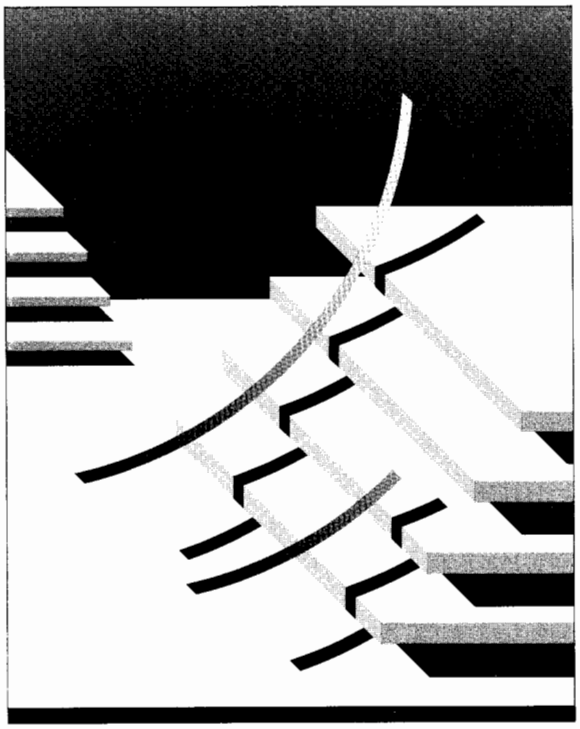
TANDEM JOURNAL

VOLUME 2, NUMBER 1

WINTER 1984



The High-Performance NonStop TXP Processor

The ENCORE Stress Test Generator for On-line Transaction Processing Applications

The 6100 Communications Subsystem: A New Architecture

The Relational Data Base Management Solution

TANDEM JOURNAL

Volume 2, Number 1, Winter 1984

Editor
Carolyn Turnbull White

Associate Editors
Kent Madsen
Anita Van Auken

Production Editor
Anita Van Auken

Technical Advisor
Geary Arceneaux

Design
Craig Frazier Design

Cover Art
Craig Frazier

The Tandem Journal is published quarterly by Tandem Computers Incorporated.

Purpose: The purpose of the Tandem Journal is to bring to Tandem users the perspectives of Tandem software developers, engineers, and support analysts on Tandem software and hardware.

Subscriptions: The Tandem Journal is offered with the Tandem Application Monograph Series in one subscription. The annual subscription rate is $100.00. Tandem bills the subscriber. *U.S. Orders* — Send directly to Tandem Computers Incorporated, Sales Administration, 19333 Vallco Parkway, Cupertino, CA 95014. *Orders outside the U.S.* — Give to your local Tandem sales office or distributor. All subscribers should address subscription problems or questions to their local Tandem sales office or distributor.

Change of address: Send all changes of address to Sales Administration (address listed above).

Comments: We welcome comments and suggestions about content and format. Please send them to Carolyn Turnbull White, Editor, Tandem Journal, Tandem Computers Incorporated, 1309 So. Mary Ave., Sunnyvale, CA 94087.

The High-Performance NonStop TXP Processor

In 1976, Tandem introduced the NonStop™ system, designed to meet the need for a truly fault-tolerant computing resource. It was targeted at those sensitive application areas in which computer down-time cannot be tolerated.

As the years went by, Tandem users discovered more and more applications that could benefit from the unique capabilities of the NonStop system. The applications grew more complex, and the number of required peripheral devices increased to the point that the capacities of the original NonStop system were strained. In response to this problem, Tandem introduced the NonStop II™ system. Its 32-bit virtual addressing capability and other enhancements removed the major limitations inherent in the original design.

Many Tandem users are now contemplating fault-tolerant computing on a very large scale. Accordingly, their concerns, and Tandem's, have shifted to cost/performance issues. With larger application loads, there is a pressing need to increase the number of transactions per second that Tandem NonStop systems can process. The NonStop TXP™ processor was designed to meet this need.

The NonStop TXP system is an all-new, high-performance processor designed to increase the throughput of Tandem systems. Benchmark tests have shown that it can handle more than twice as many transactions per second as a NonStop II processor. NonStop TXP processors currently function as the new NonStop TXP system, and in June of 1984, they will also be available to mix with NonStop II processors in the same system, running a single version of the GUARDIAN™ operating system. Strategic placement of the new processors in such an environment can significantly increase the throughput of the system, while preserving the customer's investment in NonStop II equipment.

Performance Enhancements

The performance offered by the NonStop TXP processor is the result of many elements:

- The design incorporates recent circuitry advances, including: Programmable Array Logic (PAL), Fairchild Advanced Shottky Technology (FAST), and high-speed static RAMs.
- The micro-instruction cycle time is 83.3 ns, as opposed to 100 ns on the NonStop II processor.
- The micro-instruction itself is very wide (124 bits versus 36 bits), which allows for greater parallelism.
- The processor operates on 32 bits during each cycle, using dual 16-bit data paths.
- Macro-instruction pipelining has been increased from two levels on the NonStop II processor to three levels on the NonStop TXP processor.
- A hardware-based cache memory scheme reduces effective memory access time from 400 ns to 116 ns.
- The 32-bit absolute extended addresses are generated by hardware in a single cycle.

The NonStop TXP processor has 64K bytes of memory cache. Thus, a considerable amount of information can be kept close to the central processing unit (CPU), where it is accessible in a fraction of the time that it would take to access main memory. Implemented with 16K fast static RAMs, the cache uses direct mapping and has a data block transfer size of 16 bytes. A "store through" feature ensures that main memory always holds a valid copy of all data. Cache memory has a one-cycle (83.3 ns) READ access time and a two-cycle WRITE time. Cache measurements made during application benchmarks show a better than 97% hit ratio. All cache misses are handled in the firmware.

In the NonStop II processor, 16 memory maps, containing up to 64 page-table entries each, were kept in hardware registers. In the NonStop TXP processor, these hardware registers have been replaced with a 2048-entry page-table cache, which provides fast address resolution when a cache miss does occur and physical memory must be accessed.

Physical Characteristics

The NonStop TXP processor is made up of four circuit boards plus memory. (The fourth board fits into an unused slot in the NonStop II system cabinet.) The circuit boards include:

- *The Instruction Processor (IP).* The IP board contains the main data path and arithmetic logic unit (ALU); 64K bytes of memory cache; and address translation hardware, including a separate cache containing 2048 page-table entries.
- *The Sequencer and Control Store (SQ).* The SQ board contains the microcode sequencer circuitry and the control store RAM. Special compaction techniques make it possible for this board to accommodate 8K 124-bit lines of microcode, stored as 8K 40-bit vertical control store words and 4K 84-bit horizontal control store words.
- *The Channel Control (CC).* The CC board contains the I/O channel interface, along with the diagnostic data transceiver (DDT), which provides an interface to the Operations and Service Processor (OSP); the other arithmetic logic unit; a 10-ms interrupt timer; and 8K bytes of scratchpad registers for the firmware's use.
- *The Memory Control (MC).* The MC board contains the interprocessor bus interface, the interrupt logic, the system clock, and state machines and logic used to access main memory. The error-correction control (ECC) logic also resides on this board. It uses 6 bits of control information stored with each 16-bit data word to detect and correct single-bit errors and detect double-bit errors.
- *The Memory Module (MM).* The MM board for the NonStop TXP processor has been designed with cache operations in mind. In particular, it provides access to eight bytes of memory at a time, allowing the firmware to fill a 16-byte cache block by issuing two pipelined quad-word READs. There are no lights or switches on the board; memory addressing is now configured by the DDT processor. Up to four 2-megabyte memory modules can be placed in a NonStop TXP processor.

Software Compatibility

One of the major goals of the development team was to mask all the differences between the NonStop TXP and NonStop II processors (except their speed) from as much of the software as possible. The requirement that all non-privileged software run without change on both processors was met. There were a few minor changes to I/O processes; the remainder of the new code is confined to areas such as interrupt handlers, processor loading, low-level debugging, and offline diagnostics.

The system manager need only be aware that two processor types exist. The manager must provide the SYSGEN system configuration program with both the NonStop TXP instruction set microcode files and the corresponding NonStop II files. Given both sets of files, the type of each processor in the system does not have to be specified at the time of system generation. Instead, the GUARDIAN operating system determines each processor's type at load time and then loads the appropriate instruction set microcode. This flexibility permits processor swapping without system reconfiguration, which is particularly useful in a development or benchmark environment.

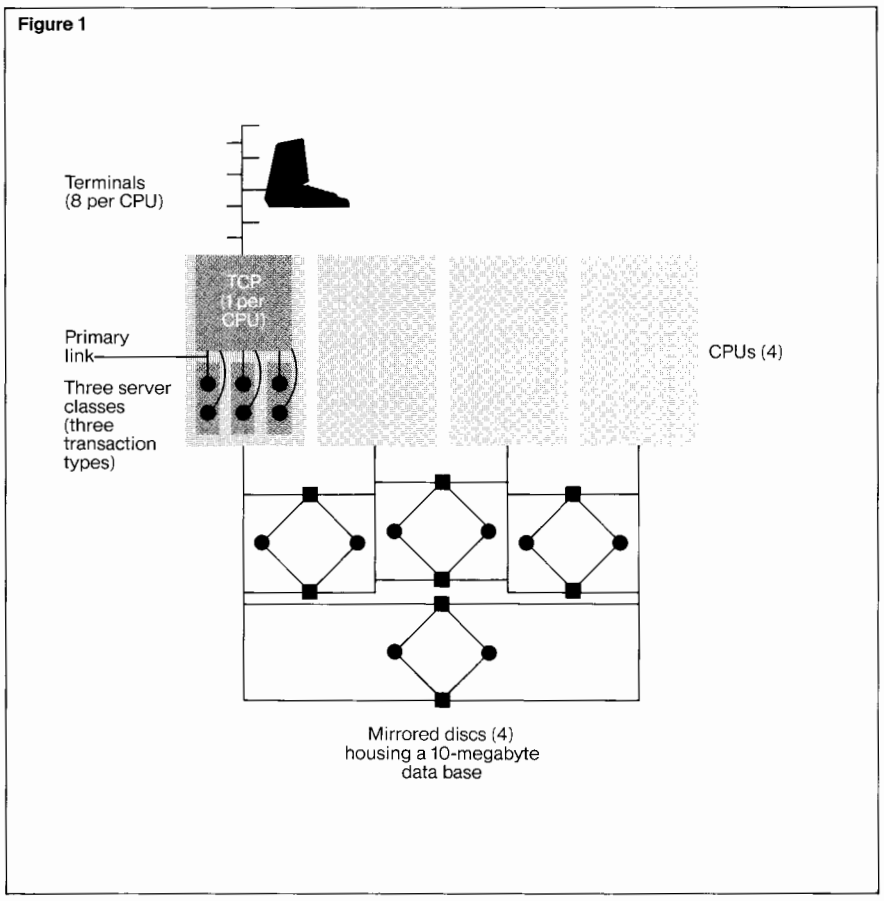


Figure 1.
Configuration of hardware and software used in benchmark tests.

A Performance Benchmark

Tandem conducted benchmark tests in which the performance of a NonStop TXP system was measured under carefully controlled conditions and compared with the performance of a NonStop II system under identical conditions. Both sets of tests were run on four-processor systems with four mirrored discs (see Figure 1). The PATHWAY™ application used in the tests effectively simulated a banking application with full teller functionality. As shown in Figure 1, there were:

- 4 PATHWAY Terminal Control Processes (TCPs) running as process pairs, 1 primary process in each CPU.
- 32 terminals (8 per TCP).
- 24 server processes (6 per CPU, belonging to 3 server classes).
- A 10-megabyte, key-sequenced data base file and an entry-sequenced log file.

The key-sequenced file, with a record size of 667 bytes and a key size of 14 bytes, was partitioned across three discs. The entry-sequenced file, a log file with a record size of 500 bytes, was on a fourth disc.

There were three types of transaction: an inquiry transaction, an update transaction, and another, referred to simply as the "big" transaction. Each one had different I/O requirements. The inquiry transaction involved one random data base READ. The update transaction involved one random data base UPDATE and one WRITE to the log file. The big transaction involved one random data base READ, two data base UPDATEs, and one WRITE to the log file. The workload consisted of 75% update transactions, 15% inquiry transactions, and 10% big transactions. The average transaction consisted of:

- One ACCEPT from the terminal.
- One SEND to a server.
- Two logical (Transaction Monitoring Facility, or TMF) updates.
- One message back to the terminal.

Each server class was made up of eight static servers distributed evenly across the system. For example, the server class handling update transactions consisted of $UPDATE0 through $UPDATE7, with $UPDATE0 and $UPDATE4 located in CPU 0, $UPDATE1 and $UPDATE5 located in CPU 1, and so on. To balance the workload, the PATHWAY system was configured in such a way that the primary links between the four TCPs and a particular server class were granted to one server per processor. (The primary linked server is the first server a particular TCP calls upon when it needs service of the kind offered by that server class. If the primary linked server is busy, the TPC calls upon the next linked server, and so on. Thus, the four primary linked servers within a given class are always busier than the other four, and placing one of them in each CPU distributes the workload evenly.)

TMF was configured to cross-audit the disc volumes with data base recovery on.

The cross-auditing for the four discs, $DATA1 through $DATA4, was as follows:

The audit file for this volume	Was located on this volume
$DATA1	$DATA2
$DATA2	$DATA1
$DATA3	$DATA4
$DATA4	$DATA3

The application system used 32 internal terminal simulators to drive the transaction throughput. The simulator "think time" was varied to increase or decrease the number of transactions per second flowing into the system so that performance at various loads could be measured.

One measure of performance in transaction processing systems is the cost of an average transaction in CPU cycles. Figure 2 shows the measured levels of CPU utilization for the NonStop II and NonStop TXP systems at various transaction rates. In both systems, the relationship between transaction rate and CPU utilization is linear. From these results, one can calculate the CPU cost per transaction for the two systems. By this measure, the performance of the NonStop TXP system is 2.2 times better than that of the NonStop II system.

Another measure of performance in transaction processing systems is response time. Figure 3 shows the measured response times for the NonStop II and NonStop TXP systems at various transaction rates, and Figure 4 shows the measured response times at various levels of CPU utilization. As shown in Figure 3, if the response time is held constant at two seconds, the measured performance in transactions per second for the NonStop TXP system is 2.8 times better than that of the NonStop II system.

This benchmark gives an indication of the performance improvements that result from moving a specific application from a NonStop II system to a NonStop TXP system of equal size. Other benchmarks, run by Tandem and by Tandem users, have shown varying transaction rate factors, ranging from 2.2 to approximately 3. The performance to be gained from a NonStop TXP system is clearly application-dependent; more comprehensive information about the behavior of NonStop TXP systems, and of mixed systems, will be made available as performance testing proceeds.

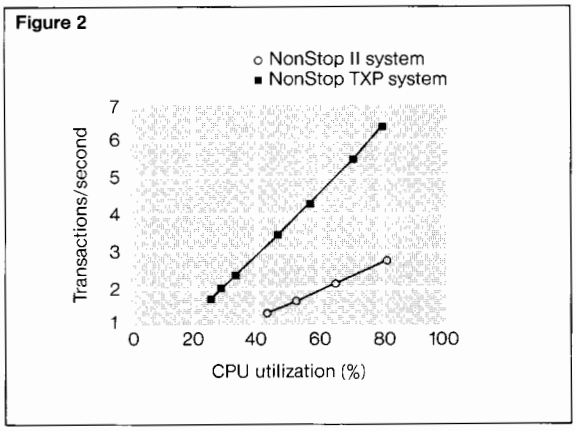


Figure 2.
Measured levels of CPU utilization at various transaction rates.

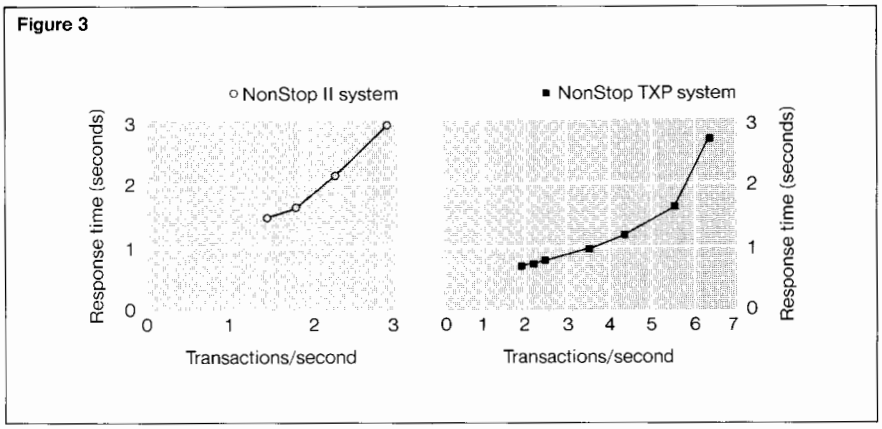


Figure 3.
Measured response times at various transaction rates. Left: NonStop II. Right: NonStop TXP.

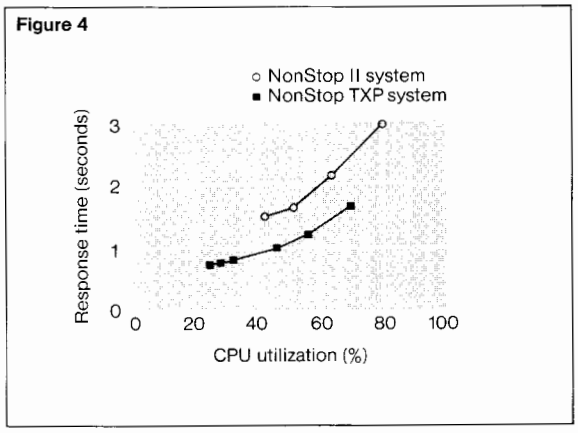


Figure 4.
Measured response times at various levels of CPU utilization.

Wendy Bartlett is Manager of the Operating Systems Group in Software Development. She developed a major portion of the operating system for the NonStop TXP system. She has been with Tandem for over five years.

Tom Houy, coordinator of the NonStop TXP System Support Team, is Manager of Customer Application Support's Performance Group. He has been with Tandem for three years. Before that, he worked for another major mainframe vendor where he maintained operating systems and analyzed system performance.

Don Meyer is currently working in the United Kingdom. Since joining Tandem in November 1981, he has also participated in presales activities and customer design reviews. Before coming to Tandem, Don worked for a large Chicago bank, where he managed a development team that installed a 15-node Tandem network. His work at the bank included design, Tandem hardware sizing, application implementation, and performance analysis.

The ENCORE Stress Test Generator For On-line Transaction Processing Applications

On-line transaction processing is one of the most rapidly expanding areas of computer automation. As in everyday business transactions, an on-line transaction takes place after one or more conversations, called interactions, between a buyer (the terminal operator) and a producer (the on-line application program).

Computer interactions are more structured than human ones and generally follow the form shown in Figure 1. First, the application prompts the operator with a data entry screen (the prompt). After a pause for reflection, the operator enters data and presses a function key. The application interprets the function key, reads the data from the terminal screen, and computes a response, which is used to complete the prompt for the next interaction.

Figure 1 also divides the time used to complete an interaction into two pieces: *think time* and *system reply time.* The think time is the amount of time used by the terminal operator to compose a query. The system reply time is the amount of time consumed by the system in returning a response.

Although operator think time is important, it is the system reply time that ultimately determines the success or failure of an on-line application. Because productivity in on-line transaction processing is intimately linked to system reply times, excessive processing delays are equated to increased costs to the user. Inadequate performance can even nullify the expected gain from automating a company process.

Because performance is so important to on-line transaction processing, specific performance objectives, such as a certain number of transactions per hour and a response time less than a certain number of seconds, are typically made a part of application design specifications. These performance objectives are as important to successful system implementation as the functional objectives.

Verifying that the performance of the application meets predetermined acceptability criteria while processing expected workloads is called stress testing. Stress testing is usually the last step in application development cycles. Should performance deficiencies be uncovered during this step, they can be corrected before the system goes live. This provides a measure of protection against costly performance errors. A performance-critical application that has not undergone a stress test is not fully tested.

Stress Testing Requirements

Stress testing consists of generating a workload and applying it to a system while measuring the system's performance. A workload is a measured collection of work to be processed by the system under test. For on-line transaction processing systems, generating a workload means to present transactions from the terminal environment to the system under test.

There are five major requirements for a stress testing tool:

- *Accurate workload presentation.* Presenting an accurate workload means closely duplicating the type and frequency of transactions presented to the application during production. Since the intent of a stress test is to verify that the application will perform adequately under live conditions, presenting a workload that does not model the live situation clearly results in an invalid test.
- *Controllability.* Controllability provides the tester with the ability to manipulate the workload to reflect different environments. Test variables might include the rate at which interactions are presented to the system, terminal start-up time, and number of terminals.
- *Repeatability.* Repeatability in a stress testing tool means the ability to accurately repeat a workload. This allows the tester to vary performance parameters in the application or reconfigure the hardware while holding the workload constant. In other words, parameters can be changed and an identical workload can be presented to the system to determine the effect of the changes.
- *Low cost.* The cost of running a stress test should not outweigh the cost of trial-and-error tuning on a live system. Testing costs are measured in terms of the personnel and hardware required to conduct the test. Expensive techniques often discourage testing in all but the most critical applications.
- *High performance.* To accommodate the demands of high-throughput applications, a suitable stress testing tool must be capable of high performance in terms of the number of interactions presented to the system per unit of time.

Figure 1

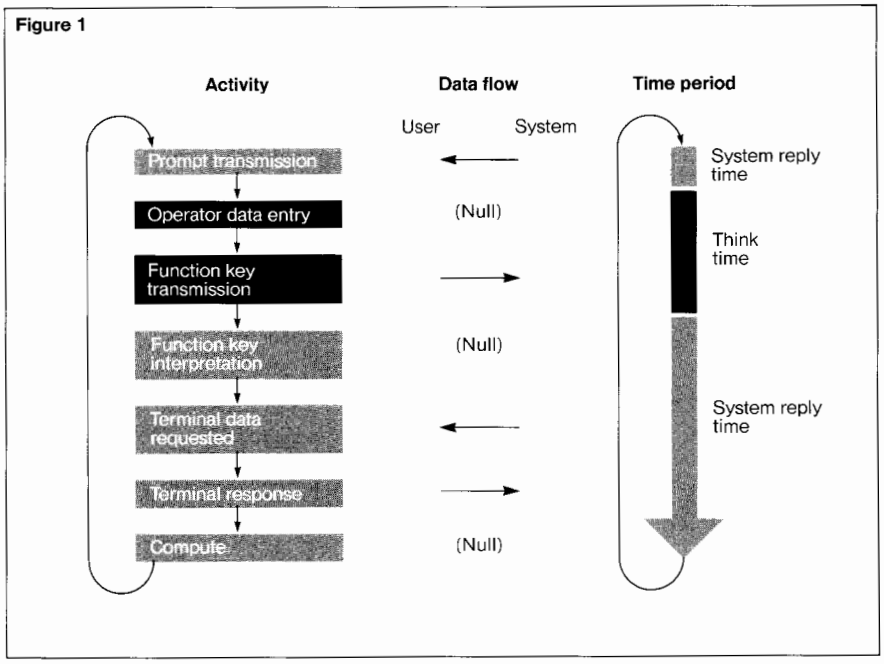
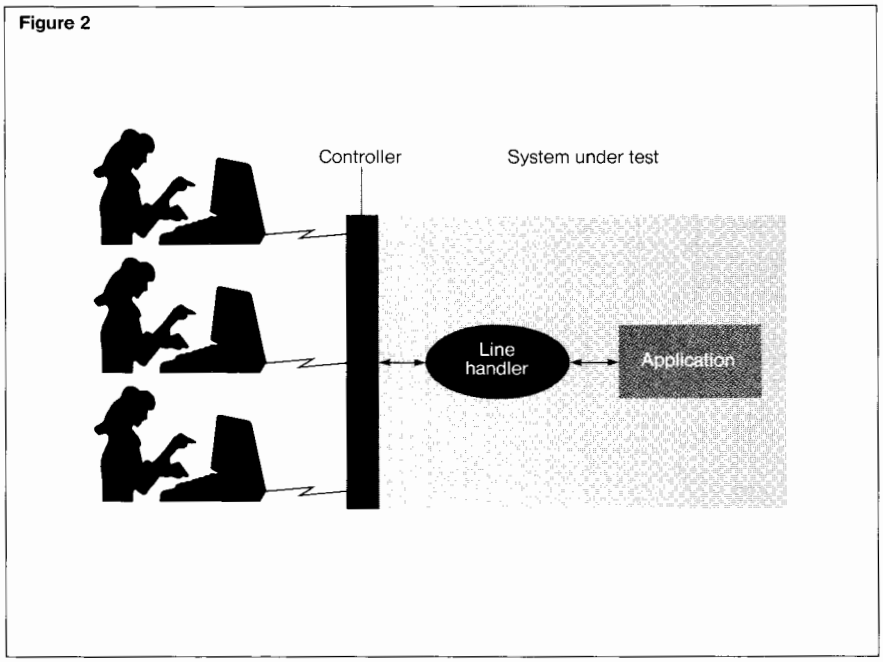


Figure 2



Stress Testing Techniques

Until recently, two major techniques were used to conduct stress tests: the *live-operators* technique and the *remote terminal emulation* technique.

The live-operators approach, illustrated in Figure 2, uses the live system's full set of hardware, including data communications lines and terminals. Operators at terminals connected to the system exercise the system by following prearranged scripts. This solution presents an accurate workload to the system, but suffers from several serious drawbacks: the high cost of hardware and personnel, the lack of control over parameters affecting the workload, and the lack of repeatability.

Figure 1.
Components of an interaction.

Figure 2.
The live-operators technique.

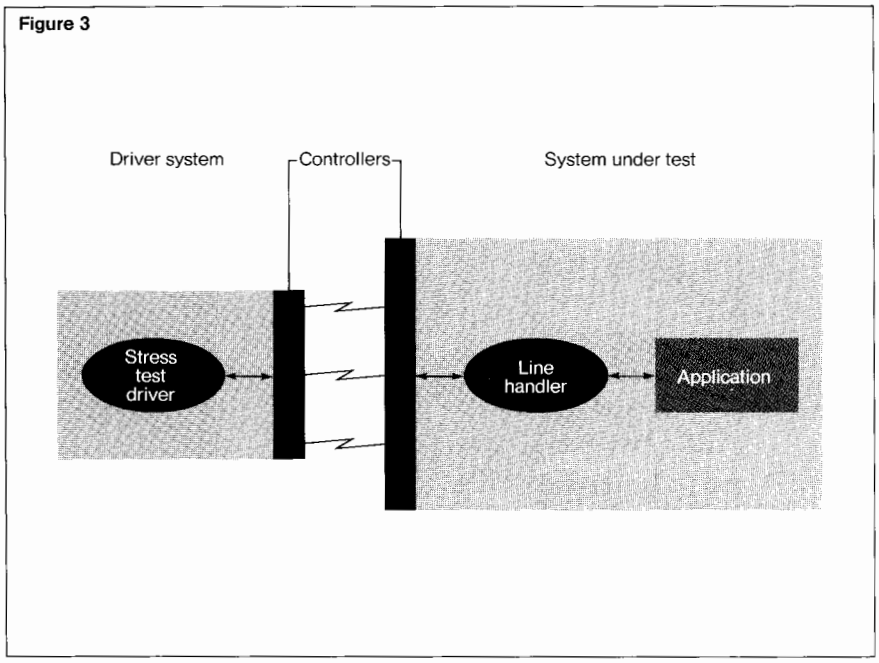


Figure 3.
The remote terminal emulation technique.

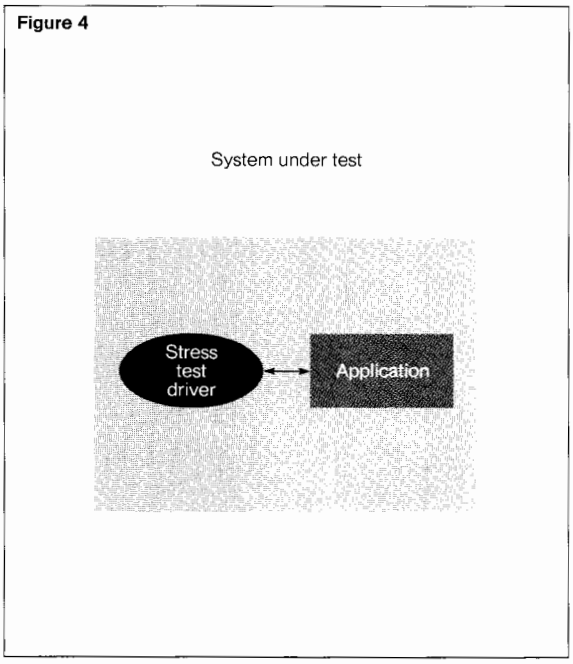


Figure 4.
The ENCORE stress test generator solution.

Remote terminal emulation is similar to the live-operators technique in that it uses most of the same hardware, but it employs one or more external computer systems to emulate the function of terminals and their operators. The driver that replaces the terminals and operators is called a remote terminal emulator (RTE). This technique is illustrated in Figure 3.

With the RTE approach, an accurate workload can be presented to the application, and repeatability and controllability can be provided through the automated driver. On the other hand, a major disadvantage of the RTE approach is its cost. Not only is there a high cost associated with procuring and installing all the communications cables and equipment, as there is in the live-operators technique, but at least one more external computer system is required as well. Modern communications terminals have evolved functionally to the point that many have several microprocessors to perform the work. Emulating them is very costly in terms of central processor consumption. These costly hardware requirements severely limit the usefulness of RTEs.

Recent research has demonstrated the viability of a third technique in which the stress test driver is located in the system under test (see Figure 4). The Tandem ENCORE™ stress test generator for PATHWAY applications uses this technique, replacing the function of the line handlers that are present under live conditions. By replacing the line handlers, ENCORE can take advantage of the system resources thus freed. Also, since ENCORE does not emulate the low-level functions of terminals, it provides high performance. In short, ENCORE presents an accurate, controllable, and repeatable workload while providing high performance at a low cost.

The ENCORE stress test generator consists of three components: CAPTURE, a mechanism to create workload scripts; EDITOR, a utility to edit workload scripts; and REPLAY, the workload generator. The rest of this article discusses some of the features of ENCORE. A basic understanding of the role of the PATHWAY Terminal Control Process (TCP) is assumed.

Figure 5

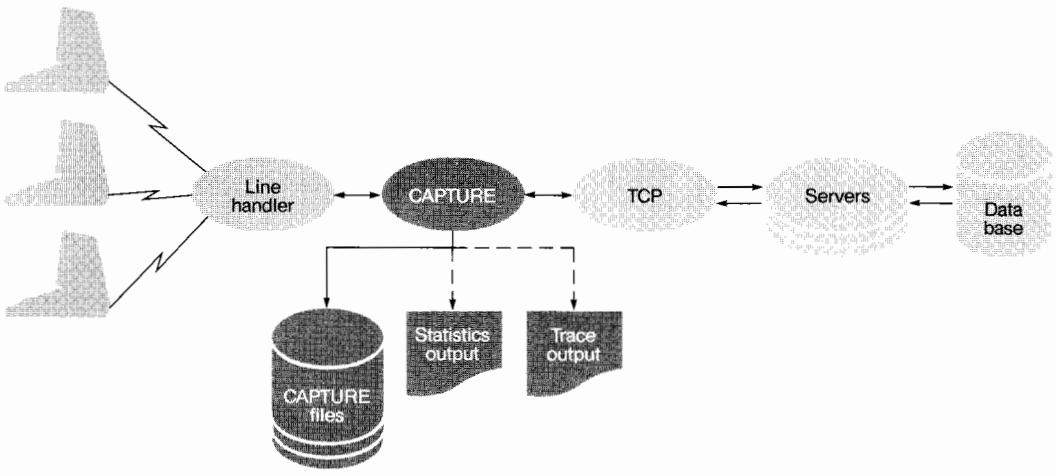


Figure 5.
Functional diagram of the CAPTURE module.

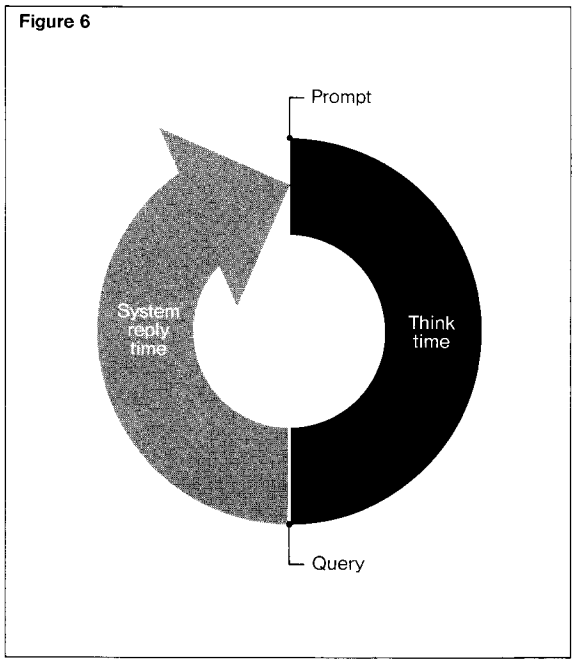
The CAPTURE Module

The CAPTURE module is the data collection component of the ENCORE stress test system. It functions simply as a recording mechanism. Optionally, the CAPTURE module provides a trace facility and summary statistics about the interactions. At least one CAPTURE process is placed between a PATHWAY TCP and its live terminals (see Figure 5). A TCP communicates with a CAPTURE process in the same manner it communicates with a line handler process. To accomplish this, the CAPTURE process operates as a mediator. That is, I/O operation requests (e.g., READs and WRITEs) are passed from the TCP to a CAPTURE process, which then communicates the request to a line handler process. When the operation is completed, the results (i.e., error codes and data) are returned from the line handler process to the CAPTURE process, which then passes them to the TCP, thereby completing the original request.

The CAPTURE module interprets each TCP request to determine what type of operation it is so the proper request can be communicated to the line handler process. Any data contained in the requests is recorded in the CAPTURE file associated with each terminal. Exactly one CAPTURE file is opened for each terminal. Input data from a terminal is recorded in its CAPTURE file and is passed, along with the operation results, back to the TCP.

The CAPTURE module times the operations and records timing information critical to performing the stress test in the CAPTURE files, along with the request data. Typical on-line transaction processing applications exhibit a sizable range of operator entry times, as some interactions require more time than others. For example, input to a menu screen usually requires very little time to be composed, whereas input to a data entry screen containing many fields may take much more time for the operator to complete. Because the module records the individual interaction timings, realistic interaction rates can be generated to accurately model the varying complexity in the interaction screens.

Figure 6.
The interaction cycle.



Components of the Interaction Cycle

The time during which an interaction cycle is completed is called the *interaction period,* and is composed of think time and system reply time. The portion of the interaction period consumed by the operator in composing a query is think time. System reply time, the other component of the interaction period, includes all the time consumed by the system in interpreting a query, reading the screen data, and writing out the next prompt screen. The cycle is illustrated in Figure 6.

Since data communication delays have a direct effect on the components of the interaction period, they are included in the timings. This allows the system-reply-time values to more accurately reflect the time required for the system to respond to a query. Likewise, the think time better reflects the amount of time before the next query is submitted to the system.

As indicated in the introduction, the performance requirements of on-line transaction processing systems are usually expressed in terms of interaction throughput and response time.

To effectively model a variety of workload environments, the tester needs the ability to vary the interaction frequency, i.e., the rate at which interactions are presented to the application. Manipulating just think time does not provide enough control of the interaction frequency because system reply time during the stress test may vary considerably from that observed during the CAPTURE phase. For example, suppose an interaction were composed equally of think time and system reply time, and the tester wanted to double the interaction rate. If the tester attempted to do this by adjusting the think time, he or she would have to reduce the think time to zero to double the interaction rate (assuming the system reply time remained the same). Since the system reply time might vary during the test, it would have to be considered when adjusting the overall interaction rate.

To properly control a stress test, the ENCORE stress test generator records the interaction period, rather than just the think time, with operator input records in the CAPTURE file. The tester can then specify the interaction rate during the stress test as a function of the rate during the CAPTURE phase. If the system reply time increases during the stress test, the REPLAY module decreases the think time to obtain the proper interaction period. This mechanism works as long as system reply time is less than the requested interaction period. If the system reply time exceeds the requested interaction period, then REPLAY can do no better than send the query immediately and record it as a late query. Late queries are discussed later in more detail.

Identification of Interaction Data

In addition to monitoring I/O requests for timing information, the CAPTURE module categorizes the requests. Categorizing the requests aids the EDITOR and REPLAY modules in interpreting a CAPTURE file. For example, many transaction processing applications place a base screen on the terminal and then accept multiple inputs from the screen without redisplaying the base screen. Because the CAPTURE module records interaction data as it is encountered, only one copy of the base screen is recorded. The records associated with the interaction immediately following the display of a base screen are located adjacent to the base screen, while subsequent interaction records are not. The EDITOR uses the base screen records identified by the CAPTURE module to reconstruct an interaction as it appeared during the record phase.

Records identified by the CAPTURE module as operator input records cause the REPLAY

module to pause, simulating operator think time. Note that output data records are recorded in the CAPTURE files for use by the EDITOR only, since only the input data is required to drive a stress test.

CAPTURE Module Commands

The user interface to the CAPTURE module consists of linking commands and output control commands.

Linking Commands. The linking commands establish a CAPTURE process between a TCP and its associated terminals, and direct the CAPTURE process to intercept and pass the data communications traffic through to the terminal or TCP. The file specification parameter in the PATHCOM configuration is modified for each terminal that is to be linked into CAPTURE. The modification specifies the CAPTURE process as the file to open instead of the terminal. A logical terminal name is also passed in the FILE OPEN to CAPTURE. This name is later used by CAPTURE to determine the physical terminals to be opened. The PATHCOM command syntax is as follows:[1]

SET TERM FILE ‹CAPTURE process name›.# ‹logical terminal name›

Example: SET TERM FILE $TSK1.#TERM01

‹CAPTURE process name› is the name of the CAPTURE process to be linked to the application. ‹Logical terminal name› is an arbitrary name of one to seven alphanumeric characters beginning with a letter. It must match a logical terminal name specified in a PARAM command given to the CAPTURE process. The PARAM syntax is as follows:

PARAM ‹logical terminal name› ‹physical terminal name›

Example: PARAM TERM01 $T1

‹Logical terminal name› is one to seven alphanumeric characters, beginning with a letter. ‹Physical terminal name› is the name of the real terminal to which the logical terminal name is to be linked. This arrangement provides the greatest flexibility in terminal naming. Also, the use of logical terminal names, because they are typically short, provides a convenient means to refer to the physical terminal names, e.g., as used by the trace facility.

Examples of the SET TERM FILE and PARAM commands are shown in Figures 7 and 8. To reduce the complexity of the figures, the role of the various hardware and software components, e.g., PATHMON, have been reduced or eliminated. Figure 7 shows the configuration of an application before the record phase. Figure 8 shows the same configuration with a CAPTURE process, $TSK1, properly inserted.

[1]Lower-case characters enclosed in less than/greater than symbols represent all variable entries supplied by the user.

Figure 7

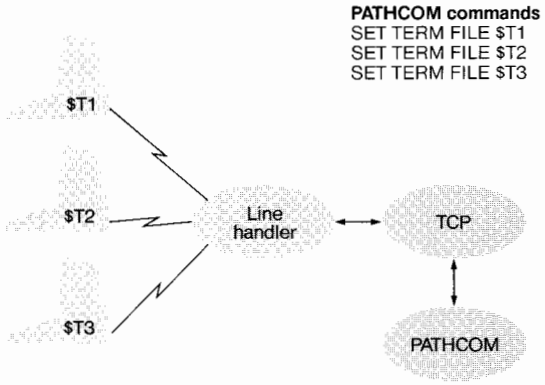


Figure 8

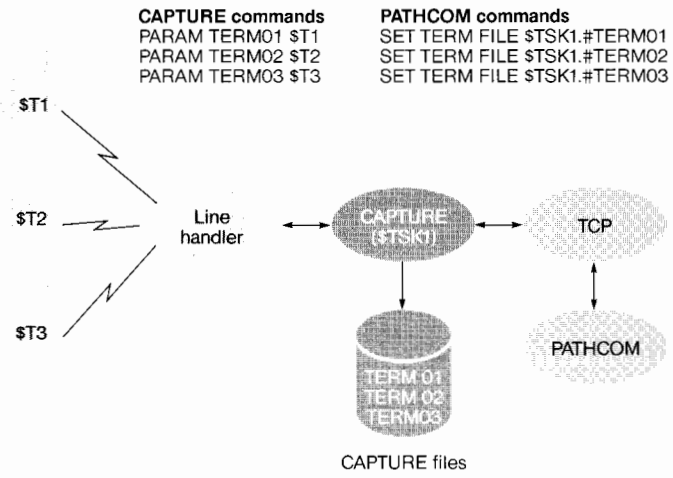


Figure 7.
A simple transaction processing environment.

Figure 8.
A simple CAPTURE configuration.

Output Control Commands. The output control commands specify the destination of trace and statistical output. Output can be directed to any output device, such as a printer, disc, or process. The *ENCORE Users Manual* contains more information about the commands.

Once configured, the CAPTURE process is started as an ordinary user program from a command interpreter. After initializing, CAPTURE waits for FILE OPEN requests from the TCP to begin processing. When a FILE OPEN request is received, the logical terminal name contained in it is extracted and used to search a table containing the terminal names specified by the PARAM commands. When the logical terminal name is located, the physical terminal name associated with the logical name is used to construct a new FILE OPEN request that is passed to the appropriate line handler process. For convenience, the logical terminal name is also used as the name of the CAPTURE file associated with the logical terminal.

The REPLAY module is the stress test driver and, in many respects, the most important module.

Once the OPENs to the terminal and CAPTURE files have completed, the CAPTURE process is established as the mediator between the TCP and the terminal. Users at the application terminals interact with the application in a normal fashion. The actions of the CAPTURE process are not visible to the terminal operators.

When all terminals controlled by a CAPTURE process are stopped, the CAPTURE process writes out its statistical output, if requested; closes the CAPTURE files; and terminates. The CAPTURE files associated with each terminal are then ready for input to the REPLAY module, or they can be edited by the EDITOR.

The EDITOR Module

The EDITOR is a screen-oriented, interactive utility that manipulates interactions and sets of interactions within CAPTURE files. An interaction is composed of a set of output streams (the prompt) and the input (query) in response to the prompt. The output streams may be as complicated as the set of terminal instructions that construct a data entry screen, or as simple as a reply from the application to the preceding operation (e.g., ADD SUCCESSFUL). Usually, the prompt and its associated query form an indivisible unit. The EDITOR combines the input data with the prompts to form complete interactions and presents them one at a time to the users, as a logical unit for EDITOR operations.

The EDITOR pieces together the components of an interaction to display it on the terminal screen as it appeared during the recording phase. The records in the CAPTURE file contain the data streams that make up the interaction; however, some interaction components may be widely dispersed in the file. This can occur when part of an interaction screen is written to the terminal as a base screen early in a CAPTURE session and ensuing interactions then make use of the base screen as part of their prompt.

An example of this might be the application heading that often appears at the top of each interaction screen (e.g., AIRLINE RESERVATION SYSTEM—CANCEL SCREEN). The heading might be transmitted when entering the cancel screen and would remain there while the user performed cancel operations. When the user changed modes, e.g., entered the add screen, the heading would then change. Because this type of heading is only written once but is used by subsequent interactions, the EDITOR uses pointers placed by the CAPTURE module to locate and identify the base screen associated with the later interactions.

Occasionally, it becomes difficult for the EDITOR to select the set of records that comprise an interaction from the continuous stream of records in the CAPTURE file. In other words, it may be difficult to determine when an arbitrary interaction starts and ends. There is no begin-interaction or end-interaction marker. In fact, some of the information that might help to identify the appropriate set of records resides in the application and with the developer. Because this information is not available to the EDITOR, reconstruction of an interaction screen can sometimes be done incorrectly.

Figure 9

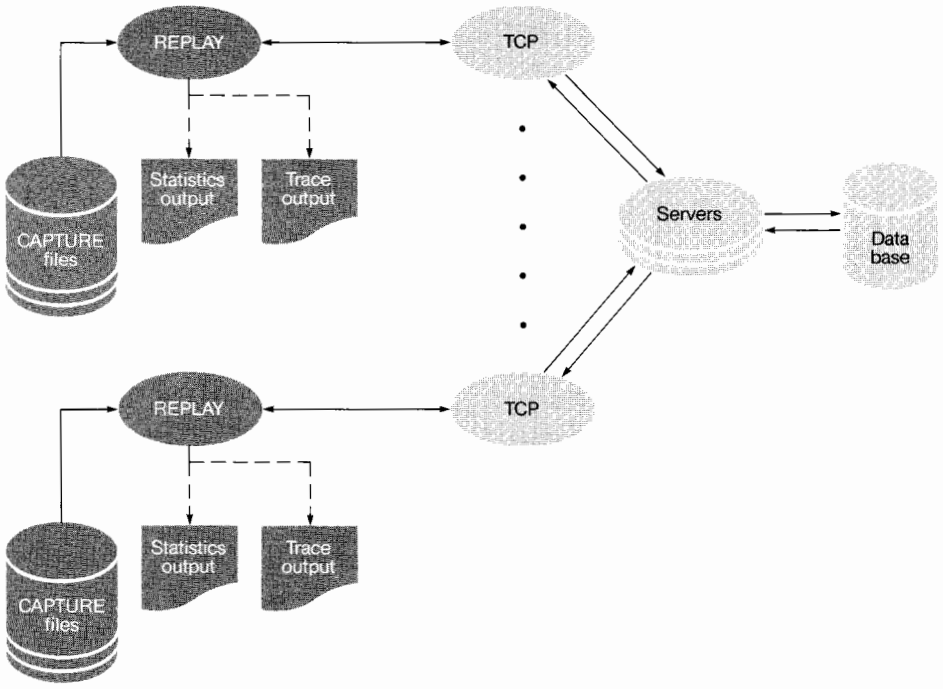


Figure 9.
Functional diagram of the REPLAY module.

It is important to be aware of this possibility and to be prepared to recognize such an occurence and recover from it. In some cases, it may be necessary to reposition to the first interaction of the CAPTURE file and single step forward to the desired interaction.

The external interface to the EDITOR uses the terminal function keys to manipulate the file. The user simply runs the EDITOR, specifying the name of the file to be edited as a parameter. A complete description of the external interface can be found in the *ENCORE Users Manual.*

The REPLAY Module

Once a CAPTURE file has been prepared and, if necessary, edited, it is ready for use in a stress test. The REPLAY module is the stress test driver and, in many respects, the most important module of the ENCORE stress test generator. As described in the introduction, the REPLAY module operates in simplex mode, i.e., it resides in the system under test. It was designed to replace the line handler process that the TCP would have used to communicate with the terminals connected to the application (see Figure 9). Optionally, the REPLAY module can produce a trace of interaction data and a statistical summary of the interactions.

From the viewpoint of the line handler process, the outgoing and incoming data transfers consist of blocks of terminal instructions and screen data. This is exactly the data that was recorded by the CAPTURE module in the CAPTURE file. When the application makes a request for input during a stress test, the request is passed to the REPLAY process. To satisfy the request, the REPLAY process retrieves the next input record from the appropriate CAPTURE file and presents this data to the requesting process. From the TCP's viewpoint, it appears as if a terminal just satisfied the input request.

Because REPLAY operates with preassembled data, CPU consumption is minimized. In fact, the REPLAY process typically consumes less of the total CPU resource than the real line handler process it replaces. This efficiency can cause problems. An application that functions adequately during the stress test could experience performance degradation due to queueing for CPU resources, if the line handler processes require substantially more of the total CPU resource than the REPLAY processes.

To prevent problems caused by CPU consumption perturbations, each REPLAY process can be configured to consume additional CPU resources. The command to configure the REPLAY process is presented later in the discussion of the user interface.

The other components of the on-line transaction processing system replaced by the REPLAY module are the data communications lines, terminals, and terminal operators. These components affect a system only indirectly by introducing delays into the flow of data. The effect of these delays is controlled by varying the interaction rate of the stress test.

Controlling the Interaction Rate

As described in the CAPTURE module section, the interaction period is composed of system reply time and operator think time. Because the system reply time is under the control of the system under test, it is not possible for REPLAY to manipulate that time to modulate the interaction rate. Control of the operator think time, however, rests with the operator. Since REPLAY replaces the function of the terminal operator, it can modify the duration of think time, thereby influencing the interaction rate. REPLAY modifies think time by inserting a delay before replying to a request for an operator input record.

Input records are classified by the CAPTURE process as operator input and terminal input when the CAPTURE files are created. Operator input records are generated by terminal operator action, e.g., function key input. Terminal input records are records generated by the terminal in response to an application request and without intervention from the terminal operator, e.g., the terminal instruction READ WITH ADDRESS. When an input request is made to a REPLAY process, REPLAY simply responds with the next input type record found in the CAPTURE file. If the input record is of operator input type, then a delay mechanism may be invoked depending on the interaction rate requested by the user.

The REPLAY module can produce either a fixed interaction rate or a variable rate based on a multiple of the period associated with each interaction in the CAPTURE file. In either case, the desired interaction period is first determined. For fixed interaction rates, the requested rate is converted into the desired interaction period. For variable rates, the interaction period in each operator input record is adjusted by a multiplier specified by the user.

Next, the system reply time is determined by saving the time at which the last query (operator input) was sent to the application to satisfy an input request, and then subtracting that time from the time at which the current input request was received. The saved time is then subtracted from the time at which the current input request was received to obtain the system reply time. Then by subtracting the system reply time from the desired interaction period, the time required to match the desired interaction period, the think time, is obtained:

$$\begin{array}{l} \text{CURRENT-REQUEST-TIME} \\ -\ \text{LAST-QUERY-TIME} \\ \hline \text{SYSTEM-REPLY-TIME} \end{array}$$

$$\begin{array}{l} \text{DESIRED-INTERACTION-PERIOD} \\ -\ \text{SYSTEM-REPLY-TIME} \\ \hline \text{THINK-TIME} \end{array}$$

If the think time is positive, the request is placed on a delay queue and resumed after the proper amount of time has elapsed.

Late Queries

Although the REPLAY module accurately computes the time at which a query must be submitted to meet the desired interaction period, a number of factors can prevent it from replying to the associated request on time. If REPLAY fails in submitting a query on time, it is called a late query. Figure 10 illustrates the situations causing late queries.

In Figure 10, T0 is the time at which the previous query was submitted by REPLAY. T1 is the time at which the query for the current request must be submitted to conform to the desired interaction period, T1-T0. The think time is the difference between the time the request is received and T1, T1-Tr. Normally, the system replies by time Tr, the REPLAY module delays for the think time, T1-Tr, and submits the query on time at Tq. If the difference between the time when the input request was received and T1 is negative, the request was received after T1, as indicated by Tr′. In this situation, REPLAY can do no better than to submit the query as soon as possible, as indicated by Tq′. This late query situation is usually caused by driving the application too hard, resulting in excessive system reply time and leaving no surplus time for operator think time.

Over-driving the REPLAY module can also cause late queries. If a large number of requests must be activated at the same time, some requests are clearly going to be activated late. REPLAY attempts to alleviate this condition by varying an activation time window to allow sufficient activation time. If late queries occur, the statistics output lists the percentage of queries that were late and the mean and standard deviation of the amount of time that the queries were late. High values warn of a possible performance deficiency during the stress test.

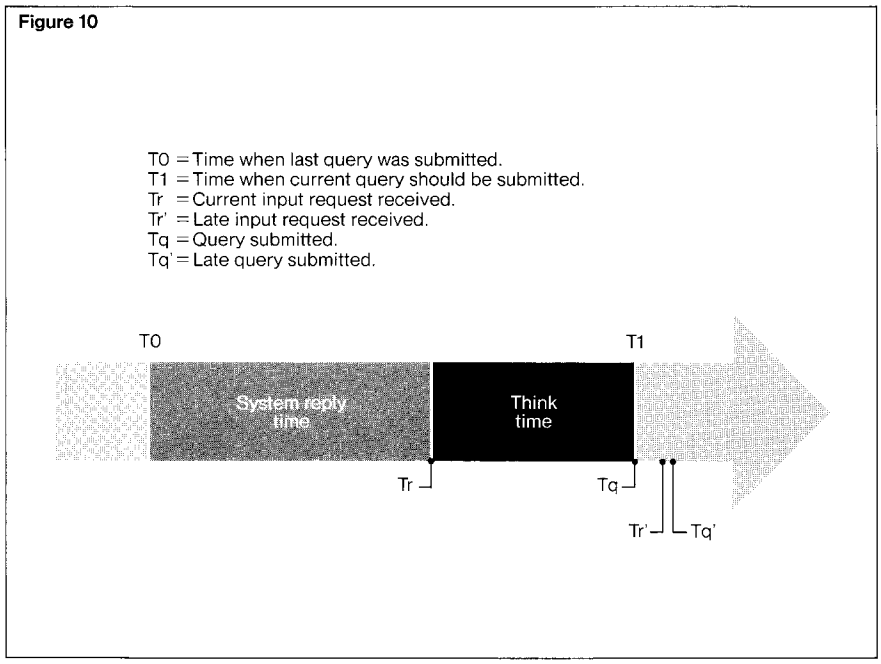


Figure 10.
Request-query timing.

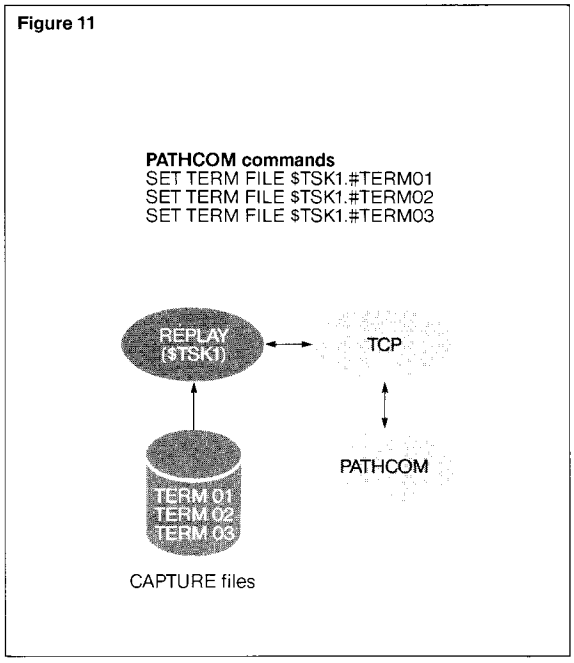


Figure 11.
A simple REPLAY configuration.

REPLAY Module Commands

The user interface to REPLAY is similar to that of the CAPTURE module. The user commands may be categorized as linking, performance control, and output control commands.

Linking Commands. The linking commands establish the connection between a REPLAY module and a TCP. The file specification in the PATHCOM configuration is modified for each terminal involved in the stress test by specifying a REPLAY process as the file to be opened by the TCP. A logical terminal name is passed to the REPLAY module to indicate which simulated terminals it is to open. The PATHCOM syntax is as follows:

SET TERM FILE ‹REPLAY process name›.# ‹logical terminal name›

Example: SET TERM FILE $TSK1.#TERM01

‹REPLAY process name› is the name of the REPLAY process to which the user desires to link the TCP. ‹Logical terminal name› is composed of one to seven alphanumeric characters, beginning with a letter. It is used as the name of the physical CAPTURE file to be associated with the terminal (see Figure 11).

‹Logical terminal name› is also used to associate the stress test parameters, start delay and interaction frequency, with a specific terminal.

Performance Control Commands. One use of the start delay parameter is to ensure that the interactions contained in identical files used by separate terminals are not presented to the application at the same time. When the interactions are staggered, accessing the same data base location at the same time is avoided, the disc heads are made to move, and the latency associated with live data base access is introduced.

A proper stress testing tool provides accurate workload presentation, controllability, repeatability, low cost, and high performance.

Another use of the start delay parameter is to simulate a terminal load that increases as the day progresses. An example of this is an airline reservation system at which only a few operators are available before 6 A.M., more come on line at 6 and 7 A.M., and at 8 A.M., the full staff is on line. With proper settings of start delay, any system of this nature can be simulated. The general syntax for the terminal stress test parameters is as follows:

PARAM ‹logical terminal name› (‹start delay›, ‹frequency›)

Examples: PARAM TERM01 (300, 0.125)
PARAM TERM02 (180, *0.5)

‹Logical terminal name› is obtained by the REPLAY module from the FILE OPEN request generated by the TCP. It is specified by the user in the SET TERM FILE command in PATHCOM. If a PARAM command is not specified for a terminal that is later opened by a REPLAY module, then global defaults are used. ‹Start delay› is used to delay the activation of a terminal until the specified number of seconds has elapsed. In the first example, a delay of five minutes has been specified, and in the second example, three minutes.

‹Frequency› is used to specify the interaction frequency for a terminal. It may be a fixed interaction frequency or, if preceded by an asterisk (*), a multiplier to be applied to the interaction period stored with each interaction in the CAPTURE file. The rate multiplier is very powerful because it retains the relative speed of operator entry with interactions of varying complexity, thus presenting a more realistic stress test. In the first example above, a fixed interaction frequency of 0.125 interactions per second has been specified, i.e., an interaction every eight seconds. In the second example, the interaction rate of the interactions in the CAPTURE file are decreased by one half.

The last performance control command is CPUBUSYPEROP. It is used to specify the CPU time to be consumed with each terminal I/O operation requested by the application. As outlined earlier, it is important for REPLAY to consume CPU resources equivalent to those consumed by the line handler process it replaces.

For simplicity, the CPUBUSYPEROP command uses the average CPU consumption for each terminal I/O operation. The syntax of the CPUBUSYPEROP command is as follows:

CPUBUSYPEROP (‹microseconds›)

Example: CPUBUSYPEROP (14850)

‹Microseconds› specifies the CPU time the REPLAY module is to consume for each terminal I/O operation performed. To obtain this value, the tester should use the XRAY performance analysis tool to determine the CPU time consumed by the line handler process. A perfect opportunity to perform this measurement is at the time the CAPTURE module is run, during the recording phase. The CAPTURE module lists, as part of its statistics output, the number of terminal I/O operations it performed. Thus, CPUBUSYPEROP parameter can be calculated simply by dividing the total CPU time that the line handler process consumed by the number of terminal I/O operations performed. In the above example, 14,850 microseconds of CPU time will be consumed by REPLAY for each terminal I/O operation performed.

Of course, the REPLAY module consumes a certain amount of the total CPU resource on its own. This consumption has been premeasured and is subtracted by the REPLAY module from the requested consumption. The difference is then consumed by REPLAY in a busy loop.

Output Control Commands. The output control commands are used to specify the destination of trace and statistical output. Output can be directed to any output device, such as a printer, disc file, or process.

Once configured, the REPLAY process is started as an ordinary user program from a command interpreter. After initializing, it waits for FILE OPEN requests from the application to begin processing. When a FILE OPEN request is received, the logical terminal name contained in the request is extracted and used to search a table containing the terminals specified by PARAM commands. If the logical terminal name is located, the stress test parameters associated with the logical terminal are applied; otherwise, the global defaults are used. Each simulated terminal then begins interacting with the application after an optional start delay.

The REPLAY process terminates when all the simulated terminals attached to it are closed. The terminals may be closed by PATHCOM commands or by the application itself via a query in the CAPTURE file; e.g., an application that terminates when the operator enters a function-key four can be terminated by setting up the last query in the CAPTURE file to submit function-key four.

Conclusion

Stress testing is an essential part of application development for any application in which performance is an issue. In a performance-critical environment, an application is not fully tested until it has been through a stress testing cycle.

A proper stress testing tool provides accurate workload presentation, controllability, repeatability, low cost, and high performance. The ENCORE stress test generator provides all these features, along with a simple, user-friendly interface. It makes stress testing a more practical and effective method of assuring the performance of an on-line transaction processing system before it is placed in production.

References

Smith, L. April 1982. *Designing a Network-Based Transaction-Processing System.* Tandem Application Monograph Series. SEDS-002. Tandem Computers Incorporated.

Spiegel, M. Summer 1981. 1981 RTE's—Past is Prologue. *ACM Sigmetrics Performance Evaluation Review.* vol 10, no 2, 66-73.

Stan Kosinski is currently a member of the Terminal Systems Group in Austin, Texas. Since joining Tandem in December 1980, he has worked on various terminal simulation projects and developed the ENCORE stress test generator while in the Operating Systems Performance Group. Before joining Tandem, Stan spent ten years in the computer field in a variety of areas, including performance prediction and analysis, artificial intelligence, computer hardware/software architecture, and data communications. While a graduate student at the University of California, Santa Barbara, he wrote a thesis on stress testing on-line transaction processing systems.

The 6100 Communications Subsystem: A New Architecture

With the 6100 Communications Subsystem™ (6100 CSS), Tandem has introduced a new integrated communications architecture. The 6100 CSS was designed to assume many of the teleprocessing control functions in Tandem NonStop systems formerly assigned to separate software and hardware components, such as interrupt handlers, communications processes, and communications controllers. It consists of a set of basic hardware and software components which can be assembled to do a variety of teleprocessing tasks, providing more capabilities, fault tolerance, and flexibility than other Tandem communications products. The 6100 CSS is supported by both the Tandem NonStop II and NonStop TXP systems. (Any references in this article to the "host" refer to the NonStop II or the NonStop TXP processors.)

Tandem communications controllers have evolved from the original 6301 (2) asynchronous controller to the recently announced 6204 bit-synchronous controller. While these controllers are limited to supporting certain protocols and electrical interfaces, the 6100 CSS architecture has been designed to support a variety of communications protocols and electrical interfaces. It provides a foundation for future communications evolution.

Motivations

The following limitations of the asynchronous, byte-synchronous, and bit-synchronous controllers were considered when the requirements for the 6100 CSS were defined:

1. Polling and being polled are nonproductive overhead within a processor.
2. Controller failures may bring down 4 lines in the best case and 32 in the worst case.
3. Most controllers are protocol-specific, prohibiting the cost-effective mixing of different line disciplines. Adding a new protocol entails adding controller microcode, a processor driver and protocol, and controller diagnostics.
4. Controllers do not provide for new electrical interface standards.

It was decided that limitations 1, 3, and 4 could be eliminated by designing a new controller that was programmable and could support the polling protocol supported at that time by the host processor. Limitation 2 could be eliminated in most configurations by redesigning the communications controller architecture. The approach agreed upon was to provide a programmable controller that had the flexibility to support multiple electrical interfaces and that diminished the possibility of a single point of failure.

The Architecture

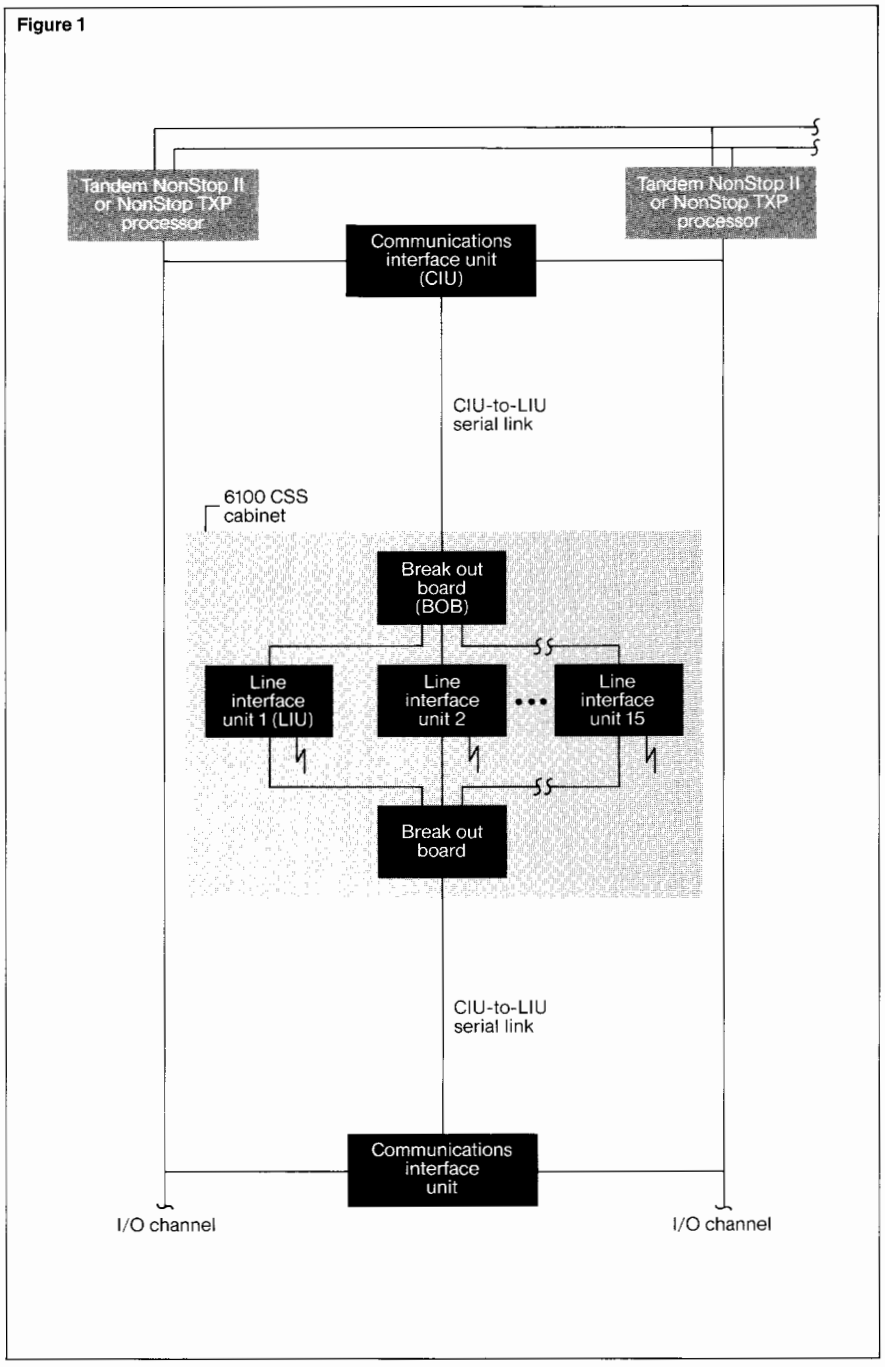
The architecture of the 6100 CSS eliminates the need to program the host in order to implement communications protocols at the Link Level or Level 2. (The term "Level 2" refers to the lowest software level of the International Standards Organization seven-layer Open Systems Interconnection reference model for data communications.) The development approach taken was to off-load the Link-Level processing onto an intelligent controller. This was done by designing a subsystem made up of easy-to-configure, easy-to-manufacture devices that can support any of the desired line types, with the loading of appropriate software from the host.

To implement the 6100 CSS design, the designers took a "building-block" approach. For example, a standard host-system interface module was designed to eliminate the need for separate controller design efforts as new communications peripherals are integrated into the system. This approach reduces the need to alter software drivers and host adapters when new devices or protocols are integrated.

A high-level language was used to develop the 6100 CSS, resulting in a shorter development period as well as a product that is much simpler to support. The use of a high-level language also ensures the transportability of the communications protocols.

Because of the programmable nature of the 6100 CSS, most device changes and corrections can now be made in software instead of hardware. This allows field upgrades to be made more easily, quickly, and inexpensively. Although Level 2 has been off-loaded, the 6100 CSS has dump and trace facilities that can be used without stopping the processor.

Figure 1



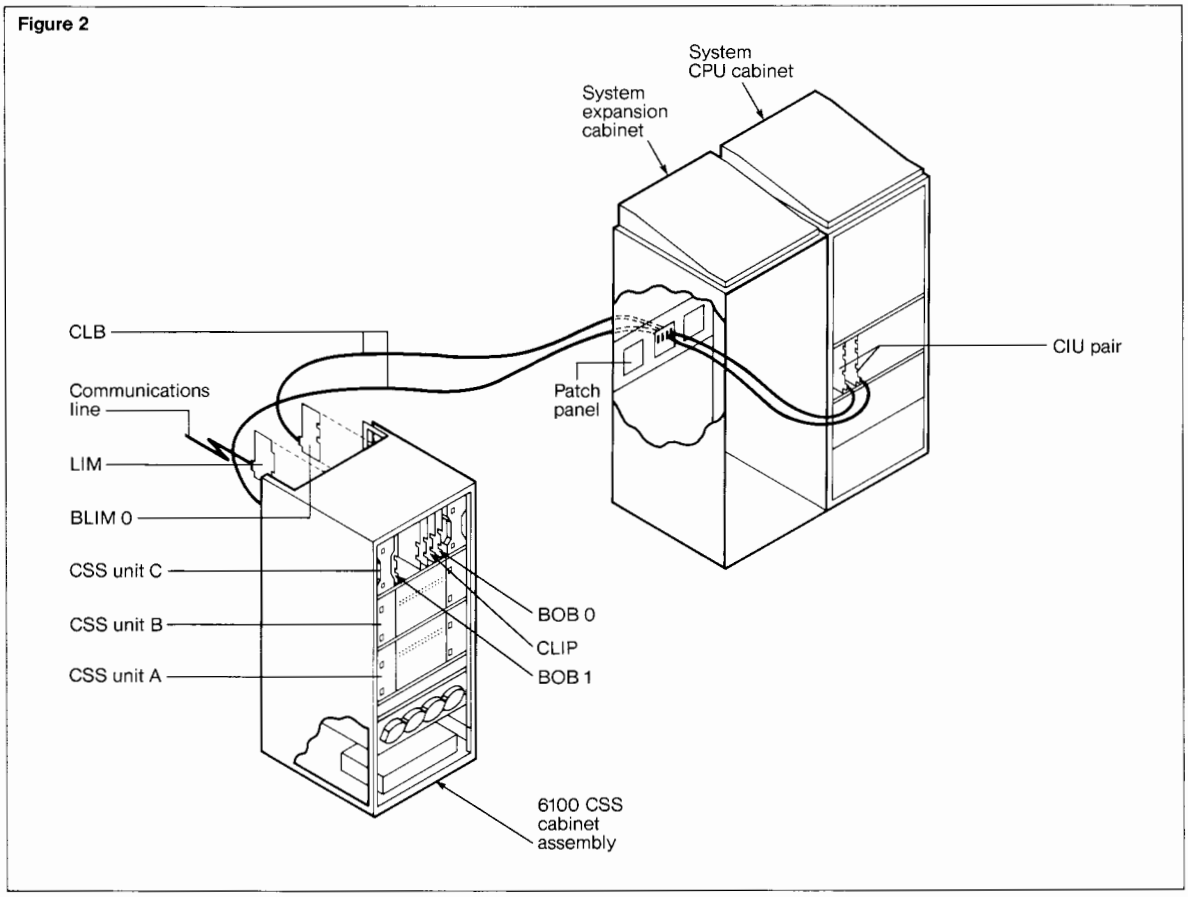
6100 CSS Hardware Component Overview

The 6100 CSS consists of three intelligent hardware components: the Communications Interface Unit (CIU), the Break Out Board (BOB), and the Line Interface Unit (LIU).

These components and their interfaces are illustrated in Figure 1. When configured together with a serial link and mechanical packaging, they provide an electrical and protocol interface to a variety of communications devices and systems. Figure 2 illustrates the physical packaging of the 6100 CSS.

Figure 1.
6100 Communications Subsystem hardware components.

Figure 2.
6100 CSS physical packaging.



The Communications Interface Unit (CIU) is the I/O controller located on the host that is the interface between the host and the remainder of the 6100 CSS. It is designed to facilitate the attachment of the 6100 CSS to a host processor's I/O channel. Each CIU is dual-ported to allow it to interface to the I/O channels of two host processors. Each 6100 CSS has two CIUs to eliminate a possible single point of failure. The CIU pair is capable of supporting up to 15 LIUs.

The CIU-to-LIU serial link (CLB) is a serial, full-duplex, point-to-point link. Address determination for the LIU is performed by the BOB. The end points of the CLB are the CIU at the host side and either the BOB or an LIU at the communications enclosure side. Transmission over the CLB is serial, using a bit-synchronous protocol and operating full-duplex at one megabit per second in each direction. The cable used for the CLB is a pair of twisted-pair wires, available in lengths of 50, 100, and 200 feet.

The cabinet in which the 6100 CSS resides is capable of accommodating up to three complete subsystems.

The Break Out Board (BOB) functions as an intelligent switch or multiplexor that controls the high-speed CLB serial link, allowing the CIU to select and talk to individual LIUs. It also provides the host with a mechanism for monitoring and controlling the 6100 CSS power supply attached to the BOB. The interface provided by the BOB allows the LIU and the application process in the host processor to act as if they have a point-to-point connection instead of a multiplexed link.

The Line Interface Units (LIUs) are microprocessor-based communications controllers responsible for handling individual communications lines. Each LIU is dual-ported on the CLB side to allow it to communicate with either of the two CIUs, each of which is

connected to two host processors. An LIU consists of two separate modules: a Line Interface Module (LIM), and a Communications Line Interface Processor (CLIP).

The LIM provides the electrical and mechanical interface to the communications line, while the CLIP provides the Level 2 protocol and the host software interface to the line. The 6100 CSS supports a mix of two CLIP types: a one-line version (CLIP-1) and a four-line version (CLIP-4). Figure 3 illustrates the physical relationship between the CLIP and LIM.

The CLIP-1 supports asynchronous, byte-synchronous, and bit-synchronous communications protocols on a single line, while the CLIP-4 supports asynchronous communications with up to four lines. The CLIP-1 supports asynchronous communications at speeds from 50 bps to 19.2K bps, byte-synchronous communications at speeds up to 9.6K bps, and bit-synchronous communications at speeds up to 56K bps full-duplex.

Figure 3

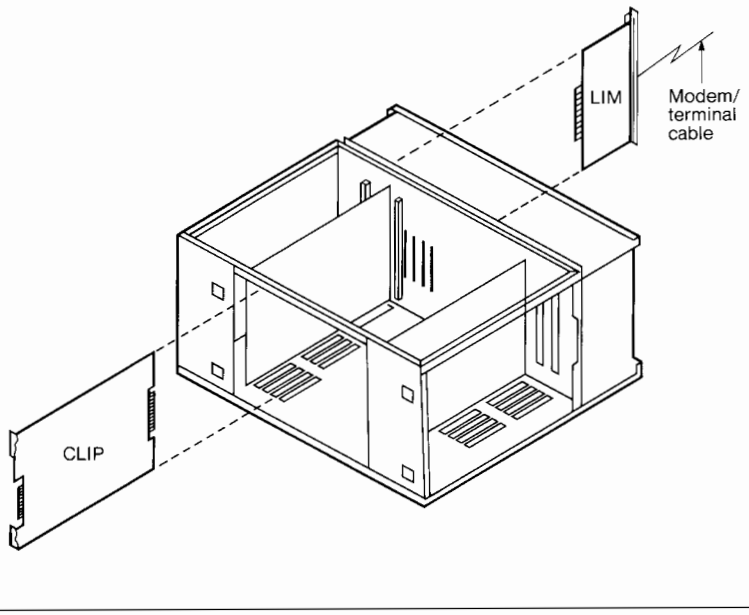


Figure 3.
6100 CSS card cage assembly.

6100 CSS Software Component Overview

As Figure 4 illustrates, the 6100 CSS software consists of a number of different components, each of which executes in one of the four processor types (the host, the CIU, the BOB, or the LIU). At a very general level, the software is structured as follows:

- The host software is loaded by, and runs under, the GUARDIAN operating system in much the same way that current I/O system code does.
- The LIU and CIU software is made up of both bootstrap software, in programmable read-only memory (PROM), and down-loaded software, in random-access memory (RAM).
- The BOB software is a microprogram in PROM that is factory-installed and not modifiable except by hardware revision.

Figure 4

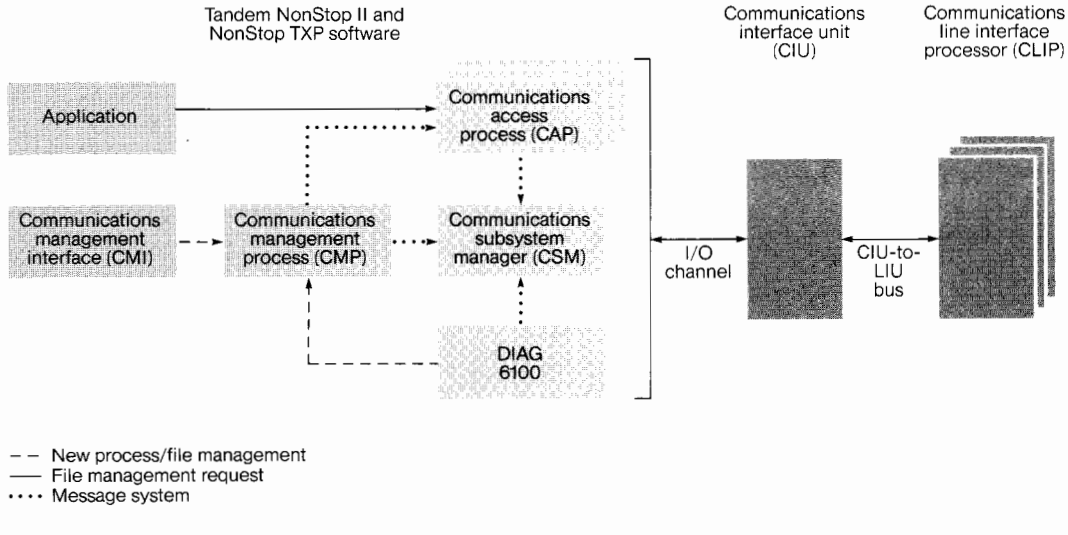


Figure 4.
Major 6100 CSS software components.

Host Software

The 6100 CSS software that runs in the host consists of the Communications Manager Interface (CMI/CMP), the Communications Subsystem Manager process (CSM), the Communications Access Processes (CAPs), and the DIAG6100 diagnostic process.

The new Communications Manager Interface (CMI/CMP) utility provides the operator interface to the 6100 CSS for configuration, control, and status display.

The Communications Subsystem Manager (CSM) is the host process that controls access to the CIU(s) for CIU and CLIP software down-loading, CIU diagnostics, and CIU tracing. It is also used to track CAP/CLIP paths, maintain configuration and status information, and monitor and control the 6100 CSS power supplies.

The Communications Access Processes (CAPs) are host processes that implement the user's file management interface to the 6100 CSS. All CAPs use a common internally defined protocol to communicate with a peer process in the LIU managing the communications line. For the initial release, most CAPs are merely modified versions of existing communications processes, with compatible application interfaces.

The diagnostics are provided by the DIAG6100 diagnostic process, a host process initiated by the user. During diagnostic testing, the CAP and CSM processes are suspended, and the DIAG6100 diagnostic process provides a direct interface to the user-specified, non-host portions of the 6100 CSS.

CIU Software

The CIU software implements the CLB protocol and the host I/O channel interface in the CIU. The CIU software object code resides in a disc file on the host system. It is down-loaded into the CIU by the CSM at system initialization.

LIU Software

Of the two cards that make up an LIU, the CLIP and the LIM, only the CLIP is an intelligent device. CLIP software implements specific communications line protocols. Each CLIP program is tailored to be accessed through a host CAP. The CLIP software object code resides in a disc file on the host system and is down-loaded into the CLIP by the CSM, at the request of the CAP.

With the 6100 CSS, much of the overhead involved in driving communications lines has been moved into the subsystem processors. In fact, all of the Level 2 processing, and in some cases, some of the Level 3 processing, has been off-loaded onto the LIUs. The host process providing the interface to code running in the LIU is the CAP. A CAP replaces each of the existing communications products, and each CAP includes the current Communications Process code above Level 2 as well as new 6100 CSS interface code.

The intermediate processing functions within the 6100 CSS that handle the control and routing of information within the subsystem are performed by the CIU (attached to the host channel interface) and by the BOB (in a separate communications cabinet). The CIU routes information from the parallel host channel interface to the serial CLIP interface via the CIU-to-LIU serial link (CLB).

6100 CSS Design Features

The 6100 CSS has the power and performance of a communications front-end without the costs normally associated with it, and has other significant design features. This discussion highlights some of these features and explains its advantages over more conventional monolithic front-end designs.

Fault-Tolerant Operation

The 6100 CSS can be configured to minimize the possibility of a single point of failure and therefore maximize its fault-tolerant aspects. This can be accomplished by having

one CIU primaried in one of the host processors and the other CIU primaried in the alternate processor. In the event of a failure, the maximum loss would involve only one LIU and its associated line(s). Any module in the 6100 CSS can be removed or replaced without affecting any other module. This includes the removal and replacement of power supplies, cables, fans, and logic modules.

Failure of a CPU, CIU, or BOB causes a "path" switch. A path switch does not take the line out of service, but end-to-end application recovery is required if the application is not using the Tandem EXPAND™ networking software.

The fault tolerance of the 6100 CSS is also ensured by the following four characteristics:

1. Host software process pairs.
2. Dual hardware components (CIUs, CLBs, BOBs, and power supplies).
3. Dual porting of CIUs and LIUs.
4. Automatic path failure recovery in the CSM and CAP processes.

Again, note that, given a properly coded application process and a fully configured 6100 CSS, the greatest loss that can occur from an LIU failure is the loss of a single line (or, with the CLIP-4, four asynchronous lines).

Adaptability

Both the 6100 CSS hardware and software are designed to allow a high degree of flexibility in configuring a communications subsystem. Because of the modular nature of the hardware and software components, many types of communications lines can be supported by a single 6100 CSS. This is especially advantageous when an installation supports a mixed network with a variety of terminals.

As with the Tandem NonStop systems, the 6100 CSS can start small and grow in incremental steps. LIUs can be added to an existing 6100 CSS, or additional 6100 CSSs can be added (up to three can be housed in one communications cabinet). This modularity allows the 6100 CSS to grow and to easily meet the needs of an expanding telecommunications installation.

Because one device can have a wide range of capabilities (bit-synchronous, byte-synchronous, asynchronous, and different protocol flavors of each), communications lines can be easily reconfigured. This increases the potential for system enlargement and development. Some line changes may require different electrical interfaces, necessitating a change to the LIM (a small inexpensive card) and its associated software driver in the CLIP. The rest of the hardware remains unchanged. Likewise, new link-level protocols can often be adapted by simply reconfiguring the line and by altering the modifiers and/or the CLIP program or line software. The hardware remains the same.

The modularity of the 6100 CSS allows it to grow to meet the needs of an expanding communications installation.

Supportability

Advanced diagnostics have been built into the 6100 CSS for easier maintenance and faster identification of problems. The modular design and lack of cabling in the 6100 CSS cabinet make it easy to replace individual hardware components simply and quickly, ensuring minimum time between fault detection and repair.

Through the DIAG6100 diagnostic process, the 6100 CSS accommodates a mechanism for initiating diagnostic (or general-purpose maintenance) routines on-line. Unlike previous Tandem diagnostics, the 6100 CSS diagnostics can be run while the system remains running.

These diagnostic routes are designed to identify problems down to the field-replaceable unit level. In addition, the CSM process periodically initiates a low-priority TEST sequence to perform backup path testing.

Also, hardware in the individual LIUs allows full loopback of a single communications line without having to remove any cables. This makes on-line testing easier and more thorough than ever before.

Another feature is that execution of the diagnostics from a remote node via EXPAND networking software allows for problem determination from a remote site.

Extensive tracing capability in both the CAP and CLIP software has been provided to assist in isolating line problems and software bugs. When required, CLIP memory can also be dumped for use in isolating software problems.

Utilizing Tandem's multi-processor architecture, the 6100 CSS allows the communications load to be distributed across any number of processors.

Performance

The 6100 CSS takes over many of the functions that were performed by the access method. This makes resources in the host processor available to handle other tasks.

Polling of asynchronous or synchronous multi-dropped lines is now done by the 6100 CSS. This eliminates the frequent interrupts that could otherwise slow down Tandem CPUs burdened with polling large numbers of terminals. The 6100 CSS also allows continuous polling. (The bit-synchronous and byte-synchronous controllers interrupt the host processor at the end of the poll list.)

Utilizing Tandem's Multi-Processor Architecture

In keeping with Tandem's multi-processor architecture, the 6100 CSS allows the communications load to be distributed across any number of processors. In other words, rather than funneling all communications lines into one processor, users can spread the load across two Tandem processors sharing a 6100 CSS, or across two or more Tandem processors, each with their own 6100 CSS.

Only the required amount of line-handling power need be purchased, and that power can be spread across the system in the most useful way. The same load-balancing features described above allow communications lines to be placed where processing power is available to handle them.

Single I/O Cardslot in the Host Processor Cabinet

Another advantage of the 6100 CSS is that it requires only one cardslot per CIU in the host I/O backplane. In the standard fault-tolerant configuration, two slots are used per 6100 CSS, which in turn can support 15 lines (or up to 60 asynchronous lines when the CLIP-4 is released). This allows more peripheral I/O equipment to be attached to each CPU.

First Release

Initial release of the 6100 CSS focused primarily on providing the pieces necessary to upgrade communications applications without redesign or additional coding. This was done by modifying existing access methods so that they interfaced to the 6100 CSS rather than to the existing communications controllers. (In effect, the access methods became CAPs.) Tandem communications products supported by the 6100 CSS are described below.

- The AM3270 Access Method provides an application process with the capability for accessing IBM 3270-type cluster controllers attached to a Tandem system via a bisynchronous multipoint communications line.
- The AM6520 Access Method provides an application process with the capability of accessing Tandem 6520 or 6530 terminals operating in block mode and connected to a multipoint communications line.
- The EXPAND networking software provides all of the components necessary to implement a network of Tandem systems, all connected to one another. It is an extension of the GUARDIAN operating system.
- The ATP6100 Access Method (previously referred to as TERMPROCESS) provides an application process with the capability of accessing asynchronous-type terminals (e.g., general TTY as well as ADM-2, 6520s and 6530s) on a point-to-point communications line.

- The X25AM Access Method provides access for Tandem NonStop systems to public X.25 packet switching networks such as DATANET-1, DATAPAC, DATEX-P, PSS, TELENET, TRANSPAC, TYMNET, or UNINET.

- The EXCHANGE™ communications subsystem provides the ability to perform remote job entry (RJE) batched communications with a host IBM mainframe, over switched or leased-line facilities, emulating either an IBM Multileaving HASP Workstation or an IBM 2780/3780 Data Transmission Terminal.

Currently, the Tandem ENVOY™ and ENVOYACP data communications managers provide a Level 2 interface; however, these products are not supported by the 6100 CSS because the Level 2 processing has been off-loaded from the system. Instead, the CP6100 CAP has been developed to allow the user direct interface to CLIP-based line tasks. Initially, CLIP line tasks were developed to support bisynchronous point-to-point and Advanced Data Communications Control Procedures (ADCCP) Level 2 protocols. The user application interface to the 6100 CSS versions of ADCCP and bisynchronous point-to-point protocols is a higher, process-level interface than the file-level interface of the ENVOY communications manager. This change in interface alleviates some of the file-level calls.

Future Releases

In the near future, the CP6100 CAP will interface to CLIP line tasks that support Texas Instruments Numeric Entry Terminals (TINET), as well as Burroughs Poll/Select protocol.

Several hardware introductions will also be made. The CLIP-4 will support four asynchronous communications lines, either in current loop or RS-232 interfaces. The X.21 and V.35 (CCITT recommendations) electrical interfaces will also be supported.

Conclusion

The 6100 Communications Subsystem originated out of a need for a programmable communications controller that supported multiple electrical interfaces and diminished the possibility of a single point of failure. As a result of these development considerations, the 6100 CSS is highly reliable and flexible, and it provides a foundation for future Tandem communications products.

Its hardware architecture, with the dual I/O controller interface, lessens the possibility of a single point of failure. Diagnostics can be run on an individual line without bringing down additional lines or host processors, and repairs can be made without affecting operational units. Also, the hardware is modularized to enable different electrical and mechanical communications interfaces to be implemented with minimal impact on the 6100 CSS. Finally, because the 6100 Communications Subsystem is a programmable product, many diverse protocols can be supported from the same hardware.

Rich Smith wrote this article from information gathered by Customer Application Support's Data Communications Group, which he joined in April 1982 when he came to Tandem. Rich has developed software courses for the 6100 CSS, and is now in Tandem's Systems Support Group supporting the product. Before joining Tandem, Rich was employed by another computer vendor as a member of the development staff for an IBM 3274 simulator.

The Relational Data Base Management Solution

Relational data base management systems originated in the early 1970s as an alternative to network systems. Due to their numerous advantages, they are a highly effective method of data base management for today's processing environment. This article provides a basic understanding of both systems, and briefly introduces the Tandem ENCOMPASS™ relational data base system. It then compares network and relational systems and discusses the advantages of relational systems. It concludes with a discussion of the role of relational systems in the software life cycle and their suitability for application development.

Trend Toward Relational Systems

From the 1960s on, computer vendors, software houses, and educational institutions devoted much of their resources to the research and development of data base management systems based on the network model. Network systems such as Cullinane's IDMS, Cincom's TOTAL, and Hewlett-Packard's IMAGE were widely used. Because of the proliferation of network systems, the Committee on Data System Languages (CODASYL) was formed in an attempt to establish network industry standards. Although CODASYL began its efforts in the early 1960s, after 20 years, the standard is still incomplete. The basis for the description of network data base management systems in this article is the CODASYL Model of Network Data Base Systems, established by the CODASYL Data Base Task Group (DBTG).

Network systems originated when hardware was slow and expensive, and system-level software was rudimentary. Because of this limited technology, the primary concern of the computer industry was user control over resources for efficient performance. During this period, network systems were the most efficient data base management systems available.

With the advances in hardware and software technology that followed, efficiency and low-level resource control were no longer the only desirable qualities for software. Additional requirements included flexibility, simplicity, and ease of use. As a result, a more effective model for data base management systems, the relational model, originated in the early 1970s. Today, many providers of data base management systems recognize that the relational model meets modern requirements better than the network model. Not only are relational systems flexible and easy to use for non-technical users, they also increase data base functionality.

As a result of extensive research by nonprofit organizations and computer manufacturers in relational system areas such as query optimization, hardware architecture, recovery, deadlock, operating system support, utilities, and distributed data bases, relational systems have become feasible for production applications. Professional groups such as the Association for Computing Machinery (ACM) and the Institute for Electrical and Electronic Engineers (IEEE) provide publications to communicate developments and exchange opinions. Also, a Data Base Systems Study Group (DBS-SG) governed by the American National Standards Institute (ANSI) is investigating the development of a relational data base standard to provide guidelines for consistent terminology, user interfaces, and functionality.

As is evident from the announcements of new data base systems in current computer publications, the industry trend is toward relational systems. New relational systems that are highly successful include Relational Technology's INGRES, Oracle Corporation's ORACLE, and IBM's DS/SQL. IBM has announced another relational data base system, DB2, which is expected to be available in late 1984. Cincom Systems, who successfully marketed the TOTAL Network Data Base System, is now marketing a relational system, TIS, as their next generation data base manager. Some vendors of network systems are even providing a layer on top of their data base software to provide some relational access capabilities.

The Basics

In Table 1, the terms for fields, records, and files used in network and relational data base management systems are listed.

The Data Model

Before any data base, network or relational, can be defined, the data base designer must first design a *logical data model.* This model captures the meaning and relationships of the data as they exist in the real world. To build a logical data model, the designer must determine which *data entities* are involved and what the *relationships* among them are.

Figure 1

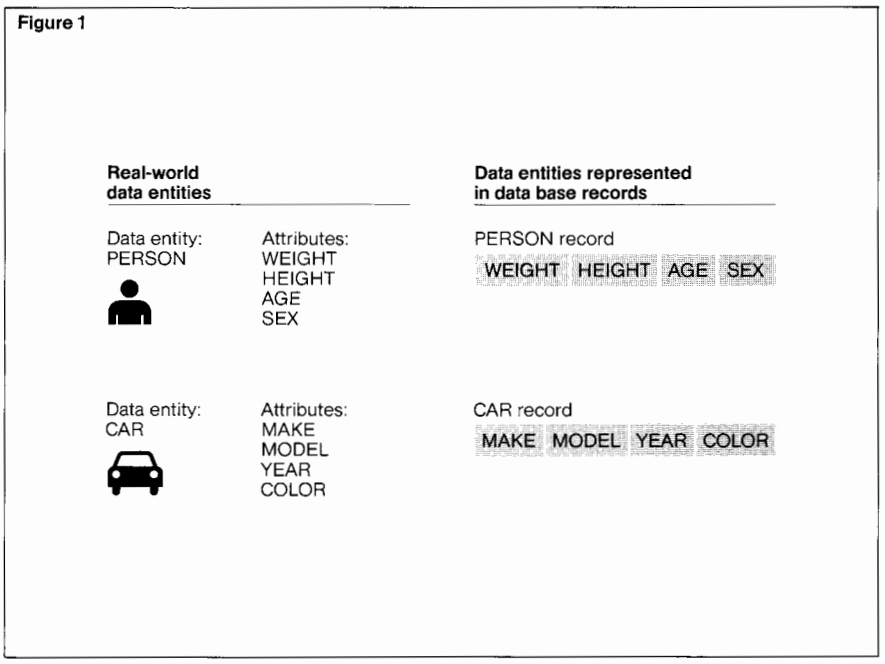


Figure 1.
Real-world entities and attributes as represented in data base records and fields.

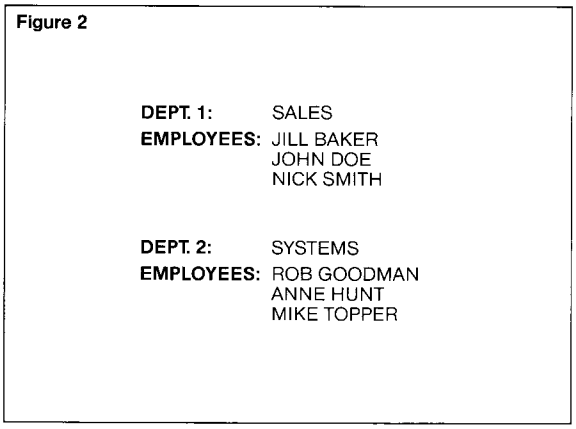
Data entities are the things that the data base will store information about. They are described by *attributes.* For example, the entity PERSON might have the attributes WEIGHT, HEIGHT, AGE, and SEX, and the entity CAR might have the attributes MAKE, MODEL, YEAR, and COLOR. Data entities in a data base are represented with records. The attributes of the data entities are represented with fields within the records. Figure 1 illustrates the representation of real-world entities and their attributes in data base records and fields.

Relationships among the data entities can be one-to-one, one-to-many, or many-to-many. The data base designer determines each relationship type by analyzing the application requirements and the user's real-world regulations for the data entities.

Table 1.
Basic terms used in network and relational data base management systems.

File System	Network	Relational	Theoretical Relational
Field	Field	Column	Attribute
Record	Record	Row	Tuple
File	Link-Set	Table	Relation

Figure 2.
An example of a one-to-many relationship is that of a department to its employees.



For example, the relationship between the entities DEPARTMENT and EMPLOYEE could be one-to-many (a department has many employees). This relationship would be valid only if the real-world regulations of the company state that a department can have many employees, but an employee can work for only one department. Figure 2 illustrates the one-to-many relationships in two departments, a sales department and a system department.

If the company allowed its employees to work in more than one department, the relationship would be many-to-many since a department could have many employees, and an employee could work in many departments.

An example of a one-to-one relationship, DEPT-CODE, could be the relationship between the data entities DEPARTMENT and DEPT-NUMBER. Each department would have a unique department number, and that department number would identify that department only. In Figure 2, the department SALES is uniquely related to the department code of 1, and SYSTEMS to code 2.

The purpose of the data model is to guide the definition and physical implementation of the data base. The data base structure should accurately reflect the relationships among the entities and provide data access paths for efficient data manipulation.

Network and relational systems are based on extremely different data models, and many of the advantages of relational systems over network systems stem from this difference. Network and relational data models are described in detail in following sections.

The Data Manipulation Language

While the data model defines entities and relationships, the data manipulation language, or DML, allows the user to manipulate the data. The user can add, delete, move, update, and read data, as well as perform other operations. The data manipulation languages for the network data model and the relational data model are as different as the models themselves.

Network Systems

Network Data Model. In a *network data model,* real-world data entities, such as people, cars, and departments, are represented with *record types.* A record type is similar to a COBOL definition of a group item or to a PASCAL record-structure type declaration. An instance of a record type is simply called a record. Relationships among the record types are defined with the *link-set,* the basic informational building block. The link-set consists of one *owner* record type and possibly many *member* record types. The owner is the parent of its member record types. For example, a salesperson (the owner record) has many customers (the member records), just as a department has many employees, and a writer writes many books. Graphically, the link-set may be represented with the tree notation or multi-list notation, as in Figure 3.

Using link-sets, a data base designer constructs a network data model by connecting the owner record types and member record types. The connections among record types are called *links* or *chains.* Member record types can themselves be owners of other record types; similarly, owner record types can be members of a parent record type. Thus, the term "network" is appropriate because link-sets can be connected in a network to form any topology.

The main restriction of the link-set is that the owner and member records cannot be of the same record type. This restriction makes it very difficult to design network data bases for certain applications, such as organizational charts and bills of material. For example, in most companies, a person can manage other people and can, in turn, be managed by someone else. Also, everyone in the company is considered an employee of the company, including the president. The

most efficient data model for this type of organization would be made up of the entity EMPLOYEE and the relationship MANAGED-BY among the employees. This would establish a link-set in which the owner record of type EMPLOYEE would be the manager of its member records of type EMPLOYEE. This link-set, however, is prohibited in the network model, since both the owner and member record types are the same. (Network data base designers commonly avoid this by defining a dummy indirect owner record of a different type.)

Figure 4 illustrates how the link-set in a network data model is used to represent the relationship of a salesperson to many customers. Using the multi-list notation, a link-set SALESPERSON-CUSTOMERS is defined. The owner record type SALESPERSON has the fields ID, NAME, and DEPT, and the member record type CUSTOMER has the fields CUSTOMER-NAME and ADDRESS. A chain from the owner record type to one or more records of the member type represents the desired relationship.

In the link-set in Figure 4, JOE has two customers (FANTASY CO and TOYBOX) in his member chain, MARY has one customer (SUITS INC), and JOHN has no customers.

Network Data Manipulation Language. The data manipulation language in the network data model is based on the concept of chains within link-sets. These chains provide logical access paths for the network data manipulation language to traverse in order to access related records. The traversal of chains is known as *navigation.* Therefore, not only does the network data manipulation language provide verbs to add, delete, update, and read records, but it also provides verbs such as CONNECT, DISCONNECT, and RECONNECT to navigate and manage chains.

For navigation, network systems provide many types of *currency pointers,* such as the current of run-unit, current of link-set-type, and current of record-type. The current of run-unit identifies the desired program process, the current of link-set-type identifies the desired link-set, and the current of record-type identifies the desired record within the current link-set. In other words, the record-type currency pointer keeps track of the user's position in a chain. Many statements of the network data manipulation language implicitly reference a currency pointer. For example, the DELETE verb deletes the record indicated by the record-type currency pointer, and the INSERT verb adds a record to a chain just after the record indicated by the record-type currency pointer.

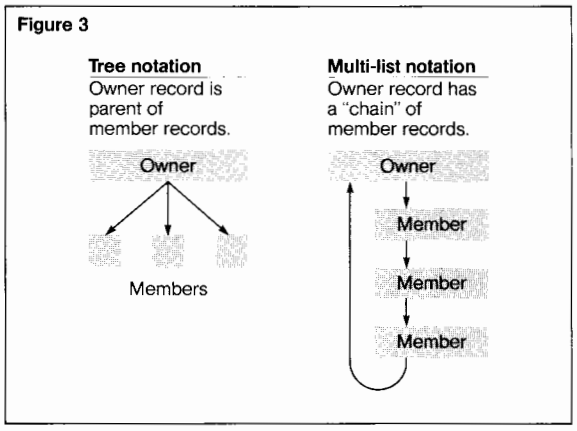


Figure 3.
Link-sets can be represented graphically by the tree notation or the multi-list notation.

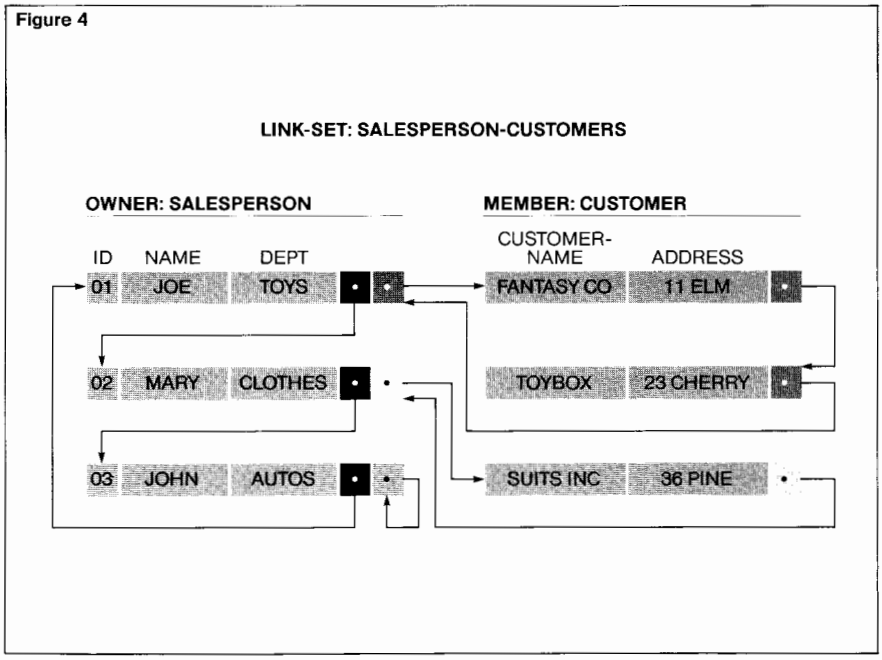


Figure 4.
A link-set in the network data model, representing the relationship of a salesperson to many customers.

Figure 5.
Example of a table in the relational data model.

Figure 5

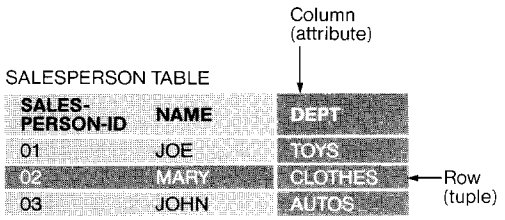


Figure 6.
The common column SALESPERSON-ID defines a one-to-many relationship between the SALES-PERSON and CUSTOMER tables.

Figure 6

SALESPERSON TABLE

SALES-PERSON-ID	NAME	DEPT
01	JOE	TOYS
02	MARY	CLOTHES
03	JOHN	AUTOS

CUSTOMER TABLE

SALES PERSON-ID	CUSTOMER-NAME	ADDRESS
01	FANTASY CO	11 ELM
01	TOYBOX	23 CHERRY
02	SUITS INC	36 PINE

For example, in a generic network data manipulation language, to answer the question, "For which customers is salesperson Joe responsible?" one could say:

```
FIND SALESPERSON RECORD
   WHERE NAME = 'JOE'
FIND FIRST CUSTOMER RECORD
IN CURRENT SALESPERSON-CUSTOMERS
WHILE NOT FAIL DO
   PRINT CUSTOMER-NAME OF CUSTOMER
   RECORD
   FIND NEXT CUSTOMER RECORD
   IN CURRENT SALESPERSON-CUSTOMERS
END
```

The first FIND initializes the link-set-type and record-type currency pointers to JOE's record in the owner chain SALESPERSON. Next, the FIND FIRST statement sets the record-type currency pointer to the first member record in JOE's chain (the member record with CUSTOMER-NAME FANTASY CO). The WHILE loop prints each customer name with PRINT and traverses the chain with FIND NEXT until the end-of-chain is reached.

Relational Systems

Relational Data Model. In the *relational data model,* data entities are represented with two-dimensional *tables.* A table is a collection of homogeneous records, and each record is composed of fields. In relational terminology, records are called *rows,* and fields are called *columns.* Figure 5 illustrates a table representing the data entity SALESPERSON with the columns SALESPERSON-ID, NAME, and DEPT.

Each column is an attribute of the data entity that the table represents. In Figure 5, each salesperson has the attributes IDENTIFICATION NUMBER, NAME, and DEPARTMENT. In order to maintain data integrity, a column can store data from a set of acceptable values only, known as the *domain* of the column. Whether or not a value is acceptable in the domain is defined by the application. For example, if the column represents CUSTOMER-ACCOUNT, its domain is taken as the list of valid customer account numbers currently on file. If the column represents GENDER, its domain is (MALE, FEMALE).

Relationships among tables are defined when the same column appears in two or more different tables. The column does not have to have the same name in each table, but it must have the same domain of acceptable values with the same real-world semantics. For example, if the common column were called DISTANCE, the tables that had this column would all have to contain values represented in the same form of measurement, such as feet, in this column. One table could not store the distance in inches, and another table in yards or metric units.

Each row in the table represents an instance of the data entity. In Figure 5, the SALESPERSON table has three rows, each representing a different salesperson (JOE, MARY, JOHN). To distinguish one salesperson from the other, the SALESPERSON-ID column stores a unique identification number for each salesperson. Other unique numbers often used to identify people are SOCIAL-SECURITY and DRIVERS-LICENSE.

Such a column used to distinguish rows in a table is called a *key.* A key can be composed of multiple columns. For example, the columns YEAR, DAY-OF-YEAR, and TIME could form a key that represents a unique timestamp. A table can be defined to allow duplicate key values or to maintain unique key values, depending on the table's role in any relationships in which it participates.

In Figure 6, a one-to-many relationship among two entities, SALESPERSON and CUSTOMER, is defined. In this relationship, a salesperson is responsible for many customers.

The SALESPERSON table and CUSTOMER table are related through a common column called SALESPERSON-ID whose values are taken from the domain of valid salesperson identification numbers. The column SALESPERSON-ID is used to establish the relationship, since it guarantees the unique identification of a salesperson. In the SALESPERSON table, the key SALESPERSON-ID can store unique key values only, since each salesperson is identified with a unique identification number. In the CUSTOMER table, the key SALESPERSON-ID allows duplicate key values, since a salesperson can have multiple customers.

A salesperson is related to his or her customers through an identification number represented in the SALESPERSON-ID column of the CUSTOMER table. For example, both the customers FANTASY CO and TOYBOX belong to the salesperson with identification number equal to "01." By looking at the SALESPERSON table, one can determine that this salesperson is JOE. In this fashion, the two tables represent the fact that JOE is responsible for the customers FANTASY CO and TOYBOX, MARY is responsible for SUITS INC, and JOHN currently is not responsible for any customers.

The common columns used to form an informative relationship are usually among different tables, but this is not mandatory. They can be in the same table. For example, as illustrated in Figure 7, only a single table is required to record information on employees and identify their immediate managers. Each row not only describes the entity EMPLOYEE but also defines the employee-manager relationship.

Figure 7

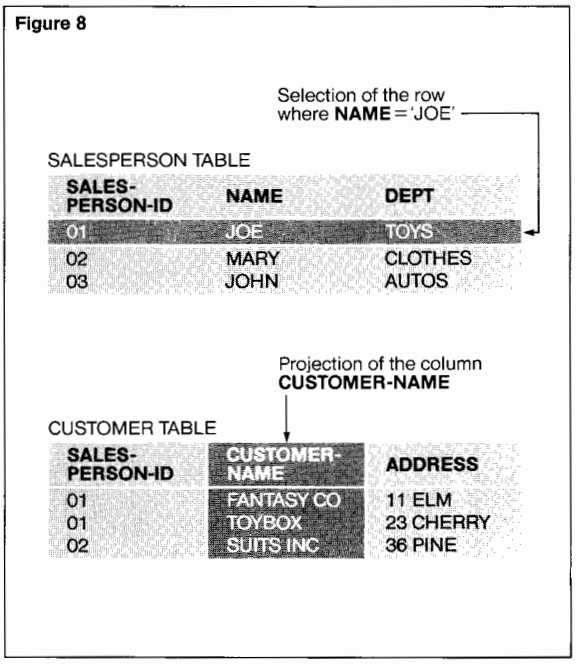
EMPLOYEE TABLE

EMPLOYEE-ID	NAME	TITLE	DEPT	MANAGER-ID
2020	JOAN	PRESIDENT	3	null
2021	SALLY	VICE PRES	4	2020
2022	MARY	SALES REP	5	2021
2023	BILL	ACCOUNTANT	5	2021
2024	JOE	CONSULTANT	4	2020

The two columns EMPLOYEE-ID and MANAGER-ID store data from the same domain of valid employee numbers. An employee, described with a unique row distinguished by the key EMPLOYEE-ID, is related to his or her manager through the manager's identification number in the row's MANAGER-ID column. For instance, both MARY and BILL work for the manager with identification number "2021." To determine whose ID this is, one can look in the EMPLOYEE-ID column and find that the ID is associated with SALLY. Therefore, this single table represents the fact that MARY and BILL work for SALLY, and SALLY and JOE work for JOAN. As president, JOAN does not have a manager.

Figure 7.
A single table can represent a relationship. In this table, the employee-manager relationship is represented.

Figure 8.
Projection selects columns; selection selects rows.



Relational Data Manipulation Language. The relational data manipulation language provides three fundamental operations that allow the user to pinpoint the data to be manipulated. The user can choose specific columns, select certain rows, and link multiple tables in order to access related data from tables.

Based on the same sales and customers example, a relational query that asks, "For which customers is salesperson Joe responsible?", would be:

```
LIST      CUSTOMER-NAME
WHERE     NAME OF SALESPERSON TABLE = 'JOE'
  AND     SALESPERSON-ID OF
          SALESPERSON TABLE =
          SALESPERSON-ID OF
          CUSTOMER TABLE
```

In this relational query, the column CUSTOMER-NAME is a *projection* of the table CUSTOMER. Since the purpose of the query is to list certain names, only the appropriate column (in this example, CUSTOMER-NAME) is projected onto the output device; all other irrelevant columns are ignored. The requirement NAME = 'JOE' is called a *selection.*

Whereas a projection selects columns, a selection selects rows. Projection and selection are the two fundamental operations that allow users to extract specific subsets of a table non-procedurally. (The non-procedural properties of a relational query language are further defined and analyzed in a following section.) Figure 8 illustrates projection and selection.

In the relational model, relationships among data entities (tables) are represented by a column common to those tables. For example, SALESPERSON-ID is common to both the SALESPERSON and CUSTOMER tables. A row in the SALESPERSON table is related to its row in the CUSTOMER table when the SALESPERSON-ID column in both tables stores the same value. Thus, to access related rows from multiple tables, the user matches rows from the tables, based on equal values of the common column. This is called a *join* operation. In the previous query, the matching of rows in the SALESPERSON and CUSTOMER tables by equating the SALESPERSON-ID columns from the two tables is an example of a join.

The join operation produces a logical view of the data to establish an informational relationship by recomposing the physical tables into a single virtual table. The set of columns of the single virtual table is the union of columns from the physical tables. Figure 9 illustrates a join where SALESPERSON-ID OF SALESPERSON TABLE = SALESPERSON-ID OF CUSTOMER TABLE. Note that since salesperson JOHN (identification "03") has no customers, he does not appear in the resultant logical view, as there is no row in the CUSTOMER table with SALESPERSON-ID equal to "03" to satisfy the join operation.

Figure 10 shows the procedure for processing the query in the example, involving projection, selection, and join operations applied together. The relational system first applies selection on the SALESPERSON table to find all rows with column NAME equal to JOE. Then, it performs the join to the CUSTOMER table using the common column SALESPERSON-ID. JOE's ID is "01", so all rows

in the CUSTOMER table with SALESPERSON-ID equal to "01" are linked. Finally, a projection applied to the linked rows in the logical view prints the desired CUSTOMER-NAMEs of FANTASY CO and TOYBOX.

A relational data manipulation language also provides other verbs to allow a user to add, update, and delete data. The user still applies projection, selection, and join operations to qualify the exact rows to be operated on. For example, the following deletes JOE's customers:

```
DELETE   ROWS FROM CUSTOMER TABLE
WHERE    NAME OF SALESPERSON
         TABLE = 'JOE'
  AND    SALESPERSON-ID OF
         SALESPERSON TABLE =
         SALESPERSON-ID OF
         CUSTOMER TABLE
```

In addition to AND, the set qualifiers OR and NOT can be used in the WHERE clause of a relational statement. For example, the following retrieves all the names of salespeople who work in the department TOYS or AUTOS:

```
LIST     NAME OF SALESPERSON TABLE
WHERE    DEPT = 'TOYS'
   OR    DEPT = 'AUTOS'
```

Also, selection and join operations do not have to be based on equality. For example, the following retrieves the names of salespeople with identification numbers greater than "02":

```
LIST     NAME OF SALESPERSON TABLE
WHERE    SALESPERSON-ID OF
         SALESPERSON TABLE > '02'
```

Summary of Network and Relational Systems

In a network system, data entities are represented with record types. Each record type has fields which are the attributes of the represented data entity. An instance of a record type is called a record. Relationships among the record types are established with link-sets, each consisting of one owner record type and possibly many member record types. All link-sets are established when the data base is initially defined and created.

Figure 9

SALESPERSON TABLE

SALES-PERSON-ID	NAME	DEPT
01	JOE	TOYS
02	MARY	CLOTHES
03	JOHN	AUTOS

CUSTOMER TABLE

SALES-PERSON-ID	CUSTOMER-NAME	ADDRESS
01	FANTASY CO	11 ELM
01	TOYBOX	23 CHERRY
02	SUITS INC	36 PINE

Join where **SALESPERSON-ID** of SALESPERSON TABLE = **SALESPERSON-ID** of CUSTOMER TABLE

LOGICAL VIEW produced by join

SALES-PERSON-ID	NAME	DEPT	CUSTOMER-NAME	ADDRESS
01	JOE	TOYS	FANTASY CO	11 ELM
01	JOE	TOYS	TOYBOX	23 CHERRY
02	MARY	CLOTHES	SUITS INC	36 PINE

Figure 10

SALESPERSON TABLE

SALES-PERSON-ID	NAME	DEPT
01	JOE	TOYS
02	MARY	CLOTHES
03	JOHN	AUTOS

Step 1. Selection of row where **NAME** = 'JOE'

CUSTOMER TABLE

SALES-PERSON-ID	CUSTOMER-NAME	ADDRESS
01	FANTASY CO	11 ELM
01	TOYBOX	23 CHERRY
02	SUITS INC	36 PINE

Step 2. Join of SALESPERSON and CUSTOMER TABLE where **SALESPERSON-ID** = '01'

LOGICAL VIEW produced by join

SALES-PERSON-ID	NAME	DEPT	CUSTOMER-NAME	ADDRESS
01	JOE	TOYS	FANTASY CO	11 ELM
01	JOE	TOYS	TOYBOX	23 CHERRY

Step 3. PROJECTION of column **CUSTOMER-NAME** lists answer to query.

Figure 9.
A join operation produces a logical tabular view of the data by recomposing the related physical tables into a single virtual table.

Figure 10.
The steps involved in processing a query.

A connection from an owner record to its member records is called a chain. Chains provide the logical paths the network data manipulation language uses to access related records. Member record types can be owners of other member types, and owner record types can be members of other owner types. Using currency pointers, the user accesses information by navigating the chains in link-sets.

In a relational system, data entities are represented with tables that have rows and columns. The columns for a table are the attributes of the data entity, and the rows are instances of the data entity. Relationships among tables are represented by a column common to those tables. The common column must have the same domain and same semantics in all related tables.

A relational query is simpler because it is non-procedural, set-based, and can be formed from a small group of verbs.

In a relational data manipulation language, projection selects columns, selection selects rows, and joining accesses related rows from multiple tables. Relationships among tables are only represented when the data base is initially defined and created. At run time, the join operation establishes the relationship and constructs a logical tabular view of the data by linking the physical tables that are related.

The term "record type" in a network system is equivalent to the term "table" in a relational system. A record type and a table both represent data entities. The terms "record" and "row" are also equivalent since they both represent instances of a data entity, and the terms "field" and "column" are equivalent because both represent attributes of the data entity. Therefore, the representation of data entities in both systems is basically the same. It is the representation of relationships among the data entities that varies significantly between the systems.

The ENCOMPASS Distributed Data Base System

The Tandem ENCOMPASS relational data base system consists of six products: the Data Definition Language (DDL), the ENFORM™ relational query language, the ENABLE™ program generator, the ENSCRIBE™ data base record manager, the Transaction Monitoring Facility (TMF), and the PATHWAY transaction processing system. This section briefly describes the role of each product.

The Data Definition Language (DDL) is primarily used to define tables and their columns. Each data base has an associated data dictionary that describes the tables and their columns in the data base. DDL is used to add, delete, or update data definitions in a data dictionary. After a table is defined, the GUARDIAN File Utility Program (FUP) is used to create the physical table. The table is physically implemented with a standard file-system file type that has records with fields. Together, DDL and FUP are used to create and maintain tables in the data base as defined by the logical and physical data models.

The ENFORM non-procedural relational query language can be used interactively or programmatically. Its interactive interface allows users to make ad hoc queries and to generate reports. Its host language interface allows programs to retrieve information from a data base using a relational access language.

The ENABLE program generator creates a simple interactive program that is based on screens and driven by function keys. It allows a user to peruse, add, delete, and update rows in a table.

The ENSCRIBE data base record manager provides high-level access to, and manipulation of, records in a data base. As an integral part of the GUARDIAN operating system, it helps ensure data integrity if a processor module, I/O channel, or disc drive fails. Some of its important features are relational access among files, record and file locking, multiple volume (partitioned) files, and mirrored discs.

The Transaction Monitoring Facility (TMF) protects the integrity and stability of information in a data base that can be distributed over many communications network nodes. If a transaction is interrupted and does not complete successfully, it may leave the data base in an inconsistent state. TMF removes the inconsistencies by restoring the data base to its original state before the execution of the transaction.

The PATHWAY transaction processing system allows users to develop fault-tolerant on-line applications in a requester-server design structure. Requester programs control terminals, screens, and the flow of transactions, while server programs perform I/O accesses to data bases. Requesters and servers communicate with each other by sending messages. This division of responsibility and functionality into independent processes makes a PATHWAY transaction processing application flexible, expandable, and easily tuneable, taking full advantage of the Tandem multi-processor hardware architecture.

Advantages of Relational Systems

The following discussion of the advantages of relational systems over network systems is based on the features and limitations inherent in their data models. Hybrid data base management systems providing some relational features for a network system are not considered, since their underlying network model does not efficiently support relational features.

Simplicity

Today, simplicity with full functionality is an important requirement of software products. In a relational system, the data model and data manipulation language are significantly simpler than in a network system. A relational system is easier to learn in less time, does not require the user to have extensive technical knowledge, allows the user to be more productive, and reduces overall costs.

Simpler Data Model. To use a relational system, the user need only understand the common concept of a table with rows and columns. It is not necessary to understand the underlying physical implementation of the data base.

On the other hand, the network data model is more complex. It involves the interconnection of owner and member record types to form link-sets. The user's view of the data is a complicated one, involving chains and access paths defined among the link-sets.

Simpler Data Manipulation Language. A relational data manipulation language is simpler than a network language because the relational language is non-procedural and set-based, and it has a small set of concise verbs to accomplish all operations. A network data manipulation language is procedural and record-oriented, and it has many verbs, all of which make it very difficult to use unless the user has expert knowledge of the language.

Earlier, for each language, an example answering the question, "For which customers is Joe responsible?" was given. In the following sections, the queries from the examples are analyzed in order to illustrate the properties of each language.

In the relational system, the example query to be analyzed is:

```
LIST      CUSTOMER-NAME
WHERE     NAME OF SALESPERSON TABLE = 'JOE'
  AND     SALESPERSON-ID OF
          SALESPERSON TABLE =
          SALESPERSON-ID OF
          CUSTOMER TABLE
```

This relational query is non-procedural because it does not specify the steps necessary to find the information; instead, it specifies the criteria the rows must satisfy, using projection, selection, and join operations of the WHERE clause. Also, it is set-based, in that all rows that satisfy the selection criteria are found, not just the first row.

Furthermore, the relational data manipulation language is easy to use because a small number of verbs can accomplish the data operations. The verbs READ (LIST), WRITE, UPDATE, and DELETE are sufficient for data manipulation; no additional verbs for chain manipulation are necessary. To access related data in multiple tables, the user simply uses the join operation. Built-in functions such as AVERAGE, SUM, MIN, and MAX, known as aggregate functions, are also available. These functions can operate on sets of rows to produce calculated results from the raw data stored in the tables.

The example network query to be analyzed is:

```
FIND SALESPERSON RECORD
   WHERE NAME = 'JOE'
FIND FIRST CUSTOMER RECORD
IN CURRENT SALESPERSON-CUSTOMERS
WHILE NOT FAIL DO
   PRINT CUSTOMER-NAME OF
   CUSTOMER RECORD
   FIND NEXT CUSTOMER RECORD
   IN CURRENT SALESPERSON-CUSTOMERS
END
```

In this query, the FIND, FIND FIRST, FIND NEXT, and WHILE statements illustrate the procedural aspects of the network language, since the user specifies the required steps to locate the information. Also, the WHILE statement illustrates the record orientation of the language, that which provides the loop control to process each record in a chain.

In addition to the standard READ, WRITE, UPDATE, and DELETE verbs, the network data manipulation language further complicates the language with several formats of the FIND verb and with the CONNECT, DISCONNECT, and RECONNECT verbs. Using the FIND verb to manipulate currency pointers, the user must navigate among the link-sets to access data records, since the network data model defines relationships with chains of logical access paths. Some formats of the FIND verb include FIND FIRST, FIND NEXT, FIND DUPLICATE, FIND OWNER, and FIND ANY. Besides navigation, the user may have to use the CONNECT, DISCONNECT, and RECONNECT verbs for the manual linking of records to, and unlinking of records from, chains.

As more relationships among record types are established, and the complexity of the network data base increases, the complexity of navigation increases also. To address this problem, some vendors of network systems provide a utility that displays a map of the data base link-sets on a terminal screen to indicate the location of various types of currency pointers. It serves as a "you-are-here" map like the ones found in large building complexes. Although this map is a very useful tool for navigating the data base, the navigation itself is an extra task for the user. In a relational data manipulation language, maps are not needed, since complex navigation problems do not exist.

Flexibility

Flexibility, another requirement of data base management systems, allows the data base designer to design a data base without knowing all the relationships and information retrieval requirements before implementation of the data base. As the application requirements change and evolve, so can the data base structure, with minimal impact on existing application software investments.

Relational systems can provide this flexibility because the rows in multiple related tables are not joined until run time, when a program or query accesses the data base. The relational system dynamically creates the logical view of the related data from the physical tables, using, among several alternatives, the most efficient access algorithm applicable, given the physical structures of the tables at the time of execution. For example, when the selection or join operation in a relational query involves a column whose physical structure is supported by a primary or alternate key index, the access algorithm accesses the rows via the index. If the query involves a non-indexed column, the access algorithm may invoke a sort-merge procedure. At the cost of some flexibility, some relational systems allow the user to invoke a pre-compiled query that has already determined the access strategy.

In a network system, this flexibility at run time is unavailable, because all relationships are established with link-sets when the data base is defined. Only those access paths provided by chains that have been

designed into the data base can exist. Relationships cannot be easily changed unless the data base is re-defined.

The following conceptual analogy, based on the elementary "connect-the-dots" game, illustrates a network system's establishment of relationships at definition time.

In Figure 11, each dot represents a table in the relational data base, and each connected pair of dots represents a link-set in the network data base. In the relational data base, the dots are not connected at definition time, while in the network data base, the dots must be connected for the data base to exist.

The run-time creation of the logical view of related data makes it possible for a relational system to be more flexible than a network system, allowing data-independence, dynamic relationships, and ad hoc query processing.

Data-Independence. In a relational system, not only is the logical tabular view of the data simpler, the physical level of the tables is separated from users and applications. Since users and applications interface with the data base via the relational data manipulation language, which is non-procedural and operates on logical tables, the users and applications are not dependent on the physical structures of the tables. This data-independence protects applications from changes at the physical level.

The data base administrator can restructure the physical level, made up of sequential, relative, and indexed file types, to improve performance or to change data definitions without affecting users or applications. For example, the data base administrator could add an alternate key index for a column in a table to enhance performance of searches based on key values of the column. To use the added index, no changes to any existing applications or queries would be required. The relational system would automatically use the alternate key index at run time, if it would improve performance. Thus, in relational systems, the physical implementation of a table is the responsibility of the data base administrator; the user simply uses the table.

In a network system, the separation of the logical and physical structure is not as distinct. The user's view of the pre-defined

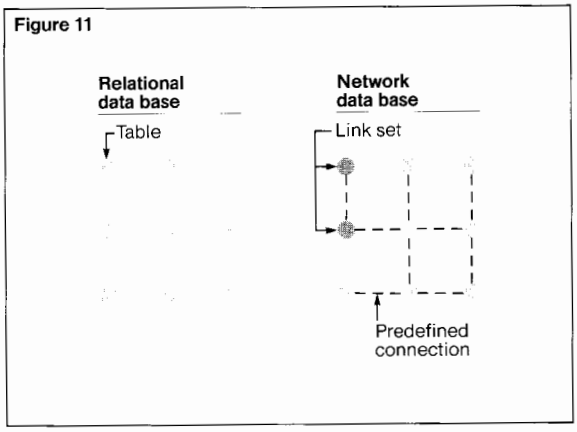


Figure 11.
Network systems link record types (or "connect the dots") when the data base is defined, prohibiting flexibility.

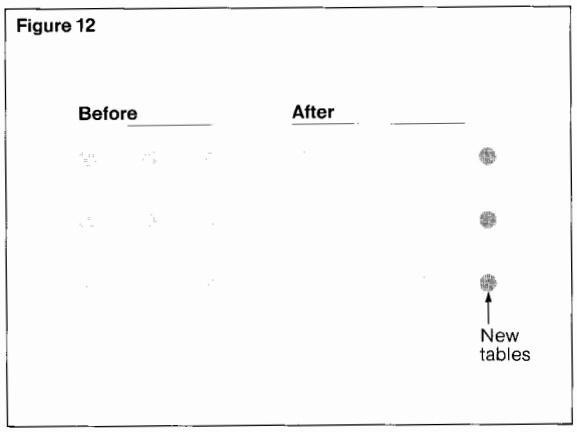
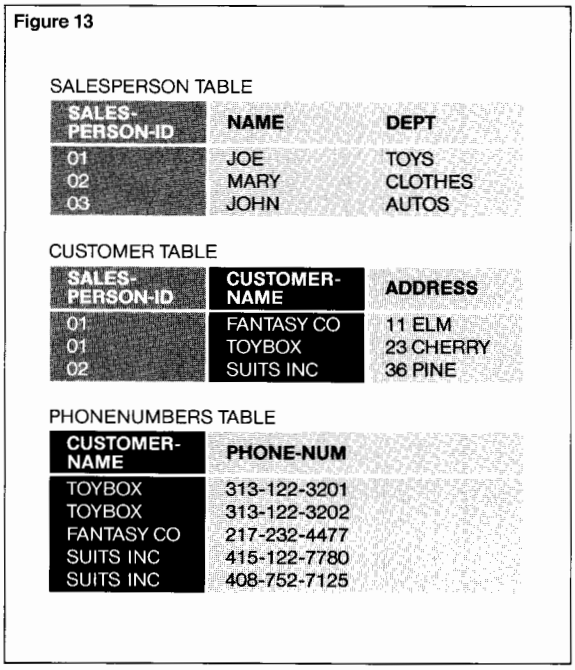


Figure 12.
In relational systems, new tables can be dynamically created to represent new relationships.

logical access paths in link-sets is dependent on the physical chains in the data base structure, and restructuring these chains changes the connections of the logical access paths. Because users and applications interact with the data base by using a procedural data manipulation language, they must navigate currency pointers among access paths, becoming dependent on the paths' existence. Thus, it is difficult to alter the structure of a network data base without affecting users and applications adversely.

Dynamic Relationships. In a relational system, because relationships are not established until run time, new relationships can be created or old ones destroyed dynamically. Information from different tables can be combined to create new tables to represent new relationships. (It is easy to add or remove dots from the picture.) This flexibility, illustrated in Figure 12, makes it easier for the data base structure to evolve as application requirements and the data model change.

Figure 13.
The new table PHONENUMBERS records multiple phone numbers for each customer. The common column CUSTOMER-NAME defines the one-to-many relationship between the CUSTOMER and PHONENUMBERS tables.



For example, in the SALESPERSON TABLE and CUSTOMER TABLE example presented earlier, a new table called PHONENUMBERS with columns CUSTOMER-NAME and PHONE-NUM could be added to represent customers with multiple phone numbers. This is illustrated in Figure 13.

Based on the new table, the following answers the query, "What are the phone numbers of the customers for which Joe is responsible?":

```
LIST      CUSTOMER-NAME OF
          CUSTOMER TABLE,
          PHONE-NUM OF PHONENUMBERS
          TABLE
WHERE     NAME OF SALESPERSON TABLE = 'JOE'
  AND     SALESPERSON-ID OF
          SALESPERSON TABLE =
          SALESPERSON-ID OF
          CUSTOMER TABLE
  AND     CUSTOMER-NAME OF
          CUSTOMER TABLE =
          CUSTOMER-NAME OF
          PHONENUMBERS TABLE
```

In most network systems (especially the older ones), adding a new relationship or destroying an old one would be extremely difficult since the record types and their connections are fixed at definition time.

To allow a customer to have a chain of phone numbers in the network data base, a PHONE-NUMBERS relationship, with a new link-set called CUSTOMER-PHONENUMBERS (where the owner record type would be CUSTOMER and the member record type would be PHONENUMBERS) would have to be added. This additional link-set would be impossible to add without re-defining the schema, restructuring the data base, and performing a data base unload/reload.

Ad Hoc Queries. Another benefit of establishing relationships at run time is that relational systems can process a wide variety of ad hoc queries, making the data base an even more valuable source of information. The flexibility inherent in relational systems allows the user to dynamically create the new relationships to satisfy ad hoc queries. Given a query at execution time, the relational system connects the dots (that is, it performs join operations among the tables) to establish the relationships necessary to satisfy the query. This is illustrated in Figure 14. This run-time flexibility is also advantageous to the data base designer, since the designer need not determine all queries to be executed before implementing the physical data base.

On the other hand, ad hoc queries can be very difficult to process on network systems, as new relationships providing new access paths cannot be formed after the data base is defined. In fact, it may not be possible to satisfy certain classes of queries at all, simply because the paths were not pre-defined.

System Optimization of Queries

A relational data manipulation language is non-procedural, in that the user specifies only the criteria for isolating the data to be manipulated, not the data manipulation steps to be performed. This allows the relational system to determine how to optimize the query process for the user. Since a network data manipulation language is procedural, the user has the responsibility of determining how to retrieve the data most efficiently.

Query optimization in relational systems involves reformulation of the user's query for better run-time performance. Relational data manipulation languages are based on theoretical relational calculus, a field of mathematics. Using the laws of relational calculus such as the commutative and associative properties, the query optimizer can re-arrange, add, and delete projections, selections, and joins to minimize:

- Unnecessary operations.
- Workspace sizes for intermediate results.
- The number of rows to be joined.

Generally, query optimization involves (1) performing selections as soon as possible in order to reduce the number of rows, and (2) combining selections and projections in the same operation. The latter reduces the length of the selected rows by eliminating all columns not required to satisfy the query.

System optimization offers two benefits over programmer optimization. First, the programmer need not be overly concerned with optimization and the development of ultra-efficient data base access algorithms. Second, as new optimization techniques are developed, they can be integrated into the existing relational system. An existing application automatically benefits from new optimization features, and thus, is not tied to the current state of technology.

Suitability for Distributed Data Bases

A relational system is better suited for the implementation of distributed data bases since there are no physical dependencies among tables. A relationship among tables is represented with the presence of a column that is common to the tables, allowing the tables to be distributed to remote nodes with the relationship intact. The related data from multiple tables can still be accessed, regardless of the geographic location of the tables.

Figure 14

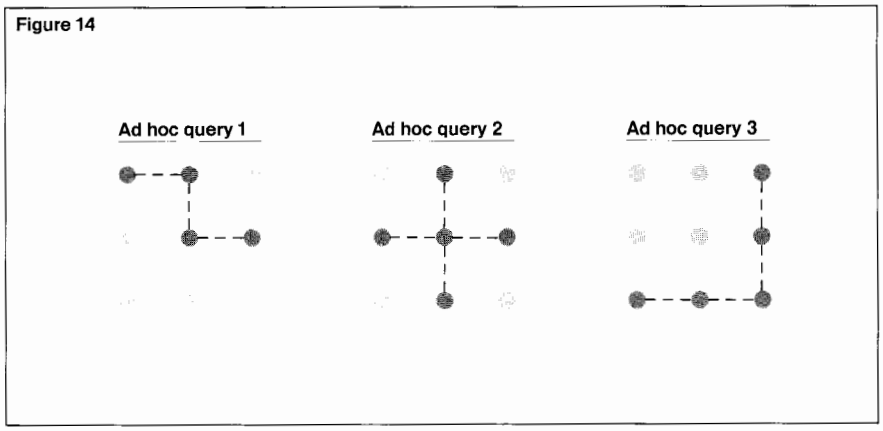


Figure 14.
In relational systems, relationships among tables are established at run time, as required, allowing ad hoc queries.

In a network system, record types are not independent from one another at the physical level. Owner record types are connected to member record types to produce a data base with intertwined access paths. This dependency makes it very difficult for network systems to implement distributed data bases efficiently because the connections are based on physical descriptions of record pointers. When physical descriptions span communications network nodes, distributed record types become bound to their locations and cannot easily be moved. For example, if record type A on node 1 is connected to record type B on node 2 by a physical description, the movement of record type B would invalidate record type A's physical description of the connection. B would no longer be located where A thinks it is.

In relational systems, the ability to place a table, whole or partitioned, on any disc attached to the same or remote node offers many performance advantages and cost savings. By spreading one or more tables and their alternate key files over many disc devices, one can increase concurrent input/output activity. While one disc is servicing a request, another disc is available for access. Moreover, since tables are geographically independent, they can be placed on the node closest to the users who use them most often. This proximity can eliminate enormous amounts of data transmission, reducing communications costs and response-time delays.

Conclusion

Network data base management systems were popular when technology dictated that efficiency and low-level resource control be of primary importance. Today, requirements have changed; user-friendliness and maximal productivity are also important. Relational systems, the latest generation of data base management systems, offer the following advantages:

- Simpler data model and data manipulation language.
- Flexibility to restructure the physical level for data-independence.
- Dynamic creation of tables and relationships.
- Easier ad hoc query processing.
- System optimization of queries.
- Easier implementation of distributed data bases.

Finally, when relational systems are used, many benefits are realized during the life cycle of an application. During the design phase, the data base definition does not have to be complete, with all paths pre-defined, as with network systems. Application requirements typically change, and the flexibility inherent in relational systems ensures easy initial data base design and ongoing growth.

In the implementation phase, that part of the data base which has already been designed can be generated immediately. Data entry into these tables and verification of the tables can proceed while the remaining portion of the data base is being designed and generated. Also, with performance-tuning parameters involving file structures, alternate keys, locking strategy, and file placement, application performance can be tuned, based on execution statistics. Significant tuning is possible, since the logical structure is clearly separated from the physical structure. This separation allows the data base administrator to tune the physical level without affecting applications that access the data base as logical tabular views.

In conclusion, a relational system is advantageous when:

- The application is not completely defined or will change over time.
- The data base is dynamic and growing.
- New, nonstandard or ad hoc reports are needed.

Today, many applications have the above properties. These applications are well suited for implementation on a relational system such as the Tandem ENCOMPASS distributed data base system.

Acknowledgements

Eileen Chan, Eric Chow, Nick Franks, Mike King, Bob Sawyer, Hal Voege, Gil Wai, and Rob Welsh provided valuable early review comments. Jim Gray, Jim Morrow, John Nauman, and Ken Schmidt provided technical expertise and guidance for later revisions. The author wishes to give his sincere thanks to all.

References

Brodie, M., and Schmidt, J. July 1982. Final Report of the ANSI/X3/SPARC Relational Data Base Task Group. *ACM SIGMOD*, vol. 12, no. 4.

CODASLYL Data Description Language Committee. 1978. *DDL Journal of Development.*

Data Base Task Group of CODASYL Programming Language Committee. April 1971. Report.

Date, C.J. 1981. *An Introduction to Data Base Systems.* 3rd ed. Addison-Wesley Publishing Company.

Digital Equipment Corporation. May 1982. *Software Product Description VAX-11 DBMS.* version 1.1.

Draffin, I.W., and Poole, F. 1980. *Distributed Data Bases: An Advanced Course.* Cambridge University Press.

Engles, R.W. November 1978. Description of the COBOL Data Base Facility. Proc. GUIDE 47.

Gray, J. March 1981. *An Approach to End-User Application Design.* Tandem Computers Incorporated.

Oracle Corporation. 1979. *Oracle Introduction.* version 1.5. Relational Technology Incorporated. 1982. *INGRES Reference Manual.*

Schmidt, K., Morrow, J., and Madsen, K. August 1983. *Application Data Base Design in a Tandem Environment.* Tandem Application Monograph Series. Tandem Computers Incorporated.

Schuster, S. February 1981. *Relational Data Base Management For On-line Transaction Processing.* Tandem Computers Incorporated.

Ullman, J.D. 1980. *Principles of Data Base Systems.* Computer Science Press, Inc.

Wiederhold, G. 1977. *Data Base Design.* McGraw-Hill.

Gary Ow is a senior systems analyst in Tandem's San Francisco sales office. He joined Tandem in April, 1983. Before joining Tandem, Gary developed languages and compilers for another major mainframe vendor. He also designed relational and network data base application systems for the petroleum industry. He has a Masters Degree in Computer Science Engineering from Stanford University and a Bachelors Degree in Computer Science from San Francisco State University.

TANDEM

NonStop™ Computer Systems

Part No. 83931

400091 01/84 Printed in USA