its local transaction log. The module then requests complete logs from any commit coordinators for transactions that are known to them but unknown to the recovering module, and runs these transactions serially in any order.

## 4. TIGHTLY COUPLED SYSTEMS WITH A SHARED DATABASE

The Synapse N+1 Computer System is an on-line transaction-processing system, developed by Synapse Computer Corporation [Jones 1983], which provides a dual path from a processor to the database stored in secondary storage devices. As Figure 8 shows, Synapse N+1 is a tightly coupled multiprocessor system with shared memory, in which processing is divided between general-purpose processors (GPP) and I/O processors (IOP), each of which is based on the Motorola 68000. Currently, up to 28 processors may be attached to a dual-bus system called the Synapse Expansion Bus.

GPPs execute user programs and most of the Synapse's Synthesis operating system from the shared memory. IOPs each manage up to 16 I/O controllers or communications subsystems. IOPs have direct memory access (DMA) capability to the shared main memory, but execute part of Synthesis from their own local memory. The Advanced Communications Subsystem (ACS) is a 68000-based communications controller, and the Multiple-Purpose Controller (MPC) is a controller for various devices. Whereas the Tandem NonStop system powers each processor with a separate power supply, the entire Synapse N+1 is powered by one set of duplicated power supplies.

Synapse enhances availability simply by having one additional processor, disk controller, and disk drive than what is necessary for satisfactory performance. This extra component is not an idle backup; it is used for normal transaction processing. When a component fails, the system sometimes has to restart, and reconfigure itself without the failed component.

The Synapse system also supports mirroring of disks. Mirroring is supported in the IOP, where two disk writes are issued for each page of data to be written to the disk. It is interesting that the Synapse system allows mirroring of disks on a logical volume basis, rather than physical volume.

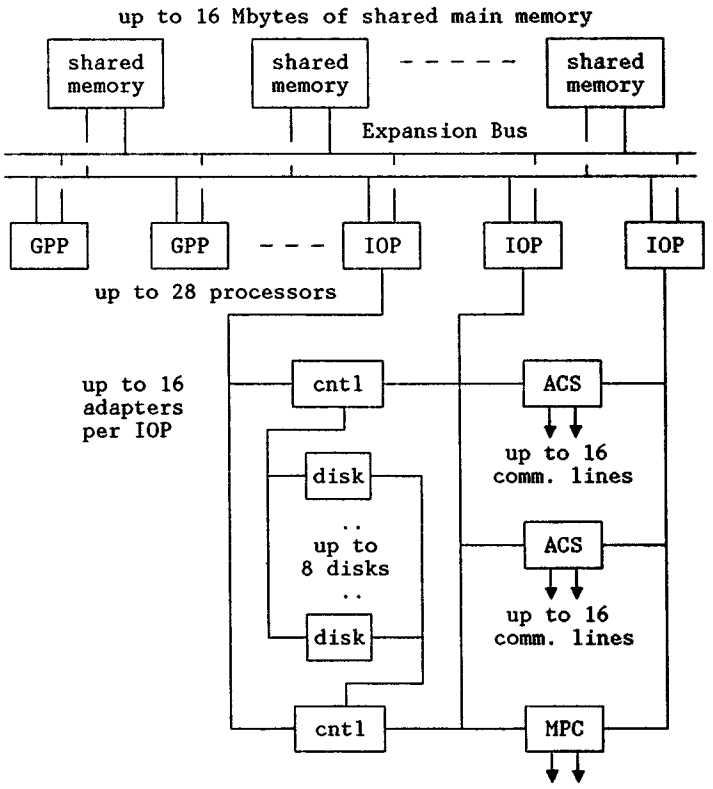


**Figure 8.** Synapse N+1 System architecture.

A logical volume may be all or part of a physical volume. Mirroring a logical volume is more flexible than mirroring a physical volume; for example, it allows storage on the same physical volume of a separate database that does not require high availability (and its associated overhead).

Synapse N+1 supports recoverable transactions using the Write-Ahead Log protocol. The Synapse database manager is a relational system. Further, the database may be migrated from one physical disk volume to another and may undergo structural changes without taking the applications off line.

The Synthesis operating system incorporates various techniques to optimize the performance of production transaction-processing systems. One is the placement of its database system directly above the kernel of Synthesis, rather than on top of the file system as has been the case with most database systems. The reason for this decision was to reduce the overhead associated with I/O requests from the database system to the operating system, both to retrieve data from disk, and to store the log of database changes after transaction processing.

An interesting side benefit of this approach is that higher levels of the Synthesis operating system can use the capabilities of the database system to query and manipulate data about operating system objects, such as files and devices. In particular, a higher level of the Synthesis operating system, called the transaction-processing domain, records the state of currently active applications in the database. An application consists of a number of programs; each program takes as input a screenful of information, processes it, and outputs a screen. The state information of an active application consists of a user identification, the identification of the current screen, the contents of its variable fields, and the next

program to execute. During restart following a crash, the transaction-processing domain creates and dispatches a task for each terminal using this state information.

Another performance optimization in Synthesis is the elimination of task-switch overhead by replacing task switches with cross-domain (address space) calls. In other words, each of the domains (layers) of Synthesis has direct addressability to a partition of the segmented virtual address space, and a request from a task for an operating system service is implemented as a jump to the address space of the server's domain.

In order to reduce contention on the shared memory, Synapse adopted a caching scheme in which modifications to the cache in each GPP are not written through (to the shared memory), and fetch requests for the portion of the shared memory read and modified by another GPP in its cache are resolved between the processors.

The Synapse system has implemented failure detection, reconfiguration, and restart procedures in a read-only memory (ROM). Its failure detection distinguishes two classes of failures: process-fatal failure and system-fatal failure. A process-fatal failure causes the process and its associated transaction to be aborted and restarted, whereas a system-fatal failure results in a restart of the entire system.

The kernel of the operating system is capable of recognizing any failures caused by internal machine checks and is responsible for initiating the reconfiguration and restart of the system. In particular, it activates self-test code to verify whether each major hardware component is operational. After any failed components are configured out of the system, each domain (layer) of the operating system is reinitialized. Since the database system is a layer directly above the kernal operating system, transaction restart and recovery take place at this time.

Since the processing is divided between the GPPs and IOPs, and the IOPs run part of the operating system from their own local memories, the risk of total system failure due to corruption of the operating system is somewhat reduced. However, as long as the GPPs execute most of the operating system out of shared main memory, this risk is still present. This problem cannot be resolved even if shared memory is made redundant. Synapse N+1 does not support a redundant shared memory.

## 5. LOOSELY COUPLED MULTIPROCESSOR SYSTEM WITH A SHARED DATABASE

General Electric's MARK III Cluster File System and Computer Consoles' Power System use the loosely coupled multiprocessor architecture, in which each processor may access any of the disks. The AMOEBA project [Traiger 1983] at IBM Research, San Jose, also uses the same architecture, but is still in the research stage.

### 5.1 GE MARK III

MARK III is a time-sharing system developed by the Information Services Business Division of General Electric to provide its customers with local-call access to MARK III computing capabilities [Weston 1978]. The primary objectives of the system were high availability, reliability, and maintainability. Three supercenters (computing centers), located in Ohio, Maryland, and Amsterdam, provide computing power to the users. The computing facilities at a supercenter typically consist of front-end processors, MARK III foreground processors, and MARK III background processors, as shown in Figure 9.

The front-end processors are network front-end processors (central concentrators). The foreground processors support interactive users, whereas the background processors provide batch-processing capabilities. The foreground and background processors are interconnected via a Bus Adapter, which allows job and file movement between the foreground and background systems.

The Bus Adapter consists of a microprocessor, programmable read-only memory (PROM) control memory, and channel interfaces to the background systems. The microprocessor polls each of the channel interfaces for data transfer requests; each request is fully processed before the next request is serviced.

In 1975 GE developed new software to allow a single foreground processor of the

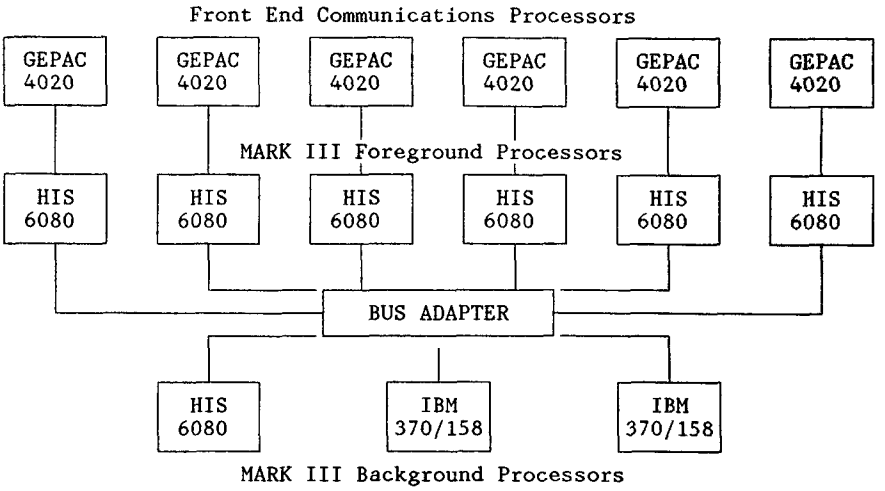


**Figure 9.** GE's MARK III Cluster File System architecture.

MARK III system to access more than one disk system at a time. This new system, called the Cluster File System, controls concurrent access to multiple-disk systems from multiple foreground processors by maintaining the access-conflict tables in one stable memory device accessible to all foreground processors. The Scratch Pad (SPAD) was developed to provide the high reliability, nonvolatility, and fast access time that such a memory device requires. All data transfers to and from the disk systems take place over normal I/O channels, but access-control decisions are made through use of the access tables in the SPAD.

Since the SPAD is the central point through which all requests are funneled, it was built with redundancy to prevent total failure of the cluster system resulting from the SPAD and Bus Adapter failure. As shown in Figure 10, the MARK III Cluster File System has two Bus Adapters, primary and secondary. Each foreground processor has access paths to both Bus Adapters. The memory and devices in SPAD dedicated to each disk system are themselves duplicated. Each of the Bus Adapters can access both the primary and backup elements of SPAD. The Bus Adapter was augmented to support the functions of, and dual access paths to, the SPAD, and the microcode in the Bus Adapter does dual read and write to the SPAD memory.

The SPAD contains 16 memory and logic devices (8 primary and 8 backup), each for a separate disk system. The Maryland supercenter supports seven disk systems. Each device contains the access-conflict table, which indicates whether a particular file in a disk system is in use and if so whether the file is sharable by other users.

One device, called the Cluster Control device (CLUSCON), is used for such global functions as processor status and recovery status. Each foreground processor periodically places its status in CLUSCON and checks to see if any other foreground processor has failed to do so in the previous interval. If it finds that another processor failed to update its status, it proceeds to clean up all resources belonging to the failed processor.

The MARK III Cluster File System is protected from single failures by redundancy in the interconnections between the front-end and foreground processors and between the foreground and background processors. The front-end processors (network central concentrators) are connected to remote concentrators, to which user terminals are connected. The connection between the central concentrators and remote concentrators is accomplished via redundant network-switching computers.

An obvious drawback of the MARK III Cluster File System is that the SPAD may become a performance bottleneck, espe-

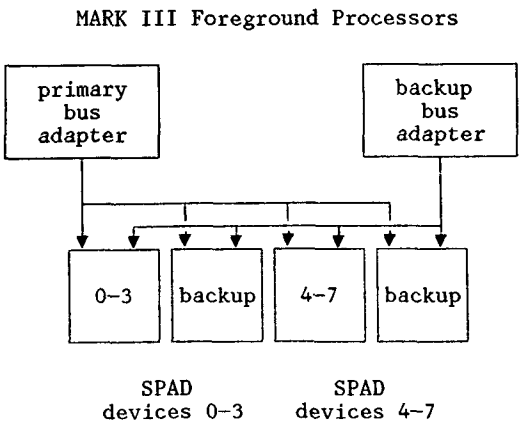


**Figure 10.** Redundancy in the Bus Adapter and SPAD.

cially with an expanded system, since each front-end processor must access it before accessing a shared file.

### 5.2 Computer Consoles' Power System

The Power Series systems have been developed at Computer Consoles, Inc., Rochester, New York [West et al. 1983]. The system is based on Motorola 68000s, and consists of a number of application processors and two coordination processors with front-end processors. The processors communicate via a dual-bus system called the Data Highway, as shown in Figure 11.

Each application processor (AP) is directly connected to all disks, and independently executes different user applications in parallel. The interprocessor coordination controllers (ICC) synchronize global operations among the APs, which consist mostly of lock requests to gain access to the shared database and system status changes due to reconfiguration. At any given time, one ICC is active and the other is a standby. The standby ICC is the only idle component of the system. The front-end processors (FEP) perform screen formatting, distribute transactions to APs, receive replies from APs, and assist in recovery from some system failures. Further, FEPs attempt to balance the load on the APs by distributing the transactions to the APs on the basis of application configuration and flow control information.

Unlike many systems that implement transaction management on top of a general-purpose operating system, the Power System combines process management and transaction management into its PERPOS operating system. The PERPOS operating system supports a number of features to enhance performance of critical applications. To allow concurrent execution of transactions with different response requirements, it provides facilities to fix critical applications in memory and run them before other applications.

Since the database manager in each AP can access the entire database, the Power System only needs the standard concurrency control and recovery techniques used for a central database. In particular, it does not need the coordinated commit protocol required by loosely coupled multiprocessors with partitioned databases.

Transaction recovery is done in a straightforward manner. The AP that receives the transaction from the FEP logs the transaction on the disk. When the FEP detects that the AP crashed, it requests another AP to abort the transaction and restart it from the log of the failed AP.

As in other systems, interprocessor message time-outs are used to detect processor failures. In addition, the primary ICC periodically polls the APs and FEPs to detect failures of processors that do not happen to be in communication with other processors. The ICC supports on-line system reconfiguration after a disk crash and on-line integration of new or repaired disks.

The primary ICC does not keep the standby up-to-date on the global lock table and the system configuration information. Rather, when the primary ICC fails, the standby requests status and lock information from all the APs and reconstructs the global lock table.

In order to reduce the communication overhead resulting from lock requests to the ICC, the system distinguishes shared files and nonshared files. When an AP opens a file for the first time, it considers the file nonshared, and does not make a lock request to the global lock manager in the ICC. When an AP opens a file currently owned by another AP, that file becomes a

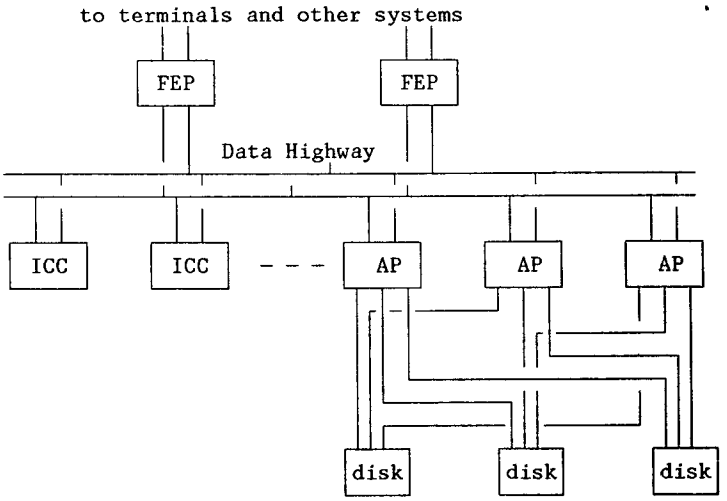


**Figure 11.** Power System architecture.

shared file under the jurisdiction of the global lock manager.

One potential drawback of the Power System architecture is that, as the number of APs increases, the ICCs may become a performance bottleneck. Further, the Power System's performance may be enhanced by generalizing its locking technique to the lock hierarchy technique similar to that found in IMS/VS [Strickland et al. 1982]. In this scheme, the database is logically partitioned, with each partition assigned to a different data manager that can acquire and release locks on data objects within its partition. The data manager consults the global lock manager only when it must lock and unlock objects outside its partition. This strategy potentially allows transfer of updated data pages from the buffer pool of one partition's data manager to another data manager. Traiger [1983] speculates on this in more detail.

## 6. REDUNDANT COMPUTATION SYSTEMS: SYNTREX'S GEMINI FILE SERVER

GEMINI is a file server developed by Syntrex, Inc., with the objective of uninterrupted operation, without backup, in the event of a single failure of any hardware or software component [Cohen et al. 1982]. Up to 14 workstations (word-processing terminals) connected to a GEMINI file server on a local-area network can share files and printers. A GEMINI system may be connected to other GEMINI systems through an Ethernet-like network.

As shown in Figure 12, GEMINI consists of two identical halves. Each half has the Aquarius interface (AI), a disk controller (DC), and a shared memory (SM), as well as a secondary storage system. The AI is a communications subsystem, implemented on an Intel 8088 microprocessor, that operates between GEMINI and the workstations, and between the AIs in each half. The DC, implemented on an Intel 8086, provides file storage and management for the workstations. Communication between an AI and a DC takes place through the shared memory (SM).

The two halves of GEMINI perform identical computations. Each half receives the same request from the workstations and retrieves and updates files in its secondary storage. When both halves are operational, one is designated the master and the other the slave. The only difference between them is that only the results from the master are returned to the workstations.

Each half continuously monitors the well-being of the other half. When one half

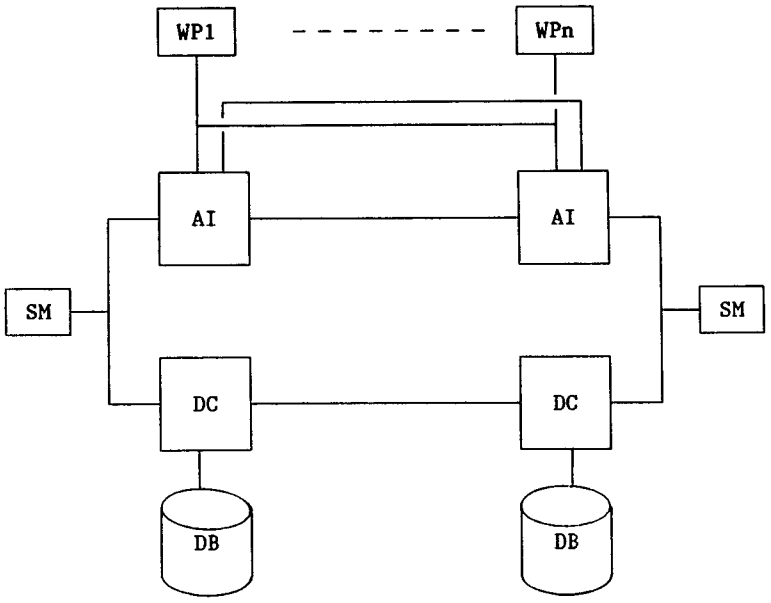


**Figure 12.** Syntrex's GEMINI file server architecture.

crashes, it is powered off and the other half continues as the master. While one half is down, its secondary storage system becomes out of date; when it is repaired, a utility is run to bring its secondary storage system up-to-date. The active system is suspended until copying is completed. Thus GEMINI does not support on-line reintegration of repaired components.

A time-out mechanism is used in the communication between the AIs of GEMINI and the workstations. If the workstation times out on its request, it retransmits the request; if GEMINI times out, it goes to receive mode and waits for the workstation to time out and retransmit the request. This means that the workstation time-out value is longer than the GEMINI time-out.

Another aspect of the reliability measures incorporated in the AI is the periodic self-checks, including auditing of input buffers, checking of the clock, memory tests, and testing of the DC and the communication link between the AIs. If any test fails, the AI logs the failure and, if possible, informs the other AI.

The heart of mutual checking in GEMINI is embedded in the AI–AI communications procedures. The availability and reliability of GEMINI critically depends on the assumption that the disk controllers (DC) of both halves receive and perform the same computation, and therefore that the two secondary storage systems are left with the same data at the end of computation. Since it is possible for one AI to receive a correctly transmitted request while the other receives the request with a transmission error, the two AIs are required to exchange status information about the requests that they received. The request is processed only when both AIs agree that they received the same request [without a character redundancy check (CRC) error].

Similarly, the AIs exchange information about the results of the computation to verify their correctness. It is possible for one AI to have completed a computation before the other is done. In such a situation, the slow half sends a notice that it is "working on the request," to prevent the other half from concluding that it is down. GEMINI takes precautions against an infinite sequence of "I am done" and "I am working on it" messages between the AIs. Since the results of computations may be too long, sometimes only the types of results are exchanged between the AIs. Therefore it appears that sometimes GEMINI may not detect conflicting results generated by the

two AIs. Further, when GEMINI does detect conflicting results, it arbitrarily assumes that the master is correct. A meaningful vote really cannot be taken with less than three processors.

## 7. A FRAMEWORK FOR THE MANAGEMENT OF SYSTEM CONFIGURATION

It is clear from the preceding discussions that the design of existing systems has been guided by the single-failure assumption; that is, these systems can become unavailable if a software or hardware component fails while another related component has failed. Despite the general success of some of these systems, notably the Tandem system, the single-failure assumption may not be valid. Future systems may be required to tolerate multiple concurrent failures.

Most existing systems and those that are currently being developed are constructed with mini- and microcomputers. If relatively expensive medium- to high-end processors were to be used, it might not be economically feasible to keep spares around for use as replacements for malfunctioning processors. In that case, the mean time to repair such processors could be relatively long, increasing the probability that other subsystems or processors might go down before the failed processors are repaired.

In addition, the software in most existing systems was developed from scratch to run on minicomputers and to support only prospective new customers. Existing database and operating systems required by medium- to high-end processors tend to be complex, and the mean time between failures for these systems due to software-induced failures is expected to be shorter than that for simple transaction-processing systems that run under relatively simple operating systems.

If a system is to be continuously operational, it must guarantee availability not only during multiple concurrent failures of software and hardware components, but also during on-line changes of software and hardware components and on-line physical reconfiguration of the database and database backups. The latter problems do not appear to be properly addressed by most systems. Syntrex GEMINI, for example, requires system shutdown when a repaired processor is reintegrated into the system, and in general most systems force applications off line when the database is physically reorganized.

An architecture of a distributed software subsystem that can serve as a framework for constructing database application systems to meet most availability requirements is outlined in the remainder of this section. This software subsystem is called an *auditor*. The description of the functions and architecture of the auditor given here is based largely on my own research. A design based on this is currently being implemented for the Highly Available Systems project at IBM Research, San Jose [Aghili et al. 1983].

An auditor is a framework for total coherent management of software and hardware components of a highly available distributed system. It will serve as the repository of failure reports from various components of the system and reconfiguration requests from the system operator (for on-line changes). It will analyze the status of all resources it manages, and compute the optimal configuration of the system in response to multiple concurrent failures of components and requests for load balancing. Further, it will initiate system reconfiguration and monitor its progress in order to effect mid-course correction of a reconfiguration that does not succeed, and finally, it will diagnose a class of failures that other components fail to recognize.

From the discussions of the survey portion of this paper, it should have become clear to the reader that most systems provide many of the functions outlined for the auditor. All systems discussed support automatic detection of process and processor failures, followed by automatic switchover to a backup or notification to the service center. Many systems also support on-line integration of new or repaired software (process) and hardware components (processor, I/O controller, disk drives).

However, the implementation of the auditor functions in many systems suffers from two shortcomings. First, these functions have often been implemented as a loose collection of specialized routines, rather than as a single coherent subsystem. Sec-

ond, the functions often are implemented to tolerate only a single failure of the system resources; as a result, the systems cannot cope with multiple failures, even when they have sufficient hardware redundancy.

Within this framework, one auditor will reside in each processor, but only one of the auditors may be designated as the audit coordinator. As pointed out by Garcia-Molina [1982], to allow each auditor to initiate reconfiguration may cause confusion or result in a less than optimal system configuration, and the notion of the audit coordinator is therefore central to the operation of the audit mechanism. If the coordinator crashes, a new coordinator must first be established, either by an election, as suggested by Garcia-Molina [1982], or by means of a dynamic succession list to which all the auditors have previously agreed [Kim 1982]. A succession list contains the system-wide unique rank for each auditor to indicate which subordinate auditor will take over the responsibilities of the audit coordinator once the corrent coordinator crashes. The authenticated version of the Byzantine consensus protocol proposed by Dolev and Strong [1982] and the version-number method discussed by Kim [1982] are possible techniques to ensure agreement on the succession list in the presence of failures of communication lines, processes, and processors.

The audit coordinator should be responsible for analyzing the states of all other subordinate auditors, analyzing the reports and initiating system reconfiguration, the replacement of failed subsystems or processors with their backups, and (re)integration of repaired (or new) subsystems or processors. The audit coordinator will also serve as the arbitrator of conflicting reports from different auditors, and is responsible for maintaining a stable configuration database which contains information about the status and physical location of each of the subsystems.

Such an auditor may be implemented as a collection of six asynchronous tasks: configuration-database task, audit task, reconfiguration task, state-report task, diagnose task, and operator-control task. The task structure of an auditor and the flow of control among the auditor tasks are illustrated in Figure 13.

The configuration-database task maintains a consistent and up-to-date configuration database. All queries and updates to the configuration database by other auditor tasks are directed to this task. Changes to the configuration database are exclusively handled by the configuration-database task of the audit coordinator. After each change, the configuration-database tasks of the subordinate auditors are given the most up-to-date copy of the configuration database. Although the configuration database viewed by a subordinate auditor may be temporarily out of date, consistency of the system configuration is not compromised since critical decisions can only be made by the audit coordinator.

The audit task is primarily responsible for coordinated surveillance of process and processor failures. The audit task of a subordinate auditor collects the local state reports from the state-report task, and delivers them to the audit task of the coordinator. The audit task of the coordinator receives these state reports from subordinate auditors, and analyzes them to determine whether any process or processor has failed.

One way for the audit coordinator to receive state reports is to periodically poll the subordinate auditors. An interesting alternative to polling by the audit coordinator is the approach proposed by Walter [1982], which requires each "auditor" to periodically send an "I-AM-ALIVE" message to its immediate neighbor on a virtual ring of "auditors." When an auditor does not receive the "I-AM-ALIVE" message within a certain time interval, it may request the audit coordinator to initiate reconfiguration.

When the audit task of the coordinator decides that a failure has occurred, or when it receives a reconfiguration request from the operator control task, it activates the reconfiguration task. Changes to system configuration are reflected in the configuration database after the reconfiguration task completes. The audit task of the coordinator is also responsible for preparing the succession list and securing its trans-

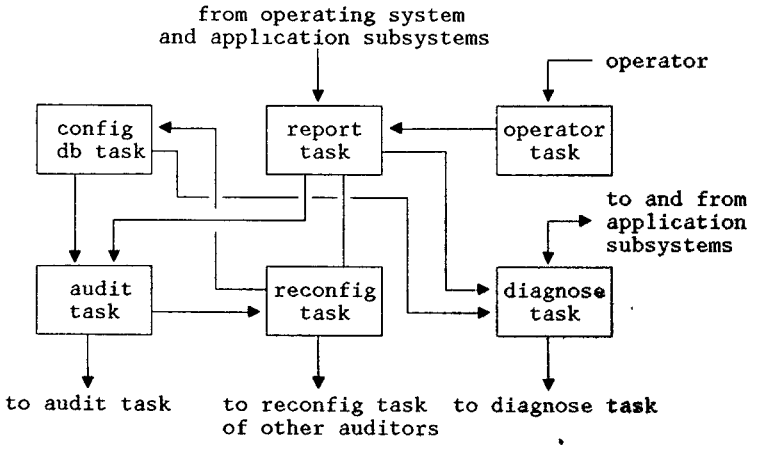


**Figure 13.** The task structure of an auditor.

mission to the audit tasks of subordinate auditors.

The reconfiguration task is responsible for processing the reconfiguration requests received from the audit task of the audit coordinator. Upon receiving a request, it initiates reconfiguration and monitors its successful completion. If the audit coordinator fails during a reconfiguration process, then the new coordinator completes the reconfiguration. Any change to the system configuration is stored in the configuration database. A report is sent back to the audit task of the coordinator upon completion of a reconfiguration request.

The state-report task receives state reports from the local database subsystems, interprocessor communication subsystem, and the operating system, as well as reconfiguration requests from the system operator. It manages this collection of state reports and makes it available to the local audit task and other auditor tasks.

The diagnose task is responsible for exposing process failures that may have gone undetected by the operating system or the process itself. It may also collect complaints from database subsystems about possible misbehavior of other subsystems (e.g., time-outs and lost messages). To establish availability or misbehavior of subsystems, it may resort to functional tests, such as checking if messages pass through queues, tracking down the messages exchanged by subsystems, and executing simple transactions whose results are known. This sometimes will require collaboration among the diagnose tasks of several auditors. Its findings are packaged into a state report and sent to a database subsystem (e.g., to report the loss of a message and the need for its retransmission) or to the state-report task (e.g., to report a subsystem crash that has remained undetected by the operating system).

The complexity of the diagnose task depends on the extent to which other software subsystems assist in identifying failures. If the operating system is capable of serving as a repository of hardware and program failures, and application software contains a reasonable amount of defensive code to detect impossible software states, the diagnose task can be quite simple. If this is not the case, the diagnose task may have to be designed in a manner similar to the Resource Manager of System D to expose the nature of failures.

The operator-control task is the auditor's interface to the system operator. By using this interface, the operator may query the system configuration or request a system reconfiguration.

## 8. CONCLUDING REMARKS

This paper has provided a survey and analysis of the architectures and availability techniques used in database application systems designed with availability as a pri-

mary objective. We found that all existing systems have been designed under the single-failure assumption, but that some of the systems contain single points of failure and cannot survive failures of some single components. Rather, these systems are designed to restart quickly to provide high overall availability.

All of the systems may be classified into four distinct architectures: loosely coupled multiprocessor systems with a partitioned database, tightly coupled multiprocessor systems with a shared database, loosely coupled multiprocessor systems with a shared database, and multiprocessor systems that perform redundant computations and compare the results. Of these, the loosely coupled multiprocessor with either a partitioned or shared database appears to offer the best framework for building a highly available system. Either architecture is conducive to incremental expansion and offers a natural boundary between data managers, which makes it difficult for a malfunctioning data manager to corrupt other data managers. Of course, both architectures require a low-overhead communications subsystem to process user requests that require access to more than one database partition. A difficult problem posed by the partitioned database, however, is that of deciding which database partition should be owned by which processor, so as to minimize the volume of processing requiring collaboration among more than one data manager. Potential drawbacks of the shared database approach are contention on shared disks and the difficulty of coordinating the global locking and the logging of database changes.

The tightly coupled multiprocessor architecture with a shared database compromises availability in favor of a potential performance advantage over the loosely coupled system with a partitioned database. However, before this potential advantage in performance can be realized, the problems of contention among processors for the use of shared memory and other shared resources, especially as more processors are added, must be resolved.

Although the redundant-computation approach may make sense for applications such as spacecraft and industrial process control, it does not appear particularly suitable for typical database applications. The exchange of status information among the replicated tasks to verify correctness of each input and output could seriously impede the performance of a production system.

A continuously operational database application systems must guarantee availability not only during multiple concurrent failures of software and hardware components but also during on-line changes of software and hardware components, on-line physical reconfiguration of the database, and generation of backup databases. An architecture was outlined in Section 7 of a distributed software subsystem called an auditor, which can serve as a framework for constructing database application systems to meet these requirements.

## ACKNOWLEDGMENTS

Irv Traiger (IBM Research, San Jose) read an initial version of this paper and made numerous valuable suggestions that helped to significantly improve the technical contents and presentation of the paper. John West of Computer Consoles, Richard Gostanian of Auragen Systems, Bob Good of Bank of America, Steve Jones of Synapse Computer, and Dave Cohen of AT&T Bell Labs kindly provided answers to various technical questions I had about their systems. The referees and Randy Katz made various constructive comments on an earlier version of this paper. Finally, a technical editor did a superb job of shaping the paper into publishable form.

## REFERENCES

AGHILI, H., ASTRAHAN, M., FINKELSTEIN, S., KIM, W., MCPHERSON, K., SCHKOLNICK, M., AND STRONG, M. 1983. A prototype for a highly available database system. IBM Res. Rep. RJ3755, IBM Research, San Jose, Calif., Jan. 17.

ANDLER, S., DING, I., ESWARAN, K., HAUSER, C., KIM, W., MEHL, J., AND WILLIAMS, R. 1982. System D: A distributed system for availability. In *Proceedings of the 8th International Conference on Very Large Data Bases* (Mexico City, Mexico D.F., Sept.), pp. 33–44.

BARTLETT, J. 1978. A nonstop operating system. In *Proceedings of the 1978 International Conference on System Sciences* (Honolulu, Hawaii, Jan.).

BERNSTEIN, P., AND GOODMAN, N. 1981. Concurrency control in distributed database systems. *ACM Comput Surv. 13*, 2 (June), 185–221.

BORR, A. 1981. Transaction monitoring in ENCOMPASS (TM): Reliable distributed transaction

processing. In *Proceedings of the 7th International Conference on Very Large Databases* (Cannes, France, Sept. 9–11). IEEE, New York, pp. 155–165. ACM, New York.

COHEN, D., HOLCOMB, J. E., AND SURY, M. B. 1983. Database management strategies to support network services. *IEEE Q. Bull Database Eng. 6*, 2 (June), special issue on Highly Available Systems.

COHEN, N. B., HALEY, C. B., HENDERSON, S. E., AND WON, C. L. 1982. GEMINI: A reliable local network. In *Proceedings of the 6th Workshop on Distributed Data Management and Computer Networks* (Berkeley, Calif., Feb.), pp. 1–22.

DOLEV, D., AND STRONG, H. R. 1982. Polynomial algorithms for multiple processor agreement. In *Proceeding of the 14th ACM Symposium on Theory of Computing* (San Francisco, May 5–7). ACM, New York, pp. 401–407.

ELECTRONIC BUSINESS 1981. October issue.

GARCIA-MOLINA, H. 1982. Elections in a distributed computing system. *IEEE Trans Comput. C-31*, 1 (Jan.), pp. 48–59.

GOOD, B. 1983. Experience with Bank of America's distributive computing System. In *Proceedings of the IEEE CompCon* (Mar.). IEEE Computer Society, Los Angeles.

GOSTANIAN, R. 1983. The Auragen System 4000. *IEEE Q. Bull. Database Eng. 6*, 2 (June), special issue on Highly Available Systems.

GRAY, J. N. 1978. Notes on data base operating systems. IBM Res. Rep. RJ2188, IBM Research, San Jose, Calif., Feb.

GRAY, J. N., MCJONES, P., BLASGEN, M., LINDSAY, B., LORIE, R., PRICE, T., PUTZOLU, F., AND TRAIGER, I. 1981. Recovery manager of a data management system. *ACM Comput. Surv. 13*, 2 (June), 223–242.

HAERDER, T., AND REUTER, A. Principles of transaction-oriented database recovery. *ACM Comput. Surv. 15*, 4 (Dec.), 287–317.

IBM 1979. OS/VS2 MVS multiprocessing: An introduction and guide to writing, operating, and recovery procedures. Form No. GC28-0952-1, File No. S370-34, International Business Machines.

IBM 1980. IBM System/370 principles of operation. Form No. GA22-7000-6, File No. S370-01, International Business Machines.

IEEE 1983. *IEEE Q Bull. Database Eng. 6*, 2, (June), special issue on Highly Available Systems.

JONES, S. 1983. Synapse's approach to high application availability. In *Proceedings of the IEEE Spring CompCon* (Mar.). IEEE Computer Society, Los Angeles.

KASTNER, P. C. 1983. A fault-tolerant transaction processing environment. *IEEE Q. Bull. Database Eng. 6*, 2 (June), special issue on Highly Available Systems.

KATSUKI, D., ELSAM, E. S., MANN, W. F., ROBERTS, E. S., ROBINSON, J. G., SKOWRONSKI, F. S., AND WOLF, E. W. 1978. Pluribus—An operational fault-tolerant multiprocessor. *Proc. IEEE 66*, 10 (Oct.) 1146–1159.

KATZMAN, J. A. 1977. System architecture for NonStop computing. In *Proceedings of the CompCon* (Feb). IEEE Computer Society, Los Angeles. pp. 77–80.

KATZMAN, J. A. 1978. A fault tolerant computer system. In *Proceedings of the 1978 International Conference on System Sciences* (Honolulu, Hawaii, Jan.).

KIM, W. 1982. Auditor: A framework for highly available DB/DC systems. In *Proceedings of the 2nd Symposium on Reliability in Distributed Software and Database Systems* (Pittsburgh, Pa., July). *IEEE* Computer Society, Silver Spring, Md., pp. 76–84.

KINNUCAN, P. 1981. An industrial computer that 'can't fail.' *Mini-Micro Syst.* (Mar.), 29–34.

KOHLER, W. 1981. A survey of techniques for synchronization and recovery in decentralized computer systems. *ACM Comput. Surv. 13*, 2 (June), 149–183.

MITZE, R. W., ET. AL. 1983. The 3B-20D processor and DMERT as a base for telecommunications applications. *Bell Syst. Tech. J. Comput. Sci. Syst. 62*, 1 (Jan.), 171–180.

SPENCER, A. C., AND VIGILANTE, F. S. 1969. System organization and objectives, *Bell Syst. Tech. J.* (Oct.), 2607–2618, special issue on No. 2 ESS.

STRATUS 1982 *Stratus/32 System Overview.* Stratus Computers, Natick, Mass.

STRICKLAND, J. P., UHROWCZIK, P. P., AND WATTS, V. L. 1982. IMS/VS: An evolving system. *IBM Syst. J. 21*, 4, 490–510.

SWAN, R., FULLER, S. H., AND SIEWIOREB, D. P. 1977. Cm*—A modular, multi-microprocessor. In *Proceedings of the National Computer Conference* (Dallas, Tex., June 13–16), vol. 45. AFIPS Press, Reston, Va., pp. 637–644.

TRAIGER, I. 1983. Trends in systems aspects of database management. In *Proceedings of the British Computer Society 2nd International Conference on Databases* (Cambridge, England, Aug. 30–Sept. 2).

TSUKIGI, K., AND HASEGAWA, Y. 1983. The travel reservation on-line network system. *IEEE Q. Bull. Database Eng. 6*, 1 (Mar.), special issue on Database Systems in Japan.

WALTER, B. 1982. A robust and efficient protocol for checking the availability of remote sites. In *Proceedings of the 6th Workshop on Distributed Data Management and Computer Networks* (Berkeley, Calif., Feb.), pp. 45–68.

WEISS, H. M. 1980. The ORACLE data base management system. *Mini-Micro Syst.* (Aug.), 111–114.

WENSLEY, J. H., LAMPORT, L., GOLDBERG, J., GREEN, M., LEVITT, K., MELLIAR-SMITH, P. M., SHOSTAK, R., AND WEINSTOCK, C. 1978. SIFT: Design and analysis of fault-tolerant computer

for aircraft control. *Proc. IEEE 66*, 10 (Oct.), 1240–1255.

WEST, J. C., ISMAN, M. A., AND HANNAFORD, S. G. 1983. Transaction processing in the PERPOS operating system. *IEEE Q. Bull. Database Eng. 6*, 2 (June), special issue on Highly Available Systems.

WESTON, J. 1978. General Electric's MARK III Cluster System. Presented to the American Institute of Industrial Engineers, Jan. 31.

Received April 1983; final revision accepted February 1984.