TANDEM

Fault Tolerance in Tandem Computer Systems

Joel Bartlett *
Wendy Bartlett
Richard Carr
Dave Garcia
Jim Gray
Robert Horst
Robert Jardine
Dan Lenoski
Dix McGuire

** Present address: Digital Equipment Corporation Western Regional Laboratory, Palo Alto, California*

Technical Report 90.5
May 1990
Part Number: 40666

TANDEM COMPUTERS

Fault Tolerance in Tandem Computer Systems

Joel Bartlett*
Wendy Bartlett
Richard Carr
Dave Garcia
Jim Gray
Robert Horst
Robert Jardine
Dan Lenoski
Dix McGuire

* Present address:
Digital Equipment Corporation Western Regional Laboratory, Palo Alto, California

Technical Report 90.5
May 1990
Part Number: 40666

Fault Tolerance in Tandem Computer Systems[1]

Wendy Bartlett, Richard Carr, Dave Garcia, Jim Gray,
Robert Horst, Robert Jardine, Dan Lenoski, Dix McGuire
Tandem Computers Incorporated
Cupertino, California

Joel Bartlett
Digital Equipment Corporation, Western Regional Laboratory
Palo Alto, California

Tandem Technical Report 90.5, Tandem Part Number 40666
March 1990

ABSTRACT

Tandem produces high-availability, general-purpose computers that provide fault tolerance through fail-fast hardware modules and fault-tolerant software[2]. This chapter presents a historical perspective of the Tandem systems' evolution and provides a synopsis of the company's current approach to implementing these systems. The article does not cover products announced since January 1990.

At the hardware level, a Tandem system is a loosely-coupled multiprocessor with fail-fast modules connected with dual paths. A system can include a range of processors, interconnected through a hierarchical fault-tolerant local network. A system can also include a variety of peripherals, attached with dual-ported controllers. A novel disk subsystem allows a choice between low cost-per-byte and low cost-per-access.

System software provides processes and messages as the basic structuring mechanism. Processes furnish software modularity and fault isolation; process pairs tolerate hardware and transient software failures. Applications are structured as requesting processes that make remote procedure calls to server processes. Process server classes utilize multiprocessors. The resulting process abstractions provide a distributed system that can take advantage of thousands of processors. High-level networking protocols such as SNA, OSI, and a proprietary network are built on top of this base.

A relational database provides distributed data and distributed transactions. Databases can optionally be replicated at several sites for protection against environmental disasters. An application generator allows users to develop fault-tolerant applications as though the system were a conventional computer. The resulting system provides price/performance competitive with traditional systems.

[1]This article is scheduled to appear as a chapter of the second edition of the book *Theory and Practice of Reliable System Design*, edited by D. Siewiorek, and published by Digital Press. It is a revision of a two-year-old version of the paper of the same title, which is now obsolete.

[2] The following are trademarks or service marks of Tandem Computers Incorporated: CLX, CYCLONE, DYNABUS, DYNABUS+, ENCOMPASS, ENFORM, ENSCRIBE, EXPAND, FOX, GUARDIAN, GUARDIAN 90, MEASURE, NonStop, NonStop SQL, PATHMAKER, TAL, Tandem, TMDS, TXP, VIEWPOINT, VLX..

TABLE OF CONTENTS

INTRODUCTION

The increasing trend for businesses to go online stimulated a need for cost-effective computer systems with continuous availability [Katzman, 1977]. The strongest demand for general-purpose fault-tolerant computing was in online database transaction and terminal-oriented applications.

In the early 1970's, vendors and customers demanding continuous availability configured multiprocessor systems as hot-standbys (see Figure 1). This preserved previous development effort and compatibility by introducing devices, such as I/O channel switches and interprocessor communications adapters, to retrofit existing hardware. These architectures, however, still contained many single points of failure. For an example, a power supply failure in the I/O bus switch, or an integrated circuit failure in any I/O controller on the I/O bus switch channel, would cause the entire system to fail. Other architectures used a common memory for interprocessor communications, creating another single point of failure. Typically, these architectures did not even approach the problems of online maintenance, redundant cooling, or a power distribution system that tolerates brownout conditions. Furthermore, these systems lacked thorough data integrity features, leading to problems in fault containment and possible database corruption.

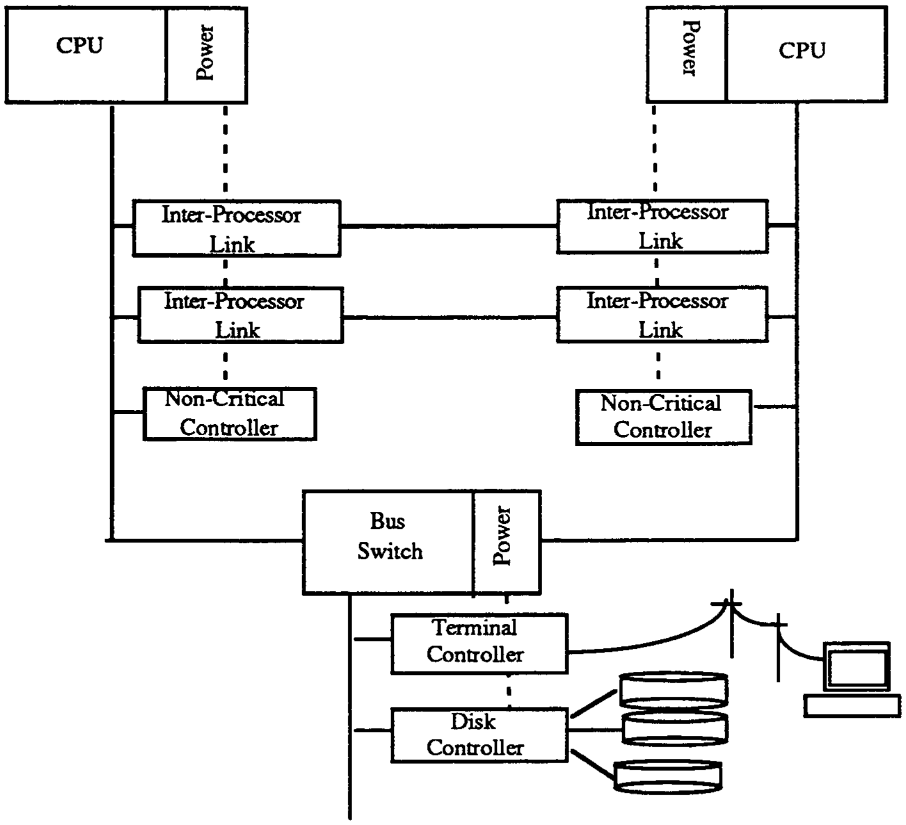


Figure 1. An example of Early Fault-Tolerant Architectures

As late as 1985, conventional, well-managed, transaction-processing systems failed about once every two weeks for about an hour [Mourad and Andrews, 1985 and Burman, 1985]. This failure rate translates to 99.6% availability, a level that reflects a considerable effort over many years to improve system availability. When the sources of faults were examined in detail, a surprising picture emerged: faults come from hardware, software, operations, maintenance, and the environment in about equal measure. Hardware could operate for two months without generating problems; software was equally reliable. The result was a one month Mean Time Between Failures (MTBF). But if operator errors, errors during maintenance, and power failures, were included, the MTBF fell below two weeks.

Based on this analysis, Tandem set out to build a system whose MTBF is measured in years[3]--more than two orders of magnitude better than conventional designs. The key design principles of the system were, and still are, the following:

- **Modularity**: Both hardware and software are based on modules of fine granularity that are units of service, failure, diagnosis, repair, and growth.
- **Fail-Fast Operation**: Each module is self-checking; when it detects a fault, the module stops.
- **Single Fault Tolerance:** When a hardware or software module fails, another module immediately takes over the failed module's function--giving a mean-time-to-repair measured in milliseconds. For processors or processes, this takeover means that a second processor or process must exist. For storage modules, it means that the modules and the paths to them are duplexed.
- **Online Maintenance**: Hardware and software can be diagnosed and repaired while the rest of the system continues to deliver service. When the hardware, programs, or data are repaired, they are reintegrated without interrupting service.
- **Simplified User Interfaces**: Complex programming and operations interfaces can be a major source of system failures. Every attempt has been made to simplify or automate interfaces to the system.

This chapter presents Tandem systems, viewed from the perspective of these key design features.

[3] The actual goal was to build a system with 100-year MTBF.

HARDWARE

Multiple hardware modules and multiple interconnections among those modules provide a basis for fault-tolerant operation. Two modules of a certain type are generally sufficient for hardware fault tolerance because the probability of a second independent failure during the repair interval of the first is extremely low. For instance, if a processor has a mean-time between failures of ten thousand hours (about a year) and a repair time of four hours, the MTBF of a dual-path system increases to about ten million hours (about one thousand years). If more than two processors were added, the further gains in reliability would be obscured by system failures related to software or system operations.

Modularity: Modularity is important to fault-tolerant systems because individual modules must be replaceable online. Keeping modules independent also makes it less likely that a failure of one module will affect the operation of another module. Increasing performance by adding modules allows customers to expand the capacity of critical systems without requiring major outages to upgrade equipment.

Fail-Fast Logic: Fail-fast logic is defined as logic that either works properly or stops. Fail-fast logic is required to prevent corruption of data in the event of a failure. Hardware checks (including parity, coding, and selfchecking), as well as firmware and software consistency checks, provide fail-fast operation.

Serviceability: As mentioned before, maintenance is a source of outages. Ideally, the hardware should have no maintenance. When maintenance is required, it should require no special skills or tools.

Price and Price/Performance: Commercial pressures do not permit customers to pay a high premium for fault tolerance; if necessary, they will use more ad-hoc methods for coping with unreliable, but cost-effective, computers. Fault-tolerant vendors have no special exemption from the requirement to use state-of-the-art components and architectures, which frequently compounds the complexity already required by fault tolerance.

HARDWARE ARCHITECTURE

The Tandem NonStop™ computer system was introduced in 1976 as the first commercial fault-tolerant computer system. Its basic architecture appears in Figure 2. The system includes from two to 16 processors connected by dual buses collectively known as the DYNABUS™ interprocessor bus. Each processor has its own memory, containing its own copy of the operating system. The processors communicate with one another through messages passed through the DYNABUS mechanism. The system can continue operation despite the loss of any single component.

Each processor has its own Input/Output bus. Dual-ported controllers connect to I/O buses from two different processors. An ownership bit in each controller selects which of its ports is currently the primary path. When a processor or I/O bus failure occurs, all controllers that were designated as primary on that I/O bus switch to their backup paths. The controller configuration can be arranged so that in a multiprocessor system, the failure of a processor causes that processor's I/O workload to be spread out over the remaining processors.

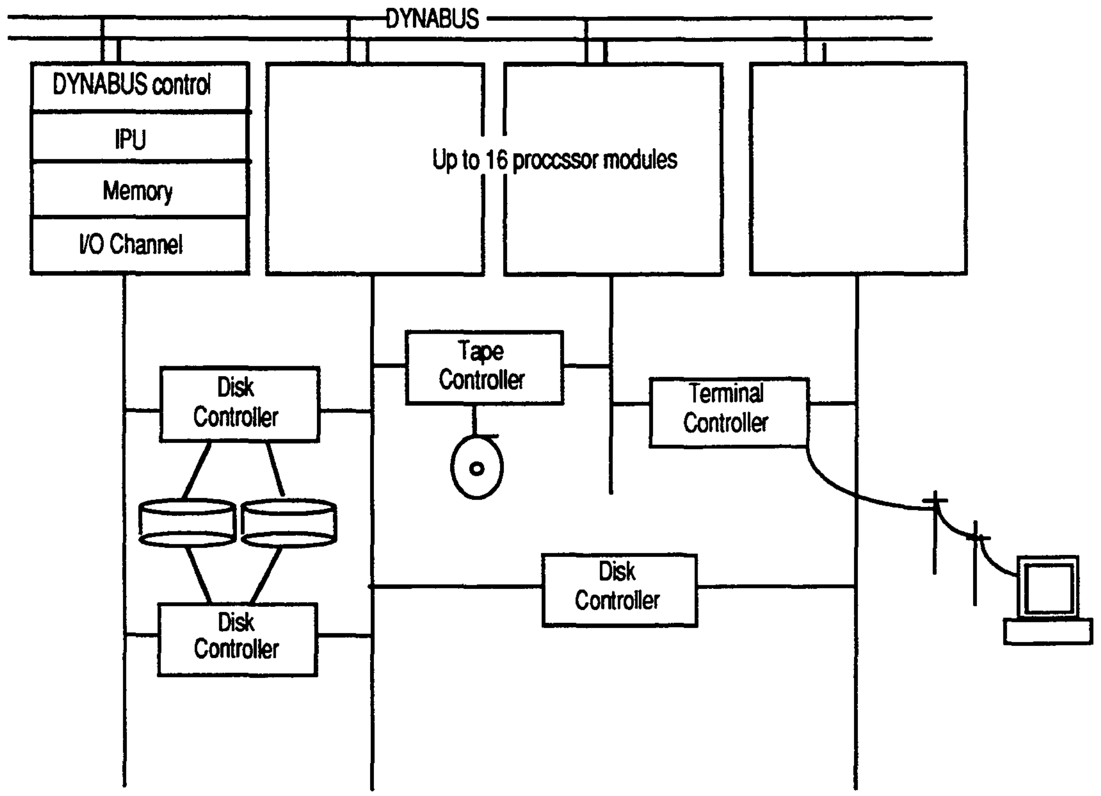


Figure 2. Original Tandem System Architecture, 1976

All subsequent systems have been upward-compatible with this basic design.

PROCESSOR MODULES

The primary components of a system are its processor modules, each of which includes an Instruction Processing Unit (IPU), memory, Input/Output channel, and DYNABUS interprocessor bus interface.

The design of the system's processor module is not much different from that of any traditional processor, with the addition of extensive error checking to provide fail-fast operation. Each processor operates independently and asynchronously from the rest of the processors. Another novel engineering requirement is that the DYNABUS interfaces must prevent a single-processor failure from disabling both buses. This requirement focuses on the proper selection of a single component type: the buffer that drives the bus. This buffer must be "well behaved" when power is removed from the processor module to prevent errors from being induced on both buses.

The power, packaging, and cabling must also be carefully considered. Parts of the system are redundantly powered through diode ORing of two different power supplies. In this way, I/O controllers and DYNABUS controllers tolerate a power supply failure. To allow online maintenance, and to allow modular growth, all boards are designed for *hot insertion*; that is, they can be inserted while the slot is powered. Battery backup power is standard in all systems. It preserves the system state for several hours in case of power failure.

The evolution of these processors is summarized in Table 1. Features common to all processors are described below. More details about the individual processors appear later in this chapter.

Each processor provides a basic set of instructions that includes operations on bits, integers, decimal numbers, floating-point numbers, and character strings; procedure calls and exits; I/O operations; and interprocessor SENDs to streamline the performance of the message-based operating system. All instructions are 16 bits long.

Table 1. Summary of Tandem Processor Evolution

Processor	NonStop I	NonStop II	TXP	VLX	CLX 600	CLX 700	Cyclone
Year	1976	1981	1983	1986	1987	1989	1989
Processor							
MIPS/IPU	0.7	0.8	2.0	3.0	1.0	1.5	10.0
Instructions	173	285	285	285	306	306	306
Technology	MSI	MSI STTL	MSI Fast PAL	ECL Gate Array	Custom 2µ CMOS	Custom 1.5µ CMOS	ECL Gate Array
Cycle Time	100ns	100ns	83ns	83ns	133ns	91ns	45ns
Microstore	- -	8k x 32b	two level: 8k x 40b 4k x 84b	10k x 120b dual	14k x 56b	14k x 56b	8k x 160b std + 8k x 160b pairs
Cache (data and instructions)	-	-	64KB direct map	64KB direct map	64KB direct map	128KB direct map	2 x 64KB direct map
Gates (approx)	20k	30k	58k	86k	81k	81k	275k
Proc. Boards	2	3	4	2	1	1	3
Procs/system	2-16	2-16	2-16	2-16	1-6	2-8	2-16
Memory							
Virtual	512KB	1GB	1GB	1GB	1GB	1GB	2GB
Physical	2MB	16MB	16MB	256MB	32MB	32MB	2GB
Per Board	64KB 384KB	512KB 2MB	2MB 8MB	8MB 16MB 48MB	4MB (on processor board) 8MB	8MB (on processor board) 8MB	32MB 64MB
Max Boards	2	2	4	2	1	1	2
Cycle Time	500ns/2B	400ns/2B	666 ns/8B	416ns/8B	933ns/8B	637ns/8B	495ns/16B (+225ns if CAM miss)
Input Output							
Interprocessor Bus Speed	2 x 13MB/s	2 x 13MB/s	2 x 13MB/s	2 x 20MB/s	2 x 20MB/s	2 x 20MB/s	2 x 20MB/s
Channel Speed	4MB/s	5MB/s	5MB/s	5MB/s	3.0MB/s	4.4MB/s	2 x 5MB/s

The Tandem NonStop I was a stack-oriented 16-bit processor with virtual memory. This instruction set has evolved to an upward-compatible 32-bit-addressing machine. Program binaries from the NonStop I will run on a Cyclone. The processor implementations have been fairly conventional, using a mix of special-purpose hardware for basic arithmetic and I/O operations, along with microcode to implement higher-level functions. Two novel features are the special hardware and micro-instructions to accelerate the sending and receipt of messages on the DYNABUS. The performance of these instructions has been a key component of the success of the message-based operating system.

Memory, as originally implemented, was designed to support a 16-bit minicomputer. In 1981, designers added a 32-bit addressing scheme that provided access to
- 4 MB of code space (for users)
- multiple 127.5 MB data spaces (for users)
- 4 MB of code space (for the system)
- 1 GB (2GB for Cyclone) of virtual data space (for the system).

The code and data spaces in both the user and the system areas are logically separate from one another.

In order to make processors fail-fast, extensive error checking is incorporated in the design. Error detection in data paths typically is done by parity checking and parity prediction, while checking of control paths is done with parity, illegal state detection, and selfchecking.

Loosely coupling the processors relaxes the constraints on the error-detection latency. A processor is required to stop itself only in time to avoid sending incorrect data over the I/O bus or DYNABUS. In some cases, to avoid lengthening the processor cycle time, error detection is pipelined and does not stop the processor until several clocks after the error occurred. Several clocks of error-detection latency is permitted in the architecture, but could not be tolerated in systems with lockstepped processors or systems where several processors share a common memory. In addition, the true fail-fast character of all processors eliminates the need for instruction retry in the event of errors.

DYNABUS INTERPROCESSOR BUS

The DYNABUS interprocessor bus is a set of two independent interprocessor buses. All components that attach to either of the buses are kept physically distinct, so that no single component failure can contaminate both buses simultaneously. Bus access is determined by two independent interprocessor bus controllers. Each of these controllers is dual-powered in the same manner as an I/O controller.

The DYNABUS controllers are not associated with, nor physically part of, any processor. Each bus has a two-byte data path and several control lines associated with it. No failed processor can independently dominate bus utilization upon failure because, to electrically transmit onto the bus, the bus controller must agree that a given processor has the right to transmit.

The original DYNABUS connected from two to 16 processors. This bus was designed with excess capacity to allow for future improvements in processor performance without redesign of the bus. The same bus was used on the NonStop II, introduced in 1980, and the NonStop TXP, introduced in 1983. The NonStop II and NonStop TXP processors can even plug into the same backplane to operate in a single system with mixed processors. A full 16-processor TXP system does not drive the bus near saturation.

A new DYNABUS was introduced with the VLX system. It provides peak throughput of 40 MB/sec, relaxes the length constraints of the bus, and has a reduced manufacturing cost due to improvements in its clock distribution. It was again overdesigned to accommodate the higher processing rates predicted for future processors. The CLX and Cyclone systems also use this bus.

For any given interprocessor data transfer, one processor is the sender and the other is the receiver. To transfer data over the DYNABUS interprocessor bus, the sending processor executes a SEND instruction. This instruction specifies the bus to be used, the intended receiver, and the number of bytes to be sent. Up to 64KB can be sent in a single SEND instruction. The sending processor continues to execute the SEND instruction until the data transfer is completed, during which time the DYNABUS interface control logic in the receiving processor is storing the data in memory. In the receiving processor, this activity occurs concurrently with program execution. Error recovery action is taken in case the transfer is not completed within a specified timeout interval.

In the DYNABUS design, the more esoteric decisions are left to the software (for example, alternate path routing and error recovery procedures); hardware, then, implements fault detection and reporting [Bartlett, 1978].

FIBER-OPTIC EXTENSION (FOX) LINKS

In 1983, a fiber-optic bus extension (FOX) was introduced to link systems together in a high-speed local network. FOX allows up to 14 systems of up to 16 processors each to be linked in a ring structure, for a total of 224 processors. The maximum distance between adjacent nodes was 1 Km on the original FOX and is 4 Km with FOX II, which was introduced on the VLX processor. A single FOX ring can mix NonStop II, TXP, VLX, and Cyclone processors.

The interconnection of systems by FOX links is illustrated in Figure 3. Each node in the group can accept or send data at rates of up to 4 MB/sec.

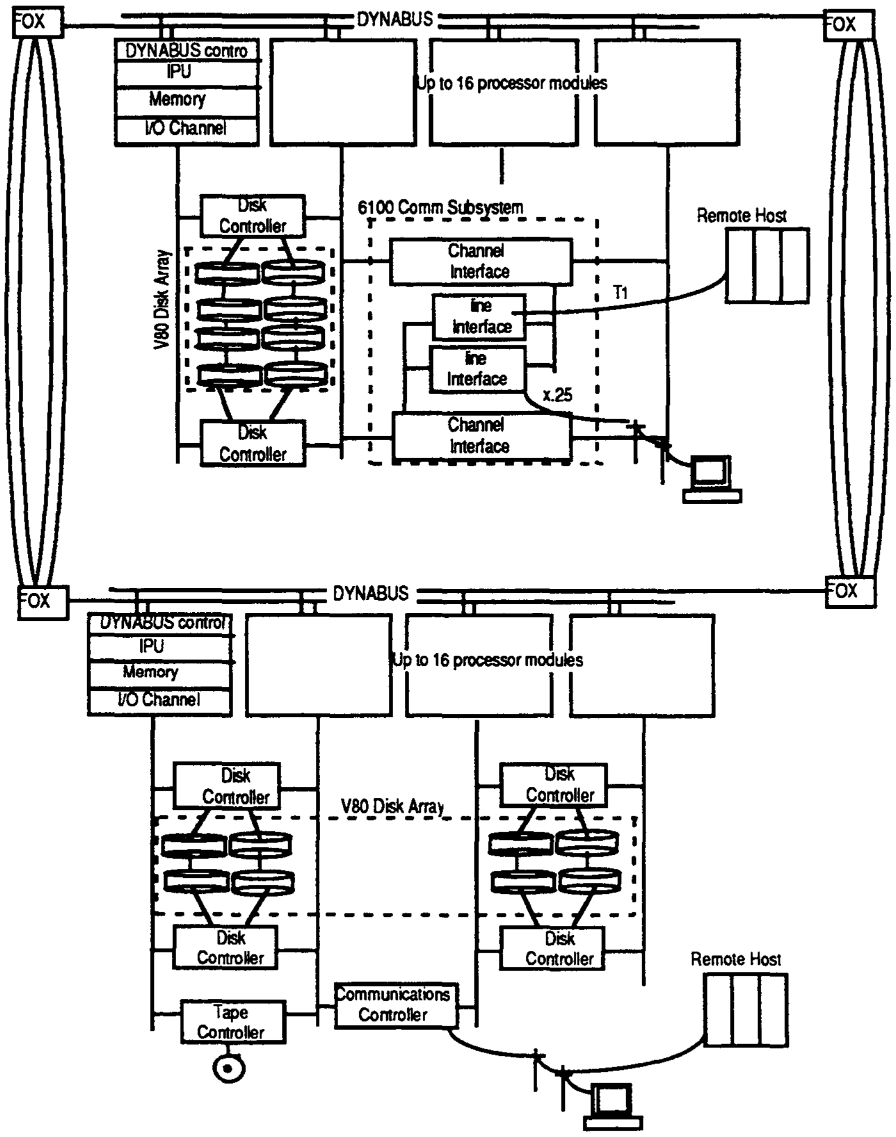


Figure 3. Tandem System Architecture of 1990

The FOX connection is based on a store-and-forward ring structure. Four fibers are connected between a system and each of its two neighbors. Each interprocessor bus is extended by a pair of fibers, which allows messages to be sent in either direction around the ring. The four paths provided between any pair of systems ensure that communication is not lost if a system is disabled (perhaps because of a power failure) or if an entire four-fiber bundle is severed.

The ring topology also has advantages over a star configuration because a ring has no central switch that could constitute a single point of failure and because cable routing is easier with a ring than with a star. In a ring structure, bandwidth increases as additional nodes are added. The total bandwidth available in a FOX network depends on the amount of passthrough traffic. In a 14-node FOX ring, if each node sends to other nodes with equal probability, the network has a usable bandwidth of 10MB/sec. With no passthrough traffic, the bandwidth increases to 24MB/second. Theoretically, an application generating 3KB of traffic per transaction, at 1,000 transactions per second, would require a FOX ring bandwidth of only 3MB/sec. In this situation, the FOX network would use less than 30% of the total available bandwidth. Transaction processing benchmarks have shown that the bandwidth of FOX is sufficient to allow linear performance growth in large multinode systems [Horst, 1985], [Englert 1989].

Fiber-optic links were chosen to solve both technical and practical problems in configuring large clusters. Fiber optics are not susceptible to electromagnetic interference, so they provide a reliable connection even in noisy environments. They also provide high-bandwidth communication over fairly large distances (4KM/hop). This lessens the congestion in the computer room and allows many computers in the same or nearby buildings to be linked. Fiber-optic cables are also flexible and of small diameter, thus easing installation.

FOX links allow computer sites to physically isolate nodes by housing them in different buildings, thereby providing a degree of fault isolation and protection against disaster. For example, a fire in the computer room in one building would not affect nodes in other buildings.

DYNABUS+ FIBER-OPTIC DYNABUS EXTENSION

With the introduction of the Cyclone system in 1989, Tandem made two additional uses of Fiber optic links; their use between peripheral controllers and Input/Output devices is described later in this chapter; their use as an interprocessor link, within a system (as compared with FOX, which is an intersystem link) is described here.

Cyclone processors are grouped into sections, each containing up to 4 processor modules. The sections may be geographically distributed up to 50 meters. Within a section, the normal (backplane) DYNABUS interface is used. Sections within a system are connected in a ring arrangement, similar to the FOX arrangement.

Individual DYNABUS+ fiber-optic links are capable of 100 Megabit/second bandpass, a good match for the 20 Megabyte/second bandpass of the DYNABUS. While increasing performance was not the major design goal of DYNABUS+, the design has resulted in additional aggregate interprocessor bus bandpass, up to 160 Megabytes/second in a system. Each section contains its own DYNABUS controllers, so message traffic local to the section can proceed concurrently with local traffic in other sections. In addition, intersection traffic can proceed concurrently on the multiple fiber-optic links.

The DYNABUS+ system is transparent to all levels of software except the maintenance subsystem (described later in this chapter). The system is self-configuring when power is first applied; Diagnostic Data Transceiver (DDT) processors within each cabinet determine the configuration and routing rules. In the event of a failure of any of the fiber-optic links or interface logic boards, the system reconfigures itself, establishes new routing paths, and notifies the maintenance subsystem.

In addition, a VLX interface to DYNABUS+ allows intermixing VLX and Cyclone processor modules within a system. This feature allows an existing VLX customer to add Cyclone processors to a system, providing a smooth upgrade path.

SYSTEM POWER AND PACKAGING

Online maintenance is a key factor in the design of the physical packaging and the power distribution of Tandem systems. Traditional designs assumed that maintenance could be done with the equipment offline and powered off.

The VLX system cabinet, shown in Figure 4, is divided into four sections: the upper card cage, the lower card cage, the cooling section, and the power supply section. The upper card cage contains up to four processors, each with its own I/O channel and private memory. The lower card cage contains up to 24 I/O controller printed circuit (PC) cards, where each controller consists of one to three PC cards. The cooling section consists of four fans and a plenum chamber that forces laminar air flow through the card cages. The power supply section contains up to four power supply modules. Multiple cabinets can be bolted together.

The system can accommodate a maximum of 16 processor modules on a DYNABUS pair. Each processor module--consisting of an IPU, memory, DYNABUS control, and I/O channel--is powered by an associated power supply. The power distribution system is shown in Figure 4.

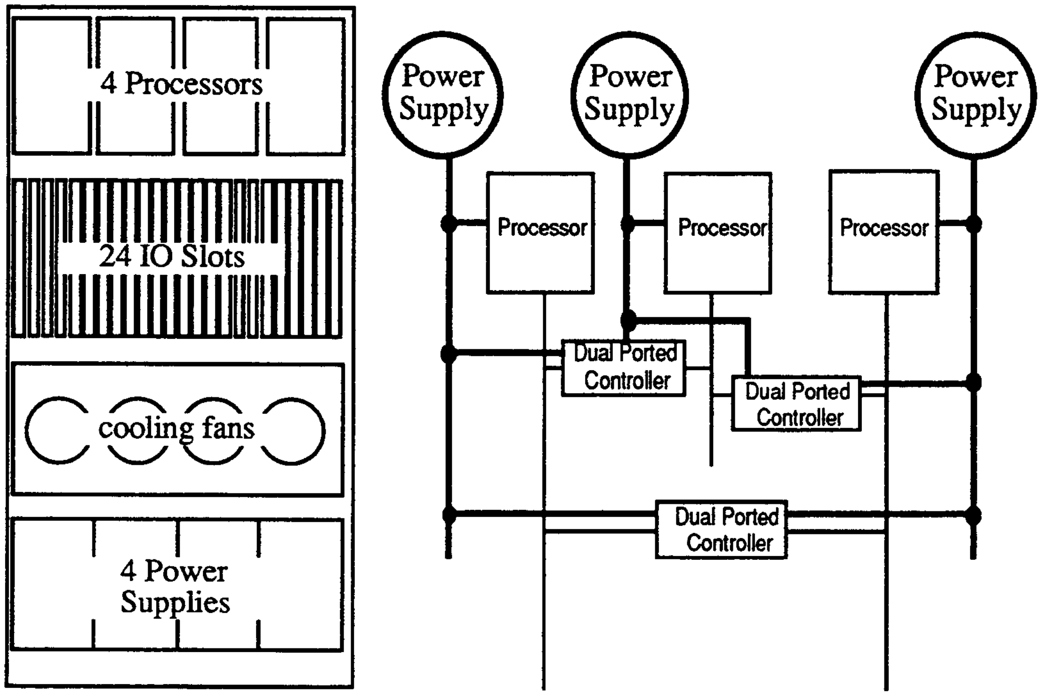


Figure 4. NonStop VLX System Cabinet (left) and power distribution (right)

If a failed processor module is to be replaced, its associated power supply is shut off, the module is replaced, and the power supply is turned on. Each card cage slot in the I/O card cage is powered by two different power supplies. Each of the I/O controllers is connected via its dual-port arrangement to two processors. Each of those processors has its own power supply; usually, but not necessarily, those two supplies are the ones that power the I/O controller (see Figure 4 again). Thus, if a power supply fails or if one is shut down to repair a processor, no I/O controllers are affected.

I/O controllers draw current from two supplies simultaneously. If a power supply fails, the second power supply provides all the current for a given controller. There is also circuitry to provide for a controlled ramping of current draw on insertion, removal, turn-on, and turn-off. This circuitry smooths power demands from a given supply, masking a potential momentary dip in supply voltage.

EVOLUTIONARY CHANGES

Processor architecture has evolved to keep pace with technology. These improvements include:
- Expansion to 1 GB of virtual memory (NonStop II system) and to 2 GB (Cyclone system)
- Incorporation of cache memory (TXP system)
- Expansion of physical memory addressability to 256 MB (VLX system) and to 2 GB (Cyclone system)
- Incorporation of separate instruction and data caches (Cyclone system)
- Incorporation of superscalar architecture (Cyclone system) [Horst, Harris, and Jardine, 1990]
- Incorporation of an independent instruction fetch unit with dynamic branch prediction (Cyclone system).

Technological improvements include:
- Evolution from core memory to 1Mb dynamic RAMs
- Evolution from Schottky TTL (NonStop I and II systems) to Programmable Array Logic (TXP system) [Horst and Metz, 1984] to bipolar gate arrays (VLX system) to silicon-compiled custom CMOS (CLX system) [Lenoski, 1988].

The Tandem multiprocessor architecture allows a single processor design to cover a wide range of processing power. Having processors of varying power adds another dimension to this flexibility. For instance, for approximately the same processing power, a customer can choose a 2-processor VLX system, a 3-processor TXP system, or a 4-processor CLX-700 system. Having a range of processors extends the range of applications from those sensitive to low-entry price to those with extremely high-volume processing needs. In a different performance range, the customer may chose a four-processor Cyclone system or a 16-processor VLX system.

PERIPHERALS

In building a fault-tolerant system, the entire system, not just the processor, must have the basic fault-tolerant properties of dual paths, modularity, and fail-fast design, as well as good price/performance. Many improvements in all of these areas have been made in peripherals and in the maintenance subsystem.

The basic architecture provides the ability to configure the I/O system to allow multiple paths to each I/O device. With dual-ported controllers and dual-ported peripherals, there are actually four paths to each device. When disks are mirrored, there are eight paths that can be used to read or write data.

In the configurations illustrated in Figure 3, there are many paths to any given disk--typically two controllers access each disk and each controller is attached to two processor channels. Software is used to mirror disks; that is, data is stored on two disks so that if one fails, the data is still available on the other disk. Consequently, the data can be retrieved regardless of any single failure of a disk drive, disk controller, power supply, processor, or I/O channel.

The original architecture did not provide as rich an interconnection scheme for communications and terminals. The first asynchronous terminal controller was dual-ported and connected to 32 terminals. The terminals themselves were not dual-ported, so it was not possible to configure the system in a way to withstand a terminal controller failure without losing a large number of terminals. The solution for critical applications was to have two terminals nearby that were connected to different terminal controllers.

The 6100 Communication Subsystem

The 6100 Communications Subsystem, introduced in 1983, helped reduce the impact of a failure in the communications subsystem. The 6100 consists of two dual-ported Communications Interface Units (CIUs) that communicate with I/O buses from two different processors (see Figure 3). Individual Line Interface Units (LIUs) connect to both CIUs, and to the communications line or terminal line. With this arrangement, CIU failures are completely transparent, and LIU failures result in the loss of only the attached line or lines. An added advantage is that each LIU can be downloaded with a different protocol in order to support different communications environments and to offload protocol interpretation from the main processors.

The 6100 Communications Subsystem is configured to have up to 45 LIUs. Each LIU can support up to 19.2 Kb/s of asynchronous communication or 64 Kb/s of synchronous communication. Redundant power supplies and cooling fans provide an extra margin of fault tolerance and permit online replacement of components.

Disk Subsystem

Modularity is standard in peripherals; it is common to mix different types of peripherals to match the intended application. In online transaction processing (OLTP), it is desirable to independently select increments of disk capacity and of disk performance. OLTP applications often require more disk arms per megabyte than is provided by traditional large (14 inch) disks. This may result in customers buying more megabytes of disk than they need in order to avoid queuing at the disk arm.

In 1984, Tandem departed from traditional disk architecture by introducing the V8 disk drive. The V8 was a single cabinet that contained up to eight 168-MB, 8-inch Winchester disk drives in six square feet of floor space. Using multiple 8-inch drives instead of a single 14-inch drive provided more access paths and less wasted capacity. The modular design was more serviceable, because individual drives could be removed and replaced online. In a mirrored configuration, system software automatically brought the replaced disk up-to-date while new transactions were underway.

Once a system can tolerate single faults, the second-order effects begin to become important in system failure rates. One category of compound faults is the combination of a hardware failure and a human error during the subsequent human activity of diagnosis and repair. The V8 reduced the likelihood of such compound hardware-human failures by simplifying servicing and eliminating preventative maintenance.

In fault-tolerant systems design, keeping down the price of peripherals is even more important than in traditional systems. Some parts of the peripheral subsystem must be duplicated, yet they provide little or no added performance.

For disk mirroring, two disk arms give better read performance than two single disks because the seeks are shorter and because the read work is spread evenly over the two servers [Bitton, 1988, 1989]. Write operations, on the other hand, do demand twice as much channel and controller time. Also, mirroring does double the cost per megabyte stored. To reduce the price per megabyte of storage, the XL8 disk drive was introduced in 1986. The XL8 had eight 9-inch Winchester disks in a single cabinet and had a total capacity of 4.2GB. As in the V8 drive, disks within the same cabinet could be mirrored, saving the costs of added cabinetry and floor space. Also, like the V8, the reliable sealed media and modular replacement kept maintenance costs low.

The V80 disk storage facility replaced the V8 in 1988. Each of the V80's eight 8-inch disk drives has a formatted capacity of 265 MB. Thus, each cabinet can hold 2.7 GB of unformatted storage, or 2.1 GB of formatted storage. Externally, the V80 resembles the V8, housed in a single cabinet that occupies six square feet of floor space. The internal design of the V80, however, extends the capacity and reliability of the V8 with a fully-checked interface to the drives. Furthermore, the design reduces by a factor of five the number of external cables and connectors between the storage facility and the control unit.

The disk drive interface is based on the emerging industry-standard IPI-2 interface design, which has parity checking on data and addressing to ensure the integrity of data and commands. (The previous SMD-based design provided only data parity.) IPI's parallel and batched data and command interface between the disks and their controller allow higher data transfer rates (2.4 MB/sec) and reduced interrupts.

A radial connection between the controller and the drives eliminates possible drive interaction that could occur with conventional bus structures. The five-fold reduction in the number of external cables and connections is achieved by placing the control logic in the disk cabinet. Within the cabinet, a new interconnect design has reduced by a factor of five the number of internal cables and connections.

In 1989, the XL80 replaced the XL8 in similar fashion, doubling the storage capacity per drive, and also moving to an IPI-2 storage interface. In addition, the XL80 cabinet contains sensors for inlet air temperature, power supply and board voltages, and fan operation; this information is polled periodically by the cabinet's maintenance subsystem and reported to the peripheral controller when an exception condition exists. A fully-configured XL80 disk subsystem, including storage modules, power supplies, and cooling fans, appears in Figure 5.

Peripheral Controllers

Peripheral controllers have fail-fast requirements similar to processors. They must not corrupt data on either of their I/O buses when they fail. If possible, they must return error information to the processor when they fail. In terms of peripheral fail-fast design, the Tandem contribution has been to put added emphasis on error detection within the peripheral controllers. An example is a VLSI tape controller which uses dual, lockstepped Motorola 68000 processors with compare circuits to detect errors. It also contains totally self-checked logic and self-checking checkers to detect errors in the ad-hoc logic portion of the controller.

Beyond this contribution, the system software uses "end-to-end" checksums generated by the high-level software. These checksums are stored with the data and are recomputed and rechecked when the data is reread.

The single-board controller supporting the V80 and XL80 disks uses CMOS VLSI technology. The controller is managed by dual, lockstepped Motorola 68010 microprocessors that provide sophisticated error-reporting and fault-isolation tools. The controller is contained on a single board; thus, it requires only half the input/output slots of previous controllers.

Other efforts to reduce peripheral prices include the use of VLSI gate arrays in controllers to reduce part counts and improve reliability and use of VLSI to integrate the standalone 6100 communications subsystems into a series of single-board controllers.

The Tandem evolution of fault tolerance in peripherals is summarized in Table 2.

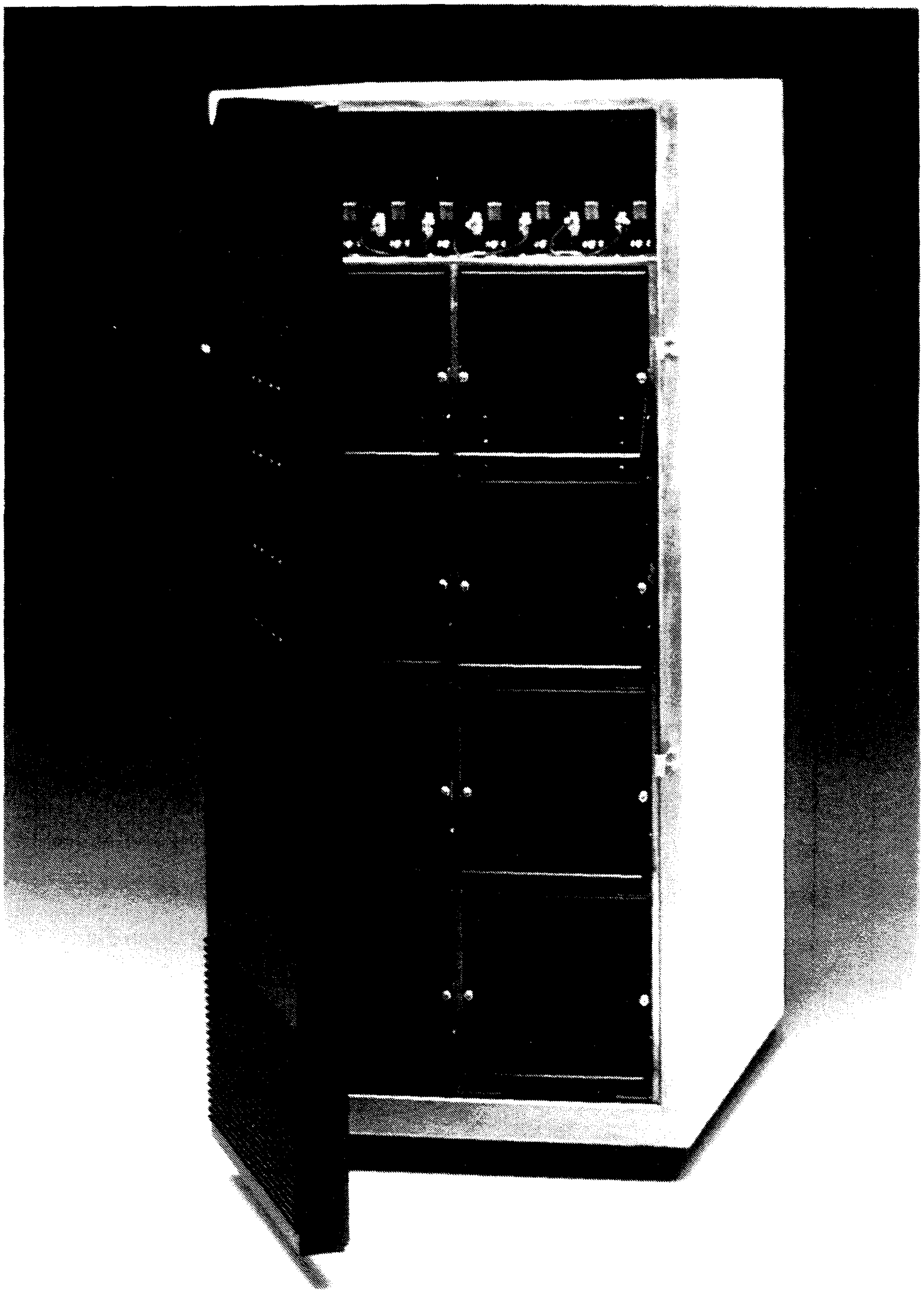


Figure 5. XL80 Disk Subsystem (Front View)

Table 2. Tandem Evolution of Peripheral Fault Tolerance

Year	Product	Contribution
1976	NonStop I System	Dual-ported controllers, single-fault-tolerant I/O system
1977	NonStop I System	Mirrored and dual-ported disks
1982	INFOSAT	Fault-tolerant satellite communications
1983	6100 Communications Subsystem	Fault-tolerant communications subsystem
1983	FOX	Fault-tolerant, high-speed, fiber-optic LAN
1984	V8 Disk Drive	Eight-drive, fault-tolerant disk array
1985	3207 Tape Controller	Totally self-checked VLSI tape controller
1985	XL8 Disk Drive	Eight-drive, high-capacity/low-cost, fault-tolerant disk array
1986	TMDS	Fault-tolerant maintenance system
1987	CLX	Fault-tolerant system that is 98% user-serviceable
1988	V80 Storage Facility	Reduced disk cabling and fully-checked disk interfaces
1988	3120 Disk Controller	Totally self-checked VLSI disk controller
1989	XL80 Storage Facility	Reduced disk cabling, fully-checked disk interfaces, environmental monitoring within disk cabinet
1989	Fiber-Optic Interconnect for V80 and XL80	Reduced cabling to a minimum, reduced transmission errors

PROCESSOR MODULE IMPLEMENTATION DETAILS

The following sections outline the implementation details of each of the Tandem processors summarized in Table 1.

NonStop I

The NonStop I processor module, introduced in 1976, included a 16-bit IPU, main memory, DYNABUS interface, and an I/O channel. Physically, the IPU, I/O channel, and DYNABUS control consisted of two PC boards that measure 16 inches by 18 inches, each containing approximately 300 integrated circuit packages. These boards employed Schottky TTL circuitry.

The processor module was viewed by the user as a 16-bit, stack-oriented processor, with a demand paging, virtual memory system capable of supporting multiprogramming.

The IPU was a microprogrammed processor consisting of: (1) an execution unit with ALU, shifter, register stack, and Program counter, (2) a microprogram sequencer with 1024 32-bit words stored in ROM, (3) address translation maps supporting system code and data, and current user code and data segments, (4) up to 512 KB of main memory, (5) 96 KB memory boards with single error correction and double error detection, and (5) battery backup for short-term main memory ride-through of power outages of up to 4 hours.

The heart of the I/O system is the I/O channel. In the NS I, all I/O was done on a direct memory access (DMA) basis. The channel was a microprogrammed, block-multiplexed channel; individual controllers determine the block size.

The channel did not execute channel programs, as on many systems, but did transfer data in parallel with program execution. The memory system priority always permitted I/O accesses to be handled before IPU or DYNABUS accesses. The maximum I/O transfer was 4 KB.

Dual-Port Controllers

The dual-ported I/O device controllers provided the interface between the NonStop I I/O channel and a variety of peripheral devices using distinct interfaces. While these I/O controllers were vastly different depending on the devices they interfaced to, there was a commonality among them that fitted them into the NonStop I architecture.

Each controller contained two independent I/O channel ports implemented by IC packages that were physically separate from each other so that no interface chip could simultaneously cause failure of both ports. Logically, only one of the two ports was active. The other port was utilized only in the event of a path failure to the primary port. An "ownership" bit, as illustrated in Figure 6, indicated to each port if it was the primary port or the alternate.

Ownership changed only when the operating system issued a TAKE OWNERSHIP I/O command. Executing this special command caused the I/O controller to swap its primary and alternate port designation and to do a controller reset.

Any attempt to use a controller that was not owned by a given processor resulted in an ownership violation. If a processor determined that a given controller was malfunctioning on its I/O channel, it could issue a DISABLE PORT command that logically disconnected the port from that I/O controller. This disconnection did not affect the ownership status. Thus, if the problem was within the port, the alternate path could be used; but if the problem was in the common portion of the controller, ownership was not forced on the other processor.

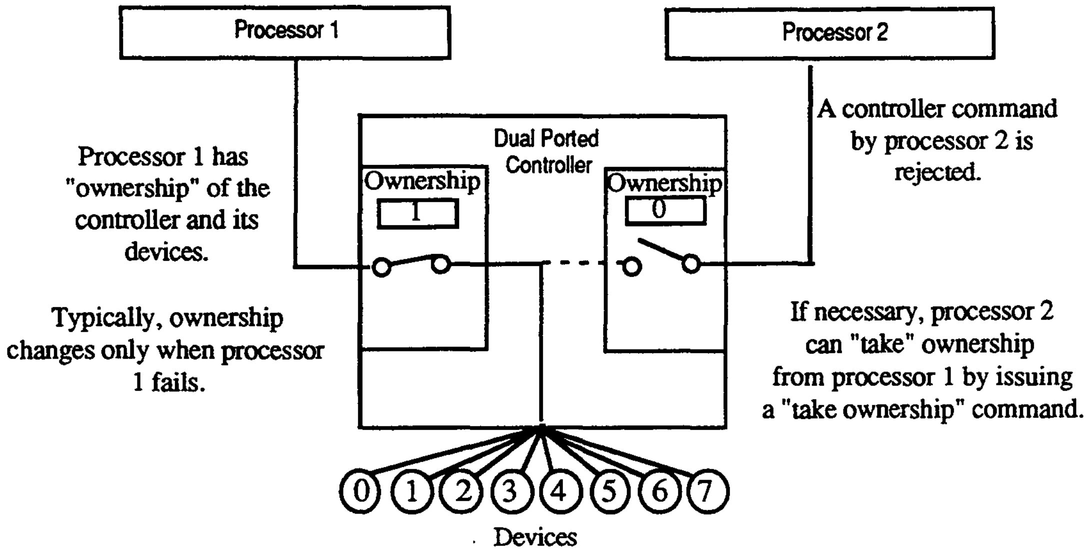

Processor 1
Processor 2
Processor 1 has "ownership" of the controller and its devices.
Typically, ownership changes only when processor 1 fails.
Dual Ported Controller
Ownership
1
Ownership
0
A controller command by processor 2 is rejected.
If necessary, processor 2 can "take" ownership from processor 1 by issuing a "take ownership" command.
0 1 2 3 4 5 6 7
Devices

Figure 6. Ownership Circuitry and Logic

Fault-Tolerant I/O Considerations

The I/O channel interface consisted of a two-byte data bus and control signals. All data transferred over the bus was parity-checked in both directions, and errors were reported through the interrupt system. A watchdog timer in the I/O channel detected if a nonexistent I/O controller was addressed, or if a controller stopped responding during an I/O sequence.

The data transfer byte count word in the channel command entry contained four status bits, including a protect bit. When this bit was set on, only output transfers were permitted to this device.

Because I/O controllers were connected between two independent I/O channels, it was very important that word count, buffer address, and direction of transfer be controlled by the processor instead of within the controller. If that information were to be kept in the controller, a single failure could fail both processors attached to it. Consider what would happen if a byte count register were located in the controller and was involved a situation where the count did not decrement on an input transfer. It would be possible to overwrite the buffer and render system tables meaningless. The error would propagate to the other processor upon discovery that the first processor was no longer operating.

Other error conditions that the channel checked for were violations of I/O protocol, attempts to transfer to absent pages (it is the operating system's responsibility to "lock down" the virtual pages used for I/O buffering), uncorrectable memory errors, and map parity errors.

NonStop II

The NonStop II was a compatible extension of the NonStop I. The major changes from the NonStop I processor module were the introduction of a 32-bit addressing scheme and a Diagnostic Data Transceiver (DDT) processor. The software for the NonStop II system was upward-compatible with the NonStop I system. Thus, application programs written for the NonStop I system could be run on the NonStop II system.

The IPU was implemented using Schottky TTL logic using a microinstruction cycle time of 100 nsec. Instructions were added to support 32-bit extended addressing. An optional floating-point instruction set was also added for high-speed scientific calculations--eventually, these instructions became a standard part of the instruction set. The instruction sets were implemented in microcode in a high-speed control store, which had 8K 32-bit words of loadable storage and 1K words of read-only storage. The loadable part of the control store was initialized when the operating system was loaded. Before loading the control store, the system performed a set of diagnostic routines to verify that the processor was operating correctly.

The processor's internal data paths and registers were parity-checked to ensure data integrity. The IPU featured a two-stage pipeline that allowed it to fetch the next instruction while executing the current instruction.

Memory boards for the NonStop II system contained 512KB, 2MB or 4MB of storage. Up to four of these boards, in any combination, could reside in one processor for a maximum of 16MB. A fully-configured 16-processor system allowed up to 64 boards with a total of 256MB of memory. The memory access time was 400 nsec.

Each memory word was 22 bits long. Six bits of the word provide an error correction code that enabled the system to correct any single-bit error and detect any double-bit error. The error- correction code also checked the address sent from the IPU to ensure that the memory access was valid.

I/O Channel

Each processor module contained a separate processor dedicated to I/O operations. Because the I/O processor operated independently from the IPU, I/O transfers were extremely efficient and required only a minimum of IPU intervention. The channel was a burst multiplexor.

Every I/O device controller was buffered, which allowed data transfers between main memory and the controller buffer to occur at full memory speed. I/O transfers had a maximum length of 64KB. The high-speed I/O channels used burst-multiplexed direct memory access to provide transfer rates of up to 5 MB/s. Thus, the aggregate burst I/O rate of a fully-configured 16-processor system was 80 MB/s.

The I/O processor supported up to 32 device controllers. Depending on the type, device controllers could support up to eight peripheral units. Therefore, as many as 256 devices could be connected to a single processor. Multipoint communication lines were treated as a single device, so each processor could support very large terminal configurations.

I/O device controllers were intelligent devices. This intelligence allowed them to relieve the central processing unit of many routine functions such as polling synchronous terminals, checking for data transmission errors, and so forth.

Diagnostic Data Transceiver (DDT)

The Diagnostic Data Transceiver (DDT) was a separate microprocessor included as part of each processor module. The DDT provided two distinct functions:

- The DDT allowed communication between a processor module and the Operations and Service Processor (OSP), which supports both operational and maintenance functions such as running diagnostics. More about the OSP appears under Maintenance Facilities and Practices, later in this chapter.
- The DDT monitored the status of the central processing unit, DYNABUS interface, memory, and the I/O processor, and reported any errors to the OSP.

Virtual Memory

The virtual memory addressing scheme, introduced by the NonStop II processor, is used by all subsequent processors. It converted the system from 16-bit addressing to 32-bit addressing. This addressing is supported by the instruction set, and is based on segments that contain from 1 to 64 pages each. A page contains 2048 bytes. Each processor can address up to 8192 segments, which provides it with a billion bytes (1 GB) of virtual memory address space (later extended to 2 GB on the Cyclone processor, introduced in 1989).

The instruction set supports standard and extended addressing modes. The standard 16-bit addressing mode provides high-speed access within the environment of an executing program. The extended 32-bit addressing mode allows access to the entire virtual memory space by privileged processes. Programs written in Pascal, C, COBOL85, and the Transaction Application Language (TAL), can use extended addressing for access to large data structures.

The instruction set supports two types of extended addressing: absolute and relocatable. Absolute extended addressing is available only to privileged users such as the operating system itself. Absolute addresses can address any byte within the virtual memory. Relocatable extended addresses are available to all users. This form of addressing can reference any byte of the current process's data space as well as one or more private relocatable extended data segments. Each extended data segment can contain up to 127.5MB.

To provide efficient virtual-to-physical address translations, each NonStop II processor included 1024 high-speed map registers. The memory maps contained absent, dirty, and referenced bits to help the software manage virtual memory.

Maintenance

A major feature of the NonStop II system was the Operations and Service Processor (OSP) located in a console supplied with the system. In addition to serving as an operations interface for communication with the system, the OSP was a powerful diagnostic and maintenance tool. The OSP is described later in this section, under Maintenance Facilities and Practices.

TXP PROCESSOR

While the NonStop II system extended the instruction set of the NonStop I system to handle 32-bit addressing, it did not efficiently support that addressing mode. The existing 5MB/s I/O channel and 26MB/s DYNABUS interprocessor bus offered more than enough bandwidth to handle a processor with two to three times the performance. The existing packaging had an extra processor card slot for future enhancements, and the existing power supplies could be reconfigured to handle a higher-powered processors. The NonStop TXP processor module, introduced in 1983, was designed in this environment.

The main problems concerned designing a new microarchitecture that would efficiently support the 32-bit instructions at much higher speeds, with only 33% more printed circuit board area and the existing backplane. This design involved eliminating some features that were not critical to performance and finding creative ways to save area on the PC board, including strategic uses of programmable array logic and an unusual multilevel control-store scheme.

The performance improvements in the NonStop TXP system were attained through a combination of advances in architecture and technology. The NonStop TXP architecture used dual 16-bit data paths, three levels of macroinstruction pipelining, 64-bit parallel access from memory, and a large cache (64KB per processor). Additional performance gains were obtained by increasing the hardware support for 32-bit memory addressing.

The machine's technology includes 25-nsec programmable array logic, 45-nsec 16K static RAM chips, and Fairchild Advanced Schottky Technology (FAST) logic. With these high-speed components and a reduction in the number of logic levels in each path, a 12-MHz (83.3 nsec per microinstruction) clock rate could be used.

The TXP's dual data-path arrangement increased performance through added parallelism, as shown in Figure 7. A main arithmetic-and-logic unit operation could be performed in parallel with another operation done by one of several special modules. Among these modules were a second ALU to perform both multiplications and divisions, a barrel shifter, an array of 4,096 scratchpad registers, an interval timer, and an interrupt controller. Other modules provide interfaces among the IPU and the interprocessor bus system, I/O channel, main memory, and a diagnostic processor.

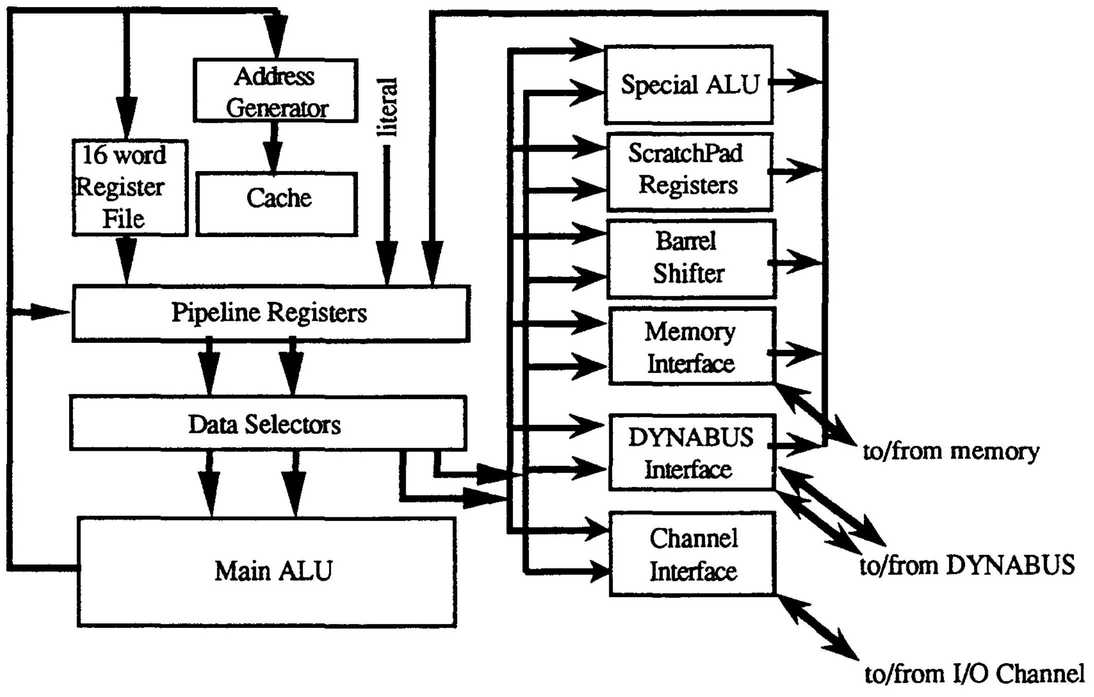


Figure 7. Parallel Data Paths of the TXP processor.

The selection of operands for the main ALU and the special modules was done in two stages. In the first stage, data was accessed from the dual-ported register file or external registers and placed into two of the six registers. During the same cycle, the other four pipeline registers were loaded with cache data, a literal constant, the results of the previous ALU operation, and the result of the previous special-module operation.

In the second stage, one of the six pipeline registers was selected for each of the main ALU inputs and another one of these registers was selected for each special-module operand. Executing the register selection in two stages, so that the register file could be two-ported rather than four-ported, greatly reduced the cost of multiplexers and control storage; the flexibility in choosing the required operands was unimpeded.

An example of the way microcode used the parallel data paths is shown in Table 3. This example shows the inner loop of the COMPARE-BYTE instruction. Each of the dual ALUs in the TXP system extracted one byte; then the extracted bytes were compared. This operation took two clock cycles on the TXP system, but would require three if the extract operations were not done simultaneously.

Table 3. Compare Byte Instructions (Inner Loop)

Clock Cycle	NonStop TXP		Traditional Architecture
	Main ALU	Special ALU	
1	Extract Byte 1	Extract Byte 2	Extract Byte 1
2	Compare Bytes	–	Extract Byte 2
3	(Repeat)	(Repeat)	Compare Bytes
4	–	–	(Repeat)

The dual 16-bit data paths tended to require fewer cycles than a single 32-bit path when manipulating byte and 16-bit quantities. However, the paths did require slightly more cycles when manipulating 32-bit quantities. A 32-bit add took two cycles rather than one, but the other data path was free to use the two cycles to perform either another 32-bit operation or two 16-bit operations. Measurements of transaction-processing applications showed that the frequencies of 32-bit arithmetic were insignificant relative to data-movement and byte-manipulation instructions, which were handled more efficiently by the dual data paths than by a single 32-bit data path. Most instructions include enough parallelism to let the microcode make effective use of both data paths.

To control the large amount of parallelism in the NonStop TXP processor, a wide control-store word was required. The effective width of the control store was over 100 bits. To reduce the number of RAMs required, the control store was divided between a vertical control store of 8K 40-bit words and a horizontal control store of 4K 84-bit words. The vertical control store controlled the first stage of the microinstruction pipeline and included a field that addressed the horizontal control store, whose fields controlled the pipeline's second stage. Lines of microcode that required the same or similar horizontal controls could share horizontal control-store entries.

Unlike microprocessor-based systems that have microcode fixed in read-only memory, the NonStop TXP system microcode was implemented in RAM so that it could be changed along with normal software updates and so that new performance-enhancing instructions could be added. Because instructions were pipelined, the TXP processor could execute its fastest instructions in just two clock cycles (167 nsec). The processor could also execute frequently-used load and branch instructions in only three clock cycles (250 nsec).

Each NonStop TXP processor had a 64KB cache holding both data and code. A 16-processor NonStop TXP system had a full megabyte of cache memory. To determine the organization of the cache, a number of measurements were performed on a NonStop II system using a specially-designed hardware monitor. The measurements showed that higher cache hit ratios resulted with a large, simple cache (directly mapped) than with a smaller, more complex cache (organized as two-way or four-way associative). Typical hit ratios for transaction processing on the NonStop TXP system fell in the range of 96% to 99%.

Cache misses were handled in a firmware subroutine rather than by the usual method of adding a special state machine and dedicated data paths for handling a miss. Because of the large savings in cache hardware, the cache could reside on the same board as the primary data paths. Keeping these functions proximal reduced wiring delays, contributing to the fast 83.3nsec cycle time.

The cache was addressed by the 32-bit virtual address rather than by the physical address, thus eliminating the extra virtual-to-physical translation step that would otherwise be required for every memory reference. The virtual-to-physical translation, needed to refill the cache on misses and to store through to memory, was handled by a separate page table cache that held mapping information for as many as 2,048 pages of 2 KB each (see Figure 8).

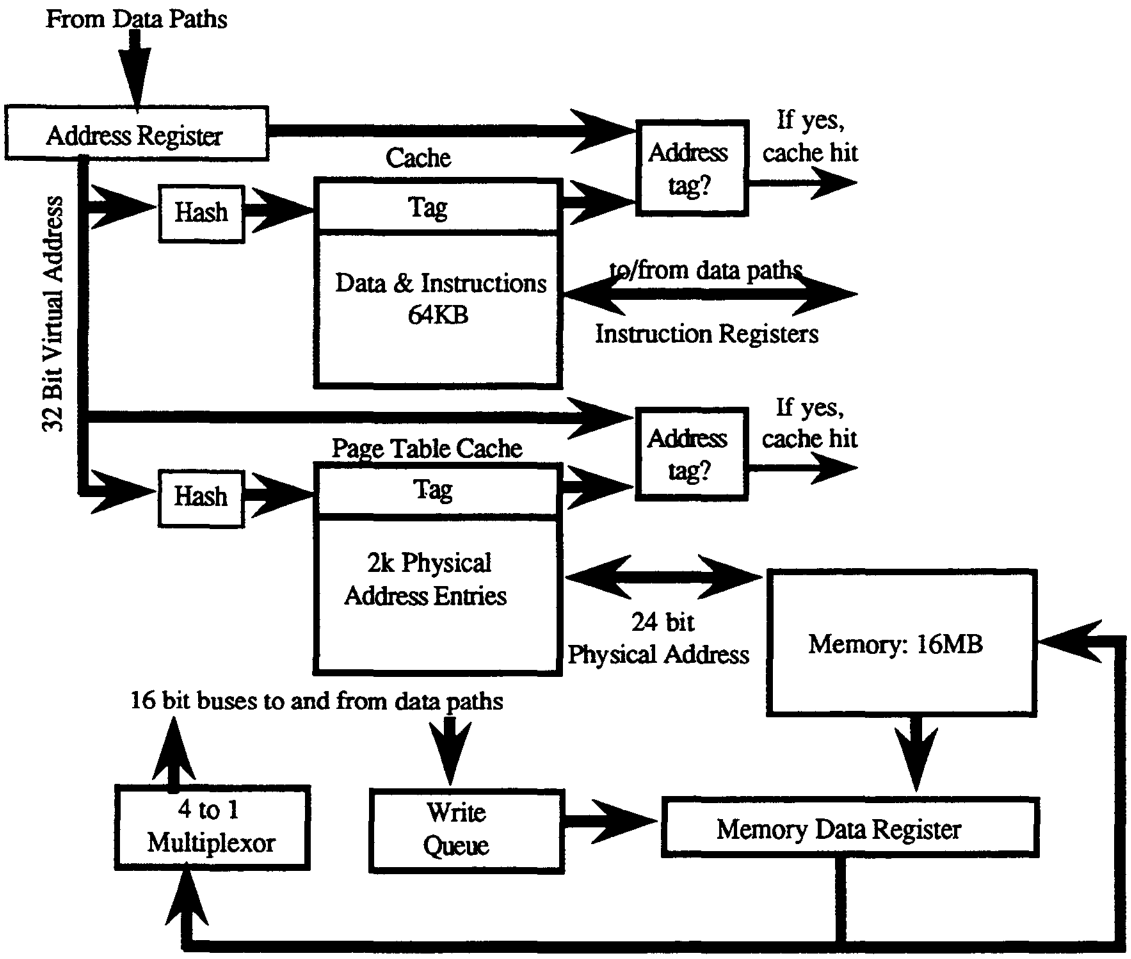


Figure 8. TXP Memory Access

A cache memory by itself does not necessarily boost a processor's performance significantly. It is of little use for the cache to provide instructions and data at a higher rate than the rest of the IPU can process. In the TXP processor, the cache's performance was tuned to provide instructions and data at a rate consistent with the enhancements to instruction processing provided by increased pipelining and parallelism.

Manufacturing and Testing

The NonStop TXP processor was implemented on four large PC boards using high-speed FAST logic, PALs, and high-speed static RAMs. Each processor module had from one to four memory boards. Each memory board contains up to 8MB of error-correcting memory. A 16-processor NonStop TXP system could therefore contain up to 256MB of physical memory.

The NonStop TXP system was designed to be easy to manufacture and efficient to test. Data and control registers were implemented with shift registers configured into several serial-scan strings. The scan strings were valuable in isolating failures in field-replaceable units. This serial access to registers also made board testing much faster and more efficient because the tester could directly observe and control many control points. A single custom tester was designed for all four IPU boards and for the memory-array board.

VLX PROCESSOR MODULE

The VLX processor module combines advanced VLSI technology with the fault-tolerant features of its predecessors. This processor module uses ECL gate array technology to implement its dual data path structure and other extensions to the TXP system. These features include:

- Dual interleaved control store
- 83.3 nanosecond cycle time
- 64 KB direct mapped, store-through cache with 16-byte block size
- Hardware cache fill
- 256 MB of physical memory addressing
- Up to 96 MB of physical memory with 48 MB memory boards
- Online power and temperature monitoring
- Four-stage instruction pipeline, supporting single clock instruction execution
- Dual 20 MB/sec DYNABUS interprocessor bus

One of the VLX processor module's printed circuit boards appears in Figure 9. The VLX IPU uses 32-bit native addressing and 64-bit main memory transfers to improve upon the transaction throughput of its predecessors, move large amounts of data, and lower the cost per transaction. Failed component sparing in cache memory allows a single malfunctioning component to be replaced by means of a logical switch to a spare--and thus, a single point of failure does not require a service call.

Chips in the VLX processors contain up to 20,000 circuits, producing modules with over three times the density of the TXP processor. This increased density adds functions that enhance error checking and fault correction, as well as performance. By making it possible to reduce the number of components and interconnections, the increased density improves both performance and reliability.

VLX processor gate arrays use Emitter-Coupled Logic (ECL) for enhanced internal performance, and Transistor-to-Transistor Logic (TTL) for input/output functions. Each VLX processor includes 31 ECL/TTL gates arrays spread over only two modules.

A major goal of the VLX processor was to reduce the cost of servicing the system. This goal was accomplished in several ways, as described below.

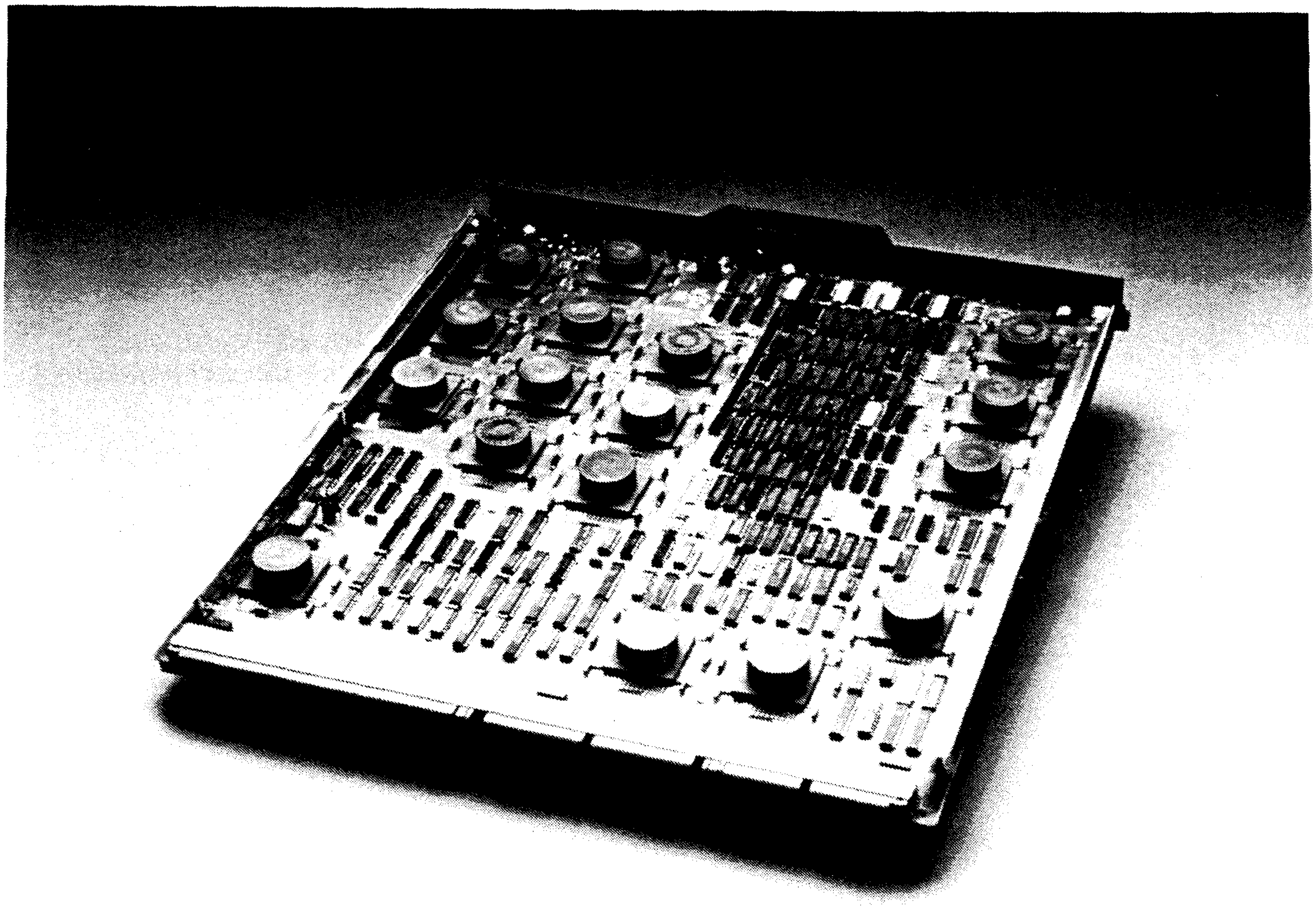


Figure 9. Printed Circuit Board from VLX Processor Module

Traditional mainframe computers have error-detection hardware as well as hardware that allows instructions to be retried after a failure. This hardware is used both to improve availability and to reduce service costs. The Tandem architecture does not require instruction retry for availability; processors can be fail-fast. The VLX processor is the first Tandem processor to incorporate a kind of retry hardware, primarily to reduce service costs.

In the VLX processor, most of the data path and control circuitry is in high-density gate arrays, which are extremely reliable. This design leaves the high-speed static RAMs in the cache and the control store as the major contributors to processor unreliability. Both the cache and the control store are designed to retry intermittent errors, and both have spare RAMs that can be switched in to continue operating despite a hard RAM failure [Horst, 1989].

The cache provides store-through operation, so there is always a valid copy of cache data in main memory. A cache parity error just forces a cache miss, and the correct data is refetched from memory. The microcode keeps track of the parity error rate; when this rate exceeds a threshold, the microcode switches in the spare RAM.

The VLX control store has two identical copies to allow a two-cycle access of each control store starting on alternate cycles. The second copy of control store is also used to retry an access in case of an intermittent failure in the first copy. Again, the microcode switches in a spare RAM online once the error threshold is reached.

Traditional instruction retry was not included due to its high cost and complexity relative to the small improvement in system MTBF it would yield.

There is also parity checking on all data paths, single-bit error correction and double-bit error detection on data in memory, as well as single-bit error detection on addresses. Bus control lines are checked for line errors, and hardware consistency checks are used throughout the system.

Each processor contains a microprocessor-based diagnostic interface, which ensures the processor is functioning properly before the operating system receives control. Pseudo-random scan diagnosis is conducted to provide a high-level of coverage and a short execution time. Correct operation of the processor is verified before processing begins.

Maintenance

For the VLX system, the Tandem Maintenance and Diagnostic System (TMDS) replaced the Operations and Service Processor (OSP) used on the NonStop II and TXP processors. Information about TMDS appears later in this chapter, under Maintenance Facilities and Practices.

CLX PROCESSOR MODULE

The CLX system was designed to fill the need for a low-cost distributed system. The design goal was to provide user serviceability, modular design, and fault tolerance with lower service and maintenance costs.

The CLX is based on a custom CMOS chip set developed using silicon compilation techniques [Lenoski, 1988]. The original CLS-600 processor was introduced in 1987 and is based on 2.0μ CMOS. The silicon compiler allowed the processor chip to be retargeted into 1.5μ CMOS for the CLX-700. The CLX-700, introduced less than 18 months after the 600, raised performance by 50%. While further process retargeting will yield even faster CLX's in the future, the performance numbers given in this section are for the CLX-700.

All CLX processors have a similar micro-architecture that integrates the features of both traditional board-level minicomputers and high-performance VLSI microprocessors. This hybrid design incorporates several novel structures, including a single static RAM array that serves three functions: writeable control store, data cache, and page table cache. The processor also features intensive fault checking to promote data integrity and fault-tolerant operation.

A fully-equipped, single-cabinet CLX system contains two processor boards with optional expansion memory, six I/O controllers, five 145MB disk drives, and one cartridge tape drive. Dual power supplies and cooling fans are also included in the cabinet. The entire system operates within the power, noise, and size requirements of a typical office environment. To expand the system, the customer simply adds more I/O or processor cabinets to the basic configuration. The CLX system architecture appears in Figure 10. A view of the actual cabinet appears in Figure 11.

As with the other Tandem processors, each processor communicates with other processors over two interprocessor buses (IPBs). Each bus operates synchronously on 16-bit wide data, and each provides a peak bandwidth of 20MB/sec. The two buses transfer data independently of one another, providing a total bandwidth of 40 MB/sec for a maximum of eight processors.

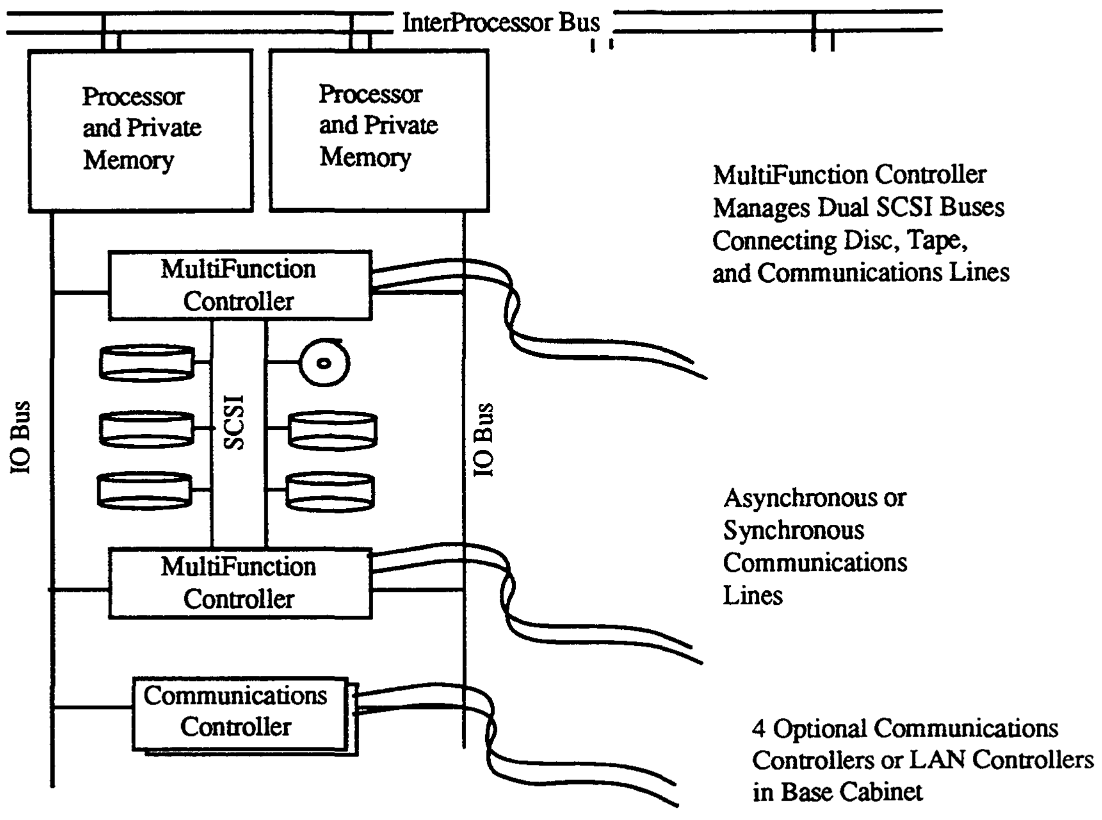

InterProcessor Bus
Processor and Private Memory
Processor and Private Memory
MultiFunction Controller Manages Dual SCSI Buses Connecting Disc, Tape, and Communications Lines
MultiFunction Controller
IO Bus
SCSI
IO Bus
Asynchronous or Synchronous Communications Lines
MultiFunction Controller
Communications Controller
4 Optional Communications Controllers or LAN Controllers in Base Cabinet

Figure 10. CLX System Architecture

Processors communicate with I/O devices either through a local I/O bus or through the IPB to another processor and its I/O bus. Each processor contains a single asynchronous, burst-multiplexed I/O bus that transfers data at a maximum rate of 4.4 MB/s to a maximum of 16 controllers. As with the other processors, these controllers are dual-ported and can be driven by either of the processors to which they are attached.

The CLX uses a Multi-Function Controller (MFC) based on a Motorola 68010 microprocessor to control dual Small Computer System Interfaces (SCSIs) supporting up to five disk drives and one tape drive. The MFC runs its own real-time operating system kernel that coordinates independent disk control, tape control, synchronous and asynchronous communication, and remote maintenance tasks.

The CLX processor module's printed circuit board appears in Figure 12.

Through the maintenance buses, maintenance and diagnostic information can flow between the system control panel, processors, multifunction controllers, and environmental monitors.

Enhanced diagnostic software and careful design of all replaceable units allows customers to service 98% of all component failures.

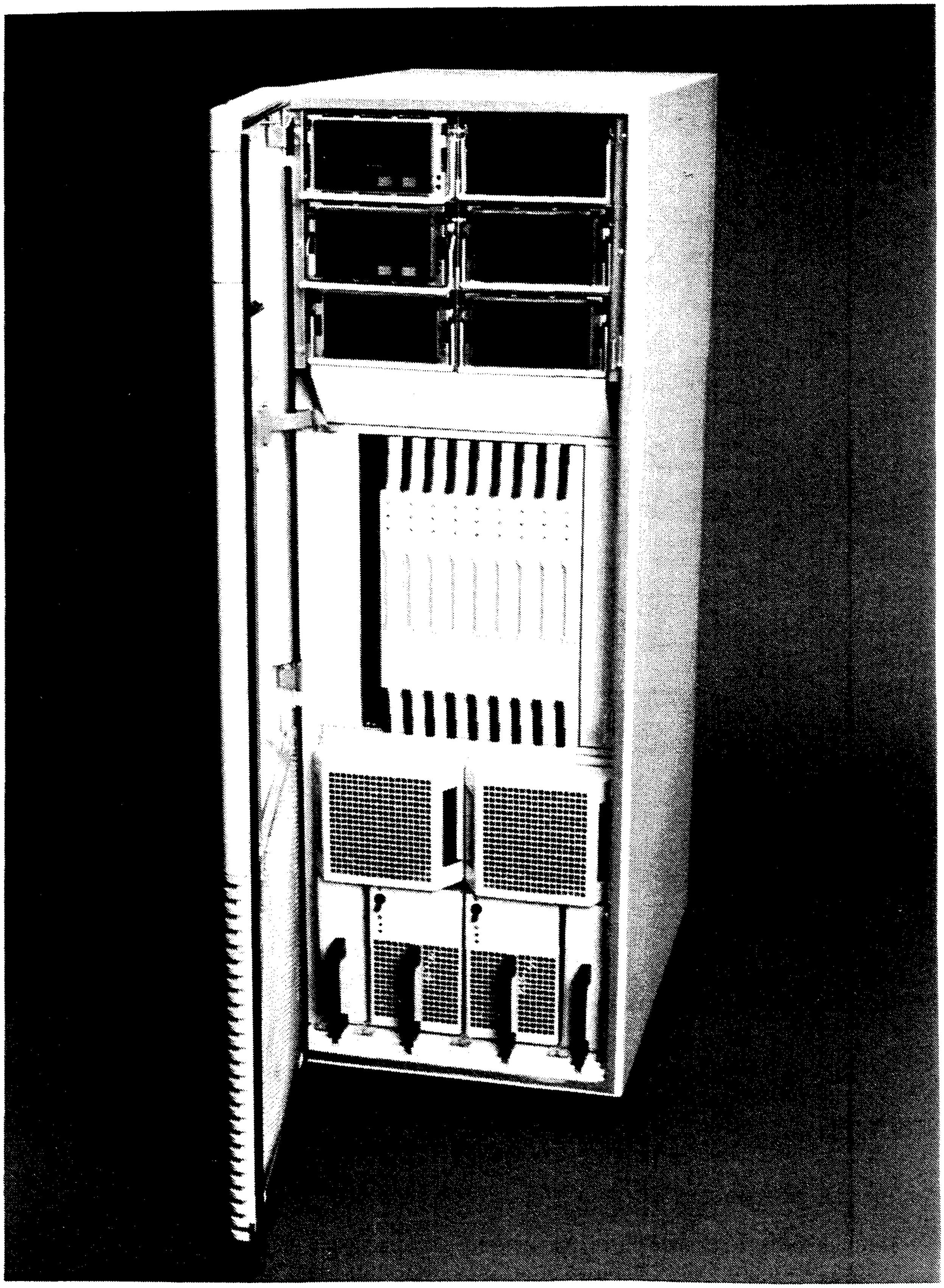


Figure 11. CLX System Cabinet (Front View), showing, from top to bottom: cartridge tape drive and SCSI disk drives, two processor modules with memory and I/O boards, dual fans, power supplies, and batteries for memory power.

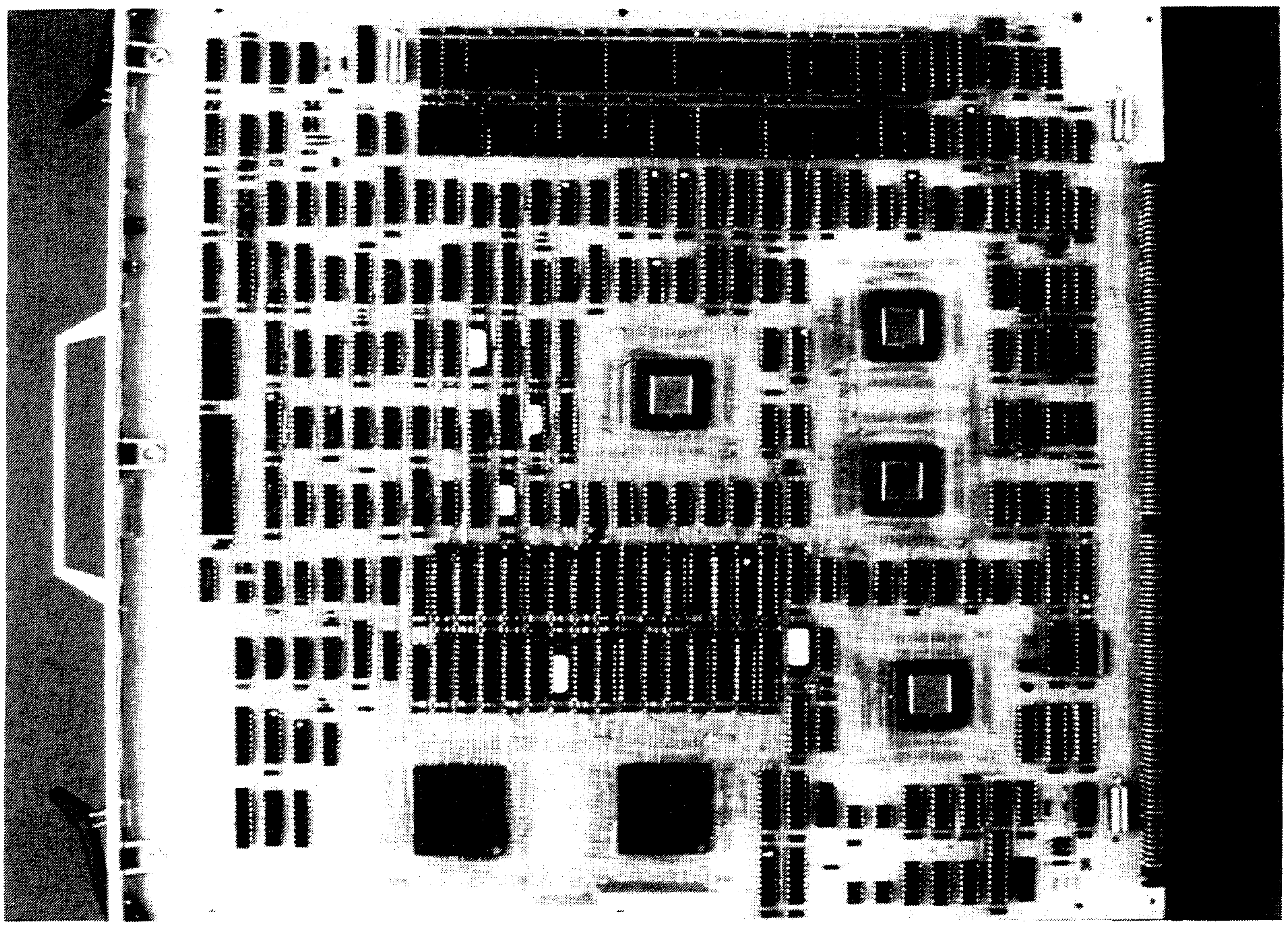


Figure 12. Printed Circuit Board for CLX Processor Module, showing dual, lockstep IPU chips (bottom center).

The processor logic resides within six custom CMOS chips, allowing the processor and main memory to be implemented in a single board. A block diagram of the processor appears in Figure 13. The chip set was designed using a silicon compiler supplied by Silicon Compiler Systems Corporation. The two IPU chips are identical, running in lock-step to form a fully selfchecking module. These chips provide the complete IPU function. They work together with a single bank of static RAM that serves as the microcode control store, page table cache, and data/instruction cache. The RAM provides for 14K x 7B of microcode and scratch pad memory, 4K entries of page table cache, and 128KB of instruction/data cache. An IPU chip appears in Figure 14.

The MC chip includes the control and ECC logic (SEC/DED) to interface to the 8 MB of on-board dynamic memory and 8 MB of expansion memory. This chip contains FIFO buffers to hold data in transit to and from the main-memory dynamic RAMs, using nibble mode accesses. The chip also features a wraparound mode to support high-speed memory-to-memory block transfers.

Each processor has one IPB chip per interprocessor bus. Each chip contains a sixteen word in-queue and a sixteen word out-queue. These queues work with on-chip state machines for sending and receiving interprocessor message packets asynchronous to processor execution.

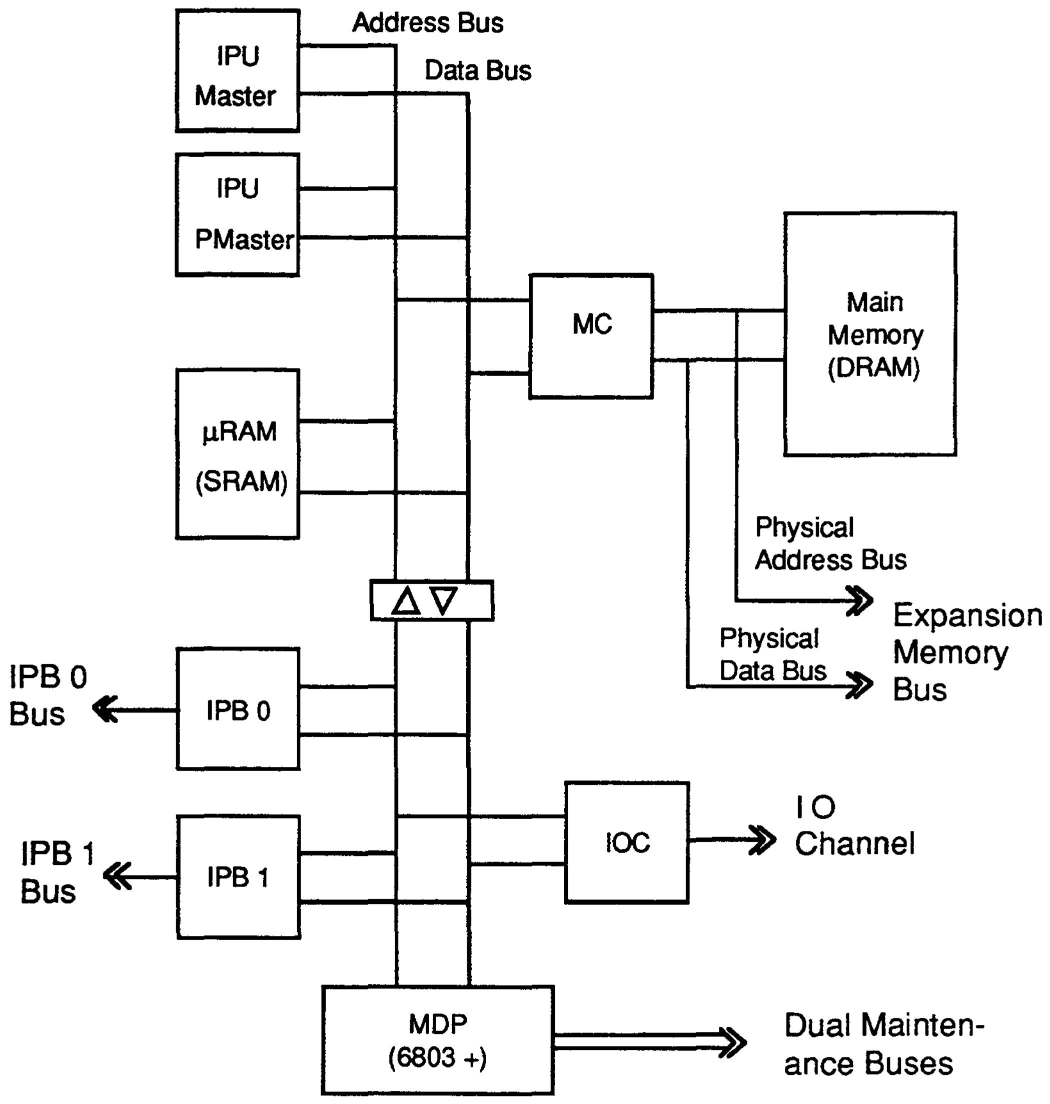


Figure 13. CLX Processor Block Diagram

The IOC chip contains the data latches and logic to control a burst-multiplexed, asynchronous I/O bus. The I/O bus is primarily controlled by the IPU, but it can also handle DMA transfer polling and selection without microcode intervention. The bus also includes priority-encoding logic to support the fair servicing of I/O interrupts.

The final component of the processor is a Motorola 6803-based maintenance and diagnostic processor. This processor furnishes overall control of the main processor, as well as a diagnostic and error reporting path for the main processor through the maintenance buses.

The IPU architecture for the CLX, as mentioned earlier, is a blend of features found in both minicomputer and microcomputer architectures. The IPU chip's external interface is similar to that of a VLSI microprocessor. For example, the interface features one address bus, one data bus, and one status bus along with miscellaneous signals such as an interrupt request, memory wait controls, and three-state bus controls. Minicomputer features, however, are manifested in the size of the address bus, which is 18 bits wide, and the data bus, which is 60 bits wide.

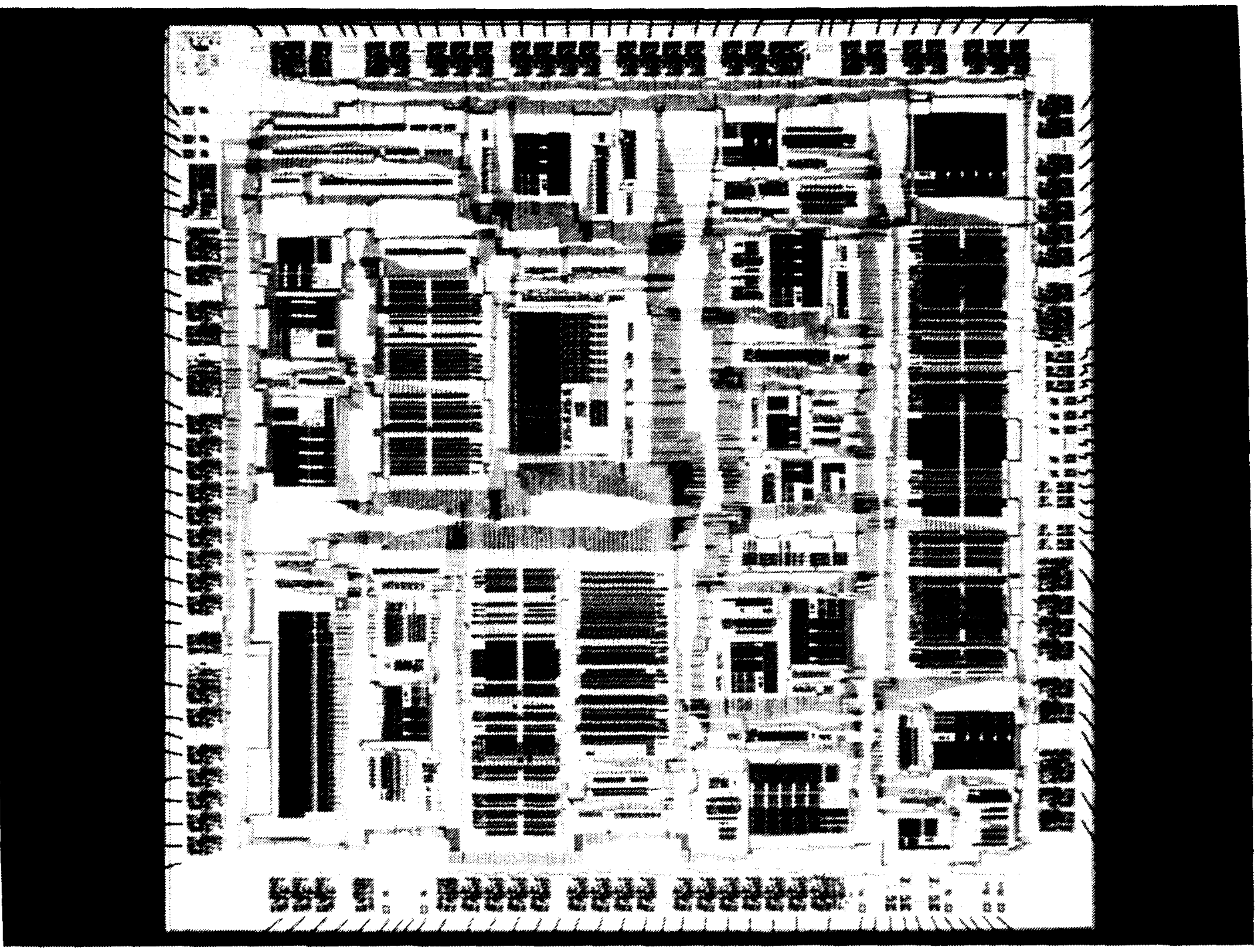


Figure 14. IPU Chip for CLX Processor Module

The IPU chip interface merges many buses that would normally be separate in a minicomputer architecture. In particular, a bus cycle on the CLX can execute any of the following functions:

- Microcode control store access
- Instruction or data cache access
- Page Table Cache (TLB) access
- Main memory access
- Microcode scratchpad memory access
- Special module (IPB, IOC, MDP) access

Merging these buses reduces the cost of the processor by decreasing the number of static RAM parts and their associated support logic and by reducing the number of pins needed on the IPU chips. If this merging were not implemented carefully, performance would have degraded significantly. To reduce the bandwidth required on the buses and to minimize the impact on performance, the designers employed a variety of techniques, including the use of:

- A small on-chip microcode ROM
- A virtually-addressed cache
- Nibble mode DRAM with block operations to the main memory controller
- Higher-level control operations for special modules

The greatest contribution toward reducing the performance impact of the merged buses comes from the on-chip micro-ROM. The micro-ROM contains 160 words of microcode (54 bits per word), with an identical format to the off-chip microcode. This ROM is addressed by either the microcode PC or through an explicit index specified in the previous line of microcode. The microcode PC addressing is used to implement the inner loops of IPB and IOC transfers, cache filling routines, and block memory moves. The explicit index is used for short sequences of common microcode. These lines overlay otherwise sequential lines of external microcode. Use of these ROM lines does not conflict with other micro-operations.

The virtually-addressed cache reduces the number of page-table accesses; thus it decreases the required bandwidth to the shared micro-RAM. Likewise, the use of block-mode commands to the memory controller reduces the number of memory commands needed during cache filling and block moves. Finally, the use of higher-level commands to the IPB and IOC reduces the control transfers needed to receive and transmit data to these devices. The on-chip micro-ROM together with these other features reduce the penalty of using a single bus approach from over 50% to less than 12%.

The main alternative to the micro-ROM used on the CLX would be an emulation scheme where a subset of instructions are implemented entirely by internal ROM, and the remaining instructions are emulated by a sequence of the simpler instructions. The micro-ROM scheme has two chief benefits when compared with emulation techniques. First, it provides much higher performance when the amount of ROM space is limited relative to the number of instructions that must be implemented. Second, the dispatch of each instruction is to external writeable control store, enabling any ROM microcode errors to be corrected externally (although with some performance penalty).

Data Integrity Through Fail-Fast Operation

In a NonStop system, fail-fast hardware operation is essential to providing fault tolerance at the system level. Fail-fast operation requires that faults do not escape detection and that the processor is halted before a fault is propagated. The CLX's processor module uses a variety of error-checking strategies to provide extensive fault coverage.

The IPU chip itself is covered by a duplicate-and-compare scheme. This scheme minimizes the amount of internal logic required for a high degree of coverage, and it maximizes the utilization of existing library elements in the silicon compiler CAD system. The implementation of the IPU's duplicate-and-compare logic appears in Figure 15.

The CLX's scheme improves the fault coverage of other duplicate-and-compare schemes by providing for a cross-coupling of data and parity outputs. One chip, designated the data master, drives all data outputs while the other chip, designated the parity master, drives all parity outputs. This action ensures that both chips' outputs and checking logic are active, and that latent errors in the checking logic cannot lead to an undetected double failure. The parity outputs of the IPU also cover the address and data lines connecting the IPU to other parts of the processor and the micro-RAM.

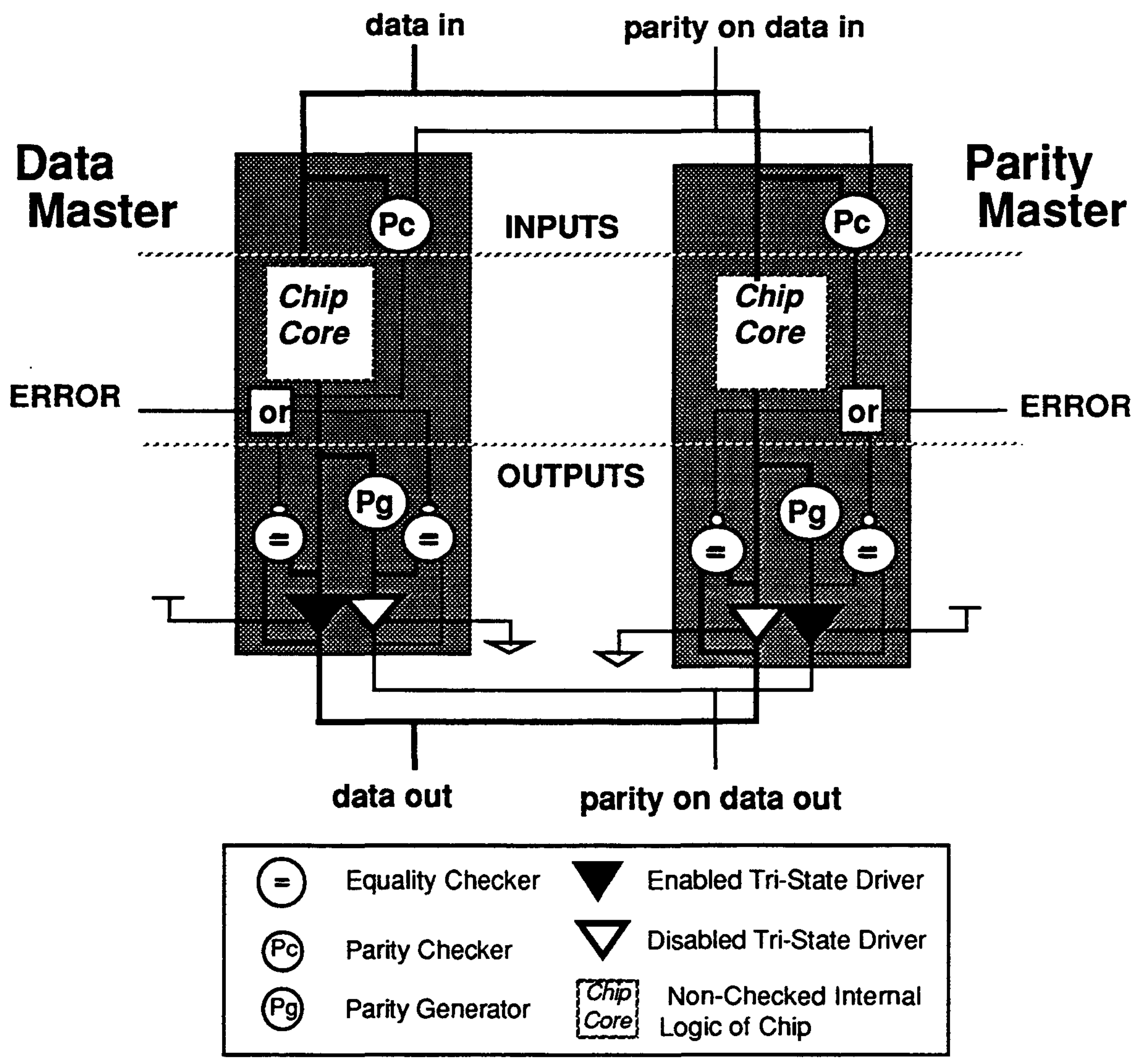


Figure 15. CLX Processor Cross-Coupled Checking

Within the memory system, ECC with encoded address parity provides checking of all memory system data paths. In addition, redundant state machines are contained in the MC chip and in the external RAS/CAS generation logic. The state transitions of these machines are encoded into CRC registers whose outputs are compared. The resulting structure produces a high fault coverage for both the data and control sections of main memory.

The IOC and IPB provide for parity protection of the data and control lines to which they are interfaced. In addition, they are protected by end-to-end checksums supported in software; these checksums guarantee the integrity of their respective buses.

CYCLONE PROCESSOR MODULE

The design goals of the Cyclone system were to significantly increase performance, while providing improvements in serviceability, manufacturability, installability, and overall cost of ownership.

The Cyclone processor, introduced in 1989 [Horst, Harris, and Jardine, 1990], provides more than three times the performance of the VLX, yet it retains full object-code compatibility. About half of the performance improvement is due to higher clock rates, and the other half is due to the new micro-architecture. Much of the architectural improvement is due to the ability to execute up to two instructions per clock cycle, a technique that has been called "superscalar" [Jouppi and Wall, 1989]. Other improvements are due to parallel data paths and new designs for the caches and main memory.

Cyclone Technology

The technology for Cyclone is a combination of ECL for speed, CMOS for high density, and TTL for standard interfaces and buses. The ECL gate arrays, jointly developed by Tandem and Advanced Micro Devices (Sunnyvale, Calif.), contain approximately 5000 gate-equivalents. These gate arrays are implemented in 155-pin grid-array packages. The pins can be individually programmed for TTL or ECL interfaces.

The processor is implemented on three 18 x 18 inch circuit boards, with a fourth board holding either 32MB or 64MB of main memory. A second, optional, memory board allows expansion up to a total of 128MB of main memory per processor (with 1MBit DRAMs). The circuit boards have 8 signal layers, four of which have controlled impedance for routing ECL signals.

Like the VLX, Cyclone uses an interleaved control store, allowing two clock cycles for access. The control store is implemented in 16KBit x 4 CMOS SRAMs, surface-mounted on a double-sided ceramic substrate, which is then vertically mounted on the main boards.

Cyclone Processor Architecture

The superscalar design of the Cyclone processor was necessitated by the fact that the VLX processor executes many frequent instructions in a single clock cycle, and the goal of three times VLX performance could not realistically be achieved based on cycle time only. Such a fast cycle time would involve higher risk, lower reliability, and higher product cost. Thus, Cyclone needed to break the 1-cycle-per-instruction barrier.

At peak rates, the Cyclone processor executes two instructions per clock cycle. To do this, it incorporates an independent, asynchronous Instruction Fetch Unit (IFU), separate large caches for instructions and data, a deep pipeline, and a dynamic branch prediction mechanism. A block diagram of the Cyclone processor is shown in Figure 16.

The Instruction Fetch Unit operates independently of the rest of the processor. It fetches up to two instructions per clock cycle from the instruction cache, decodes the instructions to determine whether they are candidates for paired execution, and presents a microcode entry address for either a single instruction or a pair of instructions to the Control Unit and Data Unit for execution. It also assists in the execution of branching instructions and of exception handling. Up to 16 different instructions can be in some stage of execution at any point in time. The IFU is shown in more detail in Figure 17.

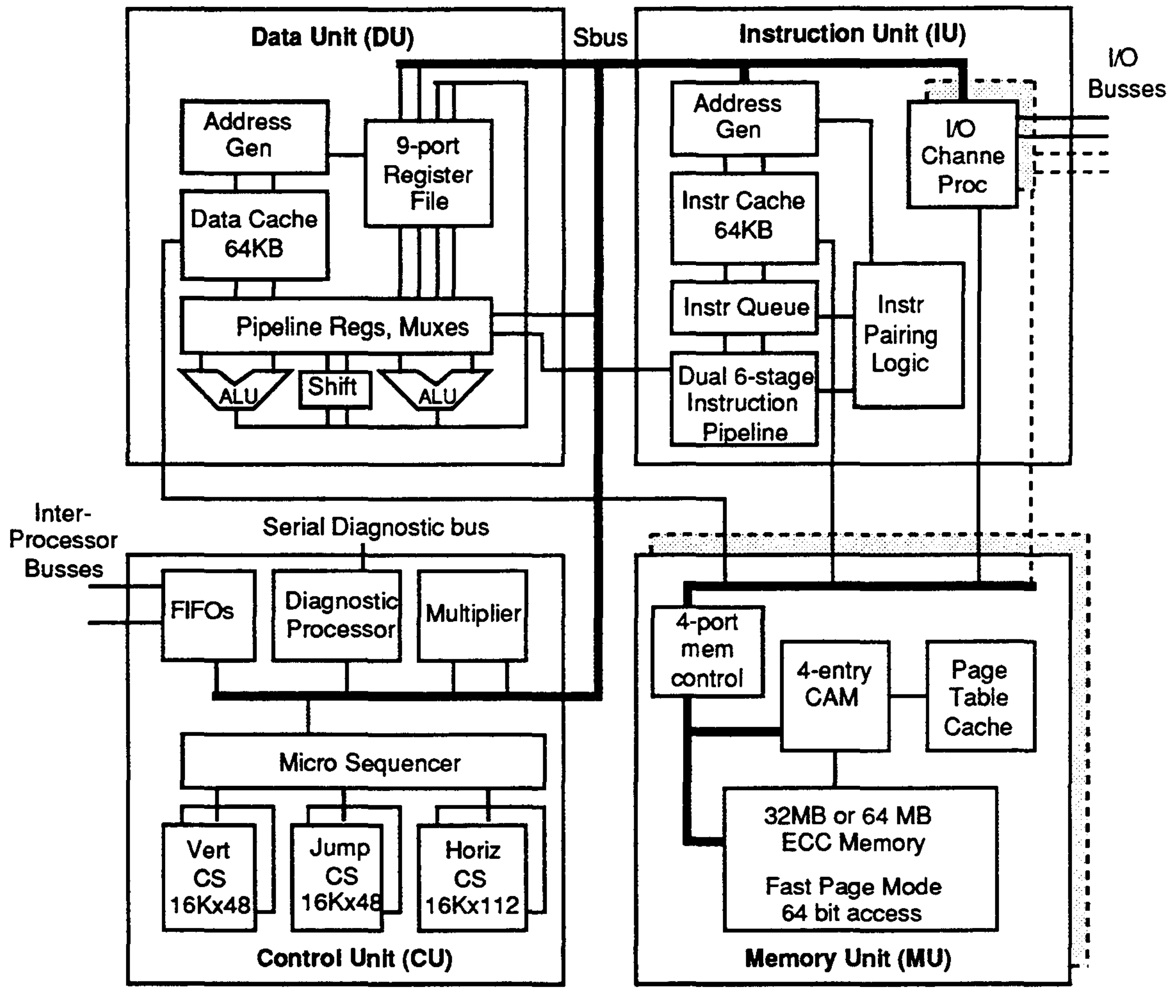
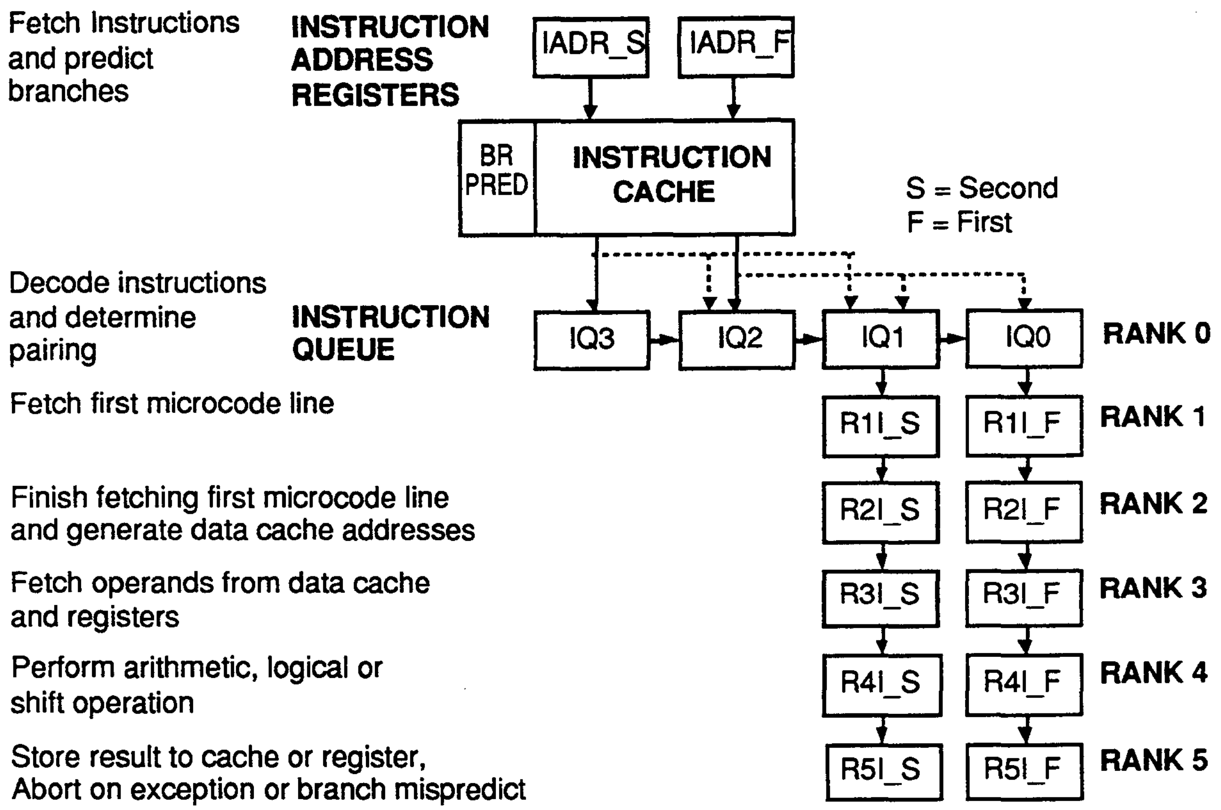


Figure 16. Cyclone Processor Block Diagram

The Cyclone processor uses a dynamic branch prediction mechanism for conditional branches. This mechanism relies on the premise that when a particular branch instruction is repeatedly encountered, it will tend to be taken (condition met) or not taken (condition not met) the same direction each time. An extra bit for each instruction is included in the instruction cache. This bit records the branch direction actually taken by the last execution of each branch instruction in the cache. When a branch is fetched from the instruction cache, the IFU "predicts" that the branch will choose the same path as the previous time, so it continues prefetching along the predicted path. When the branch instruction later enters the execution pipeline, the microcode determines whether the prediction was correct. If so, it does nothing. If not, the microcode directs the IFU to back up and resume instruction fetching along the other path. Modeling has shown that this mechanism would be correct between 85% and 95% of the time. The result is an average cost of 1.3 to 1.9 cycles per branch instruction. In addition, because the branch prediction occurs early in the prefetch queue, branches may be executed in a pair with the previous instruction, the sequentially following instruction, or the target instruction.



Fetch Instructions and predict branches
INSTRUCTION ADDRESS REGISTERS
IADR_S
IADR_F
BR PRED
INSTRUCTION CACHE
S = Second
F = First
Decode instructions and determine pairing
INSTRUCTION QUEUE
IQ3
IQ2
IQ1
IQ0
RANK 0
Fetch first microcode line
R1I_S
R1I_F
RANK 1
Finish fetching first microcode line and generate data cache addresses
R2I_S
R2I_F
RANK 2
Fetch operands from data cache and registers
R3I_S
R3I_F
RANK 3
Perform arithmetic, logical or shift operation
R4I_S
R4I_F
RANK 4
Store result to cache or register, Abort on exception or branch mispredict
R5I_S
R5I_F
RANK 5

Figure 17. Cyclone Instruction Fetch Unit

The Cyclone data path uses two 16-bit ALUs, similar to the TXP and VLX, but with two major differences. Firstly, the two ALUs are connected with the register file in a very general way. This interconnection is necessary for the execution of many of the instruction pairs, but it is also quite useful in improving the performance of many complex, multi-cycle instructions as well. In addition, the two ALUs can be linked together, so that 32-bit arithmetic can be accomplished in a single clock cycle.

Both the instruction cache and the data cache are capable of fetching two adjacent 16-bit words in a single cycle, regardless of alignment. This feature, along with the instruction pairing, the nine-port register file, the 32-bit ALU capability, and the deep pipeline, allows the execution of a double (32-bit) load instruction and a 32-bit arithmetic instruction, as a pair, in a single clock cycle.

The Cyclone sequencer is similar to the VLX sequencer, in that two copies of the control store are used to allow two-cycle access time. In addition to allowing the use of slower, denser CMOS RAM parts, the two copies provide backup for each other. In the event of an error in fetching a word from control store, the alternate bank is automatically accessed. If the error is a soft error, one of the banks can be refreshed from the other bank. In the event of a hard failure, a spare RAM can be switched in.

Part of the control store is duplicated yet again (four total copies). This duplication allows both potential paths of a microcode branch to be fetched simultaneously, thus minimizing the penalty for microcode choices.

In the Cyclone processor, virtual addresses are sent directly to the main memory. A four-entry content-addressable memory (CAM) compares each access to the row address that previously addressed a bank of dynamic RAM. When the addresses match, the dynamic RAM column address is generated by a few bits from the CAM plus the address offset. Translation from virtual to physical address is performed only on a CAM miss. Since the translation is performed infrequently, it was possible to implement the Page Table Cache (translation lookaside buffer) in relatively slow, but dense, CMOS static RAMS.

For both increased bandpass and increased connectivity, Cyclone allows the connection of up to four I/O channels per processor, whereas previous Tandem processors allowed only one channel. Two channels are supplied on the IU board, while an additional two channels are available on an optional board. The maximum Cyclone processor thus contains six boards (three processor, two memory, one optional I/O).

Cyclone Fault Tolerance, Data Integrity, and Reliability Features

Parity checking is used extensively to detect single-bit errors. Parity is propagated through devices that do not alter data, such as memories, control signals, buses, and registers. Parity prediction is used on devices that alter data, such as arithmetic units and counters. Predicted parity is based strictly on a device's data and parity inputs; it does not rely on the device's outputs, which may be faulty. Thus, an adder might generate an erroneous sum, but the parity that accompanies the sum will correspond to the correct result. Parity checkers downstream will then detect the error. Invalid-state checking or duplication-and-comparison are used in sequential state machines.

The hardware multiplier is protected by a novel technique similar to Recomputation with Shifted Operands (RESO) [Sohi, Franklin, and Saluja, 1989]. After each multiplication, a second multiplication is initiated with the operands exchanged and one operand shifted. Microcode compares the two results whenever the multiplier is needed again or before any data leaves the processor. Unlike other implementations of RESO, these checking cycles incur almost no performance penalty because they occur concurrently with unrelated execution steps.

If the processor hardware detects a fault from which it cannot recover, it shuts itself down within two clock cycles, before it can transmit any corrupt data along the interprocessor bus or I/O channel. The error is flagged in one or more of the approximately 300 error-identification registers, allowing quick fault isolation to any of the 500 hardware error detectors in each processor.

Like the VLX, Cyclone processors include spare RAM devices in all of the large RAM arrays, such as caches and control stores. These devices are automatically switched in to replace hard-failed RAMs.

Cyclone systems make extensive use of fiber-optic interconnections, which, among other advantages, increase reliability. The Dynabus+ fiber links between sections are described earlier in this chapter. In addition, Cyclone systems use fiber optic links between the disk controllers and the disk units themselves and between the communications controllers and outboard communications concentrators.

The Cyclone approach to diagnostics is similar to the approach taken on VLX, but it goes beyond in many respects. Test coverage of microprogrammed diagnostic routines has been dramatically increased, and more support has been added for Pseudo-Random Scan Test (PRST). Together, these changes improve the ability to automatically diagnose faults online and quickly pinpoint the Field Replaceable Unit responsible for the fault. In addition, a "Guided Probe" facility, which leads factory personnel through the diagnostic process, enhances the product's manufacturability.

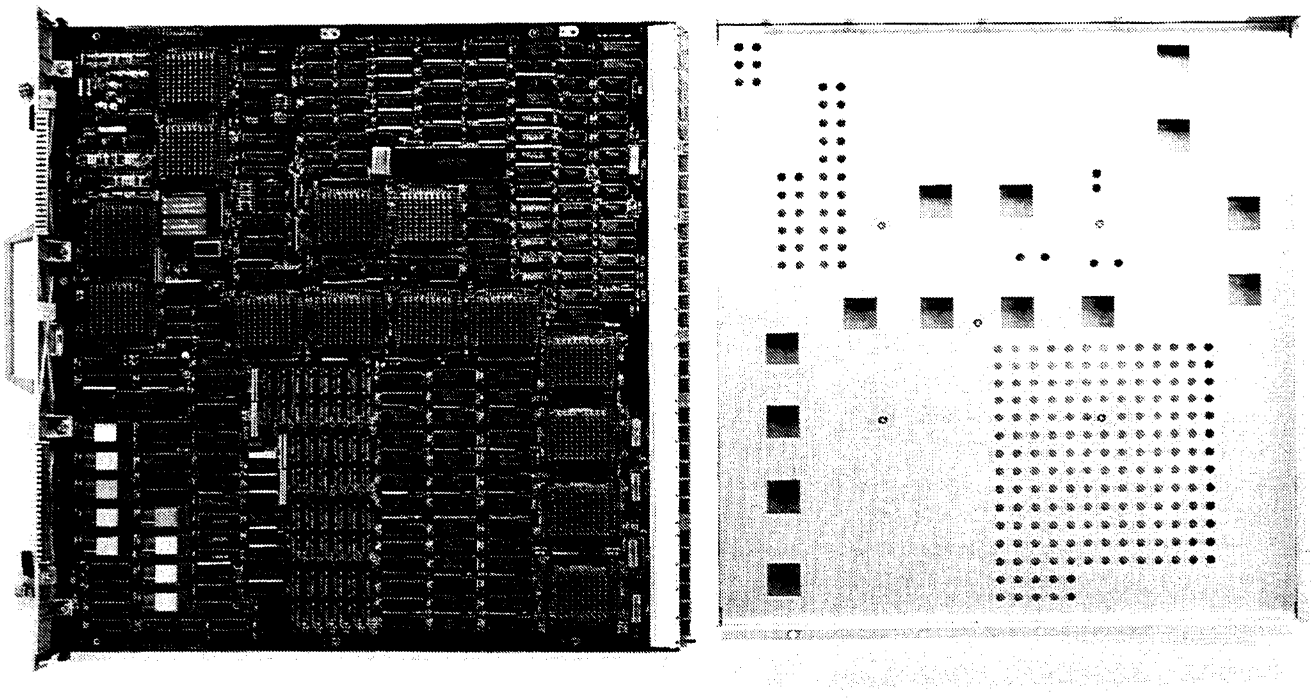


Figure 18. Cyclone Printed Circuit Board, showing Impingement Cooling.

Like the VLX processor, the Cyclone processor is implemented primarily in ECL gate arrays, although Cyclone's arrays are considerably more dense. Because of this added density and the increased clock speed, Cyclone's gate arrays dissipate up to 11 watts. In order to cool these devices without resorting to liquid cooling, Cyclone uses an *impingement* air cooling technique. Instead of blowing chilled air across the circuit board, Cyclone's boards include an orifice plate, which serves to focus the chilled air onto the hottest components. This design is shown in Figure 18. The result is that Cyclone's devices, in spite of dissipating much more power, operate at a junction temperature 10°c cooler than those in the VLX, significantly increasing reliability.

The VLX, CLX, and Cyclone processors are now the mainstays of the Tandem computer systems. The next section discusses the methods of supporting these systems.

MAINTENANCE FACILITIES AND PRACTICES

Tandem's tools, facilities, and practices for hardware maintenance have evolved considerably in the last ten years. Over time, the trend has been to increasingly share maintenance responsibility with its customers, making it easier for customers to quickly resolve hardware problems on their own.

EARLY METHODS

Early maintenance systems were mainly based on the use of online diagnostic tests to isolate the causes of readily-apparent failures. Subsequent systems, however, moved toward the ability to detect failures automatically, analyze them, report upon them, and track their repair.

The first diagnostic and maintenance tools were very primitive. For example, to support the NonStop I systems, only a set of lights and switches were available on each processor for communicating error information and for resetting and loading the processor.

In 1979, the DIAGLINK diagnostic interface was introduced to permit access to the system from remote maintenance sites. DIAGLINK featured an asynchronous modem. With DIAGLINK, customer engineers could remotely examine customers' systems, obtain system status by running operating system utilities, and execute diagnostics with customer assistance for remote, low-level debugging.

OPERATIONS AND SERVICE PROCESSOR (OSP)

The NonStop II system replaced DIAGLINK with an Operations and Service Processor (OSP). The OSP was a microcomputer system that communicated with all processors and a maintenance console. The OSP offered all of the capabilities of DIAGLINK, as well as additional features to diagnose failures and to operate a system remotely. The OSP enabled operations and support personnel to obtain an internal view of the status of each processor.

The OSP communicates with the Diagnostic Data Transceiver (DDT) included as a part of each processor module in the system. This communication allows the operator to diagnose software and hardware problems through the operator's console. The DDT monitors the status of the DYNABUS interface, I/O channel processor, memory, and IPU, including the internal data paths. For example, the DDT enables the operator to put the processor in single-step mode and monitor the contents of its registers before and after execution of a specific instruction.

The OSP includes a built-in modem that can connect it to a remote terminal or to another OSP. This connection allows an operator or customer engineer to diagnose and possibly even correct problems from the remote site. A remote customer engineer can, for example, run microdiagnostics residing on a local OSP. Alternatively, the customer engineer can download diagnostics from the remote OSP to the local OSP, remotely reset and load processors, and display error information.

For the TXP system, the OSP was enhanced to include an asynchronous modem, improved microdiagnostics, more remote operations capability, and additional remote support capabilities.

Tandem Maintenance and Diagnostic System (TMDS)

For the VLX system, the Tandem Maintenance and Diagnostic System (TMDS) replaced the OSP. TMDS provides a framework for problem detection with the VLX system, which was intended to reduce the cost of ownership in various ways. A major aspect of this attack on costs was improved diagnostic and maintenance facilities.

TMDS permitted the elimination of front panel lights and switches from the system design, dramatically streamlining maintenance activities. Unlike its predecessors, TMDS operated online without requiring significant system resources. It provided a uniform interface to many diagnostic subsystems [Troisi, 1985].

By the time of the VLX system, the maintenance strategy had evolved beyond real-time monitoring of system components to include automatic online fault analysis and automatic dial-out for online support by remote service centers [Eddy 1987]. TMDS was based on that strategy. Today, although it is known primarily for its use on the VLX, CLX, and Cyclone systems, TMDS is compatible with all NonStop systems. It runs under the GUARDIAN operating system and is distributed automatically to all customers.

Through pervasive instrumentation, an internal fault-tolerant maintenance and diagnostic subsystem continuously monitors the system's processors, power supplies, and cabinet environments. When the GUARDIAN operating system or an I/O process detects a change of state or an irregular event, it writes an event signature to an event log on disk. Then, TMDS examines each event signature. If further study seems advisable, TMDS starts a module known as an automatic fault analyzer. Thus, TMDS supports both active testing of components and symptom-based fault analysis.

TMDS fault analyzers relieve the customer of the need for an intimate knowledge of hardware, status codes, or specific error values. TMDS uses "if-then" rule-based algorithms to evaluate events against a knowledge base, automating many of the detection, interpretation, and reporting tasks previously required of a console operator. The knowledge base contains a range of acceptable component states and environmental factors. If the fault analyzer finds that an event falls within the acceptable range, TMDS saves the fault analysis in a local log--a catalog of system events and patterns that can aid future troubleshooting.

However, if a fault analyzer detects an event that suggests an active or potential problem, TMDS transmits a signal to a fault-tolerant service processor called the Remote Maintenance Interface (RMI). The RMI consists of dual Motorola 68000-based processors that communicate with each other and with other subsystems over dual bit-serial maintenance buses. The processors, FOX controllers, and power supply monitors all connect to the maintenance buses. The RMI supports all the functions of the old OSP, but does so as a much more compact unit--two circuit boards residing in one of the cabinets (VLX and Cyclone) or a part of the MFC (CLX). Through a synchronous protocol, a special communication process, and a password requirement, the RMI also greatly reduces the risk of unauthorized users gaining access to the system through the diagnostic facility.

When it receives a problem signal, the RMI alerts the on-site operator and, on the CLX, VLX, or Cyclone system, optionally dials out to a Tandem National Support Center (TNSC). Tandem staffs two such centers: one to service sites in the North America and one for sites in the Europe. Other TNSCs are planned as business needs for them develop.

Through the RMI, either the on-site operator or the remote analysts and engineers at the TNSC can review the event log, run diagnostics, test components, and isolate and diagnose the problem. On newer equipment, these actions include detecting out-of-spec intake and exhaust cabinet temperatures, malfunctioning disks or tape controllers, and faulty fans. TMDS also uses processor diagnostics to test power supplies, clocks, and batteries.

If necessary, the TNSC can dispatch a field service engineer for on-site troubleshooting or part replacement. The TNSC staff has identified and diagnosed the problem, so the service engineer is very likely to arrive with the correct replacement part in hand.

TMDS also allows analysts and engineers to run online diagnostics to identify problems that fault analyzers don't cover. In fact, many diagnostics can be run while the device being studied is online. In the worst cases, only the problem device needs to be shut down; under previous diagnostic systems, both the device and its controlling processor needed to be shut down. In any case, testing with TMDS only minimally affects the system's performance. Processing continues unhindered by the diagnostic tests, unless a processor itself is being evaluated.

As an example of how TMDS operates, imagine that a tape I/O process detects an error event involving a tape unit.
1. The tape I/O process immediately creates an error event and sends it to the TMDS event log.
2. TMDS signals the fault analyzer.
3. The fault analyzer localizes the error to a particular controller board.
4. The fault analyzer writes additional error information to the event log, specifying the probable field replaceable unit, the controller address, and the terminal error code. All of these actions take place within seconds.
5. After completing the analysis, TMDS dials out to the TNSC.
6. The TNSC dials back in to verify the analysis.
7. The TNSC dispatches a customer engineer to replace the controller board.
8. TMDS records the replacement in the event log.

TMDS event logs and the reports generated by the remote intervention are archived in a centralized support database. This database contains a history of service requests, diagnoses, and support actions for hardware and software. Experts periodically scrutinize the database, seeking out diagnostic patterns and irregularities that they can use to improve system maintenance.

TMDS continues to evolve, incorporating many new features. One of these features is a Built-In Self Test (BIST) method that uses pseudo-random test vectors and scan path design [Garcia, 1988]. On the CLX, the pseudo-random test covers several custom ICs, commercial MSI logic, a static RAM array, and their interconnects. The BIST also does a functional test of the dynamic RAM main memory and its control logic. The test is controlled by maintenance processor software, simplifying the processor board hardware dedicated to the BIST. With the BIST, hardware problems on the CLX processor can be detected and reported to TMDS without requiring downloaded, handwritten diagnostics.

In the Cyclone system, the power and environmental monitoring facilities have been significantly enhanced. In addition to sensing more different components (voltages both on the circuit boards and at the power supply outputs, intake and outlet air temperatures, battery condition, and fan rotation), sensors are polled much more frequently, and most sensors are replicated to allow differentiation between a failing component and a failing sensor. In addition, Cyclone's maintenance subsystem can detect the physical presence or absence of many components, such as cables, power supplies, fans, and bus terminators. Both the logical address and physical location (which cabinet and which slot within the cabinet) are automatically available to the maintenance subsystem, so that failed components can be easily identified and reliably replaced.

SOFTWARE

Overview

In the preceding discussion of the evolution of the Tandem system, the careful reader will find no mention of fault-tolerant hardware modules. In fact, one of the primary design criteria for Tandem hardware is to make it fault-*intolerant*. Any incorrectly functioning hardware module should detect the problem and, as quickly as possible, shut itself down. There are a few exceptions to this rule, such as error-correcting main memory, but the fundamental design of the Tandem system is to connect *fail-fast* hardware with fault-tolerant software.

Fault tolerance normally implies hardware redundancy. While this is also true for a Tandem system, the net additional "cost to the customer" has been kept surprisingly low, due to the many innovative features of the Tandem system. In most cases, the "redundant" modules each perform useful work in their own right and, except when they have failed, contribute to the capacity of the system. Failing modules can be replaced while the system is running. The net effect of a hardware failure is, at worst, a short period of slightly degraded performance. A system with a normal amount of excess capacity will survive a failure without any noticeable effect.

The key to fault tolerance without wasted redundancy is the GUARDIAN operating system. The following sections describe the many components of GUARDIAN, from the kernel (process and memory management and message system) to the transaction manager, networking support, and NonStop SQL. Each makes an important contribution, not only to the functionality of the Tandem system, but also to the support of fault tolerance.

Fault tolerance would be of little value without data integrity; business demands accurate record keeping and an inconsistent database is often worse than no database at all. Furthermore, a business that depends on its computer system must be able to grow that system at least as fast as the business. The following sections also describe how the system has been engineered to prevent data corruption and to provide expandability.

GUARDIAN: An Integrated OLTP Operating System

A basic difference between GUARDIAN and other systems is the very high level of software integration. Although there is the usual layering of software function, these layers are relatively closely tied together and interdependent. This approach has its costs, but the resulting efficiency coupled with a high level of functionality is unique in the computer industry.

There are many software components that contribute to the fault tolerance, data integrity, expandability, and basic functionality of the Tandem system. In this section, we will give a general overview of the elements that differentiate Tandem from other systems.

- The GUARDIAN kernel includes the usual support for the management of processes, memory, inter-process communication, names, time, peripheral I/O, process synchronization, and debugging. In addition the kernel detects failures of any processor, inter-processor bus, or I/O channel and performs recovery. Several innovative techniques are used to synchronize the independent processors and to provide a consistent view of time and system state despite failures and communication delays. Finally, the kernel supports the management of process pairs, which are the keystone of both hardware and software fault tolerance.

- The file system hides the distributed nature of the system from application processes and presents a conventional view of peripheral Input/Output devices. Communication with I/O devices and other processes is accomplished without regard to the location of the resource, be it in the same processor, another processor in the same system, or a processor in a remote system. The file system also provides checkpoint and retry mechanisms for hiding the effects of hardware and software failures from the application process.

- I/O processes manage peripheral devices and react to component failures by routing access over working paths and devices and then notifying operators and customer engineers about the need to repair or replace the failing component. I/O processes receive messages from application processes (via the file system) and perform the requested operations on physical devices. In the view of the application programmer, the I/O process and the device it manages are indistinguishable. There are dozens of different I/O processes, each designed to manage a particular class of device.

- The disk process is probably the single most important component of the system. It provides data-integrity and reliable access to data despite processor, channel, controller, or media failure. It supports mirrored disks efficiently, making good use of the dual access paths to data. It supports four types of file structures as well as SQL tables; it supports file partitioning, alternate indices, data security, and main memory cacheing for both reading and writing of data. It supports full transaction management, including two-phase locking and two-phase commit protocols. Last, but far from least, it can execute those parts of SQL queries that require data local to one disk, greatly reducing message traffic for many applications.

- The Transaction Management Facility (TMF) coordinates the processing of disk accesses and updates, ensuring the requirements for atomicity, consistency, isolation, and durability. An application can, in a very simple manner, request multiple database updates in many physical locations, possibly thousands of miles apart, and be assured that either all or none of them will be performed; during the "transaction" other applications will always have a consistent view of the database, never being able to see some of the updates and not others. TMF protects the database from total media failure (including the loss of a mirrored disk pair) through the technique of online dumping and roll-forward of the transaction log. TMF also supports the Remote Duplicate Database Facility, which can quickly recover from the loss of an entire computing facility.

- The Transaction Processing Monitor (PATHWAY) provides a flexible method for managing customer applications in a distributed system. PATHWAY automatically distributes application processes (called servers) to the available processors and, in the event of a processor failure, redistributes the applications to the remaining processors. Any work that was lost or compromised by the failure is automatically restarted after being rolled back to it's initial state. Customer programming is straight-forward and is not required to perform any special operations to achieve full fault tolerance.

- The Network Control Process and Line Handler Processes (EXPAND) manage communications between a system and a network of other Tandem systems. To a user on one node of a network, the rest of the network appears as a seamless extension of the local system. The requirement for local autonomy may impose access barriers and communication delays may impose performance penalties; otherwise, it is as easy to manage distributed applications and databases as it is to manage local ones.

Fundamental System Structure

A Tandem system is composed of from 2 to 16 independent processors connected by a dual, high-speed, inter-processor bus (IPB). GUARDIAN, Tandem's proprietary operating system, has two primary responsibilities:

- to maintain a consistent view of the system while allowing each processor to exercise independent autonomy, and

- to provide general services for it clients, the processes, and particularly to provide an efficient and reliable means of communication between them.

The first responsibility of the operating system requires that each processor establish and maintain communication with the other processors of the system. Continuous availability of the IPB is a fundamental assumption, since the processors must coordinate many operations and notify each other of changes in system state. If any two processors are not able to communicate with each other for any period, it is likely that they will have inconsistent views of the system state; one or the other must be restarted. Thus, a dual (fault-tolerant) IPB is an important requirement.

Except for the lowest-level functions of process dispatching, message transmission, and interrupt processing, all work in the system is managed by one or more processes. System processes manage process creation and termination, virtual memory, security, performance measurement, event management, diagnostics, and debugging. I/O processes manage access to peripheral devices and are responsible for dealing with failing components. Application and utility processes direct the operation of the system towards some useful purpose.

Messages are the primary method for process-to-process interaction, eliminating the need for applications to deal with the multiple-computer organization of the system. To applications, the system has the appearance of a conventional uniprocessor programmed in conventional programming languages such as Cobol85, Pascal, C, and Fortran. Processes interact with one another using a client-server protocol typical of the remote procedure call (RPC) protocols common in workstation-server LANs today. This client-server design is well-accepted today, but fifteen years ago it was considered novel.

Fault tolerance is provided by duplication of components in both the hardware and the software. Access to I/O devices is provided by process pairs consisting of a primary process and a backup process. The primary process must checkpoint state information to the backup process so that the backup can take over if a failure occurs. Requests to these devices are routed using the logical process name so that the request is always routed to the current primary process. The result is a set of primitives and protocols that allow recovery and continued processing in spite of bus, processor, I/O controller, or I/O device failures. Furthermore, these primitives provide access to all system resources from every process in the system.

Initialization and Processor Loading

System initialization starts with one processor being *cold loaded* from a disk attached to that processor; any processor can be used for this operation, as long as it is connected to a disk with a copy of the operating system image file. The image file contains the code of the kernel and the system processes that are automatically started when each processor is loaded.

Once any processor is loaded, it is then used to load the other processors via the IPB; all processors other than the first can be loaded in parallel. Should a processor fail or be removed for maintenance, it can be reloaded by any other processor. There is no essential difference between an initial load of a processor and a later reload of a processor after it has been repaired. A processor reload operation does not interfere with the operation of application processes.

Each processor receives an identical copy of the kernel and other system-level software, but a different processor configuration, depending upon the peripheral devices attached to the processor; each processor will start the appropriate I/O processes to manage the attached devices.

Once a processor's software and configuration is loaded, it passes through a phase in which it is synchronized with the other processors. Basically, this synchronization involves transmitting a consistent copy of the system state to the processor. Once this is accomplished, the processor becomes a fully independent entity. There is no master processor.

Processor Synchronization

Although Tandem computer systems are designed to tolerate any single fault, either hardware or software, multiple failures do sometimes occur, either through multiple hardware errors (unlikely), software errors (more likely), operation errors (even more likely), or during maintenance periods. Even when multiple faults occur, only a portion of the system (some disk volumes and/or processors) becomes unusable until it can be repaired and reintegrated into the system.

Two different mechanisms are provided for dealing with multiple faults: processor synchronization and transactions. The following sections describe the issues of processor synchronization. In addition to a simple algorithm to detect processor failure, Guardian also has three more complex algorithms to ensure that all processors of a system have closely agreeing views of processor configuration, replicated data, and time. Each of these algorithms requires a minimum of communication and is sufficiently robust without resorting to the cost and complexity of solving the Byzantine Generals problem.

I'm Alive Protocol

Once two or more processors are loaded, they must continually check on each other's health. Because the processors are designed to be fail-fast, they respond to any internal error by shutting down completely and refusing to communicate with the outside world until receiving an explicit signal that they are ready to be reloaded. Thus, it is up to the remaining processors to detect a failure through the absence of any response from the failed processor.

Using the *I'm Alive* protocol, each processor transmits a short message to each other processor, including itself, at least once every second, over each of the two interprocessor buses (the message to itself verifies that the sending and receiving bus circuitry is functioning).

Every two seconds, each processor checks to see that it has received at least one message from each processor in the interval. Any missing messages imply that a processor is not healthy and has probably failed. Normally, any processor would immediately declare such a processor "down" and proceed to cancel all outstanding communication with that processor.

Regroup Protocol

Experience showed, however, that there were rare instances in which one processor might merely be a little late in sending out its I'm Alive messages. This situation usually occurred when recovery from power failure or other high-priority error recovery momentarily usurped a processor.

Because the I'm Alive intervals are not synchronized between processors, a late I'm Alive might result in a processor being declared down by some processors and not others. Such a case, termed a "split-brain" situation, could lead to a lack of data base integrity.

Thus, a *Regroup* algorithm was implemented to handle these cases with as little disruption as possible. In essence, the slow processor is given a second chance. Whenever any processor detects a missing I'm Alive message, all processors (including the suspect processor, if able) exchange messages to decide which processors are healthy. After two broadcast rounds, the decision is made and all processors act on it.

Global Update Protocol

Certain information, notably the Destination Control Table (described later), is replicated in each processor and must be identical in each processor. Updates to replicated information, however, originate in multiple processes and multiple processors. Consistency of an update depends upon *atomicity* [Gray, 1978], which demands that (1) any update is completed within a maximum time, (2) either all replicated copies are updated or no copy is updated, and (3) all updates occur serially.

In Tandem systems, atomic update is guaranteed by the Global Update Protocol [Carr, 1985]. All such updates are performed by the Guardian kernel, so that high-priority processes cannot delay the completion of an update within a maximum time. All updates must first be sent to a *Locker* processor, which ensures that updates occur serially and also ensures that an update is propagated to all processors even if the originating processor fails, including simultaneous failure of the updating and Locker processors.

Time Synchronization

An OLTP system is designed to record and control real-world events at the time they actually occur; an important part of the information processed is the current time. Furthermore, it is important that the sequence in which events occur can be reconstructed from timestamps associated with each event.

Although it is clearly difficult to have coordinated clocks in a widely-distributed system, initial attempts to synchronize time on a local Tandem system showed that this is not a simple problem either. Although it is no great problem to keep clocks within seconds of each other, synchronization of a multiprocessor system requires that no message that can be sent from Processor A, with A's clock time T, should arrive at Processor B before B's clock time has reached T. Otherwise, time would appear to move backwards.

A novel algorithm [Nellen, 1985, 1986] passes time-adjustment packets from processor to processor; each processor not only adjusts its own clock to the average time of all processors, but it also calculates its clock error relative to the average and makes adjustments on a continual basis. This algorithm ensures that the "average time" does not fluctuate wildly when a processor fails and is replaced by a processor with a different speed clock.

The provision for an external clock can extend the local synchronization algorithm to geographically distributed Tandem systems; electronic clocks that monitor a broadcast time standard can keep Tandem systems synchronized to less than the time it takes them to communicate with each other.

GUARDIAN Kernel Services

Now that the basic problem of maintaining consistency between the distributed processors has been addressed, we can turn to the performance of useful work. As in all modern systems, GUARDIAN supports the concurrent execution of multiple independent processes. What distinguishes GUARDIAN is its heavy dependence on messages to coordinate the operations of processes. Messages are essential for the operation of both system-level software and customer applications.

The design of the system was strongly influenced by Dijkstra's "THE" system [Dijkstra, 1968] and Brinch Hansen's implementation of a message-based operating system nucleus [Brinch Hansen, 1970]. Both hardware and software have been optimized to facilitate the sending of messages between processors and between processes.

The heavy dependence upon messages, in preference to other communication and synchronization mechanisms, has been very important in the design of a system that is both distributed and smoothly expandable. Customer applications can be easily grown by simply adding more processors, disks, and other peripherals, without software changes or even reloading the system.

Because of the message-based structure, the applications are unaware of the physical configuration of the system. An application accesses a directly-connected peripheral and a remotely-connected peripheral in exactly the same way: messages are exchanged with the peripheral's manager (an I/O process); the I/O process is also unaware if the requestor is in the same or different processor.

Efficient messages have also been a key element in implementing fault tolerance. System-level software uses messages to *checkpoint* information critical to data integrity and continued operation in the event of a failure. Applications are generally unaware that a failure has occurred because messages can be automatically re-routed from a failing component to a functioning one.

Processes

A process is an independently-executable entity that consists primarily of shareable program code, private memory, and the process state vector. The process state vector includes the program counter, registers, privileges, priority, and a microsecond timer that is incremented only when the process is executing.

Once a process begins execution in some processor, it remains in that processor until it is terminated. Each process in the system has a unique identifier or name by which other processes may communicate with it on a network-wide basis. Messages to a process are addressed to its identifier, and so the identifier provides a network-wide and fault-tolerant method for process addressing.

GUARDIAN maintains a Destination Control Table which is identical in all processors of a system. This table relates the name of a process with its location (i.e., its processor and process number) so that messages to a process can be routed in an efficient manner.

Processes scheduling uses a pure priority mechanism with preemption. Scheduling is round-robin within a priority class. Considerable care is taken to avoid the priority inversion problem in which a low-priority client makes a request of a high-priority server, thereby promoting its work to high priority. This problem is unique to message-based systems and must be solved in order to provide a global priority scheduling mechanism.

Memory Management

Each process has a virtual address space containing its program and a process data stack. A program consists of *user code* and an optional *user library*. The user library is an object code file which can be shared by processes executing different user code program files. All processes executing the same program share memory for object code. The process data stack is private to the process and cannot be shared.

Processes may allocate additional *extended memory segments* which have a maximum size of 128 megabytes each. A process may access its data stack and one extended segment concurrently. If multiple extended segments are allocated, the process must explicitly request that a particular segment be placed "in use" when it is required.

Data in extended segments may be shared with other processes in two different ways:

- A read-only segment is a method of accessing the contents of a file as if it were main memory, using virtual memory to load the information on demand. Such segments can be shared among all processes in all processors, although multiple copies of the data may exist in different processors.
- A read-write segment can be shared only by processes in a single processor, since it would be impractical to update multiple copies in different processors.

The sharing of read-write data among customer application processes is discouraged so that fault containment is maintained, and so that the system load can be distributed by the placing processes in idle processors. GUARDIAN does not provide inter-process concurrency control (other than via messages) necessary for coordination of updates to shared memory.

The GUARDIAN memory manager supports virtual memory using demand paging and a CLOCK replacement strategy [Carr, 1981]. It intentionally does not support load (thrashing) control, as the performance requirements of online transaction processing do not permit the paging or swapping of processes commonly found in interactive time-sharing systems.

Interprocess Messages

The basic message system protocol follows the requestor-server model. One process sends a message with some "request" to another process and waits for a response. The second process "services" the request and replies with a result. This is considered to be a single message and can also be viewed as a remote procedure call. Naturally, this basic mechanism also suffices for simple datagrams or for transferring bulk data (up to 60KB per message) in either direction.

Multi-threaded requestor processes may have many messages outstanding to collections of server processes. Multi-threaded server processes may accept multiple requests and reply to each in any desired order. Any process may operate concurrently both as a client and as a server.

For application processes, access to the message system is through the file system, which provides some protection from abuse of privileged mechanisms and simplifies the interface for sending messages. System processes, on the other hand, generally use the message-system primitives directly, as these provide better performance.

The simplest use of messages is for application processes to access peripheral devices; the programmer is generally unaware that messages are involved, as the program makes a simple READ or WRITE of data on a "file." If, for example, the file is a disk file, then the READ/WRITE request will send a message to the manager of the disk, known as the *disk process.*

It is simple for a process to masquerade as a device manager. An example of this is the printing spooler process: applications send print output messages to a spooler (which pretends to be a physical printer process) and stores them for printing later; the spooler then sends the output, again via messages, to a true physical printer process. The application programmer need not be concerned with whether the message system routes the messages to the spooler process or directly to an I/O process controlling a physical printer; only the process name must be changed to choose the destination process.

A more interesting use of messages is in the structuring of applications as a network of communicating processes, which is described in the section on PATHWAY.

In the Tandem system, messages are a fundamental mechanism for providing fault tolerance:

- The communication protocol for the interprocessor buses tolerates any single bus error during the execution of any message-system primitive. A communication failure will occur only if the sender or receiver processes or either of their processors fail. Any bus errors that occur during a message-system operation are automatically corrected without involving the communicating processes.

- As described in the next section, the process-pair mechanism provides fault-tolerant access to peripheral devices despite processor, channel, or even most software failures. A request message that is is being processed by the failing component is automatically resent to the backup component; the application is not even aware that this has happened and need not make any provision for it.

Memory moves rather than the interprocessor buses are used for communication between processes in the same processor, but there is no apparent difference to the communicating processes. In addition to messages between processes, GUARDIAN also implements simpler *control* messages which are for communication between processor kernels. These are used as a basis of the full message-system protocol as well as an inexpensive mechanism to maintain synchronization between processors.

GUARDIAN and the inter-processor bus are highly optimized for the processing of inter-process messages, especially for short messages of 2 KB or less. A message between processes in different processors is only marginally more expensive than an intra-processor message. In fact, an inter-processor message usually has a shorter *elapsed* time than an intra-processor message, since both sender and receiver processes can execute in parallel.

A final advantage of the message system is its transparent support of both short- and long-haul networks. Except for the inevitable communication delays, the client and server can detect no apparent difference between accessing a local disk file (or process or printer) and a remote one. The message-system and file-system protocols are precisely the same in both the local and remote cases.

Tolerating Software Faults

Systems whose fault tolerance is based solely on hardware mechanisms can be designed to provide high reliability and continue to function in the presence of hardware component failures. Unfortunately, a high percentage of computer system failures are due to software.

Unlike the situation with hardware components, it is *possible* to develop perfect, defect-free, failure-proof software. It is only a matter of cost to the manufacturer and inconvenience to the customer who must wait much longer for some needed software to be delivered.

In the commercial world, customers demand a continuous flow of new software and improvements to old software. They demand that this be done quickly (more quickly than the competition) and at a reasonable price. New software systems are inevitably more functional and more complex than the systems they replace.

The use of structured programming and higher-level languages has not eliminated software errors because they have enabled the building of larger and more complicated programs. Methods to improve software quality, such as code inspections and structured testing techniques, are effective, but they only reduce the number or errors; they do not eliminate them. Therefore, in practice, even with significant care taken in software development processes, software faults are inevitable. In fact, as previously stated, software failures are typically more common than hardware failures.

To a remarkable degree (for a commercial computer), the Tandem system tolerates a great majority of critical software faults.

Software fault tolerance also leads, indirectly, to better software quality and data integrity. At Tandem, systems programmers are encouraged to make numerous consistency checks and, if a problem is detected, to *halt the processor*. (Tandem's system software probably has one of the highest densities of processor-halt instructions in the industry.) The system programmer knows that, for almost all consistency problems, the backup processes (described in the next section) will continue to provide service to the customer. This has two direct effects:

- When contamination of system data structures is detected, the processor is immediately shut down, reducing the chance that the database can become contaminated.

- All significant errors in system software become very visible and, since the entire processor state is frozen and dumped when an error is detected, it is easier to uncover the cause of the error. Thus, errors which affect system stability and data-integrity are found and corrected in a very timely manner; the result is higher quality software and fewer failures that need to be tolerated.

Process pairs provide fault-tolerant execution of programs. They tolerate any single hardware fault and most transient software faults. (Transient software faults are those caused by some untimely combination of events, such as a resource shortage occurring at the same time that I/O error must be handled.) Most faults in production software are transient [Gray, 1985], since the simpler programming errors are weeded out during testing. Process pairs allow fail-fast programs to continue execution in the backup process when the software bug is transient.

Process Pairs

The key to both hardware and software fault tolerance is the *process pair*. A process pair consists of a *primary* process, which does all the work, and a *backup* process which is passive, but is prepared to take over when the primary process fails. This arrangement is analogous to having a standby processor or computer system, except that, if properly arranged, the cost of the backup process is far less than the cost of the primary process. Generally, the memory requirements and processor-time consumption is a small fraction (usually about 10%) of the primary process.

A typical process pair is started in the same way as an ordinary process: as a single process executing in a single processor. The process then notifies the operating system that it would like to create a "clone" of itself, using the same program file, in another processor. The second process is also started in a very ordinary fashion, but with two small differences:

- The second process receives an initial message that informs it that it is a backup process. The process then goes into a passive mode, accepting only messages from the primary or from GUARDIAN (to notify the backup that either the primary process or its processor has failed).

- Both the primary and backup process share a single name. For each name, the Destination Control Table registers both the primary and backup processes. All communication to the process pair is routed to the primary process.

While the primary process executes, the backup is largely passive. At critical points, the primary process sends *checkpoint* messages to the backup. Checkpoint messages have two different forms; a process pair will normally use either one or the other, but not both:

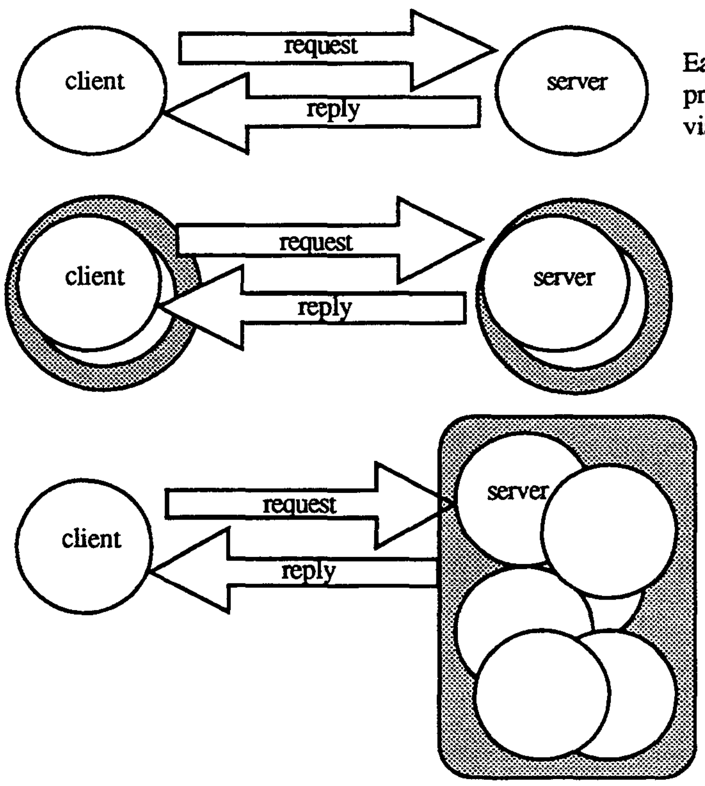
- For application software, a simple form of checkpointing is provided. Checkpoints copy the entire process state, at carefully-chosen takeover points, from the primary to the backup. Updating the process state of the backup is performed solely by the system software at the command of the primary process; the backup process is completely passive.

  At a checkpoint, usually immediately before or after some important I/O operation, the primary and backup are functionally identical. If the primary fails, the backup begins executing at the point at which the last checkpoint occurred.

- For most system software, such as I/O processes, checkpoint messages contain functional updates to the process state. The backup processor must interpret these messages and apply appropriate updates to its own local data structures; this is sometimes referred to as an *active* backup. Although the program code is identical, the contents of memory can be entirely different. A typical checkpoint would indicate that a file has been opened or closed, but, in the case of the disk process, most updates to important files will involve a checkpoint.

The active backup approach is more complicated, but it has several advantages. Less data is transmitted on each checkpoint, and information that the backup can obtain from other sources (such as a disk file) need not be checkpointed. For software fault tolerance, the active backup is better, because the backup must manage it's own state and errors in the primary process are less likely to contaminate the backup [Borr, 1984].

Since the process pair shares a single name, it appears to all other processes as a single process. When a message is sent to it, the message system automatically consults the Destination Control Table to determine which process is primary and routes the message to that location. When the primary process fails for some reason, the backup becomes the primary, the Destination Control Table is changed, and all messages are directed to the new primary process. The new primary has already received checkpoints describing the current openers of the process, so those openers need do nothing to re-establish communication.



Each process has a program and private storage. They communicate via messages.

A process pair is one logical process. The backup process is idle until the primary fails.

A server class is a group of processes spread over many processors for load balancing. Clients send messages to the class. Each messages is delivered to a particular member of the class.

Figure 19. Process Concepts in the Guardian Operating System.

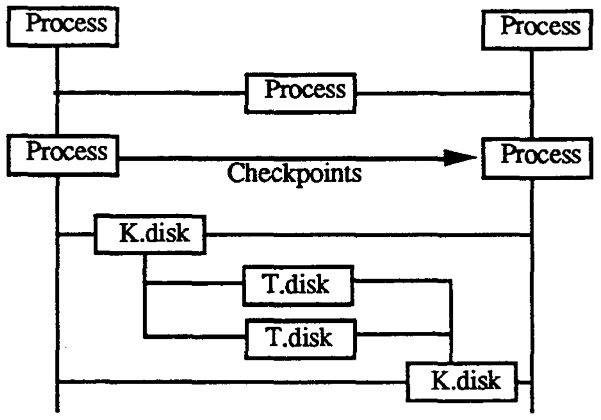
There are potential race conditions in which a server process pair has performed some request and has not replied to the client when a failure occurs. When the outstanding request is resent to the new primary process, it might perform the same operation a second time, possibly causing an inconsistency. This problem is eliminated by the associating sequence numbers with each request and the new primary simply makes duplicate responses to already-processed requests.

Similar problems could occur if a client process pair performs a checkpoint and then makes requests to a server. If a failure occurs, the backup client takes over and makes duplicate requests to the server. These are handled using the same sequence numbering scheme. During normal operation, the sequence numbers are used for duplicate elimination and detection of lost messages in case of transmission error [Bartlett, 1978].

I/O Processes

Most processes can execute on any processor. The exceptions are I/O processes, because they manage devices and each device is connected to two processors via dual-ported device controllers. When a device is configured, an I/O process pair is automatically defined for the device; one member of the pair resides in each processor connected to the device. The configuration parameters may state which processor is the normal primary access path to the device, but this choice can be altered by the operator for testing and performance tuning. In the case of a processor, channel, or controller port failure, the process that still has access to the device will take over control and all application requests will be routed to it.

When a request for an operation such as a file open or close occurs, the primary process sends this information to the backup process through the message system. These "checkpoints" ensure that the backup process has all information needed to take over control of the device. A process pair for a mirrored disk volume appears in Figure 20.



These processes represent application programs communicating with one another and with device servers.

These two processes are a pair controlling a mirrored disk. The primary process services all requests for disc operations, and checkpoints the current status to the backup process which will take ownership of the disk if the primary process fails. Similar process pairs are defined for all other peripheral devices.

Figure 20. PMS Diagram of System Structure.

Process pairs provide a uniform way to access I/O devices and all other system-wide resources. This access is independent of the functions performed within the processes, their locations, or their implementations. Within the process pair, the message system is used to checkpoint state changes so that the backup process can take over in the event of a failure.

Disk Process

Although the overall design of an OLTP application may be very complex, it is essentially composed of many simple servers that require a minor amount of computation and perform a large number of database accesses and updates. To achieve high performance in this environment requires a great deal of sophistication in the design of the database management software. In this section, we are able to give only a broad outline of the myriad responsibilities and functions of the disk process, but it clearly has the most demanding task of any component of the system.

Each disk process pair must manage a pair of mirrored disks, each connected to two controllers, which are in turn connected to two processor-channels. Thus, there are eight possible paths to the data, any component of which may fail; even in the rare case of a multiple failure, the disk process must attempt to use all available resources to provide continuous access to the data.

Mirrored Disks

Mirrored disks are, as the name implies, identical images of one another. It would appear that this is a situation where fault tolerance requires a redundant (and expensive) resource that contributes nothing to system capacity. (Even if it were true, the value of data-integrity would, in most cases, justify the expense of the redundant disks.)

Fortunately, however, even though the redundant disk does not contribute to storage capacity it usually contributes significantly to processing capacity. When data must be read from disk, the disk process can use either of the two disks, usually the disk that offers the shorter seek time. Multiple read requests can be processed concurrently and if one disk is busy, then the other disk can be be used. Because duplexed disks offer shorter seeks, they support higher read rates than two ordinary disks [Bitton 88].

Any write operation must be made to each of the mirrored disks and requires that both disks seek to the same cylinder (thereby reducing the chance of having a short-seek on the next read). Consequently, disk writes are considerably more expensive than reads, but, when performed in parallel they are not much slower than a write to a single disk. The proper use of disk cache, particularly when protected by transaction management, can eliminate a large majority of disk writes without sacrificing data integrity.

Customers who consider all or part of their database to be a non-critical resource may have unmirrored disks on a disk-by-disk basis. Modern disks are very reliable and, even when drives fail, it is exceedingly rare that the data is lost. Many activities, such as software development, would not be seriously impacted by the rare unavailability of a disk.

Disk Cache

Each disk process can be allotted many megabytes of main memory for disk cache. In 1990, the upper limit was 56 megabytes per disk volume, but this will steadily increase along with the size of main memory. The disk process uses the cache to hold recently-accessed disk pages and, under normal circumstances, can satisfy most read requests without performing any physical I/O operation. The disk process can service many requests concurrently, so it is not unusual for it to satisfy a half-dozen requests from cache while it is performing a single real disk I/O.

The worst case for cache management is an application that performs random accesses to a very large data base; the probability that it re-uses a recently-accessed page in cache would be quite low. Experience, however, shows that the typical application usually has, at most, one such file in addition to numerous smaller files that are also accessed in each transaction, so that the ratio of "cache-hits" to physical I/Os remains quite good. Even in the case of the large file, traversing the B-tree index structure to find a data record might normally require three physical I/Os that can be saved by cacheing the index blocks.

Although cache has an obvious benefit in reducing read I/O operations, the situation is not so clear with write operations. There are many situations in which disk file updates are made to blocks that were recently updated; if only part of a block is being updated this clearly saves a read to get the unchanged parts of the block. More significantly, if updated blocks could be kept in main memory and written to disk only when convenient, many disk writes could be eliminated. On a conventional system, we couldn't do this because a processor failure would lose a large number of disk updates and corrupt the entire database.

On Tandem systems, we might consider checkpointing disk updates to the backup disk process, which is much cheaper than performing an actual I/O; if a processor failed, the backup could make sure that the updates were written to disk. This approach, however, would not be safe enough, since it is possible for a long power failure, or simple bad luck, to cause a multiple failure and lose an unacceptable number of database updates. Luckily, there is a solution to this problem, but we must postpone its discussion until we have covered the basics of transactions.

Partitioning

If an OLTP application has a high transaction rate and each transaction accesses a particular file, the load on the disk process, even with perfect cacheing, may be too large to sustain on a single processor; as with the application, it is necessary to be able to easily distribute the database access load across the processors. (In general, dynamic load re-distribution has not proved necessary, but it must be easy to re-distribute in a static manner.)

The concept of file partitioning is not new, but the disk process and file system cooperate to provide a simple and flexible solution. Any file can be simply partitioned by issuing a few commands specifying the key ranges of each partition. To the applications, the partitioned file appears as a single file; the file system redirects any read or write request to the proper partition, depending on the key of the record. If a partition becomes too full, or too busy, it can be split by subdividing its key range and moving the appropriate records to the new partitions.

The partitions of a file can be distributed throughout a network and, thus, a distributed database can be created and maintained in a manner that is completely invisible to the applications, while maintaining excellent performance when accessing local data. For example, one could easily construct a 50-partition file, one for each of the United States, and physically locate each partition on a separate Tandem system in each state. On each system, local state data could be processed very efficiently, but every application process could have access to the full database as if it were a single file, subject only to the inevitable communication delays.

Locking and Transaction Protection

An OLTP application may have hundreds of independent servers performing concurrent accesses to the same set of files; the disk process must support file and record locks, both for shared and exclusive access. Locks can be obtained through explicit application request or through the operation of transaction management, which automatically provides atomicity and isolation of all database accesses in a transaction. More about transactions appears later.

File System

The file system is a set of system routines that execute in the application process and manage communication with I/O processes and other application processes. For access to I/O devices, the file system hides the process-and-message structure, and the application program appears to be issuing direct requests to a local I/O supervisor. That is, the file system provides an procedure call interface to the remote I/O processes, masking the fact that they are remote procedure calls. The application is unaware of the distributed nature of the system.

The file system automatically manages requests in order to implement partitioned files. It implements buffering, so that many sequential operations can be satisfied with one request to the I/O process.

The file system implements the first level of system security, as it does not allow an application to send a message to an I/O (or application) process unless it has first identified itself with an OPEN message that contains an authenticated user name. If the server process denies access to the object, the file system will not permit further messages (except for OPEN messages) to be sent.

The file system manages timeout and retransmission of requests; it sends the message to the backup server process if the primary fails. In addition, if the client is a process pair, the file system manages checkpointing of the process state to the backup process.

NonStop SQL

The file system and the disk server cooperate to process SQL (Structured Query Language) database operations in an integrated and efficient manner. The file system manages the SQL processing in the client and performs all the high-level operations such as sort, join, and aggregation. The disk process understands simple constructs such as table, field, and expression and will do low-level SQL operations on the data. Thus, operations can be performed at whatever level promotes the best efficiency. For example, the SQL statement

```
UPDATE ACCOUNT SET BALANCE = BALANCE + :DEPOSIT-AMOUNT
      WHERE ACCOUNT_NUMBER = :ACCOUNT-ID;
```

can be processed with a single message to the disk process. There is no need for the application to fetch the information, update it, and send it back to the disk process. In another example, the statement:

```
SELECT FIELDA, FIELDE FROM TABLEX WHERE FIELDA + FIELDB > FIELDC;
```

allows filtering to be performed at the disk process, minimizing the transfer of data to the application.

NonStop SQL is designed specifically to handle OLTP applications while achieving good performance. Because it is integrated with other system software, it can be used for OLTP applications in a geographically distributed network of systems. SQL tables can be partitioned across systems in a network. Also, applications can run at one network node while accessing and updating data at another node. Furthermore, the applications themselves can be distributed. With NonStop SQL, fault tolerance derives from the basic mechanisms of process pairs, mirrored disk, geographically distributed systems, along with node autonomy and transaction support. All transactions, local and network, are protected for consistency and error recovery.

From a fault-tolerance perspective there are two novel things about NonStop SQL. First is the design goal of node autonomy. The system is designed so that if the client and server can communicate, then the client can access the data. This simple requirement implies that all the metadata describing the file must be replicated with the file. If the file is partitioned among many nodes of the network, the catalog information describing the file must be replicated at all those nodes.

The second requirement is that no administrative operations on the data are allowed to take the database off-line. For example, taking an archive dump of the database must be done while the data is being accessed, reorganizing the data must be done on-line, and so on. Many administrative tasks are not yet on-line, but a major focus of current efforts is to make all administrative operations completely on-line.

SQL allows data administrators to attach integrity constraints to data; these may take the form of *entity constraints* that limit the values a record may have or *referential integrity constraints* that constrain the relationships among records in different tables. Placing the constraints on the data is more reliable than depending on the application program to make such checks. Updates to the data that would violate these entity constraints are rejected.

Transactions

The work involved in a computation can be packaged as an atomic unit by using the Transaction Monitoring Facility (TMF). This facility allows an application to issue a BEGIN-TRANSACTION request, make numerous database accesses and updates in multiple files, on multiple disks, and on multiple network nodes, and then issue an END-TRANSACTION request. The system guarantees that the work of the transaction will be ACID, defined as follows [Haerder & Reuter, 1983]:

- Atomic: either all of the database updates will be performed, or none of them will be. For example, if a transaction moves money from one bank account balance to another, the end result will never result in more or less money on the books.

- Consistent: each successful transaction preserves the consistency of the database.

- Isolated: events within a transaction must be hidden from other transactions running concurrently, otherwise a failing transaction could not be reset to its beginning.

- Durable: once committed, the results of the transaction must survive any failure.

Should the application or GUARDIAN (i.e., the disk process or TMF) detect a problem that compromises the transaction, either one may issue ABORT-TRANSACTION, which will cause any database updates to be undone. The system can manage thousands of concurrent transactions, keeping them all isolated from one another. The application program need not concern itself with the locking protocol, as the required locking operations are well-defined and performed automatically.

The work of a transaction can be distributed to multiple processes. As we shall see, it is normal to structure an application as client processes directing numerous server processes, each designed to perform some simple part of the transaction. Once the client has issued BEGIN-TRANSACTION all requests to servers are marked as belonging to the transaction; all database updates by the servers become part of the single transaction, until the client issues the END-TRANSACTION or ABORT-TRANSACTION request.

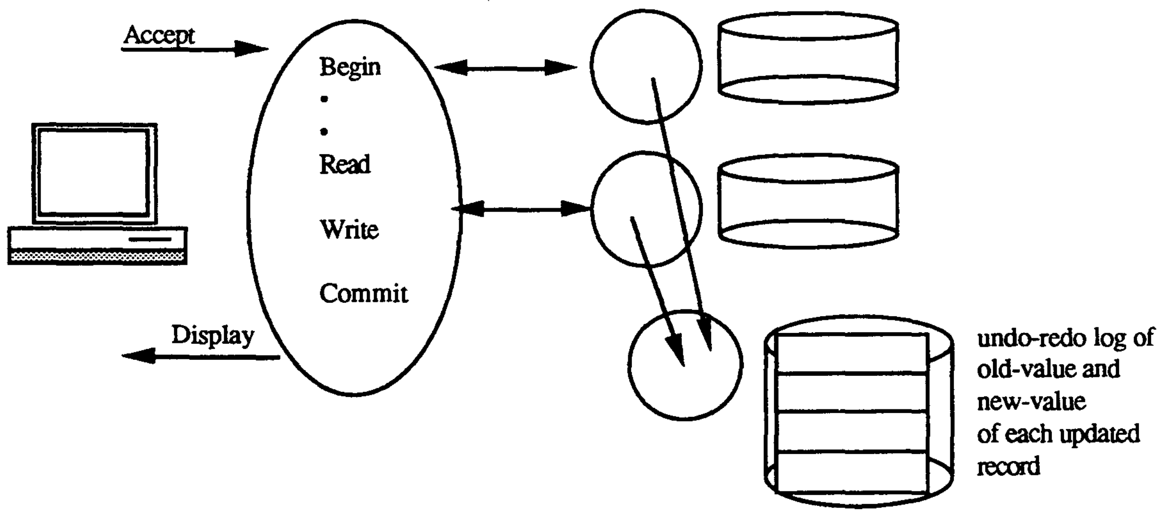
When a client issues BEGIN-TRANSACTION, the system creates a unique *transaction-identifier* and uses it to tag all messages and all database requests. If a tagged message is accepted by a server, all of its messages and database requests receive the same tag. Database locks acquired by the client or its servers are also tagged by the transaction-identifier.

During a transaction, disk processes hold the updates in their caches. If there is insufficient cache, the disk process may update the database before the transaction completes, but first it must generate *undo* and *redo* records that allow the transaction to be either undone in case it aborts or redone in case it commits and a later failure occurs. When the client issues END-TRANSACTION, each disk process with updates for the transaction must generate undo and redo log records before it updates the disk.

The undo and redo records are written to specially-designated transaction log files called the *audit trail.* Normally, these files are on disks that are separate from the database, and the undo and redo records must be written to the audit trail disk before the main database is updated; thus, even a total system failure will not lose transactions that are committed or allow the database to become inconsistent. Even if a mirrored disk pair were to be destroyed, the database and all transactions can be recovered from an archival copy of the disk and the transaction log.

Any process participating in the transaction can unilaterally abort it. The system implements two-phase locking and uses a non-blocking, grouped, presumed-abort, two-phase commit protocol.

It might appear that support of transactions adds considerable overhead to the basic operations of accessing and updating the data base. Surprisingly, TMF *improves performance* while enhancing fault tolerance and data-integrity. As described previously, updates to a database should be written to disk immediately in order to prevent their loss in case of a failure. This increases disk traffic and lengthens transaction response time.



Accept
Begin
Read
Write
Commit
Display
undo-redo log of old-value and new-value of each updated record

Figure 21. The structure of the Transaction Management Facility (TMF).

When database operations are protected by TMF, a description of all updates is written to the audit trail; it becomes unnecessary for the disk process to write the updates to the database, except when it is convenient to do so. As soon as the audit trail records are reliably stored on disks, the application can be notified that the updates have been permanently recorded and will survive any failure. Writing the updates to the audit trail is considerably more efficient than writing the updates to the database, because many updates from a single transaction (and, in a busy system, from multiple concurrent transactions) are blocked together and written in a single I/O operation. Further, the audit trail is written sequentially and writing is performed with a minimum of seeks, while database updates are random-access and imply numerous seeks. Finally, the disk process performs less checkpointing to its backup, because uncommitted updates do not need to be protected against processor failures.

The result of these effects is that the logging and recovery of TMF is a net savings over the less functional store-thru-disk cache. TMF converts random main memory database access to sequential accesses, dramatically reducing the density of I/O transfers. Benchmarks have demonstrated that I/O density can be reduced by a factor of two or three when TMF is used [Enright, 1985].

While the Tandem system provides high availability through single-fault tolerance, TMF provides multiple-fault tolerance for the critical element of transaction processing systems--the database. Although multiple faults are exceedingly rare, the consequent cost of database loss is very high.

Before the introduction of TMF, application programmers relied on process pairs and forward error recovery to provide fault tolerance. Whenever an error was detected, the backup process resumed the computation from the last checkpoint. Process pairs were difficult to design and implement and reduced the productivity of application programmers. Applications implemented with TMF are much simpler to program and achieve the equivalent level of fault tolerance. Because transactions imply an automatic locking protocol, it is much easier to maintain a consistent database.

Process pairs are still an important concept and are used for system and I/O processes, as well as specialized utility processes such as the print spooler. They are the fundamental basis on which TMF and other system software are built, so that the customer can write fault-tolerant applications without regard for fault tolerance.

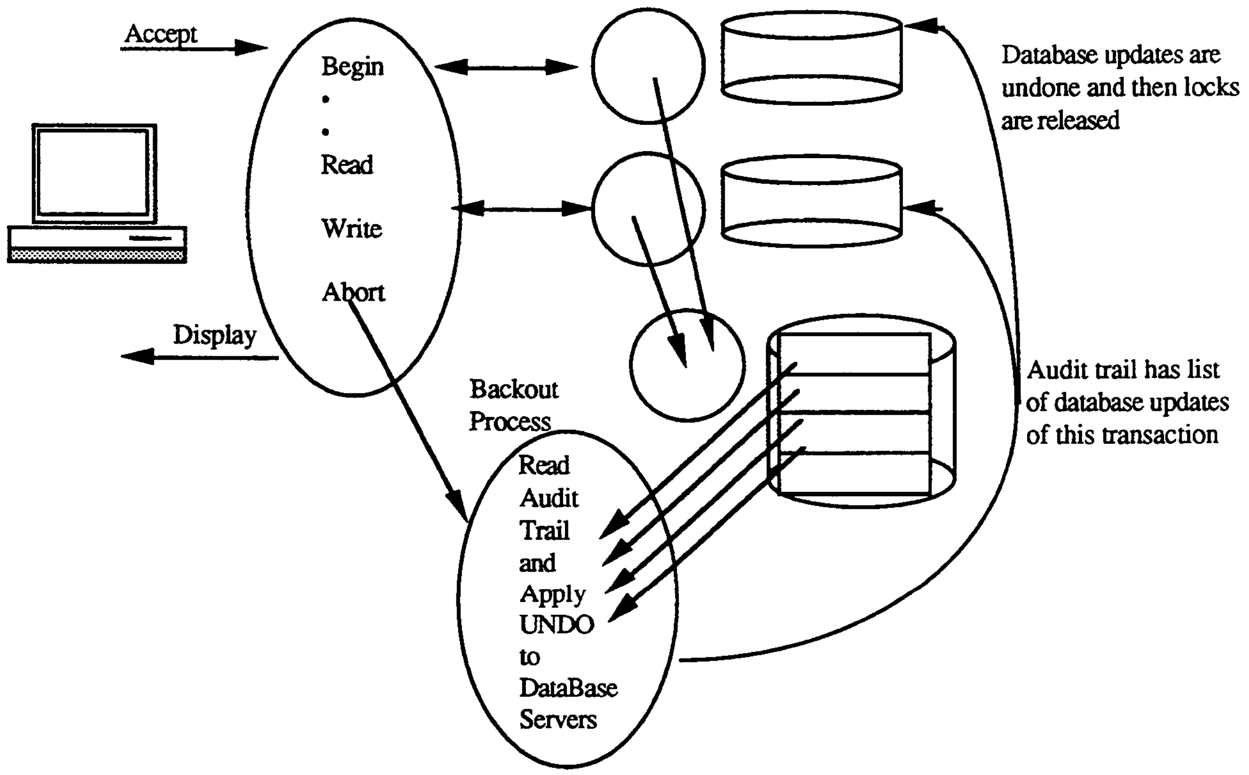

Accept
Begin
•
•
Read
Write
Abort
Display
Database updates are undone and then locks are released
Backout Process
Read Audit Trail and Apply UNDO to DataBase Servers
Audit trail has list of database updates of this transaction

Figure 22. TMF transaction backout structure

Transaction Processing Monitor (PATHWAY)

Applications are structured as client (requestor) and server processes. The clients are responsible for presentation services and for managing the user interface. Such user interfaces range from a forms-oriented interface to an electronic mail or home banking system, to real-time interfaces where the "user" is an automated warehouse, gas pump, or telephone switch. The servers are programmed to perform specific functions, usually a set of related database accesses and updates. In the electronic mail example, one server looks up names while another is responsible for routing messages to other network nodes and to gateways. In the gas pump example, one server does user authentication while another does billing. Typically, applications are structured as hundreds of services. Breaking an application into requestors and servers promotes software modularity, allows on-line change and growth of applications, and exploits the multicomputer architecture. With the advent of intelligent terminals (workstations, automated teller machines, and other computers acting as clients and servers), the client is migrating to the workstation, and the client-server architecture is becoming the standard structure for all transaction processing applications.

The application designer specifies the programs and parameters for each client and server. The servers can be programmed in any language, but the clients have traditionally been programmed in a Cobol dialect called Screen Cobol. This interpretive language automatically manages transactions and process pairs. In case the client process fails for any reason, the backup process takes over, reprocesses the input message if necessary, and redelivers the output message. This gives exactly-once semantics to transaction processing. Screen Cobol relieves the application programmer from needing to understand how to write process pairs. It is the most common way that customers get message integrity.

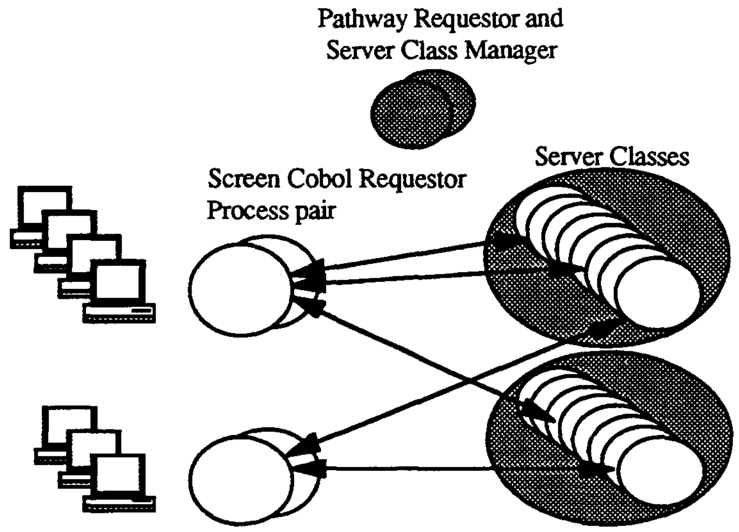


Figure 23. The structure of the Pathway Transaction Processing Monitor

The Transaction Processing Monitor, PATHWAY (Figure 23), is responsible for managing the application's requestors and servers. It creates requestors and servers at system startup, maintains a configuration database that can be altered on-line by operator commands, and load-balances the system by creating and deleting server instances as the load changes and as processors come and go from the system.

Process Server Classes

To obtain software modularity, computations are broken into several processes. For example, a transaction arriving from a terminal passes through a line-handler process (for instance, X.25), a protocol (for example, SNA), a presentation services process to do screen handling, an application process that has the database logic, and several disk processes that manage disks, disk buffer pools, locks, and transaction audit trails. This method breaks the application into many small modules, which serve as units of service and of failure. If one unit fails, its computation switches to its backup process.

If a process performs a particular service--for example, acting as a name server or managing a particular database--then traffic against this server is likely to grow as the system grows. Gradually, the load on such a process will increase until it becomes a bottleneck. Such bottlenecks can be an impediment to linear growth in performance as processors are added. The concept of process *server class* is introduced to circumvent this bottleneck problem.

A server class is a collection of processes that all perform the same function, typically spread over several processors. Such a collection can be managed by PATHWAY. Requests are sent to the class rather than to individual members of the class. As the load increases, members are added to the class. If a member fails or if one of the processors fail, the server class migrates into the remaining processors. As the load decreases, the server class shrinks. Hence, process server classes are a mechanism for fault tolerance and for load balancing in a distributed system [Tandem, 1985]. The application designer specifies the program, parameters, minimum size, maximum size, and distribution of the server class. PATHWAY reads this configuration database at system startup and manages the server class, growing it as the load increases, and shrinking it as the load decreases.

Networking

The process- and message-based structure of GUARDIAN naturally generalizes to a network operating system. A proprietary network, called EXPAND [Tandem, 1987], enlarges the original 16-processor design to a 4080-processor network. EXPAND uses a packet-switched, hop-by-hop routing scheme to move messages among nodes; in essence, it connects all of the inter-processor buses of remote systems. EXPAND is now widely used as a backbone for corporate networks or as an intelligent network, acting as a gateway among other networks. The fault tolerance and modularity of the architecture make it a natural choice for these applications.

Increasingly, the system software supports standards such as SNA, OSI, MAP, SWIFT, TCP/IP, Named Pipes, and so forth. These protocols run on top of the message system and appear to extend it.

The fault tolerance provided by the system extends to the network. Networking software allows a session to continue even if a communication link breaks. For example, SNAX, Tandem's implementation of IBM's System Network Architecture (SNA), provides continuous operation by transparently checkpointing at key points the internal information needed to sustain operation. This enables SNAX to maintain all active sessions in the event that a single processor or line fails. Similar provisions exist in the Open System Interconnection (OSI) software.

Fault tolerance also underlies the Distributed Systems Management (DSM) products for globally managing NonStop systems and EXPAND networks. For example, with the Event Management Subsystem (EMS), an event recorder process executing as a NonStop process pair provides for graceful recovery from single-process failures. That is, if the primary event recorder process fails or is stopped, the backup process continues recording in the appropriate event logs.

Disaster Protection

In conventional disaster recovery operations, when a disaster happens, people at a standby site retrieve the database tapes from archival storage, transfer them to the standby site, establish a compatible operating system environment, restore the data from tape to disk, switch communication lines to the backup site, and restart application programs. This is a complex, labor-intensive, and error-prone process that commonly takes from 12 to 48 hours.

The issues surrounding disaster recovery change dramatically when one moves from a traditional batch environment to the world of on-line transaction processing. OLTP customers require recovery within a matter of seconds or minutes with little or no lost transactions or data. Symmetric network and application designs based on the Remote Duplicate Database Facility (RDF) software give this kind of disaster protection.

Operations personnel can use RDF to get applications back on-line within five minutes of a disaster. RDF stores the database at two systems typically at two distinct geographic sites. For each data item there is a "primary" copy and a "backup" copy of record. All updates are made to the primary copy and the TMF log records for the primary are sent to the site holding the backup copy of the record where they are applied to the backup copy of the data. The customer can select one of three options for the sending of these log records:

2-Safe: No lost transactions at takeover. In this case the transaction is recorded at both sites prior to committing the transaction to the client. It implies slightly longer response times because of the added network delay.

1-Safe: The last few transactions may be lost at takeover because they were not recorded at the backup site. 1-safe has better response time and may be appropriate if each transaction is of little value or is easily reconstructed at a later time.

Electronic vaulting: The log of the system is simply transmitted to a remote site and stored there. It is not applied to the remote database until there is an actual disaster. This is similar to the standby site scheme, but avoids the movement of data to the standby site when the disaster happens.

RDF makes it possible to maintain a current, on-line copy of the database at a secondary node, as illustrated in Figure 24. The secondary database can be located nearby or across the nation. The use of RDF is completely transparent to the application programmer. Any TMF application can be converted to an RDF application without change. Only the database configuration and operations procedures change [Lyon, 1990].

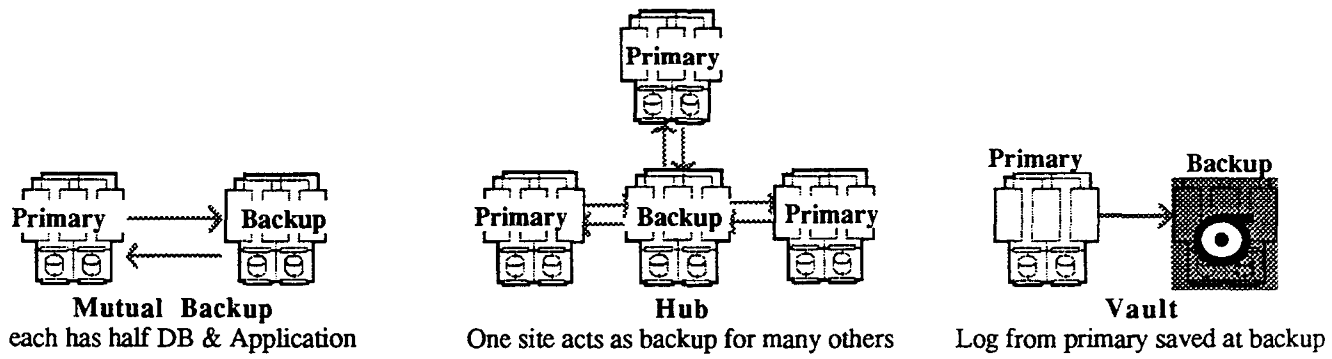

Primary
Backup
Mutual Backup
each has half DB & Application
Primary
Primary
Backup
Primary
Hub
One site acts as backup for many others
Primary
Backup
Vault
Log from primary saved at backup

Figure 24. Remote Duplicate Database Facility (RDF)

To support its backup capabilities, RDF monitors and extracts information from TMF audit files and sends this information over the EXPAND network to a corresponding RDF process on the second node. This extraction usually takes place within seconds of the audit being created by TMF. The RDF process on the second node receives the audit transactions and stores them in disk buffers. Another RDF process on the second node then applies these transactions to the database, thus maintaining the duplicate database. The second copy of the database is usually current within seconds of the primary database.

Updating activities on the second database can be temporarily suspended without compromising the integrity of the on-line backup process. The audit transactions accumulate in disk buffers at the secondary site until updating is resumed. Then all accumulated updates are automatically applied in the correct sequence.

RDF has some additional fault-tolerance benefits. If one of the sites needs to be taken off-line for a software upgrade, hardware upgrade, facilities move, or just a "fire drill" to test the disaster recovery facility, the load can be switched to the other node without interrupting service.

Operating System Summary

The innovative aspects of the GUARDIAN operating system do not entail new concepts; instead, they are a synthesis and integration of pre-existing ideas. Of particular importance are the low-level abstractions: processes and messages. These abstractions allow all processor boundaries to be hidden from both application programs and most system software. These initial abstractions are the key to the system's ability to tolerate failure. They also provide the configuration independence that is necessary in order for system and application software to run on systems of many sizes. Process pairs are a natural extension of the process concept to fault-tolerant execution. Transactions have been integrated with the operating system and appear as a natural part of the execution environment. Spooling the transaction log to a remote site is the basis of the disaster recovery facility. Extending the message-based system to a long-haul network makes it geographically distributed.

The operating system provides the application programmer with general approaches to process structuring, inter-process communication, and failure tolerance. Much has been documented about structuring programs using multiple communicating processes, but few operating systems support such structures.

Finally, the design goals of the system have been demonstrated in practice. Systems with from 2 to 200 processors have been installed and are running on-line applications. Many of these systems are members of multi-node networks. They are recovering from failures and failures are being repaired on-line, with little or no impact on the system users.

Application Software

Application software provides a high-level interface for developing on-line transaction processing applications to run on the low level process-message-network system described in the preceding sections. The basic principle is that the simpler the system, the less likely the user is to make mistakes.

For data communications, high-level interfaces are provided to "paint" screens for presentation services. Furthermore, a high-level interface is provided to SNA to simplify the applications programming task.

For database management, the relational data model is adopted and the NonStop SQL software provides a variety of easily-implemented functions. A relational query language integrated with a report writer allows quick development of ad hoc reports.

System programs are written in the Transaction Application Language (TAL), which is a high-level block-structured language similar to ALGOL or Pascal; TAL provides access to machine instructions. Most commercial applications are written in COBOL85 or developed through application generators.

In addition, the system supports FORTRAN, Pascal, C, BASIC, MUMPS, and other specialized languages. A binder allows programmers to combine modules from different languages into a single application, and a symbolic debugger allows them to debug in the source programming language. The goal, however, is to reduce such low-level programming by moving to application-specific fourth generation languages.

A menu-oriented application generation system, PATHMAKER, guides developers through the process of developing and maintaining applications. Where possible, it generates the application code for clients and servers based on the contents of an integrated system dictionary.

The application generator builds most clients from the menu-oriented interface, although the user can tailor the client by adding COBOL statements. The template for the servers is also automatically generated, but customers must add the semantics of the application, generally using COBOL. Servers access the relational database either through COBOL record-at-a-time verbs or through set-oriented relational operators.

Using automatically-generated clients and the transaction mechanism, customers can build fault-tolerant distributed applications with no special programming required. PATHMAKER provides maximum protection against failures through its reliance on TMF.

Ongoing investigations support the hypothesis that, as programmers migrate from language to language, human error rates remain nearly constant. That is, a programmer will produce about the same number of errors for every 1,000 lines of code regardless of the language being used. Thus, the higher the level of language a programmer uses, the smaller will be the number of errors in the object code. On this basis, Tandem urges its customers to use high-level tools like the PATHMAKER application generator and other products that incorporate fourth-generation language concepts for OLTP program development. These tools include SQL Forms from Oracle, Applications By Forms from Ingres, and Focus from Information Builders. These tools greatly simplify the programmer's work.

New application software is under development that takes advantage of the growing number of personal computers and other workstations in the business world. This influx of desktop computers has dramatically influenced the way that business people do their computing. As these machines become faster and provide more memory and disk storage, businesses are expected to want system software for them that parallels software running on mainframes and minicomputers. This software, in turn, will generate a growing number of applications.

As stressed earlier in this section, customers demand good price/performance from fault-tolerant systems. Each Cyclone processor can process about twenty-five standard transactions per second. Benchmarks have demonstrated that 32 processors have 32 times the transaction throughput of one processor: that is, throughput grows linearly with the number of processors, and the price per transaction declines slightly [Tandem Performance Group, 1988]. Tandem believes a 50 processor Cyclone system is capable of 1,000 transactions per second. The price per transaction for a small system compares favorably with other full-function systems. This price per transaction demonstrates that single-fault tolerance need not be an expensive proposition.

OPERATIONS

Errors originating with computer operators are a major source of faults. Operators are often asked to make difficult decisions based on insufficient data or training. The system attempts to minimize operator actions and, where required, directs the operator to perform tasks and then checks the operator's actions for correctness [Gray, 90].

Nevertheless, the operator is in charge and dictates what orders the computer must follow. This relationship poses a dilemma to the system designer: how to limit the actions of the operator. First, all routine operations are handled by the system. For example, the system automatically reconfigures itself in the case of a single fault. The operator is left with only exceptional situations. Single-fault tolerance reduces the urgency of dealing with failures of single components. The operator can be more leisurely in dealing with most single failures.

Increasingly, operators are given a simple and uniform high-level model of the system's behavior that reflects physical "real-world" entities such as disks, tapes, lines, terminals, applications, and so on, rather than control blocks and other abstractions. The interface is organized in terms of actions and exception reports. The operator is prompted through diagnostic steps to localize and repair a failed component.

Maintenance problems, discussed earlier in this chapter, are very similar to operations. Ideally, there would be no maintenance. Single-fault tolerance allows hardware repair to be done on a scheduled basis rather than "as soon as possible," since the system continues to operate even if a module fails. This approach reduces the cost and stress of conventional maintenance.

The areas of single-fault-tolerant operations and single-fault-tolerant maintenance are major topics of research at Tandem.

SUMMARY AND CONCLUSIONS

Single-fault tolerance is a good engineering tradeoff for commercial systems. For example, single disks are rated at a MTBF of five years. Duplexed disks, which record data on two mirrored disks connected through dual controllers to dual processors, raises the MTBF to 5000 years (theoretical) and 1500 years (measured). Triplexed disks would have theoretical MTBF of over one million years, but because operator and software errors dominate, the measured MTBF would probably be similar to that of duplexed disks.

Single-fault tolerance through the use of fail-fast modules and reconfiguration must be applied to both software and hardware. Processes and messages are the key to structuring software into modules with good fault isolation. A side-benefit of this design is that it can utilize multiple processors and lends itself to a distributed system design. Modular growth of software and hardware is a side-effect of fault tolerance. If the system can tolerate repair and reintegration of modules, then it can tolerate the addition of brand new modules.

In addition, systems must tolerate operations and environmental faults. Tolerating operations faults is the greatest challenge.

REFERENCES

[Bartlett, 1978] Bartlett, J.F. "A 'NonStop' Operating System," Proceedings, Hawaii Int. Conf. of System Sciences. Honolulu, HI 1978, pp. 103-119

[Bartlett, 1981] Bartlett, J.F., "A NonStop Kernel,", Proc. 8th SIGOPS, ACM Press, Dec. 1981, pp 22-29.

[Bitton, 1988] Bitton, D. and Gray, J., "Disk Shadowing", Proc. VLDB Conference, Sept. 1988, pp 331-338.

[Bitton, 1989] Bitton, D., "Arm Scheduling in Shadowed Disks", CompCon 1989, IEEE Press, March 1989, pp 132-136.

[Borr, 1981] Borr, A., "Transaction Monitoring in ENCOMPASS," Proc. 7Th VLDB, September 1981. Also Tandem Computers TR 81.2.

[Borr, 1984] Borr, A., "Robustness to Crash in a Distributed Database: A Non Shared-Memory Multi-processor Approach," Proc. 9th VLDB, Sept. 1984. Also Tandem Computers TR 84.2.

[Brinch Hansen, 1970] Brinch Hansen, P. "The Nucleus of a Multi-programming System," Comm ACM, 13 (April 1970): pp 238-241

[Burman, 1985] Burman, M. "Aspects of a High Volume Production Online Banking System", Proc. Int. Workshop on High Performance Transaction Systems, Asilomar, CA Sept. 1985.

[Carr, 1981] Carr, R.W., "Virtual Memory Management", Stanford University, 1981.

[Carr, 1985] Carr, R.W., "The Tandem Global Update Protocol", Tandem Systems Review, V 1.2, June 1985.

[Dijkstra, 1968] Dijkstra, E.W. "The Structure of the 'THE' Multiprogramming System, Comm ACM, 11 (1968): pp 341-346

[Eddy, 1987] Eddy, John. "Remote Support Strategy." Tandem Systems Review, 3, no. 1 (March 1987): pp. 12-16

[Englert, 1989] Englert, S., Gray, J., Shah, P., "A Benchmark of NonStop SQL Release 2 Demonstrating Near-Linear Speedup and Scaleup on Large Databases", Tandem TR 89.4, 1989.

[Enright, 1985] Enright, Jim "DP2 Performance." Tandem Systems Review, 1, no. 2 (June 1985): pp. 33-43

[Garcia, 1988] Garcia, David J. "Built-In-Self-Test for the Tandem NonStop CLX Processor" Digest of Papers, CompCon Spring 1988, San Francisco, CA 1988.

[Gray, 1979] Gray, J., "Notes on Database Operating Systems", in *Operating Systems, an Advanced Course*, Springer Verlag, 1979.

[Gray, 1985] Gray, J., "Why Do Computers Stop and What Can Be Done About It?", Tandem TR85.7, June 1985, Tandem Computers, Cupertino, CA.

[Sohi, Franklin, and Saluja, 1989] Sohi, G.S., Franklin, M., and Saluja, K.K., "A Study of Time-Redundant Fault Tolerance Techniques for High-Performance Pipelined Computers", in proc. Ninteenth International Symposium on Fault Tolerant Computing, Chicago, IL, pp.436-443, June 1989.

[Tandem, 1985] Tandem Computers. Introduction to PATHWAY, Cupertino, CA, 1985.

[Tandem, 1987] Tandem Computers. EXPAND Reference Manual, Cupertino, CA, 1987.

[Tandem Database Group, 1988] "NonStop SQL, A Distributed High Performance, High Availability Implementation of SQL", *Proceedings of 2nd High Performance Transaction Processing Workshop*, D. Gawlick Ed., Springer Verlag, 1989.

[Tandem Performance Group, 1988] Tandem Performance Group, "A Benchmark of NonStop SQL on the DebitCredit Transaction", SIGMOD 88, ACM, June 1988.

[Tom, 1988] Tom, G., "Tandem's Subsystem Programmatic Interface", Tandem Systems Review, V4.3, 1988.

[Troisi, 1985] Troisi, Jim "Introducing TMDS, Tandem's New On-line Diagnostic System." Tandem Systems Review, V 1.2 . June 1985.

[Uren, 1986] Uren, S., "Message System Performance Tests", Tandem Systems Review, V3.4, Dec. 1986.

[White, 1987] White, L., "Enhancements to TMDS", Tandem Systems Review, V3.2, June 1987.

[Gray, 1985] Gray, J., Anderton, M., "Distributed Database Systems--Four Case Studies", Tandem Computers TR 85.5.

[Gray, 1990] Gray, J., "A Census of Tandem System Availability: 1985-1990", Tandem Computers TR 90.1. 1990.

[Haerder & Reuter, 1983] Haerder, T and Reuter, A, "Principles of Transaction-Oriented Database Recovery", ACM Computing Surveys, Vol. 15.4, December 1983.

[Homan, 1988] Homan, P., Malizia, B., Reismer, E. "Overview of DSM", Tandem Systems Review, V. 4.3, Oct 1988.

[Horst and Chou, 1985] Horst, R. and T. Chou, "The Hardware Architecture and Linear Expansion of Tandem NonStop Systems," Proceedings of the 12th International Symposium on Computer Architecture, June 1985.

[Horst and Metz, 1984] Horst, R. and S. Metz, "A New System Manages Hundreds of Transactions/Second," Electronics, April 19,1984, pp. 147-151.

[Horst, 1989] Horst, R. "Reliable Design of High-speed Cache and Control Store Memories," in proc. Nineteenth International Symposium on Fault Tolerant Computing, Chicago, IL, pp. 259-266, June 1989.

[Horst and Gray, 1989] Horst, R., Gray, J., "Learning from Field Experience with Fault Tolerant Systems", Proceedings of International Workshop on Hardware Fault Tolerance in Multiprocessors, University of Illinois, Urbana IL, June 19-20, 1989, pp 77-79.

[Horst, Harris, and Jardine, 1990] Horst, R., Harris, R., and Jardine, R., "Multiple Instruction Issue in the NonStop Cyclone System", to appear in 17th International Symposium on Computer Architecture, Seattle, WA, May 28-31, 1990.

[Jouppi and Wall, 1989] Jouppi, N.P. and Wall, D.W., "Available Instruction-Level Parallelism for Superscalar and Superpipelined Machines", in proc. Third International Conference on Architectural Support for Programming Languages and Operating Systems, Boston, MA, 1989.

[Katzman, 1977] Katzman, J. A. "System Architecture for NonStop Computing," CompCon, 1977, p. 77-80.

[Lenoski, 1988] Lenoski, Daniel E. "A Highly Integrated, Fault Tolerant Minicomputer: The NonStop CLX," Digest of Papers, CompCon Spring 1988, San Francisco, CA 1988.

[Lyon, 1990] Lyon, J., "Tandem's Remote Data Facility", To appear in Proceedings of CompCon 90, Feb. 1990, San Francisco, CA., IEEE Press.

[Mourad and Andrews, 1985] Mourad, S. and Andrews, D., "The Reliability of the IBM/XA Operating System", Digest of 15th Annual Int. Sym. on Fault-Tolerant Computing. IEEE Computer Society Press, June 1985.

[Nellen, 1985] Nellen, Eric, "New GUARDIAN 90 Timekeeping Facilities", Tandem Systems Review, V1.2, June 1985.

[Nellen, 1986] Nellen, Eric, "Managing System Time Under GUARDIAN 90", Tandem Systems Review, V2.1, February 1986.

Distributed by
TANDEM
Corporate Information Center
10400 N. Tantau Ave., LOC 248-07
Cupertino, CA 95014-0708