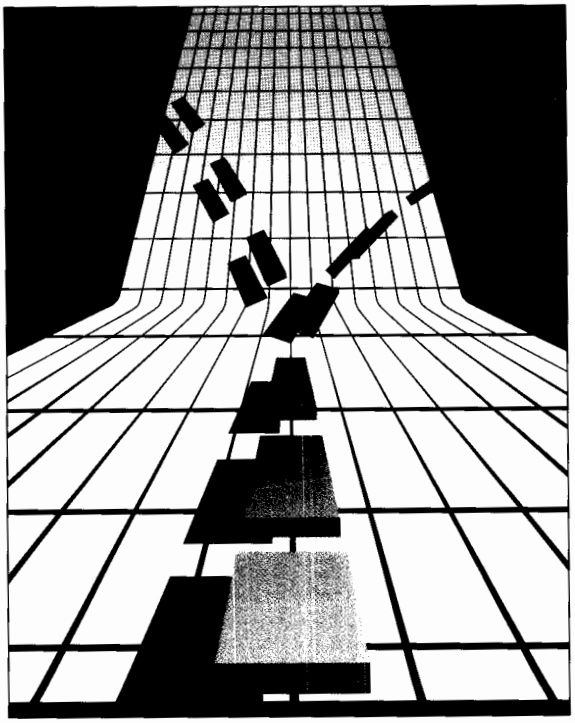
T A N D E M

SYSTEMS REVIEW



TNS/R Overview

Improving TNS/R System Performance

Debugging on TNS/R Systems

Event Management Performance

Product Update • Ongoing Support

T A N D E M

SYSTEMS REVIEW

SPRING 1992 • VOLUME 8, NUMBER 1

TANDEM

Tandem Systems Review is published quarterly by Tandem Computers Incorporated. All correspondence and subscriptions should be addressed to *Tandem Systems Review*, 18922 Forge Drive, Loc 216-05, Cupertino, CA 95014.

Subscriptions: $75.00 per year; single copies: $20.00. Detailed subscription information is provided on the subscription order form at the end of this book.

Editor's Note

Tandem has introduced RISC technology into its Cyclone and CLX NonStop computer families. With TNS/R systems, Tandem users benefit from the power and economy of RISC technology. These systems provide transaction processing power comparable with that of systems that require special facilities and staffs. Yet they are compact, do not require computer rooms, and can be installed and maintained by users. These factors greatly improve the economics of large-scale online transaction processing.

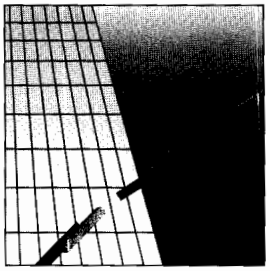
This issue includes three articles on TNS/R systems: "Overview of Tandem NonStop Series/RISC Systems" by Faby and Mateosian, "Improving Performance on TNS/R Systems With the Accelerator" by Blanchet, and "Debugging Accelerated Programs on TNS/R Systems" by Cressler.

The feature article entitled "Measuring DSM Event Management Performance," by Stockton, discusses performance issues related to event management in the Tandem DSM environment.

This issue introduces two new departments that will appear regularly in the *Tandem Systems Review*. "Product Update" includes brief descriptions of the Tandem products and enhancements that have been announced in the last three months.

The "Ongoing Support" department describes a support service available from Tandem. This issue discusses Professional Services, a group of consulting packages provided by Tandem analysts.

This issue also includes an index of *Tandem System Review* articles. The index is a list of all articles by subject and product. — *SWT*

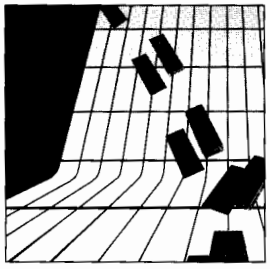


TNS/R

8 Overview of Tandem NonStop Series/RISC Systems
Les Faby, Richard Mateosian

16 Improving Performance on TNS/R Systems With the Accelerator
Manon Blanchet

28 Debugging Accelerated Programs on TNS/R Systems
Diane Cressler



FEATURE

42 Measuring DSM Event Management Performance
Mark Stockton

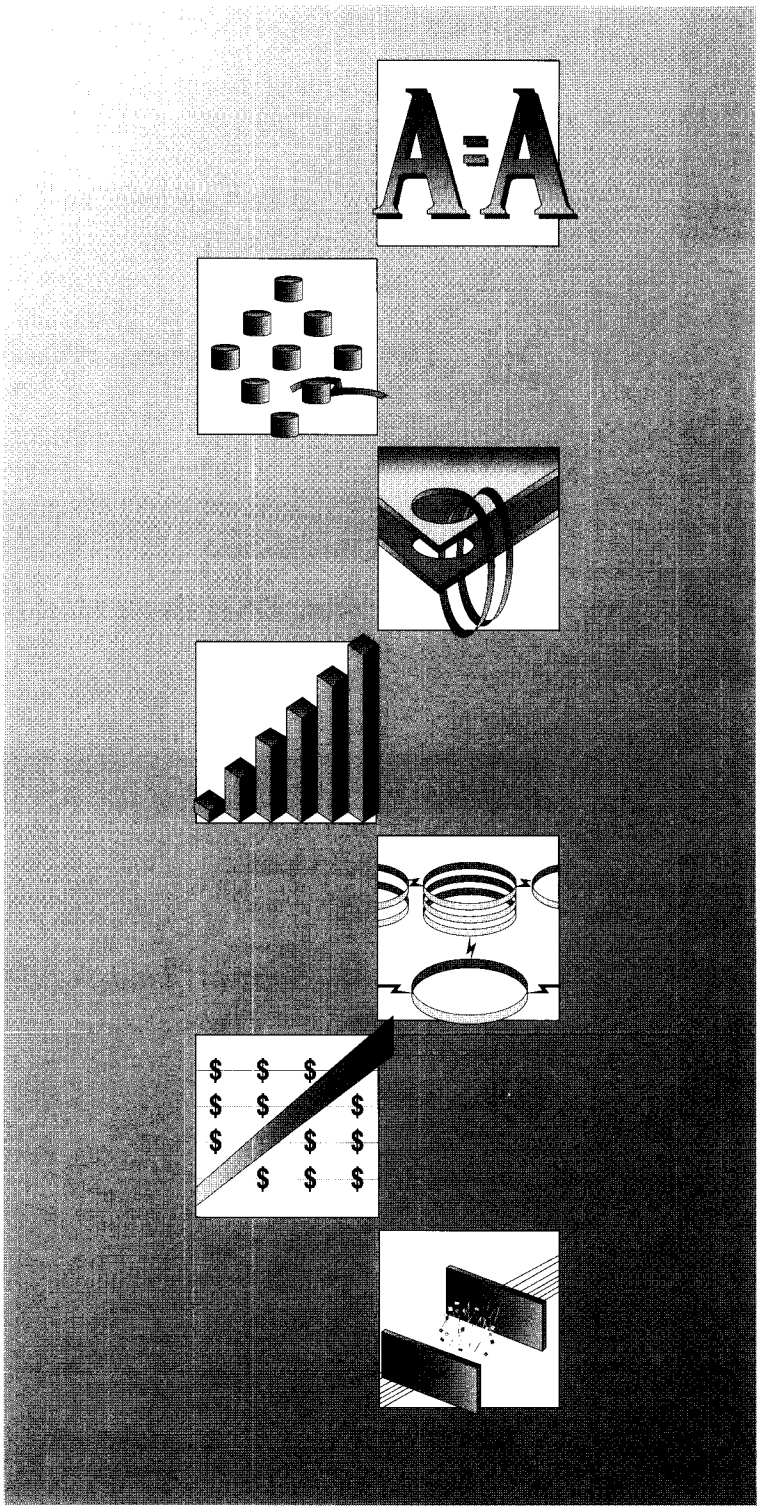
DEPARTMENTS

2 Product Update
Donna Carnes

60 Ongoing Support: *Tandem Professional Services*
Mary Ann Whiteman

63 Index of Articles

Product Update



Systems Products

NonStop Cyclone/R
October 1991

The NonStop Cyclone/R is an entry-level Cyclone that provides users with 50 percent of the Cyclone transaction power at an economical price. The Cyclone/R makes large OLTP applications available to more users by combining RISC technology with the Guardian 90 operating system and the NonStop system architecture. The Cyclone/R is the first high-performance OLTP system that does not require a computer room.

The Cyclone/R provides object-code compatibility for existing NonStop system applications; it runs the same object code as all other Tandem NonStop systems. Thus, existing user applications can run on the Cyclone/R without any reprogramming.

NonStop CLX/R
October 1991

The NonStop CLX/R is an entry-level system that supports a full-function, highly distributed platform for OLTP applications. The CLX/R uses RISC technology in a compact system with full Guardian 90 operating system compatibility. The CLX/R provides object-code compatibility for existing NonStop system applications; it runs the same object code as all other Tandem NonStop systems.

Integrity

Integrity Family: New Products and Unix Systems Enhancements
October 1991

The Integrity product line offers users two new systems: Model 300 and Model 100E. Model 300 performs up to 100 percent faster than its predecessor. The increased performance is attributable to the faster RISC microprocessor, faster access to memory, additional hardware enhancements, and the new software technology in the NonStop-UX operating system, Release 1.2. Existing Integrity systems can be upgraded on-site to Model 300 by replacing the existing CPU modules with the three new CPU modules.

Integrity system Model 100E is a new entry-level system. It has CPU performance equivalent to the Model 200, but more limited configurability.

Two new subsystems for the Integrity systems family include the Reliable Ethernet subsystem and the Four Port Synchronous Communications Controller. The Reliable Ethernet provides LAN environments with the Integrity architecture's reliability and availability, which means that no single hardware failure can prevent access to the LAN. The Four Port Synchronous Communications Controller provides bit-synchronous port for use with SNA or X.25 communications.

Storage Products

4500 Disk Subsystem
October 1991

The new 4500 disk subsystem provides high-performance, high-capacity external disk storage for the NonStop Cyclone, Cyclone/R, and CLX 800 systems. The 4500 delivers higher capacity than the XL80 storage facility and higher performance than the V80. It is packaged in the compact Modular Storage System, which provides floorspace utilization, location flexibility, reliability, and modularity.

The 4500 offers 37.7 GB of formatted capacity in a footprint of 6.3 square feet. It can be connected to any NonStop Cyclone, Cyclone/R, or CLX 800 system through the Tandem 3128 disk controller. Disk subsystems can be located as far as 1,640 feet away from the host. Individual drives can be serviced without shutting down other drives or the rest of the system.

4240 Disk Drive
October 1991

The new 4240 disk drive provides high-performance, high-capacity internal disk storage for the NonStop Cyclone, Cyclone/R, and CLX systems. The 4240 has a formatted capacity of 1.038 GB and is contained in a standard customer-replaceable unit (CRU).

The 4240 offers the highest capacity of any of the Cyclone/R, CLX, and CLX/R internal disk storage devices. Using the 4240 disk storage device, a fully configured 16-processor Cyclone/R system (with expansion cabinets and a total of 96 drives) can provide up to 99.6 GB of formatted internal storage.

5175 Tape Drive
October 1991

The 5175 is a compact, dual-density, streaming tape subsystem for NonStop Cyclone/R, CLX 800, and CLX/R systems that is designed to fit in Tandem's Modular Storage System. The 5175 features automatic tape threading and tensioning for error-free loading, automatic power-up diagnostics, adjustment-free operation, and user-performed preventative maintenance. It stores up to 180 MB (unformatted) of data in a 2,400-foot tape reel.

5180 Cartridge Tape Subsystem on NonStop Cyclone/R and CLX 800 Systems
October 1991

The 5180 tape subsystem is a highly reliable streaming cartridge tape device for NonStop Cyclone/R and CLX 800 systems. It increases operator productivity and provides automatic tape handling, high performance, and configuration flexibility.

The 5180 is fully compatible with the IBM 3480 and the recently announced 18-track 3490 cartridge tape subsystems. The 5180 enables users of the NonStop Cyclone/R and CLX 800 systems to exchange data with IBM and IBM/PC systems that have converted their tape processing operations to the 3480 cartridge format. Additional applications include large database support, TMF, and any operator-intensive application that could benefit from the 5180 automatic tape-loading feature.

5200 Optical Storage Facility (5200 OSF) on NonStop CLX 800 and Cyclone Systems
October 1991

The 5200 OSF brings the functionality and benefits of optical storage to the NonStop CLX 800 and Cyclone platforms. The 5200 OSF provides CLX 800 and Cyclone users with online, large-capacity storage, archival, and image processing features. It accesses data faster than tape or microfiche and reduces data archive management costs.

Users can connect the 5200 OSF to Cyclone processors to support the highest volume requirements for large data-archiving applications. When connected to a NonStop CLX 800 system, the 5200 OSF can manage applications for storing graphics, high-resolution image data, textual data, and facsimile-scanned documents. The new connectivity capability of the 5300 OSF enables existing 5200 OSF users to migrate their storage applications to the Cyclone and NonStop CLX 80 platforms.

Guardian 90 Based Software

EMS Analyzer
January 1992

The EMS Analyzer provides analysis and reports on Event Management Service (EMS) log files. It provides definable search criteria that lets users select particular EMS events from the logs. The selected events are routed to a designated terminal, spooler, database, or CSV file. If events are directed to a database or CSV file, users can generate graphics and special reports by using tools such as Enform or Microsoft EXCEL on a PC or Macintosh.

Object Monitoring Facility (OMF)
January 1992

The OMF operations application allows operators to set up monitoring parameters for key system objects. OMF provides information such as critical events by object type, device and subsystem availability, system availability, and application availability.

OMF monitors objects in a Tandem system or network and reports informative, critical, or abnormal events to the local EMS collector. Informative or critical events are displayed on the OMF general status and detail screens. Events can also be written to the Viewpoint console or made available to management applications such as Programmatic Network Administrator (PNA).

Workstation and Terminal Products (TSC)

PSX EP386SX/20 Personal Computer
December 1991

The PSX EP386SX/20 is an entry-level workstation for personal computer LANs. It can be used for almost any horizontal application (such as call center, EDI, and image capture and retrieval) that requires economical LAN workstations or a communication gateway.

The PSX EP386SX/20 is available in two models: Model 43 and Model 0. Model 43 includes 2 MB of memory, a 3.5-inch, 1.44-MB floppy drive, a 40-MB hard drive, mouse, and Windows 3.0. Model 0 does not provide disk drives or installed memory. Both models feature a built-in VGA video adapter, two serial ports, one parallel port, and a mouse port. PSX workstation options include Net/One Ethernet, X.25/SNA, and specialized products.

PSX CP486SX/20 Personal Computer
October 1991

The PSX CP486SX/20 is a high-performance workstation for personal computer LANs. It offers the performance of a 33-MHz 80386 processor and features expandability, flexibility, and an upgradable processor.

The PSX CP486SX/20 is available in two models: Model 83 and Model 3. Model 83 includes a 4-MB hard drive, 80-MB hard drive, mouse, and Windows 3.0. The basic Model 3 is user-configurable; it does not include installed memory. Both models feature a 3.5-inch, one-third-height floppy drive and a built-in VGA video adapter. Users can add up to 80 MB of memory and can upgrade the processor by either adding an 80387 math coprocessor or installing a faster processor card.

PC6530 Release G31 and the TELNET Driver Option

November 1991

PC6530 G31 terminal emulation software and the TELNET Driver Option let users directly connect a PSX workstation on an Access/One or Ungermann-Bass LAN to a Guardian 90 host through TCP/IP. Using PC6530 G31 and the TELNET Driver Option, a workstation can function in true 6530 block and conversational mode and run multiple 6530 sessions. Updating PC6530 to the G31 release does not require purchasing updates for the alternate input device or X.25 connectivity options.

AST Premium Exec Notebook Computer

December 1991

The AST Premium Exec Notebook computer, now available through Tandem, provides desktop PC power and storage capacity in a compact and portable package. The Notebook computer weighs less than 7 pounds, fits inside a briefcase, and has a rechargeable NiCad battery pack that provides power for up to 3 hours.

Standard features include a 20-MHz 803865x processor, 2 MB of RAM, 3.5-inch floppy drive, an internal 60-MB hard disk, and an integrated 640x480 VGA screen that provides 32 shades of gray. Options include a 4-MB memory SIMM, a spare battery backup, and a battery charger. An optional, internal 2400-baud modem is available in the U.S.

Image Storage Server (ISS)

October 1991

Tandem's LAN network-based ISS provides high-speed, secure access to large image databases. The ISS manages images for maximum efficiency in Guardian 90 based Tandem applications; it can readily store and manage as much as a terabyte of information. Frequently accessed images are stored in a large RAM and magnetic memory cache for immediate LAN availability; less frequently accessed images are kept in optical storage. Both erasable and WORM optical media are available.

Communications, applications, and workstations can access and command the ISS by using File Transfer Protocol (FTP) on an Ethernet network. ISS is available in four expandable models, which offer different storage capacities.

Security and POS Products (Atalla)

NDX UNIX System Servers
October 1991

Tandem NDX UNIX system servers combine the power of SCO UNIX System V with the high performance and reliability of the NDX ST486/33 EISA computer. Each NDX UNIX system server provides up to 80 MB of RAM and 2 GB of storage and can support up to 64 personal computers on a TCP/IP LAN or 128 asynchronous devices.

An optional X.25 facility supports up to 128 communications channels, which allows the NDX UNIX system server to conduct multiple sessions with Tandem's Guardian 90 based NonStop systems, Tandem's UNIX-based Integrity systems, or other hosts in remote locations.

Cryptographic Security Manager (CSM)
October 1991

The Tandem/Atalla CSM is a Guardian 90 based server that enables application programmers to easily incorporate cryptographic data security functions into their applications without requiring detailed cryptographic knowledge. CSM supports services such as Wholesale Banking Server, Authentication Server, and SNAX/CDF Cryptographic Server.

CSM substantially reduces the time required to add cryptographic protection to Guardian 90 applications andis the foundation of an integrated end-to-end security architecture thatprotects network high-exposure transactions from compromise. CSM issupplemented by optional servers that provide specific higher-level functions such as ANSI and ISO cryptographic and key management standards.

Product Programs

CD Read
October 1991

Tandem's CD Read provides a complete set of Guardian 90 operating system software manuals on a single CD-ROM disc. CD Read provides a menu-driven graphical user interface as well as powerful commands that make it easy to locate information. For example, users can perform keyword searches within graphics as well as text. CD Read can be connected to LANs and works with a variety of workstations, including IBM PCs and Macintosh computers.

Overview of Tandem NonStop Series/RISC Systems

Tandem™ NonStop™ computer systems and the Tandem Guardian 90™ operating system have long provided excellent solutions for a wide range of applications. Tandem has now introduced Tandem NonStop Series/RISC (TNS/R) systems into its Cyclone™ and CLX™ NonStop computer families. TNS/R systems maintain complete compatibility with the Guardian 90 operating system and with users' existing application software.

TNS/R systems vary in expandability, connectivity, and transaction processing power, but all use a common central processing unit based on the approach to computer design called reduced instruction set computing (RISC). TNS/R systems allow Tandem users to benefit from the power and economy of RISC technology. They are compact, do not require computer rooms, and can be installed and maintained by users. Nonetheless, they provide transaction processing power comparable with that of systems that require special facilities and staffs. These factors greatly improve the economics of large-scale online transaction processing (OLTP) and extend downward the range of applications for which OLTP is economically feasible.

TNS/R systems are based on the same parallel processing architecture as other NonStop systems. For Tandem CLX systems, users can transform their existing equipment into TNS/R systems simply by replacing the processor and memory boards. This capability protects users' investment in their existing hardware.

This article describes the approach Tandem used to produce RISC-based NonStop systems. It includes a summary of the fundamentals of RISC technology and an explanation of how this technology benefits users of Tandem NonStop systems.

The Technical Challenge

In planning for its TNS/R products, Tandem faced a classic decision: whether to design from scratch or use commercially available components. Tandem chose to use a commercially available microprocessor that was designed using RISC technology. This decision allowed Tandem to provide the new computer designs quickly and at an economical price, while still allowing users to benefit from the many millions of development dollars that have been and continue to be spent by the microprocessor manufacturer. Because Tandem chose a microprocessor with a well defined future growth path, Tandem's design engineers can look forward to working with increasingly more powerful and economical components within an architecturally stable framework.

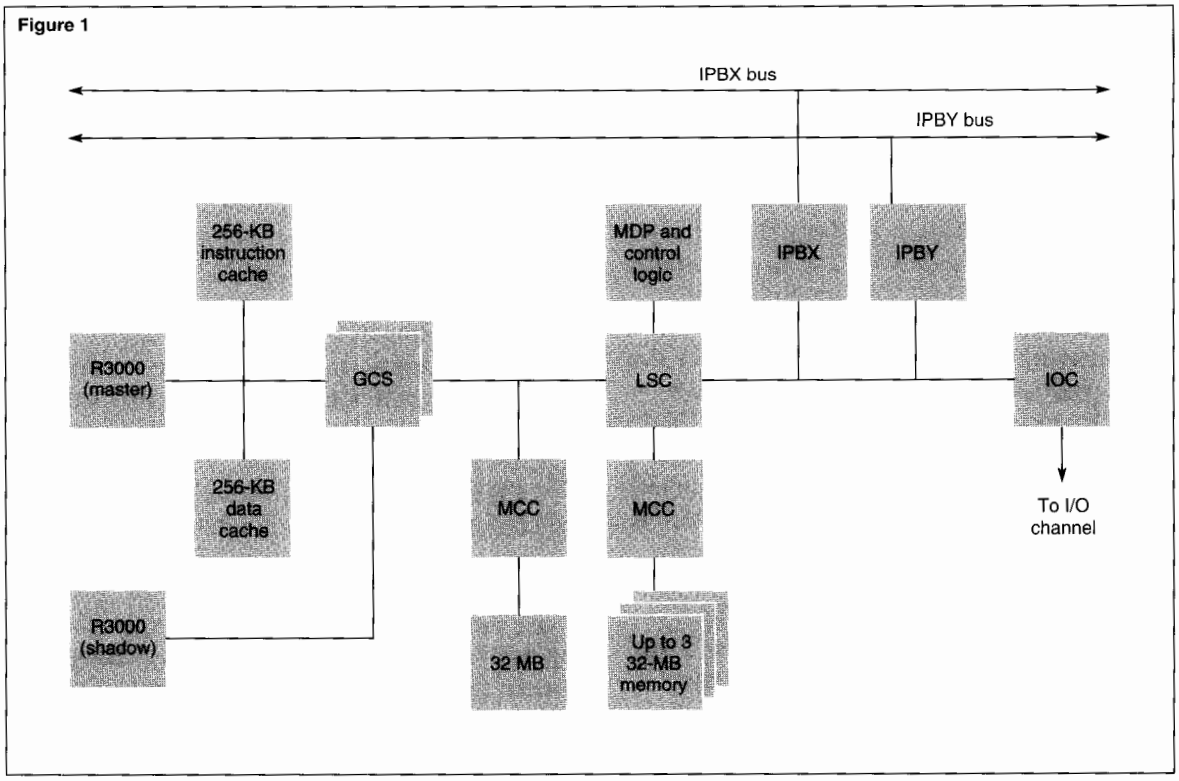
The use of a commercially available microprocessor presented Tandem with a difficult technical problem: how to maintain the necessary compatibility with existing applications. Tandem's solution allows users to move their existing applications to TNS/R systems without change and to use their current programming languages, operating system, and debugging tools to develop new applications for the new systems.

Before the introduction of TNS/R systems, all Tandem NonStop Series (TNS) computers used a common instruction set, called the TNS instruction set. The TNS instruction set is based on an approach to computer design called complex instruction set computing (CISC). The TNS/R systems have their own, entirely different instruction set based on the RISC approach. This article uses the term TNS to refer to Tandem's architecture, languages, and computers based on the CISC approach. Tandem developed technology to allow RISC systems to execute the CISC instruction set. This means that TNS/R systems can execute existing TNS object files.

Running unmodified TNS object files on TNS/R systems achieves only part of the performance improvement possible with the new technology. To allow users to achieve even greater performance improvement, Tandem has developed its Accelerator software product. The Accelerator processes TNS object files and produces accelerated object files that use the TNS/R instruction set. Tandem has applied this process to all of the performance-critical parts of the Guardian 90 operating system and the associated system calls. Users can optionally accelerate their application object modules for still more performance improvement.

Tandem ensured that TNS/R systems can function in networks of Tandem NonStop systems by designing a special object file format that contains both accelerated (TNS/R) and nonaccelerated (TNS) object code. This presents users with a much simpler management task than requiring two separate object files for each program. Existing TNS systems can execute programs that have been accelerated for TNS/R systems simply by ignoring the TNS/R portion of the object code.

Figure 1.
Block diagram of TNS/R CPU board.



Overview of the TNS/R Hardware

All TNS/R systems are based on the same basic CPU, called a NonStop System RISC Model L (NSR-L) processor. TNS/R systems use from 2 to 16 of these processors. Figure 1 is a diagram of the NSR-L processor. The processor contains two Mips R3000 RISC microprocessors and separate 32-bit-wide instruction and data caches[1]. The microprocessor can access both a 32-bit instruction and a 32-bit data item on every cycle. The basic 32 megabytes of main memory can optionally be expanded to 128 megabytes. Main memory is connected to the CPU by memory control chips (MCCs), which perform the memory refresh function, bank selection, and interleaving. They also perform error detection and correction. Tandem designed special circuitry called the gateway chip set (GCS) to achieve fault tolerance with commodity RISC microprocessors. Each NSR-L processor has two RISC microprocessors, the master and the shadow. Each time the master R3000 microprocessor performs an operation, the GCS causes the shadow R3000 to perform the same operation with the same operands one cycle later. The GCS then compares the results of the two operations and halts the NSR-L processor if the results are not the same. By allowing the shadow microprocessor to do the operation after the master microprocessor, Tandem can run the NSR-L microprocessors at full speed.

[1] A cache is a relatively small, fast memory in which frequently accessed portions of the relatively larger, slower main memory are kept temporarily. The circuitry accompanying a cache decides which portions of the main memory are to be kept in the cache at any time. The same circuitry recognizes the memory access requests that can be satisfied from cache and intercepts them.

The gateway chip set also implements the memory access breakpoint (MAB) register, the hardware timers, and the interrupt registers (INTA and INTB). These registers are needed by TNS applications, but they have no counterparts in the R3000 microprocessor. The GCS also provides the interface between the R3000 chips and the rest of the processor module, including memory.

Tandem made TNS/R systems compatible with the controllers and peripherals of the CLX and Cyclone computer systems. This helps CLX users preserve their investments when upgrading to TNS/R systems. Many of the support chips on the NSR-L processor board are the same as on CLX CPUs. The I/O controller (IOC) and interprocessor bus chips (IPBX and IPBY) are the same. The maintenance diagnostic processor (MDP) is similar.

The logic sequencing chip (LSC) is a new circuit that Tandem designed to serve as an interface between the 16-bit I/O bus of Tandem NonStop systems and the 32-bit I/O bus of the R3000 microprocessor. The LSC also connects expansion memory and the MDP to the gateway chip set.

Overview of the TNS/R Software

Nearly all TNS programs, even most privileged programs, can run without change on a TNS/R system. The Guardian 90 operating system, version C30.06 and beyond, runs on both TNS and TNS/R systems. Users developing new software for TNS/R systems can write and debug TNS programs exactly as on TNS systems. The user data stack and the extended data stack are bit-for-bit the same as on TNS systems. At the beginning of each high-level language source statement, the user data memory is exactly the same as on TNS systems.

A very small number of application programs will not run unchanged on TNS/R systems. For example, if the program explicitly modifies trap return addresses, it may have to be changed. (See *Programmer's Guide for TNS/R Systems*, 1991.) During the extensive beta testing period for the TNS/R systems, only one TNS application program required changes to enable it to run on a TNS/R system.

Although most privileged TNS programs can run directly on TNS/R systems, a few might need modification. In general, Tandem discourages users from writing privileged code for any of its systems. Users who must do so or who have trouble moving privileged programs to TNS/R systems should consult a Tandem analyst for assistance.

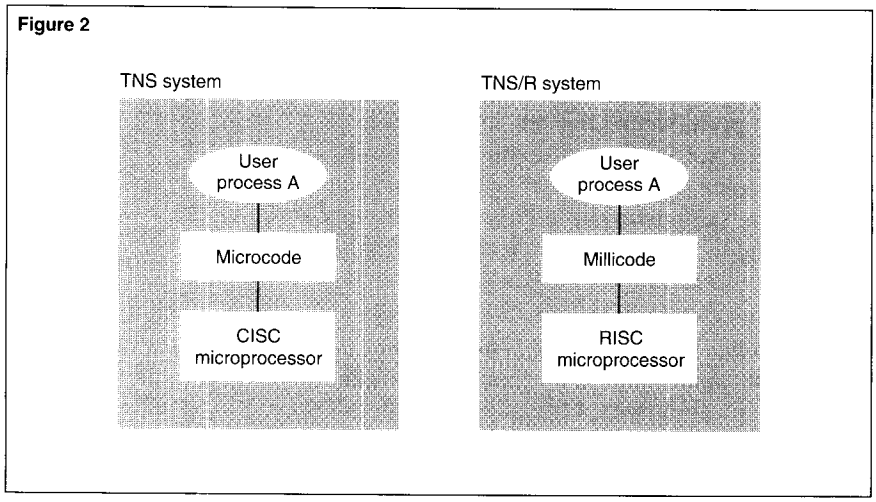


Figure 2.
Execution of TNS CISC instructions on TNS and TNS/R systems.

Implementing TNS Instructions on TNS/R Systems

Figure 2 shows how Tandem has implemented the TNS instruction set on TNS/R systems. The left side of the diagram shows how TNS systems execute TNS instructions. Programmers write programs (for example, an application requester) that use TNS instructions. These are the instructions generated by the TNS compilers that users are familiar with. As the TNS system receives each instruction, it directs its internal circuitry to execute a corresponding sequence of steps. The programs describing these steps are called *microcode*. They reside in a memory built into the CPU of the TNS system.

The right side of Figure 2 shows how a TNS/R system executes the same TNS instructions. As the TNS/R CPU receives each TNS instruction, it directs the RISC microprocessor to execute a corresponding set of TNS/R instructions. Tandem calls these instructions *millicode*. Tandem chose the term millicode to emphasize the similarity between the way TNS/R systems handle TNS instructions and the way TNS systems do. The term also reflects the difference between the two approaches.

Microcode on TNS systems controls movement of data along hardware paths and into internal registers. It mobilizes the functional units of the CPU. These paths, registers, and functional units are internal to the CPU and most of them cannot be addressed directly by programmers. Furthermore, each microcode instruction directs as many operations as the CPU can carry out simultaneously. Millicode on TNS/R systems consists entirely of manipulations of the programming model of the RISC microprocessor. That is, it consists of programs written in the TNS/R instruction set. The RISC microprocessor executes TNS/R instructions in approximately the same way the TNS system executes microcode.

The Accelerator

Millicode on TNS/R systems allows users to move TNS programs directly to TNS/R systems. This simple approach achieves some of the benefits of moving to the RISC microprocessor without any of the costs of adapting the application to the new computing environment. TNS/R users can achieve even greater performance gains by adapting their existing applications to the RISC environment.

Tandem developed its Accelerator product to adapt TNS programs to the TNS/R environment. The Accelerator is an optimizing compiler that accepts TNS object format as its source language and generates RISC instructions to perform the same task. For example, a TNS instruction that places a 5 into a register also indicates that the condition code is positive. The Accelerator can look ahead in the program and check if the program tests that condition code. If it does not, the Accelerator eliminates the extra RISC instructions that would perform condition-code setting.

In general, the Accelerator generates only those instructions required to execute the program correctly. It also performs other optimizations that reduce accesses to memory and unnecessary recalculations.

The Accelerator runs on any TNS or TNS/R system. It reads a TNS object file and produces a new object file that contains both the original TNS object program and an equivalent TNS/R program, which executes more efficiently than millicode. The same object file might be executed sometimes by a TNS/R system and at other times by a TNS system. The TNS system simply ignores TNS/R code. The TNS/R system uses the TNS/R code to execute the function more effectively.

Because the Accelerator produces a new object file, acceleration is a one-time process. Tandem has already applied this process to the performance-critical portions of the Guardian 90 operating system and the programs that implement its user services. Figure 3 shows how both nonaccelerated and accelerated user processes can execute in the TNS/R environment.

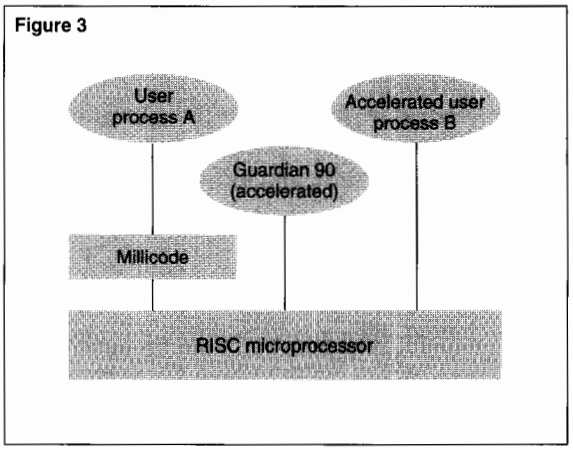


Figure 3.
TNS/R execution environment showing both nonaccelerated and accelerated user processes and the Guardian 90 operating system.

Typical applications, which spend most of their time executing system code or waiting for I/O devices to perform their functions, generally do not need to be accelerated. Acceleration has a few costs, principally the time required to run the Accelerator and the increased sizes of object files on disk and of executable programs in memory. Users need to weigh these costs against the potential benefits of higher performance in deciding whether or not to accelerate some of their application code. The article by Blanchet in this issue of the *Tandem Systems Review* explores the tradeoffs involved in this decision.

Users who decide to accelerate their programs can almost always do so directly from object files, without ever looking at the source code. The object programs produced by high-level language compilers like TAL™ and COBOL generally give users no problems. Programs that explicitly manipulate the register stack, use CODE statements, or implement trap handlers will work in most cases. For exceptions, see *Programmer's Guide for TNS/R Systems*, 1991. The article by Cressler, also in this issue of the *Tandem Systems Review*, addresses the process of moving TNS programs to TNS/R systems and shows how to handle the few special cases in which programs cannot be moved readily.

RISC Technology

The RISC approach to computer design, which arose in the 1980s, responds to two basic trends in computer technology. Memory chips have become denser and faster, eliminating the advantage of on-chip microcode in microprocessors. High-level languages (HLLs) have become more widespread and more efficient, reducing the need to cater to assembly language programmers. The new RISC designs take advantage of these trends by relying on software for many functions that were handled by hardware in previous computers.

The rule for RISC is to measure each design decision by its effect on the performance of typical large HLL programs. If a hardware feature yields a substantial performance gain for such programs, the designers include it. Otherwise, they rely on software to perform that function. Using this approach, designers found that they could best use the available technology by making their processors adhere to the following principles:

- Use a few simple instructions and memory addressing methods.
- Use on-chip circuitry rather than microcode to implement instructions.
- Use an assembly-line technique, called pipelining, for instruction execution.
- Rely on optimizing compilers to enforce hardware restrictions and maximize efficiency.
- Restrict off-chip memory accesses to the loading or storing of on-chip memory locations, called registers.
- Use operands in registers for all arithmetic operations.
- Dedicate the chip area saved by the above simplifications to providing a large on-chip set of registers as well as reducing the overall size of the chip.
- Rely on internal and external high-speed memory, called caches, for fast access to data and programs.

These principles allow the instruction pipeline to run smoothly. The RISC pipeline improves performance by increasing the instruction throughput.

The speed of execution of any program depends upon two factors: the number of instructions and the average length of time to execute each instruction. The RISC approach achieves faster execution than older design approaches because it allows a large decrease in the average length of time to execute each instruction but entails only a small increase in the number of instructions needed for typical HLL programs.

Conclusion

Tandem's TNS/R systems improve the economics of large-scale OLTP and broaden the range of applications for which OLTP is economically feasible. TNS/R systems provide substantial capabilities and lower costs by combining the technological advances of RISC architecture with the advantages of Tandem NonStop systems.

The TNS/R design protects users' hardware investment. TNS/R systems are compatible with the controllers and peripherals of the CLX and Cyclone lines. Existing CLX systems can be upgraded to TNS/R systems simply by replacing processor and memory boards.

TNS/R systems allow migration and expansion of existing TNS applications. Existing TNS applications run on TNS/R systems without modification, thereby preserving users' software investment. Users can develop new applications for TNS/R systems by using the same languages and tools they currently use for TNS systems. Tandem's millicode approach and its Accelerator product make this possible.

References

Blanchet, M. 1992. Improving Performance on TNS/R Systems With the Accelerator. *Tandem Systems Review*. Vol. 8, No. 1. Tandem Computers Incorporated. Part no. 65250.

Cressler, D. 1992. Debugging Accelerated Programs on TNS/R Systems. *Tandem Systems Review*. Vol. 8, No. 1. Tandem Computers Incorporated. Part no. 65250.

Programmer's Guide for TNS/R Systems. 1991. Tandem Computers Incorporated. Part no. 63927.

Acknowledgments

Many people helped with this article. The authors especially wish to thank Mark A. Taylor and Dave Garcia for describing the hardware; Duane Sand and Kristy Andrews for an introduction to RISC and the Accelerator; Alan Rowe for explaining the Guardian 90 changes; Mitch Butler for explaining the memory layout; Charles Levine and Art Sheehan for providing performance statistics.

Les Faby develops and teaches courses covering system software internals. He is responsible for TNS/R software training within the Software Development group. From 1983 to 1987, he supported low-level database products and the Guardian 90 operating system. Before joining Tandem, Les worked as a systems programmer and software developer.

Richard Mateosian is a freelance writer and computer systems consultant. He has worked as a systems programmer, a university lecturer, and a strategic marketing manager for microprocessors. His published works include books on microprocessors and programming and many technical articles. Richard is a regular columnist in *Micro*, a publication of the Computer Society of the IEEE.

Improving Performance on TNS/R Systems With the Accelerator

Tandem™ has recently introduced Tandem NonStop™ Series/RISC (TNS/R) systems, based on reduced instruction set computing (RISC) technology. TNS/R systems are compatible with existing Tandem NonStop Series (TNS) systems, so existing applications can run unchanged on TNS/R systems. They combine the performance and economy of RISC systems with the advantages of the Tandem NonStop architecture.

TNS applications running on a TNS/R system benefit only partially from the power and economy of RISC technology. A one-time process called acceleration improves the performance of applications by allowing them to use the RISC technology more effectively. The acceleration process is accomplished by Tandem's Accelerator software product, which operates on a TNS object file and produces a TNS/R, or accelerated, object file. The accelerated object file consists of the original object file and additional information to allow it to run more efficiently on TNS/R systems. The accelerated file runs exactly as it did before on TNS systems and can run several times faster on TNS/R systems.

Tandem has already applied the acceleration process to the performance-critical portions of the Tandem Guardian 90™ operating system and its associated utilities. Users receive the performance benefits of this acceleration, because most online transaction processing (OLTP) applications spend a large proportion of time executing system code. Users can usually achieve further performance increases by accelerating their own application code, but the benefit is different for each application.

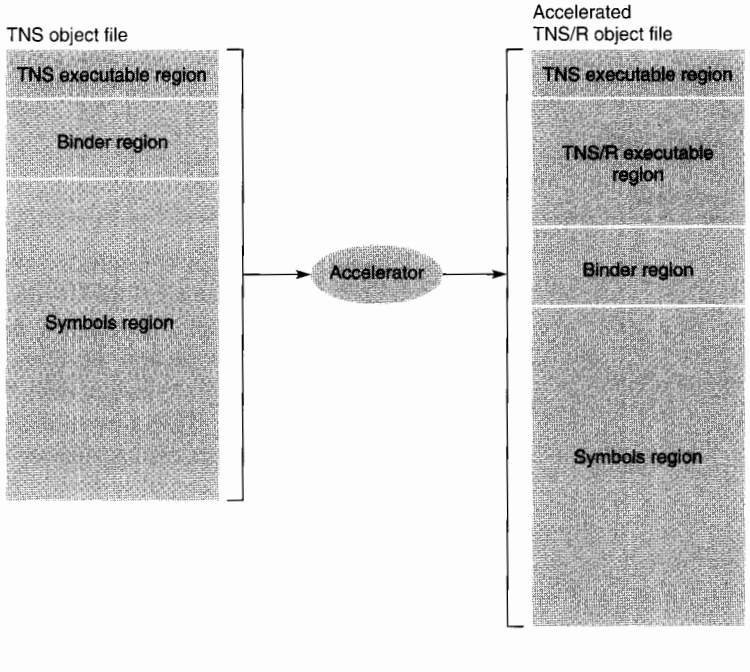
Users must weigh the costs and benefits of accelerating all or part of their application code. This article discusses the issues that users must consider in deciding whether or not to accelerate specific application programs. It describes a set of tools designed to help in the analysis and selection of programs to accelerate. The article is directed at application programmers and system managers. The reader does not need a technical knowledge of TNS/R systems.

The TNS/R Environment

TNS/R systems bring the power and economy of RISC technology to OLTP applications. Faby and Mateosian, elsewhere in this issue of the *Tandem Systems Review*, describe the approach Tandem used to achieve this technological advance while remaining compatible with existing TNS systems.

A TNS/R system includes both an outer and an inner instruction-processing environment. The outer environment is the familiar TNS architecture used in all Tandem computer systems. Tandem has implemented the TNS environment on TNS/R systems using a set of routines called *millicode*. (See Faby and Mateosian, 1992.) The inner environment, called the TNS/R environment, is the RISC architecture defined by the microprocessor that Tandem has chosen as the heart of its TNS/R systems. Existing TNS programs can run unchanged in the outer layer. When they do so, they already benefit from the fast RISC microprocessor and from the accelerated performance of the Guardian 90 operating system. The Accelerator program allows TNS programs to exploit even more of the power of the innerRISC layer without source code changes.

Figure 1



The Accelerator

Figure 1 shows how the process of acceleration transforms a TNS object file into an accelerated TNS/R object file. The Accelerator is supplied with the Guardian 90 operating system (version C30.06 or later) and can run on TNS or TNS/R systems.

Figure 1.
The process of acceleration.

Table 1.
Effects of migration and acceleration.

	Relative execution speed		
Program	**TNS system**	**TNS/R system**	**TNS/R system (accelerated)**
Program A (mostly application code)	1	3	6
Program B (mostly system code)	1	4	4.1

The Accelerator accepts TNS object files as input and produces accelerated object files. Because accelerated object files contain both the original TNS object code and the corresponding RISC code, they are larger than the original TNS object files. This allows accelerated object files to run either on TNS systems or on TNS/R systems.

The Accelerator functions like an optimizing compiler. When a TNS program runs in the outer environment of a TNS/R system, the millicode treats each TNS instruction like a separate program. The Accelerator treats successions of separate millicode routines as larger program units. Depending on user-specified options, these larger units correspond either to source statements or to entire procedures. The Accelerator applies optimizing techniques to these units to take advantage of features of the RISC architecture. For example, the RISC microprocessor has a large number of registers. If a variable is used by several successive TNS instructions, the Accelerator generates RISC code in which the variable is read from memory into a register the first time, and thereafter read from the register.

One of the most important optimizations performed by the Accelerator concerns the condition codes of TNS systems. Millicode must update flags corresponding to TNS condition codes each time it processes a TNS instruction. The Accelerator determines which of these codes the TNS program will actually use. It omits the TNS/R instructions that would otherwise update the corresponding flags needlessly.

Users can control the degree of optimization that the Accelerator performs. The Accelerator provides several options that alter the balance between error checking and execution speed. These are grouped into three option packages, called *safe*, *common*, and *fast*, which users can select at the object file level when they run the Accelerator. At the individual procedure level, users can make more specific tradeoffs by specifying options in the source code. For example, users who are sure that the code that implements a procedure never generates an overflow condition can eliminate the test for overflow from the RISC code that the Accelerator generates for that procedure.

Acceleration does not affect the performance of the program in the TNS environment but can result in large improvements when the RISC code runs on TNS/R systems. More information about the Accelerator appears in the *Accelerator Manual* (1991).

Considerations for Accelerating Applications

Tandem has accelerated all system code (SC) and all system library code (SL) of the Guardian 90 operating system. Applications spending a significant number of CPU cycles in SC or SL benefit immediately from this. Applications that spend a significant number of cycles in their own code can benefit from acceleration.

Table 1 illustrates the potential benefits of acceleration. The numbers indicate relative performance. Each program's performance on a CLX™ 600 system is taken to be 1.0, and the other numbers indicate execution speed after migration to a TNS/R system. Program A spends most of its CPU cycles in application code, while Program B spends most of its time in system code.

The example shows why it is important to make performance measurements both before and after acceleration. In the case of Program A, acceleration doubles the performance. In the case of Program B, most of the performance benefits come from the new hardware and the faster operating system. Acceleration adds little.

The Accelerator provides few performance benefits for I/O-intensive applications. The performance of these applications depends largely on the speed of physical I/O devices.

Most OLTP applications spend a large portion of their time executing Guardian 90 system calls, which have already been accelerated, and the rest of their time executing application code. The more time a program spends executing its own code, the greater the performance improvement from acceleration. Typical OLTP applications spend 85 percent of their time executing system calls, so they do not benefit much from acceleration.

Users should consider both the benefits and costs of acceleration before deciding which application programs to accelerate. These considerations, coupled with the set of tools and recommendations discussed later, can help users make educated decisions.

Table 2.
Approximate acceleration rates on a lightly loaded VLX system.

Procedure size (no. of instructions)	Instructions per second
Less than 20	15
Between 20 and 50	25
Between 50 and 500	40
Between 500 and 2000	25
Greater than 2000	15

Time Required to Run the Accelerator

The following steps must be followed to proceed from a source program to an accelerated object file:

1. Compile the individual Transaction Application Language (TAL™), COBOL, or other source programs to produce object files.
2. Bind the object files into a single TNS object file.
3. Accelerate the TNS object file to produce an accelerated object file.
4. Perform an SQL compilation (SQLCOMP) on the accelerated file if the original source programs contain SQL statements.

Only step 3 is new. The other steps are required for any TNS program. The entire sequence of four steps must be performed each time the source code changes.

A number of factors influence the elapsed time required to run the Accelerator. They include the complexity of the code; the size of the program, including the size of the procedures; and the workload, speed, and memory configuration of the CPU used. The Accelerator processes TNS instructions at a rate that depends roughly on the procedure size. Table 2 shows this relationship on a lightly loaded Tandem VLX™ system.

Users can use the Tandem Binder program to determine the sizes of procedures. The time required to accelerate an object file containing several procedures is the sum of the times required for the individual procedures. For very large procedures, users can expect acceleration to consume about 45 minutes per code segment on a VLX system, or about half that time on a TNS/R system.

Size of Accelerated Object Files

Figure 1 shows a TNS object file and the corresponding accelerated object file. The TNS executable, Binder, and symbols regions are identical in both files. The accelerated file also contains a TNS/R executable region, which is larger than the TNS executable region. It takes more of the simple RISC instructions to do the same job as the complex instructions included in the TNS executable region.

Occasionally the Accelerator will not convert a section of TNS code completely into TNS/R instructions because of the way Tandem has implemented the TNS environment on TNS/R systems. When such points are reached during execution, the program switches from TNS/R code to TNS code, after which it continues executing TNS code until it reaches a point at which it can conveniently switch back. These switches between instruction sets are called *transitions.*

Transitions are normal events that do not affect the correctness of programs. The need for them explains why accelerated object files must contain both TNS and TNS/R executable regions, even when there is no need for the object file to be portable between TNS and TNS/R environments.

The ratio between the size of the accelerated object file and the size of the TNS object file depends on a number of factors. It is possible to remove the Binder and symbols regions from a TNS object file. Such a file is called a *stripped* object file. The Accelerator can process a stripped TNS object file to produce an accelerated object file. The resulting TNS/R executable code will probably execute more slowly than the code that would result if the TNS object had not been stripped. This happens because the TNS/R code will be forced to make more transitions into TNS code at run time. Therefore, Tandem recommends that only unstripped object files be accelerated. The accelerated file can then be stripped of Binder and symbols regions to save disk space in the production environment.

An accelerated object file is about twice the size of the original TNS object file. A stripped accelerated file is about four times the size of the stripped TNS object file. These ratios will vary among applications, but they can be used for planning purposes. They give a rough idea of the amount of disk storage that accelerated object files require.

The larger size of accelerated object files means that they also require more memory at run time. A rule of thumb is that an accelerated program needs three times the code space, while the amount of data space is unchanged. Thus if a TNS application uses 0.25 megabytes of main memory for code and 0.75 megabytes for data for a total of 1 megabyte, the corresponding accelerated application will use 0.75 megabytes for code and the same 0.75 megabytes for data, for a total of 1.5 megabytes. The ratio of memory use in this case is 6:4, a 50 percent increase. A TNS/R CPU board contains 32 megabytes of memory.

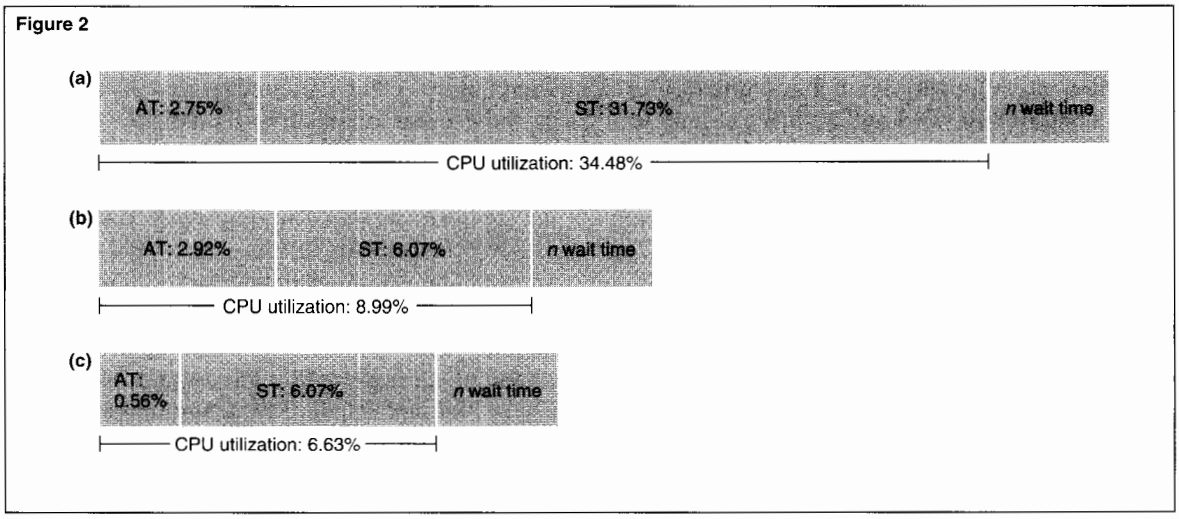


Figure 2.

A sample application profile of CPU utilization–application time (AT) plus system time (ST)–for three different scenarios. (a) Process A executing on a TNS system. (b) Process A running on a TNS/R system. (c) An accelerated Process A running on a TNS/R system.

Testing of Applications on TNS/R Systems

Testing is normally the final step in the process of placing an application program into production. This is no different for accelerated programs, but several cases need to be discussed in detail.

If an application is in production on a TNS system, Tandem advises first updating the Guardian 90 operating system on the TNS system to C30 and verifying that the application runs correctly there. Users should follow their usual testing practices for moving to a new Guardian 90 release. They then should follow the steps in the next paragraph to upgrade the hardware.

If an application is in production on a TNS system running the C30 version of the operating system and the underlying hardware is upgraded to a TNS/R system, users can move their unmodified, unaccelerated application code to the TNS/R system and perform the same level of testing that they normally do when moving to a new hardware platform. A very few programs that run properly on TNS systems will require modification before they run correctly on TNS/R systems.

Tandem advises users wishing to accelerate some of their unmodified application code to test it sufficiently after acceleration to verify that it still works properly. Users should bear in mind that there is an extremely small class of TNS programs that run correctly on TNS/R systems but still require modification before their accelerated versions run correctly. (See *Programmers Guide for TNS/R Systems*, 1991, when either TNS or accelerated program modifications are necessary.)

When users wish to modify application code or develop new applications for TNS/R systems, Tandem recommends performing the iterative cycle of debugging and source code modification in the TNS environment, then accelerating the working application and testing it. This is because debugging TNS programs is easier than debugging the corresponding accelerated programs. (See the article by Cressler in this issue of the *Tandem Systems Review.*)

Profiling an Application

Users should be able to estimate the costs of acceleration by applying the guidelines described earlier. The tradeoff is the performance improvement attributable to acceleration. The process of profiling described here will help users estimate the amount of improvement to be expected from accelerating any particular application program. Users must then examine their service-level objectives to decide whether the available improvement is worth the cost.

Tandem recommends the following profiling procedure for determining which programs to accelerate.

1. Place the program into one of three categories, based upon how its performance affects overall application performance: critical to overall performance, affecting overall performance, or not likely to affect performance. In doing so, look at the relation of the application to components such as memory, disks, and communications, which also affect throughput, path length, and response time. If the program is unlikely to affect overall performance, do not accelerate it. If the program is critical to overall performance, follow the procedures described in the next section. If the program falls into the middle category, continue with the following steps.
2. Determine whether the program is CPU intensive or I/O intensive. Users should know where the program is spending most of its execution time, either by understanding its design or by measuring its performance. The Accelerator will improve the performance of a CPU-intensive program but is less likely to benefit an I/O-intensive program.
3. Determine the CPU utilization of each program. Accelerating a program that consumes a small percentage of CPU time will have a small impact on system performance. In typical systems, users should concentrate on the 20 percent of user processes that consume 80 percent of the CPU cycles (the 80-20 rule). This will give a good performance gain at a reasonable cost. Measure™, Tandem's system performance measurement product, can assist in this task.
4. Select user processes with significant time in user code (UC) and user library code (UL). As a rule of thumb, if a program spends 85 percent or more of its time in system code (SC) and system library code (SL), the main performance gain has already been achieved because the SC/SL has already been accelerated on TNS/R systems. To help assess the relative amounts of time spent in UC/UL and SC/SL, users can obtain a set of tools called the application profiling tools from their Tandem analysts.

Figure 2 illustrates the process of profiling a typical OLTP application. The first bar, (a), shows that over 90 percent of the application's time is spent in SC/SL, represented here by ST. The performance improvements gained from acceleration are strongly related to the time spent in application code, represented by AT. In this example, overall performance would benefit only slightly from acceleration, because AT represents only a small percentage of total CPU utilization. (AT + ST = total CPU utilization.) However, if the percentages on the TNS system were reversed, that is, AT = 31.73 percent and ST = 2.75 percent, the application would benefit greatly from acceleration.

Figure 3.
A sample Enform report.

Figure 3

Application Characterization report for System \D (C30).

Name	PID	Program-Name			CPU Util	Process UC/UL Util	Process SC/SL Util	CPU UC/UL Util
.								
.								
.								
$P1CC	5, 22	$DD12	XB10OBJ	IASLO	11.94%	49.59%	50.41%	5.59%
$XD04	5, 10	$SYSTEM	SYS34	X25OBJ	10.91%	71.75%	28.25%	7.74%
$XD02	5, 11	$SYSTEM	SYS34	X25OBJ	10.53%	70.22%	29.78%	7.25%
$XD03	5, 9	$SYSTEM	SYS34	X25OBJ	10.42%	69.78%	30.22%	7.18%
$XD01	5, 12	$SYSTEM	SYS34	X25OBJ	10.21%	69.90%	30.10%	7.04%
	5, 4	$SYSTEM	SYS34	OSIMAGE	.78%	6.41%	93.59%	.04%
$MONITOR	5, 0	$SYSTEM	SYS34	OSIMAGE	.02%	42.85%	57.15%	.00%
					54.81	380.50	319.50	34.84
.								
.								
.								
$DD12	7, 6	$SYSTEM	SYS34	OSIMAGE	13.56%	62.04%	37.96%	8.12%
$P1CE	7, 29	$DD12	XB10OBJ	IASLO	12.14%	50.65%	49.35%	5.93%
$PD04	7, 21	$DD13	XPSIMOBJ	PSIM	5.60%	24.77%	75.23%	1.34%
$PD01	7, 25	$DD13	XPSIMOBJ	PSIM	5.60%	25.32%	74.68%	1.40%
$PD02	7, 30	$DD13	XPSIMOBJ	PSIM	5.55%	23.80%	76.20%	1.27%
$PD03	7, 26	$DD13	XPSIMOBJ	PSIM	5.49%	25.65%	74.35%	1.37%
$DD12	7, 12	$SYSTEM	SYS34	OSIMAGE	3.86%	67.89%	32.11%	2.53%
					54.91	642.25	757.75	22.87
					398.45			197.52

CPU Util: For each process, the percentage of CPU utilization consumed during the measurement window (default 15 minutes)

Process UC/UL Util: Of the CPU Util, percentage of time spent in user code (UC.n and/or UL.n).

Process SC/SL Util: Of the CPU Util, percentage of time spent in system code (SC.n and/or SL.n).

CPU UC/UL Util: Percentage of total CPU utilization spent executing in user code (UC.n and/or UL.n). This is equivalent to (CPU Util) x (Process UC/UL Util).

Programs Critical to Overall Performance

For most programs, the Accelerator's default behavior provides most of the improvement possible and requires little effort on the part of users. However, some programs are so critical to achieving overall performance goals that users are willing to take extra steps to obtain small additional improvements in execution speed. These steps are summarized in order of importance:

- Be sure that the Binder and symbols regions are included in the original TNS object file.
- Determine whether accelerator options can be safely used to trade error checking for speed.
- Modify the source code to reduce situations that force the Accelerator to generate transitions to TNS code.
- Modify the source code to avoid situations, called *compatibility traps*, that TNS/R systems handle automatically but less efficiently (for example, misaligned pointers).

Information on all of the above items appears in the *Programmers Guide for TNS/R Systems*, 1991.

Application Profiling Tools

Tandem developed application profiling tools to help users decide which programs to accelerate. These tools include a Tandem Advanced Command Language (TACL™) macro to configure a performance measurement using Measure and a set of Enform™ query language/report formatter queries. The tools can be used on either TNS or TNS/R systems running the C30 release of the Guardian 90 operating system.

The profiling tools are not supported as standard products because users can modify them easily to conform to their own specific needs. These tools were used extensively prior to the introduction of the TNS/R systems. The profiling tools are easy to use and are documented in *Application Profiling Tools: Deciding What to Accelerate.*

The TACL Macro

The TACL macro collects Measure data on all PROCESSH entities for 15 minutes. Users can modify the time interval. Tandem recommends that a long enough time be used to give a representative result. The decision about whether to accelerate a program might otherwise be based on inadequate information.

Enform Queries: Application Characterization Report

Users can easily customize the Enform queries to reduce the amount of data reported. They can eliminate data about processes pertaining to OSIMAGE, SYS*nn*, and SYSTEM. To facilitate data analysis, users can perform a modified Enform run to group programs by program name. The Enform queries provided in the profiling tool package give the basic information that most users need, and they serve as a model for additional queries that users may wish to include.

Figure 3 shows an example of the Enform report. The measurements were taken on a TNS system. The information appears in descending order of CPU utilization. This is further broken down by CPU and system totals.

Figure 4

```
RUN $SYSTEM.SYSTEM.VPROC
Enter filename:
> $system.sys34.x25obj
 $SYSTEM.SYS34.X25OBJ
        Binder timestamp:  30MAY  14:21:55
      Version procedure:  T9060C20^15JUN91^X25AM^ABL02A
             Target CPU:  TNS/R
      AXCEL timestamp:  30MAY91  14:26:22
```

Figure 4.
Using VPROC to determine whether or not a program has been accelerated.

From the data presented in Figure 3, one can readily see which of the processes would benefit most from acceleration. Process 5,22 ($DD12.XB10OBJ.IASLO), for example, seems like a suitable candidate. It uses 11.94 percent of the CPU time and spends 49.59 percent of its time in UC/UL, which is equivalent to overall CPU utilization of 5.59 percent for this process.

$SYSTEM.SYS34.X25OBJ also seems like an excellent program to accelerate, but it is not a user program. This code is part of the X.25 subsystem provided by Tandem, so users should not try to accelerate it. Acceleration of Tandem code by users is not supported under any circumstances. Tandem tests each release of software in its entirety. Users can jeopardize the stability and integrity of their system environments if they accelerate Tandem modules.

To verify whether a program has been accelerated, users can use the Binder command SHOW INFO, or VPROC, as shown in Figure 4. The phrase AXCEL timestamp in Figure 4 indicates that $SYSTEM.SYS34.X25OBJ has already been accelerated by Tandem.

As shown previously in Figure 3, the last two Tandem processes in CPU 5 use the program $SYSTEM.SYS34.OSIMAGE. This is where system code resides. Tandem has accelerated all system code and system library code.

In CPU 7, process 7,29 also appears to be a good candidate for acceleration. Its statistics are similar to those of process 5,22. Thus, if program $DD12.XB10OBJ.IASLO is also accelerated, performance improvements will be obtained in both CPU 5 and CPU 7.

On an individual process basis, program $DD13.XPSIMOBJ.PSIM does not seem like a candidate for acceleration. However, when the results of all four processes are combined, the numbers are quite different: 22.24 percent CPU utilization and 5.38 percent CPU UC/UL utilization.

From this exercise, one can conclude that processes 5,22 and 7,29 both have excellent profiles. Accelerating these programs would improve their performance and reduce the CPU consumption in CPU 5 and CPU 7. The extra CPU cycles could be used for additional work, load balancing, or simply to reduce a CPU bottleneck.

Similar conclusions apply to program $DD13.XPSIMOBJ.PSIM. Although a single occurrence of this program does not account for significant CPU UC/UL utilization, when CPU utilization from all four processes are combined, this program becomes a good candidate for acceleration. The same applies in general to any server program. The final decision whether or not to accelerate these programs should be made after considering the associated costs and desired performance-level objectives.

Conclusion

Tandem's Accelerator program and application profiling tools help users manage the performance of their applications on TNS/R systems. Acceleration provides large performance gains for CPU-intensive programs that spend a significant portion of their time in application code rather than system code. These gains must be weighed against the costs of acceleration in time, disk space, and memory requirements. Users must consider all of these factors in light of their own performance-level objectives and decide whether or not to accelerate specific programs.

References

Accelerator Manual. 1991. Tandem Computers Incorporated. Part no. 63928.

Application Profiling Tools: Deciding What to Accelerate. 1991. Tandem Computers Incorporated. Support Note S91111. Available from Tandem representatives upon request.

Cressler, D. 1992. Debugging Accelerated Programs on TNS/R Systems. *Tandem Systems Review.* Vol. 8, No. 1. Tandem Computers Incorporated. Part no. 65250.

Faby, L., and Mateosian, R. 1992. Overview of Tandem NonStop RISC Systems. *Tandem Systems Review.* Vol. 8, No. 1. Tandem Computers Incorporated. Part no. 65250.

Programmer's Guide for TNS/R Systems. 1991. Tandem Computers Incorporated. Part no. 63927.

Acknowledgments

I wish to thank Diane Cressler, Kevin Deyager, Steven Kahn, Richard Mateosian, Bob Metter, Mike Noonan, and Dean Wakashige for their contributions to this article.

Manon Blanchet joined Tandem in 1989 and has worked in the data processing field since 1979. She is currently a member of Large Systems Marketing Support, working as the CSO technical program manager for future operating systems. Her prior experience includes field support in migration to RISC, performance analysis, disaster recovery, and system management.

Debugging Accelerated Programs on TNS/R Systems

Tandem™ NonStop™ Series (TNS) computer systems and the Tandem Guardian 90™ operating system have long provided an excellent performance-to-price ratio (PPR) over a broad spectrum of applications. Recently Tandem has introduced a new line of computers based on reduced instruction set computing (RISC) technology. These Tandem NonStop Series/RISC (TNS/R) computer systems provide an even better PPR than existing TNS systems.

Although TNS/R systems have a new underlying instruction set, Tandem has designed them to execute existing TNS programs. This allows users to move existing applications directly to TNS/R systems and enjoy the benefits of the new technology without reprogramming.

The direct execution of TNS programs on TNS/R systems achieves only part of the performance improvement possible with the new technology. To allow users to achieve greater performance improvement, Tandem has developed its Accelerator software product. The Accelerator processes TNS object files and produces accelerated object files that use the new technology more efficiently.

Almost all existing TNS object files execute correctly on TNS/R systems without change. These programs can be accelerated to provide higher performance, and almost all accelerated programs run correctly with no further effort by the programmer.

For debugging nonaccelerated TNS programs, there are no significant differences between TNS systems and TNS/R systems. Debugging accelerated programs differs slightly. Programmers debugging accelerated programs have fewer debugging commands available.

To minimize the need to debug accelerated programs, programmers can follow a simple sequence of steps to move programs from TNS systems to TNS/R systems. If a programmer needs to debug an accelerated program, Tandem's tools provide the most frequently needed source-level debugging capabilities.

This article describes a sequence of steps that programmers can follow to move existing TNS applications to TNS/R systems or to develop new applications for TNS/R systems. The article helps programmers understand the factors that influence debugging and describes the actions that programmers can take while debugging accelerated programs. The article assumes the reader is familiar with programming TNS systems and with the Tandem Inspect™ debugger.

Executing and Debugging Programs on TNS/R Systems

TNS/R systems execute TNS programs by invoking millicode[1] subroutines, which provide the same function as microcode on TNS systems. Running and debugging a TNS program on a TNS/R system is the same as running and debugging the program on a TNS system. To maximize program performance on TNS/R systems, programmers can accelerate TNS programs by using the Accelerator. Tandem designed the Accelerator to improve program performance and to maintain compatibility with existing TNS programs. The Accelerator compiles the TNS code into TNS/R instructions that are optimized for execution by TNS/R processors. The accelerated code can run substantially faster than the TNS code. The accelerated object file contains the unmodified TNS program and the accelerated TNS/R instructions.

An accelerated program behaves in the same way as the original TNS program; however, the code optimizations that make an accelerated program perform faster than a TNS program can make an accelerated program more difficult to debug. The TNS/R instruction sequence does not perform steps in the same order as the original TNS instructions.

Optimizations in the accelerated code include the following:

- Reordering instructions to take advantage of the TNS/R processor's instruction pipeline (Kane, 1989).
- Minimizing loads from memory by reusing values previously loaded into registers (Kane, 1989).
- Eliminating unnecessary TNS machine side effects[2].

The ease of debugging accelerated programs on Tandem systems compares favorably with the difficulty of debugging optimized code on other systems. A programmer who can debug a TNS program without knowing the TNS instruction set and machine registers can debug an accelerated version of the program without knowing the TNS/R instructions and machine registers. When debugging accelerated programs, programmers can set breakpoints on statements, step by statements, trace the call sequence (stack frames), and examine variables in memory.

Application Migration

TNS/R systems run the C30 release of Guardian software. If a program executes correctly on a TNS system running C30 software, it should execute correctly on a TNS/R system. The simplest and most likely migration path is to start with a working program, optionally accelerate it, test it on a TNS/R system, and find that it works there without change. Programmers should debug new or modified TNS programs before accelerating them.

[1] Millicode is TNS/R code that implements TNS low-level functions such as exception handling, real-time translation, and the TNS instruction set. TNS/R millicode is functionally equivalent to TNS microcode.

[2] Most TNS instructions cause changes to registers such as E, CC, and K. Because these registers are not present in TNS/R systems, the millicode maintains a view of them when executing TNS instructions. Accelerated code updates the values of these registers only when they are needed by subsequent instructions.

Figure 1

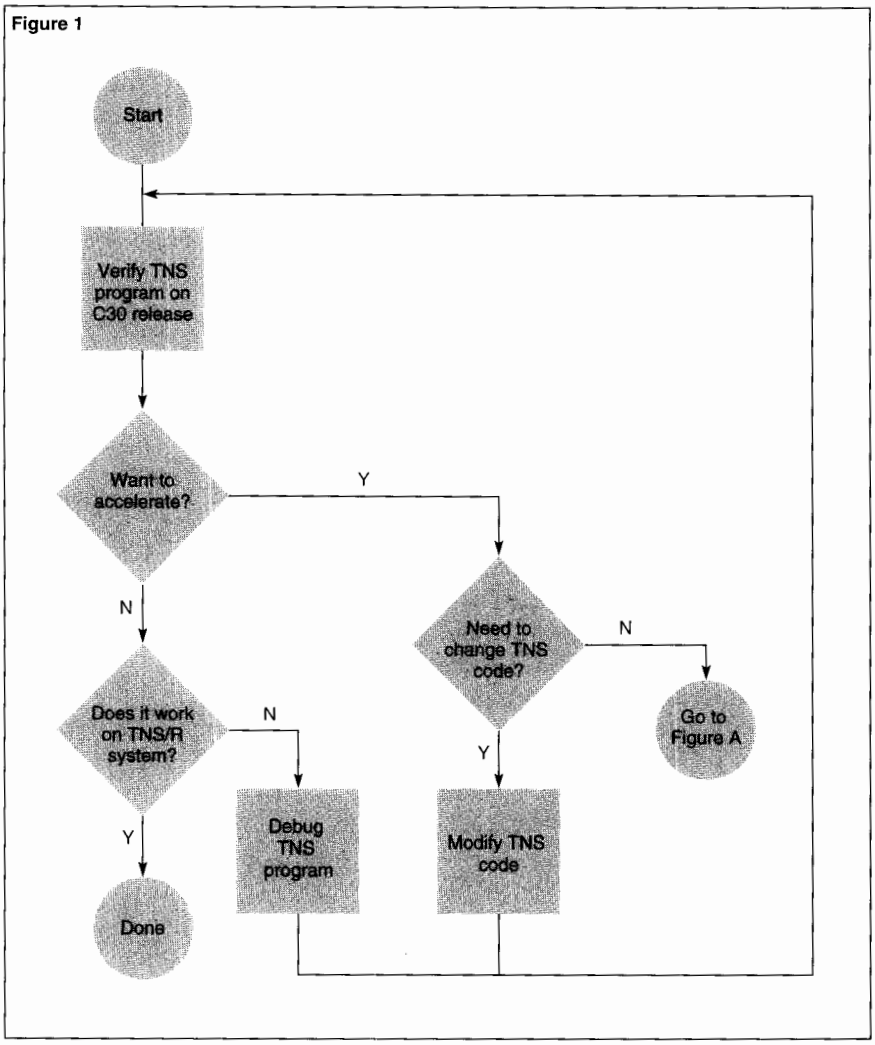


Figure 1.
Moving TNS programs to TNS/R systems.

Problem Checklist

Sometimes a TNS program executes correctly but the accelerated program does not. When that happens, programmers should follow the steps listed.

Step 1. If a TNS program runs correctly but fails when accelerated, the most likely cause is that the programmer has made one of the following common mistakes:

- Accelerating with incorrect Accelerator options and failing to check the Accelerator output listing for the resulting warning messages.
- Binding a program after accelerating it.
- Failing to SQLCOMP a program with embedded SQL after accelerating it.

Step 2. If the programmer has not made any of the above mistakes, the most likely cause of the problem is one of the following:

- Differences between TNS and TNS/R systems.
- Timing problems.

Step 3. If the programmer cannot identify one of the above problems as the cause, the situation may require the assistance of a Tandem analyst.

Often, the programmer has made one of the common mistakes listed in step 1. In other cases, Transaction Application Language (TAL™) programs that contain privileged or machine-dependent code (for example, CODE statements) require modification because of differences between TNS and TNS/R processors. (See *Programmer's Guide for TNS/R Systems*, 1991.) In rare cases, a program that runs correctly on a TNS system does not run correctly on a TNS/R system. For example, some timing-sensitive programs may encounter timing problems on the TNS/R system. The programmer may have to debug such a program on a TNS/R system.

The flow charts in Figures 1 and 2 show how to move programs from TNS systems to TNS/R systems. In only a few cases do programmers have to debug accelerated programs. The remainder of this article focuses on those cases. It provides hints on how to accelerate programs to satisfy performance and debugging needs and how to debug accelerated programs using Inspect or Debug.

Accelerator Debugging Features

The Accelerator provides two levels of optimization. One level provides easier debugging than the other. With either level, the Accelerator labels each location in the code to indicate which debugging capabilities are valid at that point. The programmer need not know about the underlying TNS/R instructions.

StmtDebug and ProcDebug Options

The Accelerator provides two options to define the boundaries for optimizations. At these boundaries within accelerated programs, all debugging capabilities are available. The StmtDebug option directs the Accelerator to optimize instructions within the code produced for each source statement. Instructions are not optimized across statements. (Statements include sentences and verbs in COBOL.) At a statement boundary, all of the underlying machine instructions for previous statements and none for following statements have been executed. All debugging capabilities are available at statement boundaries.

The ProcDebug option directs the Accelerator to perform optimizations within each procedure. (A procedure is a program in COBOL.) Optimizations can cross statement boundaries. As a result, programmers have fewer debugging capabilities at statement boundaries. For production use, programmers should use the ProcDebug option, because it provides the best performance. The Accelerator uses ProcDebug when neither option is specified.

The StmtDebug option produces code that is easier to debug, while the ProcDebug option produces more highly optimized code. A program accelerated using the ProcDebug option has approximately 10 percent fewer TNS/R instructions to execute than the same program accelerated using the StmtDebug option. A program accelerated with either option executes faster than the nonaccelerated program.

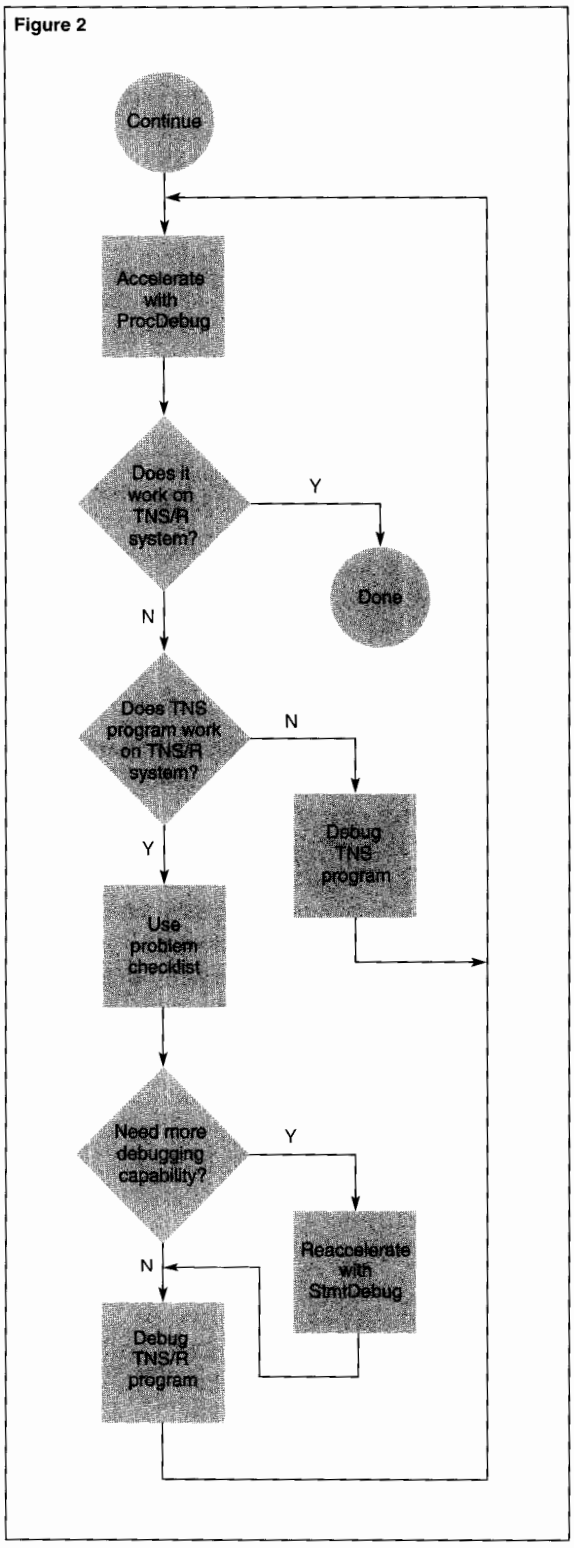


Figure 2.
Accelerating TNS programs for execution on TNS/R systems.

Figure 3

Source statement	TNS/R instructions (StmtDebug)	TNS/R instructions (ProcDebug)
1. a := b + c;	**RE** LOAD B -> REG1 LOAD C-> REG2 NOP ADD REG1, REG2 -> REG3 STORE REG3 -> A	LOAD B -> REG1 LOAD C -> REG2 LOAD X -> REG4 ADD REG1, REG2 ->REG3 STORE REG3 -> A
2. p := x - c;	**RE** LOAD X -> REG1 LOAD C -> REG2 NOP SUB REG1,REG2 -> REG3 STORE REG3 -> P	- - X already loaded - - C already loaded **ME** SUB REG4, REG2 -> REG3 - - store to P later
3. if x > 0 then begin	**RE** LOAD X -> REG1 NOP BLEZ REG1 NOP	- - X already loaded BLEZ REG4 STORE REG3 -> P
4. call procx;	**RE** CALL PROCX	**ME** CALL PROCX
5. a := p end;	**RE** LOAD P -> REG1 NOP STORE REG1 -> A	**RE** LOAD P -> REG1 NOP STORE REG1-> A

RE Register-exact
ME Memory-exact

Figure 3.
TNS/R instructions generated by the StmtDebug and ProcDebug options.

Figure 3 compares TNS/R code generated using the StmtDebug and ProcDebug options. The StmtDebug option results in code that is longer by six instructions but is easier to debug. The two-letter code to the left of the first TNS/R instruction corresponding to each source statement indicates the degree of debugging difficulty. The codes stand for register-exact (RE), and memory-exact (ME). Unmarked TNS/R instructions are called non-exact points.

Programmers have the most debugging options at register-exact points and the fewest at non-exact points. The StmtDebug option produces a register-exact point corresponding to each source statement. The ProcDebug option produces memory-exact points corresponding to three of the source statements and no point at all corresponding to statement 3.

Table 1 shows the relative performance and debugging capabilities of nonaccelerated TNS programs, programs accelerated with the StmtDebug option, and programs accelerated with the ProcDebug option. Programs accelerated with the StmtDebug option contain register-exact points at most statement boundaries and programs accelerated with the ProcDebug option contain memory-exact points at most statement boundaries. For simplicity, the table entries reflect these typical cases.

Programmers debugging nonaccelerated programs on TNS/R systems have all the debugging capabilities that they have on TNS systems. When they debug programs accelerated with the ProcDebug option, they can set breakpoints at the beginnings of most statements and display data in memory. If they need to modify data or resume execution at arbitrary statements during debugging, they must accelerate the program using the StmtDebug option.

Table 2 shows how to create an accelerated program from a nonaccelerated program and vice versa. Because accelerated object files still contain the nonaccelerated TNS code, programmers can use the following Tandem Binder command to disable the accelerated code:

CHANGE AXCEL ENABLE OFF *filename*

When the TNS/R system executes this file it will use the TNS code rather than the accelerated code. The programmer can then use the following Binder command to re-enable the accelerated code:

CHANGE AXCEL ENABLE ON *filename*

Accelerator State

The Accelerator associates with each code location an attribute called its accelerator state. The accelerator state tells programmers which debugging commands can be used when execution stops at that location.

Table 1.
Relative performance and debugging capabilities of TNS programs and accelerated programs.

		Debugging at statement boundaries						
Program	**Performance**	**Exactness**	**Breakpoint at statement**	**Step statements**	**Display variables**	**Modify variables**	**Resume at another statement**	**Debugging at TNS machine level*** **
TNS program	Slowest	—	Yes	Yes	Yes	Yes	Yes	Yes
Accelerated program (StmtDebug)	Faster	Usually register exact	Yes	Yes	Yes	Yes	Most**	At statement boundaries
Accelerated Program (ProcDebug)	Fastest	Usually memory exact	Yes	Yes	Yes	No*	No	At procedure boundaries

* Modifying the values of variables may have no effect.

** Execution can resume at most statement boundaries. Exceptions include the first statement in some subprocedures and COBOL paragraphs, and some labeled statements.

*** TNS machine-level capabilities include setting breakpoints at TNS instructions, stepping TNS instructions, and displaying and modifying TNS register values.

The accelerator state can be register-exact, memory-exact, or non-exact. At register-exact points, all debugging capabilities are valid. At memory-exact points, programmers can set breakpoints and display memory. At non-exact points, debugging operations might produce unreliable results.

Most statement boundaries are register-exact points or memory-exact points. Most other TNS instructions are non-exact points. Whether a statement boundary is register-exact or memory-exact depends on whether the programmer used StmtDebug or ProcDebug to accelerate the program.

Register-Exact Points. A location is a register-exact point if the state of the accelerated program is the same there as at the corresponding point of the nonaccelerated TNS program. The Accelerator suppresses optimizations across locations that are register-exact points and ensures that the TNS register state (values of R0-R7, E) is up to date[3]. This is important to programmers who debug at the TNS instruction level.

Table 2.
Moving between levels of acceleration.

	Go to		
Go from	**TNS code**	**Accelerated code (StmtDebug)**	**Accelerated code (ProcDebug)**
TNS code	—	Accelerate	Accelerate
Accelerated code (StmtDebug)	Use Binder	—	Reaccelerate
Accelerated code (ProcDebug)	Use Binder	Reaccelerate	—

[3] At a register-exact point, a TNS register such as R0-R7 or CC is guaranteed to contain the same value it would have on a TNS system only if that register value is needed subsequently by the program.

At register-exact points, programmers can use all debugging capabilities with reliable results:

- Display and modify data in memory.
- Resume execution at a different register-exact point.
- Specify a register-exact or memory-exact point as the target location of a breakpoint.
- Display and modify TNS register values.

In programs accelerated with the ProcDebug option, relatively few code locations are register-exact points. These include:

- Procedure entry points.
- Some subprocedure entry points (paragraphs in COBOL).
- Some labeled statements (if the object file being accelerated contains symbols).
- The TNS instruction after a procedure or subprocedure call.

In programs accelerated with the StmtDebug option, register-exact points include:

- All locations that are register-exact with the ProcDebug option.
- The beginnings of most statements[4].

Figure 3 shows this difference between the StmtDebug and ProcDebug options. The StmtDebug option generates TNS/R instructions for which there are register-exact points at all statement boundaries. The TNS/R code generated by the ProcDebug option contains only one register-exact point, at statement 5, which is the first instruction of the statement after a procedure call. The ProcDebug option intermingles TNS/R code for statements 1, 2, and 3.

Memory-Exact Points. The Accelerator ensures that programmers can set breakpoints and display data at memory-exact points. A memory-exact point is a location in the code where memory is up to date with respect to the source code. That is, at a memory-exact point, the computer has performed all memory-modifying operations (for example, store to memory) for preceding statements and has performed no memory-modifying operations for subsequent statements. The Accelerator may perform other optimizations that cross memory-exact points. For example, the TNS/R instructions that implement a subsequent statement can use a value that is already in a register.

[4]Exceptions include any labeled statement that immediately follows a procedure call, the first statement in some subprocedures, and the first statement in most COBOL paragraphs. These statements are memory-exact points.

If the Accelerator has used the ProcDebug option, the beginnings of most statements are memory-exact points. Programmers can display memory accurately at memory-exact points. They can also specify memory-exact points as target locations for breakpoint requests. They might, however, encounter limitations with other debugging operations at memory-exact points.

Debugging limitation at memory-exact points include the following:

- Memory values modified during debugging might not be used in subsequent operations.
- Displayed TNS register values might not be accurate.
- Inspect will not allow modification of TNS register values.
- Inspect will not accept a RESUME AT command if either the current location or the target location is a memory-exact point.

If the programmer has accelerated the code in Figure 3 using the ProcDebug option, statement 2 is memory-exact. The programmer can stop execution at that point by specifying statement 2 as the target of a breakpoint. A display of the memory value of *a* at that point reflects the assignment from statement 1. The memory value of *p* does not yet reflect the assignment from statement 2. If the programmer modifies the value of *c* in memory at this point, the program will not use the new value in calculating the value to be assigned to *p* in statement 2. The accelerated program uses the value that was loaded into REG2 during the execution of statement 1.

After performing debugging operations at a breakpoint at statement 2, the programmer can set another breakpoint and direct Inspect to continue execution of the program from where it left off. Inspect will not allow the programmer to specify a location at which to resume execution, since it can only perform that operation reliably if both the current location and the target location are register-exact points.

Non-Exact Points. The Accelerator labels all code locations that are neither memory-exact nor register-exact as non-exact. Most statement boundaries are memory-exact or register-exact points. Most TNS instructions that do not correspond to the beginnings of statements are non-exact points. Any type of optimization may cross a non-exact point. At non-exact points, the location in the source code (or TNS code) does not map to any point in the TNS/R code. If the programmer has accelerated the code in Figure 3 using the ProcDebug option, statement 3 is a non-exact point.

The most useful debugging action at a non-exact point is to step the program to the next memory-exact point or register-exact point, using the Inspect command STEP STATEMENT, STEP VERB, or BREAKPOINT. All other debugging actions are either not permitted or not guaranteed to produce reliable results.

Programmers must be aware of the following restrictions at non-exact points:

- All debugging restrictions at memory-exact points also apply at non-exact points.
- Source statements or TNS instructions that are non-exact points are not valid target locations for breakpoint requests.
- Displaying the values of variables yields unpredictable results.

Figure 4.
Identification of exact points at statement boundaries by the SOURCE command in Inspect.

Figure 4

```
-EXY1XLP-source
    #14    end
    #15
    #16  proc mainp main
    #17    begin
 *  #18      a := b + c;
    #19      p := x – c;
 —  #20      if x > 0 then
    #21        begin
    #22          call procx;
 @  #23          a := p

    @   Register-exact
blank   Memory-exact
    —   Non-exact
    *   Current location
```

Figure 5.
Identification of exact points of TNS instructions by the SOURCE ICODE command in Inspect.

Figure 5

```
-EXY1XLP-source icode
   #14      end;
   #15
   #16  proc mainp main;
   #17      begin
 * #18        a := b + c;
        @ LOAD G+001  LOAD G+002  IADD
          STOR G+000
   #19        p := x - c;
        >  LOAD G+003  LOAD G+002  ISUB
          STOR G+004
 — #20        if x> 0 then
   #21          begin
          LOAD G+003  CMPI  +000     BLEQ  +003
   #22          call procx;
        >  PCAL 002
 @ #23          a := p
        @ LOAD G+004  STOR G+000

    @   Register-exact (source line or TNS instruction)
    >   Memory-exact (TNS instruction)
    —   Non-exact (source line)
blank   Non-exact (TNS instruction)
    *   Current location
```

Debugging Accelerated Programs at the TNS Instruction Level

In most cases, programmers should debug programs before accelerating them or use the source-level debugging capabilties that Inspect provides for accelerated programs. While the statement remains a valid level of granularity for debugging accelerated programs, debugging within a statement at the TNS instruction level is severely limited because most TNS instructions are non-exact points. The Accelerator does not preserve debugging capabilities at every TNS instruction, because this would sacrifice performance excessively. Programmers who must set breakpoints within statements and examine the register state at those points in an accelerated program might need to debug at the TNS/R instruction level.

Accelerator State Information in Inspect

The SOURCE, SOURCE ICODE, ICODE, and low-level I commands in Inspect can help programmers determine valid BREAKPOINT and RESUME AT locations. Inspect annotates the output of the SOURCE command by marking the beginning of each source line or statement with a single character, as follows:

- @ if it is a register-exact point.
- – if it is a non-exact point.
- *blank* if it is a memory-exact point.

Inspect annotates the output of the SOURCE ICODE, ICODE, and low-level I commands by marking each TNS instruction with a single character, as follows:

- @ if it is a register-exact point.
- *blank* if it is a non-exact point.
- > if it is a memory-exact point.

Figures 4 and 5 shows sample output from the SOURCE and the SOURCE ICODE commands, respectively.

Programmers can set breakpoints at any source or TNS code location except non-exact points; therefore, most statements are valid breakpoint locations regardless of the accelerator option used. The RESUME AT command, however, can only be used if both the current and target locations are register-exact points.

In order to know which debugging commands are valid at the code location where execution is currently suspended, the programmer needs to know the accelerator state there. To get this information, the programmer can include a new token, called ACCELERATOR STATE, in the definition of the Inspect prompt or status line. When the prompt or status line is defined with this token, Inspect displays the current accelerator state. For example, after the command:

```
SET PROMPT = "[", ACCELERATOR
STATE ,"]"
```

each Inspect prompt for the debugging session will be one of the following, based on the current accelerator state:

- [Memory-exact]
- [Register-exact]
- [Non-exact]
- [ ]

Empty brackets indicate that TNS code is running. This happens if either a nonaccelerated program is running or the accelerated program is executing TNS code. An accelerated program executes nonaccelerated TNS code at the start of the program and when a transition to TNS code has occurred[5]. The examples in the remainder of this article assume that the programmer has set the prompt to display the accelerator state.

Table 3 summarizes the debugging capabilities available when the current program location is register-exact, memory-exact, or non-exact. Inspect issues warnings or error messages when requested commands are not available or might produce unexpected results because of the accelerator state of the current location.

[5]Occasionally the Accelerator cannot translate a section of TNS code to TNS/R instructions. When such points are reached during execution, the program makes a transition from TNS/R code to TNS code, after which it continues executing TNS code until the next procedure call or return that is a register-exact point.

Table 3.
Inspect debugging capabilities at exact points.

		Accelerated program		
Action	**TNS program**	**Register-exact**	**Memory-exact**	**Non-exact**
Add code breakpoint	Yes	Yes	Yes	Yes
Statement stepping	Yes	Yes	Yes	Yes*
Display variables	Yes	Yes	Yes	Yes**
Modify variables	Yes	Yes	Yes**	Yes**
Resume at	Yes	Yes***	No	No
Instruction stepping	Yes	No	No	No
Display TNS registers	Yes	Yes	Yes**	Yes**
Modify TNS registers	Yes	No	No	No

* To the next exact point.

** Inspect issues a warning message when the action might have no effect or the displayed values might not be up to date.

*** To a register-exact point.

Debugging Accelerated Programs Using Inspect

Debugging events suspend program execution. Code breakpoint events leave programs at memory-exact points or register-exact points. Data access breakpoint events, HOLD request events, and Debug process request events usually leave programs at non-exact points. When one of these events occurs, Inspect helps the programmer find the current program location, determine available debugging commands, and move to another location in the program.

Figure 6

Source code	TNS instructions			TNS/R instructions (ProcDebug)		
* #19 p := x - c;	%000011: >	LOAD	G+003	%h70420064: >	SUBU	s0,t4,t5
— #20 If x > 0	%000012:	LOAD	G+002	%h70420068:	BLEZ	t4,0x70420084
then	%000013:	ISUB		%h7042006C:	SH	s0,8($0)
#21 begin	%000014:	STOR	G+004			
	%000015:	LOAD	G+003			
	%000016:	CMPI	+000			
	%000017:	BLEQ	+003			
#22 call procx;	%000020: >	PCAL	002	%h70420070: >	JAL	PROCX
				%h70420074:	LI	a0,17

Figure 6.
Code block.

Finding the Current Program Location. Inspect reports the current location in a format determined by the setting of its LOCATION FORMAT parameter. The location formats are:

- Line
- Statement
- Line plus TNS offset
- Statement plus TNS offset

For TNS programs, Inspect reports the current location accurately when the location format is line plus offset or statement plus offset, because it indicates the exact spot in the executing TNS code where execution stopped.

Accelerated code consists of a sequence of blocks. A block of code always begins at a memory-exact point or a register-exact point. It consists of that point and any following non-exact points up to but not including the next memory-exact or register-exact point (or the end of the program). Figure 6 illustrates this. It contains two blocks of code separated by a horizontal line. Lines 19 and 22 are exact points. The first block consists of the code implementing lines 19 through 21. The second consists of the code for line 22. For each block, the corresponding source code, TNS code, and TNS/R code are listed from left to right. Inspect reports the current location accurately if and only if the currently executing TNS/R instruction is the start of a block. That is, for accelerated programs, Inspect reports the current location accurately if and only if the current accelerator state is memory-exact or register-exact.

If the program stops at the first TNS/R instruction in a block (an exact point), then Inspect reports the current location accurately. For example, if the program stops at location %h70420064 in the TNS/R code in Figure 6, then Inspect reports the current location and accelerator state as follows:

251,01,082 EXY1XL #MAINP.#19(EXY1)

[Memory-exact]

If the program stops at a TNS/R instruction within a block (a non-exact point), then the current location reported by Inspect is approximate. For example, if the program stops at %h70420068 in the TNS/R code, then Inspect will report the current location as the last exact point passed and will issue a warning as follows:

251,01,084 EXY1XL #MAINP.#19(EXY1)

**** WARNING 359 **** Current location is not a memory-exact point; displayed values may be out of date; the location reported is an approximate TNS location

[Non-exact]

When the accelerator state is non-exact, the program has stopped somewhere between the reported location (line 19) and the next exact point (line 22). Therefore, this program is suspended somewhere in the midst of executing lines 19, 20, and 21.

Determining Available Debugging Commands. Table 3 shows which debugging commands are valid for each accelerator state. In a save file the current location is likely to be at a non-exact point, especially in a save file created because the program abended. At non-exact points the values of variables updated in the current block of code are unpredictable, but the programmer can examine variables that are not updated by the current block. For example, if the program in Figure 6 is at a non-exact point, and the reported current location is line 19, then all the variables except *p* can be examined reliably, because *p* is the only variable modified by the statements on lines 19, 20, and 21. The value of *p* may or may not reflect the assignment on line 19, because the program stopped in the process of executing instructions for the statements on lines 19, 20, and 21.

Moving to Another Location in the Program. The BREAKPOINT and STEP commands move a program forward to a specific location. Inspect accepts breakpoint requests for lines, statements, and TNS code addresses when the specified location is memory-exact or register-exact. It rejects breakpoint requests at non-exact points, because there is no meaningful point in the TNS/R code that corresponds with the requested line, statement, or TNS code address. For example, a breakpoint request for line 20 in Figure 6 would produce the Inspect error message:

-EX1XL-break #20

**** ERROR 197 **** Location deleted by optimizations

Inspect refers to statement boundaries that are non-exact points as deleted. The function of the statement in this case has not been deleted. Rather, Inspect has deleted this location from the set of allowed breakpoint targets, because the underlying TNS/R code blurs the boundary between this and the previous statement.

The STEP command is valid for statements and verbs. It causes execution to proceed from the current location to the next statement or verb that is an exact point. To achieve finer granularity, such as setting breakpoints at TNS/R instructions, programmers must use Debug.

Figure 7

```
      18.000     a := b + c;

000005:000000:  @  LOAD   G+001    0x70420050:  LH     s0,2($0)
000006:000001:     LOAD   G+002    0x70420054:  LH     t5,4($0)
000007:000002:     IADD           0x70420058:  LH     t4,6($0)
000010:000003:     STOR   G+000    0x7042005C:  ADDU   s0,s0,t5
                                   0x70420060:  SH     s0,0($0)

      19.000     p := x - c;
      20.000     if x > 0 then

000011:000004:  >  LOAD   G+003    0x70420064:  SUBU   s0,t4,t5
000012:000005:     LOAD   G+002    0x70420068:  BLEZ   t4,0x70420084
000013:000006:     ISUB           0x7042006C:  SH     s0,8($0)
000014:000007:     STOR   G+004
000015:000010:     LOAD   G+003
000016:000011:     CMPI    +000
000017:000012:     BLEQ    +003

      22.000     call procx;

000020:000013:  >  PCAL   PROCX    0x70420070:  JAL    PROCX
                                   0x70420074:  LI     a0,17

      23.000     a :=p

000021:000014:  @  LOAD   G+004    0x70420078:  LH     s0,8($0)
000022:000015:     STOR   G+000    0x7042007C:  NOP
                                   0x70420080:  SH     s0,0($0)

   @  Register-exact
   >  Memory-exact
blank  Non-exact
```

Figure 7.
Examining TNS/R instructions with the ICODE command in APE.

Considerations for Using Inspect Commands. Inspect commands and their syntax are the same for accelerated and nonaccelerated programs. The default access type for data breakpoints is CHANGE under the C30 version of Inspect. This represents a change from the default of WRITE used under earlier WRITE versions of Inspect[6].

[6] Data breakpoints of type CHANGE occur only if the value of the variable has changed. Writes that store the same value already contained in the variable do not cause program execution to be suspended.

Some commands have new options to provide information about accelerated programs. The ACCELERATOR STATE token for the SET PROMPT and SET STATUS commands is such an option. Also, the output for some commands has been expanded for accelerated programs. For example, the SOURCE command output indicates the accelerator state for each source line listed. (See Figure 4.) There are also differences in the behavior of stepping and of data access breakpoints between TNS programs and accelerated programs. The *Inspect Manual* (1991) contains detailed descriptions of these differences and the new command options and output.

Debugging at the TNS/R Instruction Level

Although it is rarely necessary, programmers can use Debug and Inspect to debug an accelerated program at the TNS/R instruction level.

In Debug, the programmer can perform the following actions:

- Set breakpoints at TNS/R addresses.
- Display and modify the values of TNS/R registers.
- Display the TNS/R instructions for a specified address range
- Display corresponding blocks of TNS and TNS/R instructions.

In Inspect, the programmer can perform the following actions:

- Display and modify the values of the TNS/R registers.
- Display the TNS/R instructions.
- Display blocks of corresponding source, TNS, and TNS/R instructions.

Inspect creates save files that contain both the TNS state and TNS/R state of the program. Inspect does not allow setting breakpoints at TNS/R code addresses. However, the Inspect command SELECT DEBUGGER DEBUG allows the programmer to call Debug and use it to set TNS/R breakpoints. The *Debug Manual* (1991) and the *Inspect Manual* (1991) provide details of features for TNS/R instruction-level debugging.

A new tool, called the Accelerated Program Examiner (APE), allows the programmer to examine an object file that is not running. APE can display the mapping between source statements, TNS instructions, and TNS/R instructions, and it can provide other information about the accelerated object file. Inspect and Debug, which operate on running object files, provide some of the same information. For example, the APE command ICODE, the Inspect command ICODE, and the Debug command PMAP all display the mapping between TNS instructions and TNS/R instructions. Figure 7 provides an example of the output of the APE command ICODE.

In order to use the TNS/R debugging capabilities in these tools, the programmer must understand both the TNS and TNS/R instruction sets, TNS/R addressing, and how accelerated code is executed on the TNS/R system. Few programmers will need to debug at the TNS/R instruction level if they follow the recommendations in this article. In addition to the Debug and Inspect manuals already cited, Kane (1989), the *CYCLONE/R System Description Manual* and *APE Accelerated Program Examiner* (1991) provide information about TNS/R level debugging.

Conclusion

Programmers can debug both TNS programs and accelerated programs on TNS/R systems using the same tools and commands that are available on TNS systems. Current users of source-level debugging capabilities (where the unit of work is the statement, not the machine instruction) can continue to debug their programs with the same knowledge they must have on TNS systems, namely knowledge of their own source code. The programmer need not be concerned with the underlying TNS or optimized TNS/R instructions to isolate and analyze program bugs. Programmers who currently debug their programs using machine-level capabilities (where the unit of work is the TNS instruction) can continue to do so before accelerating them. In the rare event that the programmer needs to debug an accelerated program at the instruction level, Inspect, Debug, and a new tool, APE, provide the needed functions.

Acknowledgments

Special thanks to Kristy Andrews, Seth Hawthorne, Alan Rowe, Duane Sands, and Steven Watanabe for sharing their technical knowledge and answering my many questions about the Accelerator product and debugging programs on TNS/Rsystems. Thanks to Clark Gehrke for providing me with program examples for this article.

References

APE Accelerated Program Examiner. 1991. Tandem Computers Incorporated. Available in the APEDOC online file in distribution subvolume for product T9292.

CYCLONE/R System Description Manual. 1991. Tandem Computers Incorporated. Part no. 43572.

Debug Manual. 1991. Tandem Computers Incorporated. Part no. 56984.

Kane, G. 1989. MIPS RISC Architecture. Prentice-Hall, Inc.

Inspect Manual. 1991. Tandem Computers Incorporated. Part no. 57887.

Programmer's Guide for TNS/R Systems. 1991. Tandem Computers Incorporated. Part no. 63927.

Diane M. Cressler joined Tandem in May 1986. She is currently part of the Large Systems Marketing Support group, providing product support and introduction services. During the past year Diane co-developed and taught the TNS/R Migration Training course for beta customers and Tandem support personnel.

Measuring DSM Event Management Performance

Operations organizations must be able to monitor the current status of an online system in order to maintain high availability for the system's users. Tandem™ Distributed Systems Management (DSM) architecture and products can provide monitoring information to system operators by delivering and presenting accurate and timely events.

The challenge for Tandem system managers is to supervise both the monitoring information and the performance of the DSM event delivery tools. Delivering reliable information rapidly must be balanced against overtaxing the available system resources. Moreover, the event information must be limited to ensure that operators notice and respond quickly to important events.

Application developers and system managers can influence the effectiveness of DSM software at each stage of the event management process. By designing events carefully, application developers can reduce the number of extraneous events and provide accurate, useful information to operators. By configuring system resources to support the various components of DSM software, system managers can enhance the performance of event management. Proper resource planning also minimizes the CPU cost of event management, which lowers the cost of owning the equipment. Finally, by designing effective filters that eliminate insignificant events, system managers can reduce event noise, increase operators' productivity, and thereby increase system availability.

This article discusses the stages of event management in the Tandem DSM environment. It describes the functions and performance implications of Event Management Service (EMS), a component of the Guardian 90™ operating system, and the ViewPoint™ operations console facility. It also describes how to configure the components of EMS and ViewPoint to reduce the CPU costs of event management.

The article is intended mainly for system managers and performance analysts responsible for system resource planning. It assumes the reader is familiar with DSM. Overviews of DSM, EMS, and ViewPoint appear in the October 1988 issue of the *Tandem Systems Review* (Hansen and Stewart, 1988; Homan et al., 1988; and Jordan et al., 1988). It also assumes the reader understands performance modeling and is familiar with the Tandem Measure™ system performance measurement product.

Figure 1

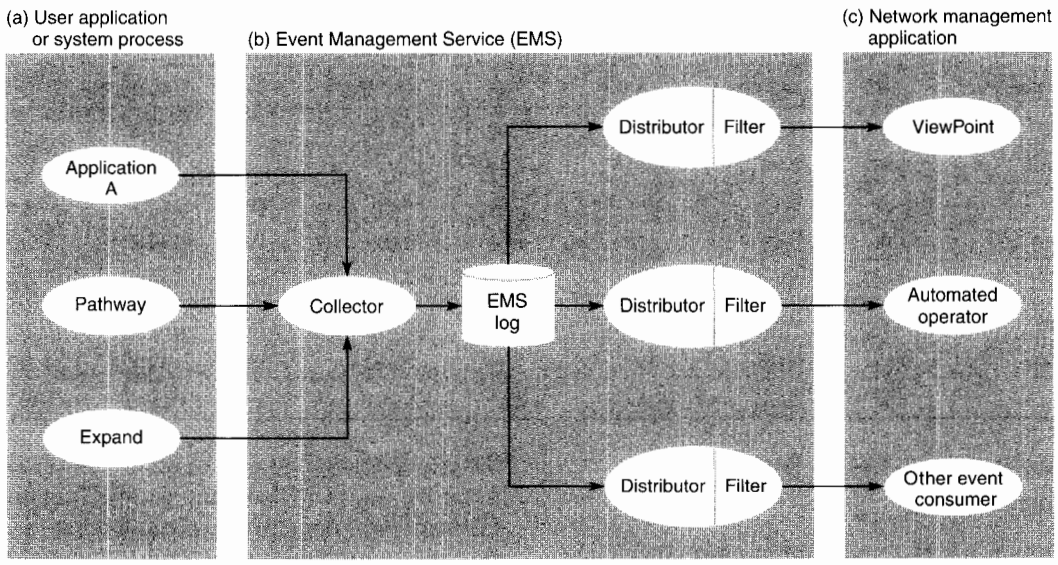


Figure 1.
Event management in the DSM environment. (a) A user application or system process performs event generation. (b) EMS performs event collection and distribution. (c) A network management application performs event consumption.

Event Management in the DSM Environment

Figure 1 shows the basic components of event management in the DSM environment:

- A system or application process generates the event and passes it to EMS.
- EMS collects and stores the event in an event log file called the EMS log.
- EMS filters (evaluates) the event. If the event meets the filter's selection criteria, EMS distributes it to event consumers.
- The event consumer uses the event data to perform system or network management tasks. For example, the ViewPoint product processes and presents the event to an operator.

Event management begins when a process generates an event to report a change or problem in the system or user application. For example, an event is generated when a terminal starts or a network communication line goes down. Either a Tandem subsystem or a user application can generate an event. Application programmers design and code the events generated by user applications. (In general, application programmers should not have to create events for errors that do not originate in the application.)

Events that contain useful diagnostic information increase the availability of the application by helping the operator to resolve problems rapidly. This benefit applies equally to human operators and automated operations software.

Moreover, the *event rate*, the number of events generated per second, can greatly affect the performance of event management software, and therefore of the entire system. Application developers can reduce the event rate by making sure that only one event is generated for each problem or change in condition (state change). Unnecessary events should be eliminated. Dagenais, in the October 1991 issue of the *Tandem Systems Review*, discusses how to design application events to support effective problem resolution.

Table 1.
CPU scaling information showing the relative performance capabilities of Tandem systems.

	Ratio							
CPU ratios	**NSII**	**CLX 600**	**CLX 700**	**TXP**	**CLX 800**	**VLX**	**Cyclone/R**	**Cyclone**
Cyclone	0.09	0.10	0.16	0.20	0.24	0.29	0.50	**1.00**
Cyclone/R	0.19	0.21	0.32	0.41	0.49	0.59	**1.00**	2.00
VLX	0.32	0.36	0.55	0.69	0.83	**1.00**	1.70	3.40
CLX 800	0.38	0.43	0.67	0.83	**1.00**	1.21	2.06	4.11
TXP	0.46	0.52	0.80	**1.00**	1.20	1.45	2.47	4.94
CLX 700	0.58	0.65	**1.00**	1.25	1.50	1.82	3.09	6.17
CLX 600	0.89	**1.00**	1.55	1.94	2.32	2.81	4.78	9.57
NSII	**1.00**	1.12	1.74	2.17	2.60	3.15	5.36	10.71

Resource Planning for Event Management

To support resource planning and performance tuning, the system manager or performance analyst must be able to predict the CPU cost of event management functions. These functions include event collection, distribution, filtering, consumption, processing, and presentation. With accurate estimates, the analyst can identify the CPU resources needed to support event management. The sizing information provided in this article can help the analyst to set up a balanced configuration of event management components, thus minimizing the impact of event management on system performance.

The formulas in this article identify atomic values for sizing event management functions. The atomic values quantify the CPU demand, or service cost, to process a single event. The analyst can then multiply the atomic demand by the expected arrival rate of the events to determine the total CPU demand for the function. Furthermore, once the total demand is identified, the analyst can estimate CPU capacity by dividing the target utilization of a CPU (for example, 75 percent busy) by the CPU demand.

Each formula is shown twice, first with generic terms and then with specific sample values. The sample values are based on a study that used Tandem's Measure product to gather performance data on a Tandem NonStop™ VLX™ system. The performance testing environment was based on a 4-processor VLX system, each configured with 16 megabytes of memory, using the C20 release of Guardian 90. The C21 releases of EMS and ViewPoint were used to develop the formulas.

To make estimates for a different Tandem system, the analyst can extrapolate from the VLX values shown here by using the matrix of relative performance capabilities shown in Table 1. For example, to estimate CPU costs on a Tandem CLX™ 800 system, the analyst can multiply the VLX values by 1.21.

In each column shown in Table 1, each system is scaled in relation to the one showing the baseline value of 1.00. The scalings are intended to be a rule of thumb[1].

[1]The validity of the scaling values is subject to a few constraints. First, sufficient memory must be configured so that the system does not incur swapping and no portion of disk cache is confiscated by the Guardian 90 memory manager. Second, no more than two mirrored disk devices should be configured per pair of disk controllers. Third, the Tandem Cyclone™ system must be configured with two I/O subsystems (four I/O channels) per CPU.

The analyst will have to supply certain values specific to the user environment. For example, the analyst must know how many EMS distributors use an EMS collector log. Furthermore, he or she must be able to evaluate the filters that restrict the number of events passed to the event consumers. In addition, the analyst must determine the event rate (for event generation and consumption) by gathering performance data with the Measure product and viewing Measure reports.

The analyst should use these formulas with caution. The results of the VLX performance study are approximations derived from a controlled environment; their accuracy, applied to a user system, may vary. The values derived from the VLX system do not include operating system overhead costs such as interrupts or message handling. A conservative guideline for calculating these additional system costs is to add 20 percent to the results of the formulas.

EMS Event Collection and Storage

The EMS collector receives events generated by Tandem subsystems and application processes and stores them in the EMS log. The generating process constructs the event by using a procedure to initialize the event buffer and add the tokens required by EMS. Optionally, it can issue a series of calls to add tokens and token values to the event buffer. When the event is constructed, the generating process issues a call to forward it to EMS.

The EMS collector and the EMS log disk process manage event collection. When the EMS collector receives the event, its tasks involve:

- Sending a checkpoint message to the backup EMS collector.
- Replying to the process that generated the event.
- Inserting a logging timestamp into the event.
- Sending the event to the EMS log disk process.
- Sending a confirmation to the backup EMS collector.

Figure 2

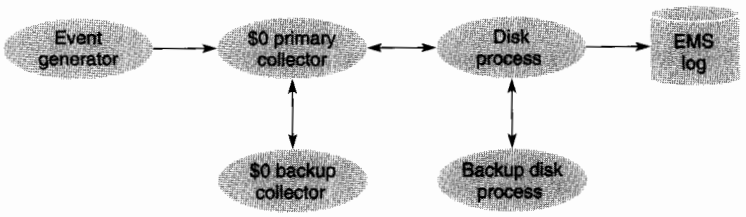


Figure 2.
The EMS event collection process.

Figure 2 shows the processes involved in EMS event collection. The primary EMS collector ($0 process) uses low-level Guardian 90 message system services to communicate with its backup process and the EMS log disk process. To distribute the CPU consumption of event collection to different CPUs, the system manager can add alternate EMS collectors to which applications report events. An alternate collector operates just as the primary collector does, but it uses less efficient Guardian 90 file system calls to communicate with its backup process and the disk process.

Design Goals of Event Collection

The EMS collection function meets several performance-related design goals. First, event collection does not depend on event distribution because the EMS collector and EMS distributors can use the disk-based queue (EMS log) asynchronously. Thus, when an event consumer performs slowly, it has no impact on the performance of the event generator.

Figure 3

$$Eg = Er \times (Ge + (Nt \times Gat) + Ce + Dw + (Nd \times Dwr))$$

The VLX performance study produced the following results:

$$Eg = Er \times (5\ ms + (Nt \times 0.4\ ms) + 4\ ms + 4\ ms + (Nd \times 7.5\ ms))$$

where

Eg	=	CPU cost, in CPU ms/sec, of generating and collecting events
Er	=	system event rate in events/sec
Ge	=	CPU time, in ms, the source process uses to generate an event
Nt	=	number of optional tokens added to the event by the event generator
Gat	=	CPU time, in ms, the source process uses to add a token to the event
Ce	=	CPU time, in ms, the primary and backup collector processes use to log the event*
Dw	=	CPU time, in ms, the disk process uses to write the event to the EMS log
Nd	=	number of EMS distributors using the collector's event log
Dwr	=	additional CPU time, in ms, the collector uses to service an event notification request from a distributor. Use this value only at lower event rates.

* On a VLX system, the primary collector ($0) uses approximately 2.5 ms/event. The backup collector process uses 1.5 ms/event.

Figure 3.
Formula for estimating the CPU cost of EMS event collection.

Second, the EMS collector ensures the integrity of event data by sending a checkpoint to its backup process when it receives an event, before it replies to the event generator. If the primary collector process fails, the event data is saved in the backup process.

Third, by replying promptly to the event generator, the EMS collector minimizes its impact on the event generator's performance. The event generator can continue processing without having to wait for the event to be written to the EMS log.

Fourth, direct communication between the collector and distributors reduces the demand on the EMS log disk process. Instead of burdening the EMS log disk process with excessive read requests (polls) after it reaches an end-of-file condition on the EMS log, the distributor sends a request to the collector, asking to be notified of a new event.

Calculating the CPU Cost of EMS Event Collection

By using the formula shown in Figure 3, the analyst can calculate the CPU cost (in milliseconds per second) of EMS event collection. One can use this information to estimate the CPU resources needed for EMS event collection and to predict required system capacity.

Figure 3 shows the generic formula and the VLX performance test results. These results supply specific values for the CPU resources needed by the following processes to collect one event:

- The subsystem or application process that generates the event (*Ge* and *Nt* x *Gat*).
- The primary and backup EMS collector processes (*Ce* and *Dwr*).
- The primary and backup EMS log disk processes (*Dw*).

To complete the calculation, the analyst must determine the event rate in events per second (*Er*), the number of optional tokens added to the event by the event generator (*Nt*), and the number of EMS distributors using the collector's EMS log (*Nd*).

Determining the System Event Rate

To determine the system event rate (*Er*) for event collection, one should understand how the primary EMS collector ($0) processes the event.[2] The $0 process uses Guardian 90 message system services both to log the event (one message) and to notify its backup process of current status information (two messages).

[2] It is difficult to determine how many events the $0 process is processing because it communicates with the EMS log disk process through message system facilities rather than standard file system calls. The FILE counter of the Measure product collects data on file activity only by counting file system procedure calls. Also, while the $0 process accepts events on its $RECEIVE file, the EMS distributors also send other event-related requests to the $0 $RECEIVE file. Given these considerations, one cannot determine event rates by counting messages on the $0 $RECEIVE file.

Thus, the primary $0 process sends three messages per event, and its backup process receives two messages per event. The analyst can determine the event rate by viewing the Measure PROCESS report for the $0 process. One can either divide the number of messages sent by the primary $0 process by three or divide the number of messages received by the backup $0 process by two.

For the alternate collector, the best way to calculate the event rate is to use the Measure FILE entity on the alternate collector's event log. One can examine the Writes counter to determine the number of write operations from the alternate collector to the EMS log file. A second method is to divide the number of messages received by the backup alternate collector process by two (just as for the backup $0 process).

Determining When to Use Alternate Collectors

Using alternate collectors can improve system performance when the overall system event rate exceeds the capacity of the primary EMS collector or the primary EMS log disk process. Alternate collectors allow one to distribute the event collection load to other CPUs.

Performance Impact on the EMS Log Disk Process. An EMS collector issues one write request to the EMS log to store an event. Each EMS distributor also issues I/O requests to the EMS log. At lower event rates, each EMS distributor issues two I/O requests to the EMS log for each event. (The first I/O results in a successful read of an event record; the second I/O results in an end-of-file condition.) In systems with low event rates and many EMS distributors, this I/O activity can increase the CPU cost of the EMS log disk process.

To avoid the potential impact on system performance under these conditions, the analyst can distribute the event processing load among several collectors that use EMS logs on different CPUs.[3] The analyst should, however, weigh the benefits of using alternate collectors against the potential complexity of configuring and managing multiple collectors and distributors.

The Cost of the Alternate Collector. An alternate collector (the EMSACOLL process) consumes more CPU resources in processing events than does the primary EMS collector (the $0 process). For a VLX system, one should use a value of 19.4 milliseconds per event to estimate the CPU cost of the alternate collector (shown as *Ce* in Figure 3). The primary process of the alternate collector consumes 14.5 milliseconds per event. Its backup process consumes 4.9 milliseconds per event.

[3] With a recent release of the EMS distributor, interim product maintenance (IPM) AAG, the distributor invokes a delay parameter detecting an end-of-file condition. It waits 500 milliseconds and then retries the read on the EMS log file. If this read also results in an end-of-file condition, the distributor contacts the EMS collector for new event notification. The 500 milliseconds is a default value that can be modified through a Subsystem Programmatic Interface (SPI) command.

Figure 4

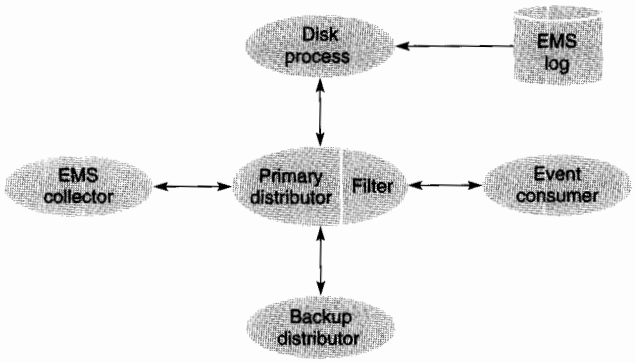


Figure 5

***Ed* = *Er* x ((2 x *Dr*) + *Ef* + *Fe* + (*Pr* x *Ec*))**

The VLX performance study produced the following results:

***Ed* = *Er* x (4.0 ms + 13.1 ms + *Fe* + (*Pr* x 9.6 ms))**

where

Ed	=	CPU cost, in CPU ms/sec, of filtering and distributing events
Er	=	collector (system) event rate, in events/sec
Dr	=	CPU time, in ms, the EMS log disk process uses to read the EMS log*
Ef	=	CPU time, in ms, the EMS distributor uses to fetch an event from the EMS log and set it up for filter evaluation
Fe	=	CPU time, in ms, the EMS distributor uses to perform a filter evaluation of the event
Pr	=	percentage of events that pass the filter evaluation
Ec	=	CPU time, in ms, the EMS distributor uses to forward the filtered event to the consumer

* This value is multiplied by 2 to include a successful read of an event followed by an end-of-file condition.

Figure 4. *The EMS event distribution process.*

Figure 5. *Formula for estimating the CPU cost of EMS event distribution.*

EMS Event Filtering and Distribution

After the EMS collector stores an event in the EMS log file, the EMS distributor filters and distributes the event. To filter an event, the EMS distributor evaluates its contents to see if it meets the selection criteria of the event filter. The filter is written by the user in the EMS filter language, compiled, and loaded into the EMS distributor. If the event satisfies the filter criteria, the EMS distributor forwards it to an event consumer. An event consumer is any user-written management application or Tandem DSM product requesting events from the EMS distributor.

Figure 4 shows the processes involved in event distribution. Effectively, distribution begins when the event consumer requests an event from the EMS distributor.[4] The EMS distributor performs the following tasks:

- It receives an event request from the event consumer.
- It retrieves the next event from the EMS log disk process, which reads the EMS log.
- It sends a checkpoint message to the backup EMS distributor.
- It evaluates the event contents by using the filter.
- If the event passes the filter evaluation, it forwards the event to the consumer.
- If the event fails the filter evaluation, it requests the next event from the EMS log disk process.
- If an end-of-file condition on the EMS log is encountered, it requests notification of a new event from the EMS collector.

[4] Once the EMS distributor has had its filter defined by the event consumer, it immediately issues a read request on the EMS log. As soon as the event is returned to the event consumer, another read on the log occurs. This overlapping of read requests on the EMS log before the event consumer's next request for an event provides a better response time to the event consumer.

Calculating the CPU Cost of EMS Event Distribution

Figure 5 shows a formula for calculating the CPU cost of EMS event distribution. The results of the VLX performance study supply specific CPU costs for the operations listed below. The EMS log disk process reads the EMS log; the EMS distributor performs the other operations listed below:

- Read the EMS log (*Dr*).
- Fetch the event and prepare it for filter evaluation (*Ef*).
- Evaluate the event contents by using the filter (*Fe*).
- Forward the event to the event consumer (*Ec*).

The formula accounts for two read operations on the EMS log, one to read an event record and a second to encounter an end-of-file condition. The end-of-file read occurs when the EMS distributor processes events more quickly than the system generates them.

The analyst must determine the EMS collector event rate (*Er*). This is the same value as the event rate (*Er*) shown in Figure 3. In addition, the analyst must determine the CPU costs of filter evaluation (*Fe*) and the percentage of events that pass the filter criteria (*Pr*). The following section, on filter evaluation, describes how to determine (*Fe*). To determine (*Pr*), the analyst can use the Measure PROCESS entity and divide the number of messages received by each of the distributors by the system event rate.

Filter Evaluation

The CPU cost of filter evaluation (*Fe*) is the most significant performance issue in event distribution. Filtering can greatly reduce the number of events forwarded to event consumers, which has two main benefits. First, event consumers such as the ViewPoint product perform better because they have fewer events to process. Second, operators (human or automated) can identify and resolve problems quickly because they are not inundated with irrelevant event messages and can focus on significant events. However, if a filter is not designed properly, filter evaluation can consume a great deal of CPU resources. Therefore, analysts must weigh the benefits of filtering against its potential impact on performance. A well-designed filter can accomplish the objectives of filtering while minimizing its impact on the performance of event distribution.

Several factors affect the CPU cost of filter evaluation:

- The event rate. (This affects all aspects of event distribution.)
- The number of tokens being tested in the event.
- The total length of the event message and the placement within the event of the tokens being tested.
- The type of testing. For example, the EQUALS test, which compares token values, executes more quickly than the MATCH test, which compares text strings.
- The data type being tested. For example, an integer data type is tested more quickly than a text data type.
- The number of nesting levels in the filter.
- The sequence of testing specified by the filter.

Figure 6

SSID test:	IF SSID = some^ssid THEN . . .
Event number test:	IF EVENTNUMBER = some^event^number THEN . . .
Text match test:	IF MATCH(some^text, match^text) THEN . . .
Sender ID test:	IF SENDERID = some^process THEN . . .
Event subject test:	IF EVENTSUBJECT = some^subject THEN . . .
Event flag tests:	IF EMPHASIS^TKN = true THEN . . .
	IF ACTION^NEEDED = true THEN . . .

Figure 6.
Common filter statement primitives.

Clearly, the structure of a filter can affect the performance of filter evaluation. In order to understand the structure of a filter, one can break it down into individual filter statements, or primitives. Figure 6 shows some common filter primitives.

These primitives are combined to establish evaluation blocks. An evaluation block is a set of Boolean-operator linked tests performed on a token value. For example, Figure 7 shows a common evaluation block, which tests a series of event numbers for a specific subsystem ID (SSID).

To construct a filter, one can nest evaluation blocks within IF statements. A completed filter can combine evaluation blocks in various ways. Figure 8 shows an example of a simple filter that combines different evaluation blocks for events generated by three different subsystems (applications). In addition, the sample filter passes all events containing action-needed tokens and sets Viewpoint display attributes.

The EMS distributor executes the compiled filter code by fetching the token and comparing the token value to the value specified in the filter. For each statement in the filter, the EMS distributor fetches the appropriate token and tests it. Most filters use nesting; if the first-level test is passed, a second level of testing takes place. To enhance the performance of filter evaluation, the EMS distributor caches tokens in memory after they are extracted from the event. If another test invokes the same token, the distributor uses its cached token to avoid issuing a token get call.

Determining the CPU Cost of Filter Evaluation

To determine the CPU cost of filter evaluation (Fe), the analyst must know the filter as well as the values (CPU time in milliseconds) of the various filter primitives. Measurements taken on a VLX system showed the CPU costs of the following filter primitives:

- SSID tests cost 0.2 milliseconds each.
- Event number tests cost 0.3 milliseconds each.
- Text match tests cost 0.3 milliseconds plus 0.008 milliseconds per byte tested.[5] On a 100-byte text string where no match occurs, this results in 1.1 milliseconds per test.
- Sender ID tests cost 0.7 milliseconds each.
- Event subject tests cost 0.6 milliseconds each.
- Emphasis tests cost 0.2 milliseconds each.

By applying these numbers to the sample filter shown in Figure 8, one can determine the CPU cost of evaluating particular events. For example, evaluating an event containing SSID APPL.104.0 would cost 0.8 milliseconds. This evaluation involves the entire first level of the filter: three SSID tests plus one event flag test. The CPU cost would be the same whether or not the event contained an action-needed flag and was passed to the consumer.

[5] Using wild cards in string matching does not significantly affect the cost.

Evaluating an event containing SSID APPL.102.0 and event number 2 would cost 1 millisecond. This evaluation involves two SSID tests and two event number tests.

Guidelines for Constructing Filters

The filter evaluations discussed in the above example show how important it is to construct a filter that will evaluate events efficiently. First, the filter execution time is affected by the order of the token tests within the filter. If a critical token is tested at the end of the filter, the entire first level of the filter must execute before an event containing that token can be evaluated. For events that occur frequently, this can greatly increase the overall execution time of the filter.

Therefore, token value tests for events that occur most often should be placed at the beginning of the filter. The filter writer should follow the same guideline at all levels of nesting. For example, for events containing a particular SSID, one might test first for the event number that occurs most often.

The same principle applies to events that do not contain any tokens tested by the filter. When these events are evaluated, the entire first level of the filter will execute. If these events occur frequently, it may be more efficient to insert statements into the filter that will identify and explicitly fail these events.

Matching text strings uses substantially more CPU time than testing integer values. Therefore, one should avoid the MATCH verb. One should, instead, test structured tokens such as integers, file names, or SSIDs whenever possible.

Real filters are much more complex than the example shown in Figure 8. The analyst can scan event logs to determine the frequency of specific events. On the basis of this information, the analyst can construct a filter that responds efficiently to the specific characteristics of the user environment.

Figure 7

```
IF SSID = selected^ssid THEN
  BEGIN
  IF   (EVENTNUMBER = first^event^number OR
        EVENTNUMBER = second^event^number OR
        . . .
        EVENTNUMBER = nth^event^number)        THEN PASS
                                               ELSE FAIL
  END;
```

Figure 8

```
FILTER EXAMPLE^FILTER;
BEGIN
(Fail events from this talkative subsystem.)
IF ZSPI^TKN^SSID = SSID (APPL.101.0) THEN FAIL;
(Pass specific event numbers from this subsystem.)
IF ZSPI^TKN^SSID = SSID (APPL.102.0) THEN
  BEGIN
  IF (EVENTNUMBER = 1                  OR
      EVENTNUMBER = 2                  OR
      MATCH (TEXT,first^text)      OR
      MATCH (TEXT,second^text)        )
                                       THEN PASS
                                       ELSE FAIL;
  END;
(Check only for critical events from this subsystem.)
IF ZSPI^TKN^SSID = SSID (APPL.103.0) THEN
  BEGIN
  IF EMPHASIS = true THEN PASS 2
                     ELSE FAIL;
  END;
(Pass all events with action needed token. Reverse video)
(for action requests, normal video for action completions.)
IF TOKENPRESENT (ACTION^NEEDED) THEN
  BEGIN
  IF ACTION^NEEDED = true THEN PASS 1
                          ELSE PASS;
  END;
END;
```

Figure 7.
Example of combining filter primitives to create an evaluation block. This example shows an SSID-event number block.

Figure 8.
Example of a simple filter that combines evaluation blocks.

The Effect of Event Rates on the EMS Sizing Formulas

The formulas for estimating event collection and distribution costs are accurate as long as the event rate and system load do not exceed a certain limit. The formulas assume that the EMS distributor processes events more quickly than the events are stored on the EMS log.

Because the arrival of the next event takes longer than filter evaluation at these lower event rates, the distributor executes a second, unsuccessful read of the EMS log after each successful retrieval of an event. The term (2 x *Dr*) in the distribution formula shown in Figure 5 accounts for this. The end-of-file condition on the EMS log triggers the distributor's event request to the EMS collector. The term *Dwr* in the collection formula shown in Figure 3 accounts for this.

If the event rate rises or the execution of the distributor slows down, the above assumption is no longer valid. In this case, the distributor never reaches the end of the EMS log. For each event it retrieves, it sends only a single read request to the EMS log, and it does not send an event request to the EMS collector.

Many factors can slow down the execution of the distributor. These include a heavy system load, a low CPU priority for the distributor, a complex filter, and the addition of other requesters or distributors requesting services from the EMS log disk process.

Event Processing and Presentation by ViewPoint

When an event passes the defined filter conditions, the EMS distributor forwards it to an event consumer. The consumer is not a part of EMS. It can be a user-written management application or a Tandem DSM product such as the Distributed Systems Management/Problem Management (DSM/PM), which performs problem tracking, or the Programmatic Network Administrator (PNA) automated operations software, which provides rules-based, programmed responses to system problems.

The ViewPoint product is currently the most widely used DSM tool for event monitoring. This article discusses ViewPoint event consumption and provides sizing information to help the analyst predict the CPU costs of ViewPoint functions. The article assumes that one is using the C21 version of ViewPoint. One can divide ViewPoint event consumption into two steps: event processing and event presentation.

Event processing involves obtaining the event from the EMS distributor and preparing it for presentation. ViewPoint servers perform this work.

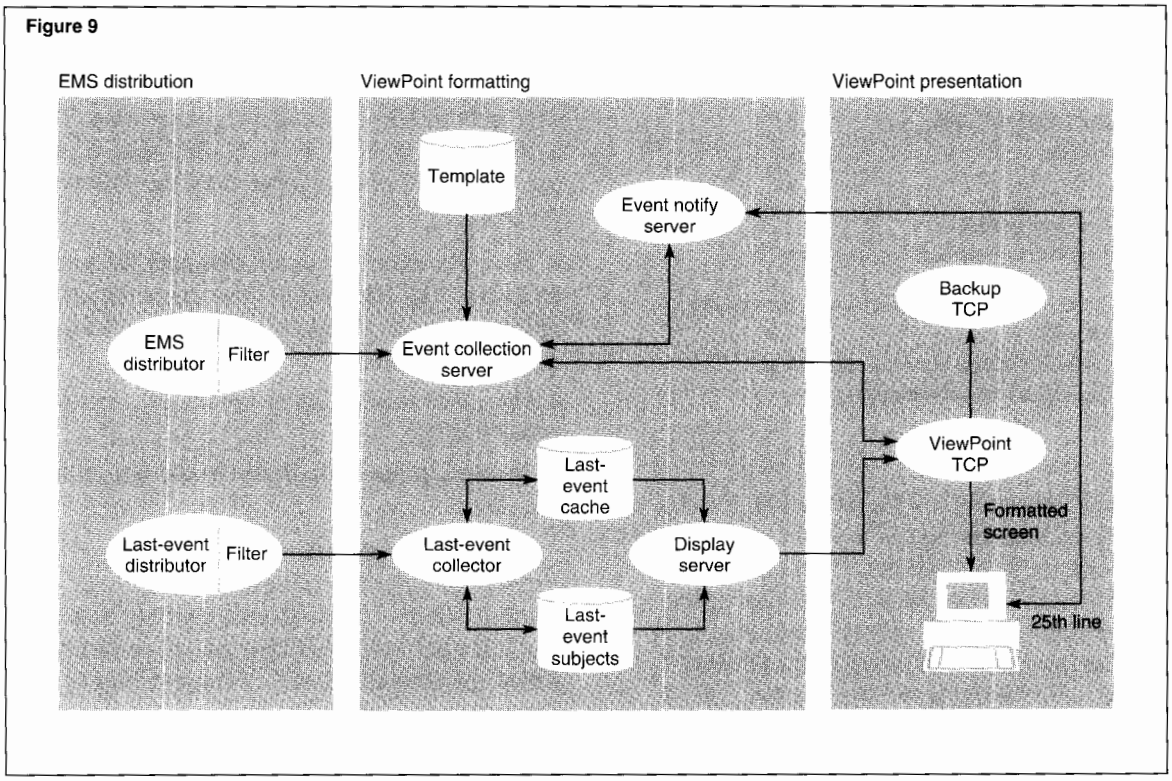


Figure 9.
Event processing and presentation by ViewPoint.

Event presentation involves constructing the event display and writing it to the operator terminal. The terminal control process (TCP), part of Tandem's Pathway distributed transaction processing system, performs this work.

Figure 9 illustrates the processes involved in event processing and presentation with ViewPoint. ViewPoint servers collect events from the EMS distributors and format the events. The ViewPoint servers then notify the ViewPoint TCP, which requests the events from the servers, constructs the screen displays, and sends them to the appropriate operator consoles.

Event Processing

In event processing, the ViewPoint event collection server obtains an event by sending an event request to the EMS distributor. The collection server formats the text for display. Next, the server stores the event in memory for each TCP thread (for each primary and alternate event display that requires this event). If an event is marked as critical, the collection server sends it to the event notify server. The event notify server writes the critical or action message directly to the 25th line of the operator console; it does not use the ViewPoint TCP to display events.

Figure 10

$$Ep = Fr \times (Cp + (Cr \times (Cn) + (Dr))$$

The VLX performance study produced the following results:

$$Ep = Fr \times (14.5\ ms + (Cr \times (10\ ms) + 4\ ms))$$

where

Ep	=	CPU cost, in CPU ms/sec, of event processing
Fr	=	filtered event rate, the sum of the events forwarded by all distributors, in events/sec
Cp	=	base CPU time, in ms, the collection server uses to collect and format an event
Cr	=	percentage of critical events received
Cn	=	base CPU time, in ms, the notify server uses to process a critical event
Dr	=	CPU time, in ms, the disk process uses to read the template file

Figure 10.
Formula for estimating the CPU cost of ViewPoint event processing.

The C21 version of ViewPoint allows the user to take advantage of a performance option that lowers the overall cost of event processing. In previous versions of ViewPoint, the ViewPoint collection server stored events in the display cache and event terminal files. These were then read by the display server and prepared for display. The C21 version of ViewPoint removes much of the disk access from event processing by retaining the current event list in the collection server's memory. Thus, the ViewPoint TCP can use the collection server to provide events for presentation. With this option, the display server is no longer involved in event processing.

Calculating the CPU Cost of ViewPoint Event Processing

Figure 10 shows a formula for estimating the CPU cost of ViewPoint event processing. The VLX performance study provided specific values for operations performed by the following processes:

- The collection server fetches and processes an event (*Cp*). The collection server incurs additional costs during event presentation, as described later.
- The event notify server processes a critical event (*Cn*).
- The disk process reads the event template file (*Dr*).

The analyst must determine the filtered event rate (*Fr*), which includes the sum of events collected from all related distributors, in events per second. The analyst can obtain *Fr* by viewing either one of these Measure report items:

- The number of messages received by the EMS distributors (the Measure PROCESS entity).
- The number of requests issued by the ViewPoint collection server to the EMS distributors (the Measure FILE entity).

When calculating the value of *Fr*, the analyst must include both the primary distributor and all active alternate distributors. This cost is incurred whether or not the ViewPoint alternate event displays are active.

In addition, the analyst must determine the percentage of events received by the collection server that are critical (*Cr*). Critical events add to the CPU cost because they require additional processing by the event notify server. To determine *Cr*, the analyst can use Measure to calculate the number of messages received by the event notify server.

Last-Event Processing

In addition to displaying the most recent events, ViewPoint provides a last-event function, which allows the operator to view system events that concern a particular event subject. To plan CPU resources to support ViewPoint, the analyst must include the CPU costs associated with the processing of last events, whether or not the last-event display functions are invoked.

The last-event collection server functions the way the collection server did in versions of ViewPoint prior to C21. It uses its own EMS distributor and a user-defined filter to store events by subject. For each filtered event it receives, the last-event collection server sends read and write requests to the last-event cache file and the last-event subject files.

Calculating the CPU Cost of Last-Event Processing

Figure 11 shows a formula for estimating the CPU cost of ViewPoint last-event processing of one event. The VLX performance study provides specific values for the CPU costs to support the last-event collection server process (*Lc*) and last-event disk process (*LDp*).

Figure 11

***Lp* = (*Fr* x (*Lc* x *LDp*))**

The VLX performance study produced the following results:

***Lp* = (*Fr* x (29 ms + 15 ms))**

where

Lp	=	CPU cost, in CPU ms/sec, of last-event processing
Fr	=	filtered event rate in events/sec
Lc	=	CPU time, in ms, the last-event collection server uses to process the event
LDp	=	CPU time, in ms, the last-event collection disk process uses to read and update the event files

The analyst must determine *Fr*. This event rate is based on the frequency of filtered events delivered by the last-event collection server's EMS distributor. The analyst can obtain this value by viewing the Measure FILE requests on the last-event distributor.

Figure 11.
Formula for estimating the CPU cost of ViewPoint last-event processing.

Figure 12

$$Cd = (Cf + ((On - 1) \times Cad)) \times Rr$$

The VLX performance study produced the following results:

$$Cd = (238.1 \text{ ms} + ((On - 1) \times 41.0 \text{ ms})) \times Rr$$

where

Cd = CPU cost, in CPU ms/sec, of the collection server to support primary display processing by ViewPoint

Cf = CPU time, in ms, the collection server uses to process the first primary display

On = number of active operators viewing the primary display

Cad = CPU time, in ms, the collection server uses to process additional primary displays

Rr = user-configured ViewPoint refresh rate, in refreshes per second

Figure 12.
Formula for estimating the CPU cost of the ViewPoint collection server to present primary displays.

Event Presentation

In event presentation, ViewPoint displays an up-to-date list of events on the operator's console. The display shows the last 16 events received within a user-defined interval. The default interval is 10 seconds.

First, the collection server receives an event intended for a particular operator terminal (either a primary or alternate display). (See Figure 9.) The collection server then places an unsolicited message processing (UMP) message on the TCP's terminal thread.

Asynchronously, the TCP delays the specified refresh interval and then checks to see if there is an UMP message outstanding. If there is, the TCP replies to the UMP message and requests the list of events directly from the collection server.

When sizing event presentation, the analyst must distinguish between primary and alternate display presentation. When multiple operators view a primary display, they all use one filter and EMS distributor. Thus, they share one event list, which lowers the processing costs. However, each alternate display is processed independently, using its own filter and EMS distributor.

Calculating the CPU Cost of Primary Displays

Event presentation of a ViewPoint primary display involves two steps:

- The collection server processes the event before the display.
- The ViewPoint TCP displays the formatted list of events on the terminal.

Figure 12 shows a formula for estimating the CPU cost incurred by the collection server to process a primary display. After it processes the initial display (*Cf*), the collection server incurs a small processing cost (*Cad*) for each additional active operator viewing the primary display.

Figure 13 shows a formula for estimating the CPU cost incurred by the ViewPoint TC to process a primary display. The TCP incurs a cost for the initial primary display (*Td*) and each additional display (*Tn*). The backup TCP also affects the cost of TCP display processing (*Tb*).

The VLX performance study provides specific values for *Cf*, *Cad*, *Td*, *Tn*, and *Tb*. (See Figures 12 and 13.) The analyst must know the number of active operator terminals (*On*) and the ViewPoint refresh rate (*Rr*).

Assume, for example, that two operators are viewing the primary display and the refresh rate has the default value of 0.1 (the event list is refreshed every 10 seconds). Figure 14 shows how to calculate the CPU cost of event presentation in this example.

These calculations assume that at least one filtered event will arrive within the refresh interval. If no events pass through the collection server during this time, no costs for event presentation occur.

The analyst should also consider the CPU cost of the I/O process that drives the terminal. The event presentation formulas do not include this cost. For TCP TERMPROCESS configurations on a VLX system, the cost is 28 milliseconds per display or 28 milliseconds x *On*. For Tandem 6100-type communications subsystems (terminal controllers) on a VLX system, the cost is 45.3 milliseconds per display or 45.3 milliseconds x *On*.

Calculating the CPU Cost of Alternate Displays

Sizing Viewpoint's alternate display processing is similar to sizing for the primary display. The TCP costs are the same. Therefore, one can use the formula in Figure 13 to calculate the TCP costs of alternate display processing.

Figure 13

$$Tp = (Td + (Tn \times (On - 1)) + (On \times Tb)) \times Rr$$

The VLX performance study produced the following results:

$$Tp = (105.7\ ms + (45.3\ ms \times (On - 1)) + (On \times 10.2\ ms)) \times Rr$$

where

- *Tp* = CPU cost, in ms/sec, of the ViewPoint TCP to support primary display processing by ViewPoint
- *Td* = CPU time, in ms, the TCP uses to present a single primary event display
- *Tn* = CPU time, in ms, the TCP uses to present each additional primary event display
- *On* = number of active operators viewing the primary display
- *Tb* = CPU time, in ms, the backup TCP uses to process each ViewPoint primary event display
- *Rr* = user-configured ViewPoint refresh rate, in refreshes per second

Figure 14

Assume that two operators view the primary display and the default refresh rate = 0.1.

CPU cost of primary display processing:

$Cd = (238.1\ ms + ((2 - 1) \times 41.0\ ms \times 0.1 = 27.91\ ms/sec$

and

$Tp = (105.7\ ms \times 2) + (10.2 \times 2)) \times 0.1 = 23.18\ ms/sec$

Total CPU cost for primary display (*Pd*) processing:

$Pd = 51.09\ ms/sec$

Figure 13

Formula for estimating the CPU cost of the ViewPoint TCP to present primary displays.

Figure 14.

Sample CPU cost of ViewPoint primary display processing on a VLX system. These calculations use the formulas shown in Figures 12 and 13.

Figure 15

$$Ca = (Cf + ((On - 1) \times Cad)) \times Rr$$

The VLX performance study produced the following results:

$$Ca = (238.1 \text{ ms} + ((On - 1) \times 204.4 \text{ ms})) \times Rr$$

where

Ca	=	CPU cost, in ms/sec, of the collection server to support alternate display processing
Cf	=	CPU time, in ms, the collection server uses to process the first alternate display
On	=	number of active operators viewing the alternate display
Cad	=	CPU time, in ms, the collection server uses to process additional alternate displays
Rr	=	user-configured ViewPoint refresh rate, in refreshes per second

Figure 15.
Formula for estimating the CPU cost of the ViewPoint collection server to present primary displays.

However, with primary displays, the cost of supporting multiple displays is relatively small. With alternate display processing, the collection server treats each display independently. Each display has its own memory cache, filter, and EMS distributor. Thus, each additional display costs about the same as the initial one. Figure 15 shows a formula for estimating the CPU cost of processing alternate displays.

The Impact of Event Displays on ViewPoint Performance

The number and type of event displays affect the performance of ViewPoint. When several operators se the primary display, the CPU cost of event processing remains fixed. However, when additional alternate displays are configured, the CPU cost of event processing rises, because the event collection server must handle additional events from more EMS distributors.

In previous releases of ViewPoint, these costs clustered in the same CPU because the EMS distributors were placed in the collection server's CPU. In versions of ViewPoint after C21, starting with interim product maintenance (IPM) AAG, the analyst can designate the CPU in which an EMS distributor will execute.

The analyst can also configure multiple ViewPoint environments to handle different functional areas such as telecommunications, the host system, and application events. Or one can divide the environments according to management responsibility, assigning, for example, specific network nodes to specific ViewPoint environments.

The CPU cost of event presentation varies according to the frequency of the terminal display. Fortunately, the CPU cost is spread over the duration of the display interval. Also, the event list is redisplayed only if a new event has arrived (has been filtered) during the display interval. The analyst can increase the default value of 10 seconds, which will lower the impact on system performance by spreading the CPU cost over a longer interval. This is a useful alternative if the operator can perform well with less frequent event notification.

Conclusion

DSM products such as EMS and ViewPoint provide an environment that supports effective event management. By designing events that provide information efficiently, application developers can enhance the effectiveness of event management. By applying the formulas in this article to their installations, system managers can accurately configure the Tandem software tools that collect, distribute, process, and present events, thus enhancing the performance of event management.

References

Dagenais, J. 1991. Instrumenting Applications for Effective Event Management. *Tandem Systems Review*. Vol. 7, No. 2. Tandem Computers Incorporated. Part no. 65248.

Hansen, R. and Stewart, G. 1988. VIEWPOINT Operations Console Facility. *Tandem Systems Review*. Vol. 4, No. 3. Tandem Computers Incorporated. Part no. 15748.

Homan, P., Malizia, B., and Reisner, E. 1988. Overview of DSM. *Tandem Systems Review*. Vol. 4, No. 3. Tandem Computers Incorporated. Part no. 15748.

Jordan, H., McRee, R., and Schuet, R. 1988. Event Management Service Design and Implementation. *Tandem Systems Review*. Vol. 4, No. 3. Tandem Computers Incorporated. Part no. 15748.

Acknowledgments

Special thanks are due to Art Sheehan for providing the CPU scaling information shown in Table 1.

Mark Stockton has worked with Tandem as a consultant for over six years. Before the C00 release of Guardian 90, Mark worked with Tandem developers in the requirements, definition, and design of the DSM products. In 1989, he began working with the DSM Applications group as a performance consultant. Mark is currently designing a Tandem message switch and trading system for the Paris Stock Exchange.

Ongoing Support

Tandem Professional Services

To help users develop applications quickly and maintain them efficiently, Tandem™ recently introduced packaged Professional Services. This program provides trained Tandem experts who deliver standardized technical consulting services at the user site, helping users to take advantage of the latest tools and technology to implement solutions quickly.

Tandem Professional Services offers a range of service packages to assist users during all phases of system planning, design, implementation, and production. The services are designed as independent modules, so users can choose specific areas where they need assistance. Three services, which can be completed in two to four weeks, are available now:

- Project Definition
- NonStop™ SQL Physical Database Design
- NonStop SQL Performance Tuning

Before a service is delivered, the Tandem service manager reviews the goals, tasks, and deliverables of the service. A Tandem consultant then works onsite with a user team to accomplish those objectives. Working with the user team, the Tandem consultant provides technical training, which can be applied to other phases of the project and to future projects.

Current Service Products

The Professional Services products now available are briefly described below. Together, these three products can help users at various stages of application design, development, and tuning.

Project Definition
This service helps users develop a strategic plan for a Tandem development project. Tandem project management specialists work with a user team to define the project requirements, outline the major phases and associated tasks required to complete the project, and identify the roles and responsibilities of the people who will execute the project. In addition to project management expertise, the Tandem specialists provide extensive experience with online transaction processing (OLTP) applications as well as Tandem concepts and products.

NonStop SQL Physical Database Design
This service helps users design a high-performance relational database. Tandem database specialists assist the user database design team in translating their logical database design into an effective physical database design optimized for NonStop SQL. As they work with the user team, the Tandem specialists also provide coaching and advice, transferring database design expertise and methodology to the user database staff.

NonStop SQL Performance Tuning
This service helps users tune NonStop SQL applications for maximum performance. Tandem database specialists work closely with the user database team to measure, analyze, and optimize the NonStop SQL application environment. The Tandem specialists also give advice and help transfer performance analysis and tuning skills to the user database staff.

Created and Delivered by Technical Experts

The goal of the Professional Services program is to provide the best approach to solving a system requirement. Tandem is experienced in tailoring advanced technological solutions for companies and has developed efficient methods to attain successful results. By using a proven systematic approach, Professional Services ensure consistent, high-quality delivery and substantial time savings for users.

The program gives users access to Tandem's technical experts, who have extensive experience in OLTP and who understand the requirements of implementing complex applications cost-effectively.

Each service is created by a team of Tandem's senior technical people who have years of experience in helping users in the specific subject area. The Tandem team develops the methodology, principles, and tools for the service according to uniform standards. The materials are reviewed by other Tandem experts and tested at multiple customer sites.

After the service has been developed, the best technical people from Tandem and Tandem Alliance partners are selected to become service consultants. Each consultant completes a certification program to ensure quality and consistency in delivery. The consultant attends a training session and receives mentoring and evaluation from a senior expert on the service process and in the subject area.

Most services not only help users accomplish practical goals, but they also build users' technical skills. The specialized skills of the consultants complement the user team's understanding of their business and application requirements. By working side by side with users, consultants can enhance the user team's skills and the project's success.

Service Management

To assure a smooth and timely delivery, a Tandem service manager oversees the implementation of each service. The service manager makes sure that users are satisfied with all phases of the service, from selection to final evaluation.

Before starting the service delivery, the service manager reviews the service description and service prerequisites with the user, handles any requests for changes to the standard service, and works with the user on scheduling. The service description provides specific information about the delivery of each service, including the objective, scope of work, deliverable results, user and Tandem roles and responsibilities, project approach, and tasks.

To benefit fully from the service, the service manager often recommends that user participants attend specific Tandem courses listed in the service description. In addition, the service description may describe project phases, documents, or materials that need to be completed before Tandem begins the service.

To learn more about the Tandem Professional Services program and individual consulting services, users should contact their local Tandem sales office.

TANDEM SYSTEMS REVIEW INDEX

The *Tandem Journal* became the *Tandem Systems Review* in February 1985. Four issues of the *Tandem Journal* were published:

Volume 1, Number 1	Fall 1983	Part no. 83930
Volume 2, Number 1	Winter 1984	Part no. 83931
Volume 2, Number 2	Spring 1984	Part no. 83932
Volume 2, Number 3	Summer 1984	Part no. 83933

As of this issue, 17 issues of the *Tandem Systems Review* have been published:

Volume 1, Number 1	February 1985	Part no. 83934
Volume 1, Number 2	June 1985	Part no. 83935
Volume 2, Number 1	February 1986	Part no. 83936
Volume 2, Number 2	June 1986	Part no. 83937
Volume 2, Number 3	December 1986	Part no. 83938
Volume 3, Number 1	March 1987	Part no. 83939
Volume 3, Number 2	August 1987	Part no. 83940
Volume 4, Number 1	February 1988	Part no. 11078
Volume 4, Number 2	July 1988	Part no. 13693
Volume 4, Number 3	October 1988	Part no. 15748
Volume 5, Number 1	April 1989	Part no. 18662
Volume 5, Number 2	September 1989	Part no. 28152
Volume 6, Number 1	March 1990	Part no. 32986
Volume 6, Number 2	October 1990	Part no. 46987
Volume 7, Number 1	April 1991	Part no. 46988
Volume 7, Number 2	October 1991	Part no. 65248
Volume 8, Number 1	Spring 1992	Part no. 65250

The articles published in all 21 issues are arranged by subject below. (*Tandem Journal* is abbreviated as TJ and *Tandem Systems Review* as TSR.) A second index, arranged by product, is also provided.

Index by Subject

Article title	Author(s)	Publication	Volume, Issue	Season or month and year	Part number
Application Development and Languages					
Ada: Tandem's Newest Compiler and Programming Environment	R. Vnuk	TSR	3,2	Aug. 1987	83940
A New Design for the PATHWAY TCP	R. Wong	TJ	2,2	Spring 1984	83932
An Introduction to Tandem EXTENDED BASIC	J. Meyerson	TJ	2,2	Spring 1984	83932
Debugging TACL Code	L. Palmer	TSR	4,2	July 1988	13693
Instrumenting Applications for Effective Event Management	J. Dagenais	TSR	7,2	Oct. 1991	65248
New TAL Features	C. Lu, J. Murayama	TSR	2,2	June 1986	83837
PATHFINDER—An Aid for Application Development	S. Benett	TJ	1,1	Fall 1983	83930
PATHWAY IDS: A Message-level Interface to Devices and Processes	M. Anderton, M. Noonan	TSR	2,2	June 1986	83937
State-of-the-Art C Compiler	E. Kit	TSR	2,2	June 1986	83937
TACL, Tandem's New Extensible Command Language	J. Campbell, R. Glascock	TSR	2,1	Feb. 1986	83936
Tandem's New COBOL85	D. Nelson	TSR	2,1	Feb. 1986	83936
The ENABLE Program Generator for Multifile Applications	B. Chapman, J. Zimmerman	TSR	1,1	Feb. 1985	83934
TMF and the Multi-Threaded Requester	T. Lemberger	TJ	1,1	Fall 1983	83930
Writing a Command Interpreter	D. Wong	TSR	1,2	June 1985	83935

Article title	Author(s)	Publication	Volume, Issue	Season or month and year	Part number
Customer Support					
Customer Information Service	J. Massucco	TSR	3,1	March 1987	83939
Remote Support Strategy	J. Eddy	TSR	3,1	March 1987	83939
Tandem's Software Support Plan	R. Baker, D. McEvoy	TSR	3,1	March 1987	83939
Data Communications					
An Overview of SNAX/CDF	M. Turner	TSR	5,2	Sept. 1989	28152
A SNAX Passthrough Tutorial	D. Kirk	TJ	2,2	Spring 1984	83932
Changes in FOX	N. Donde	TSR	1,2	June 1985	83935
Introduction to MULTILAN	A. Coyle	TSR	4,1	Feb. 1988	11078
Overview of the MULTILAN Server	A. Rowe	TSR	4,1	Feb. 1988	11078
SNAX/APC: Tandem's New SNA Software for Distributed Processing	B. Grantham	TSR	3,1	March 1987	83939
SNAX/HLS: An Overview	S. Saltwick	TSR	1,2	June 1985	83935
TLAM: A Connectivity Option for Expand	K. MacKenzie	TSR	7,1	April 1991	46988
Using the MULTILAN Application Interfaces	M. Berg, A. Rowe	TSR	4,1	Feb. 1988	11078
Data Management					
A Comparison of the B00 DP1 and DP2 Disc Processes	T. Schachter	TSR	1,2	June 1985	83935
An Overview of NonStop SQL Release 2	M. Pong	TSR	6,2	Oct. 1990	46987
Batch Processing in Online Enterprise Computing	T. Keefauver	TSR	6,2	Oct. 1990	46987
Concurrency Control Aspects of Transaction Design	W. Senf	TSR	6,1	March 1990	32968
Converting Database Files from ENSCRIBE to NonStop SQL	W. Weikel	TSR	6,1	March 1990	32986
DP1-DP2 File Conversion: An Overview	J. Tate	TSR	2,1	Feb. 1986	83936
Determining FCP Conversion Time	J. Tate	TSR	2,1	Feb. 1986	83936
DP2's Efficient Use of Cache	T. Schachter	TSR	1,2	June 1985	83935
DP2 Highlights	K. Carlyle, L. McGowan	TSR	1,2	June 1985	83935
DP2 Key-sequenced Files	T. Schachter	TSR	1,2	June 1985	83935
Gateways to NonStop SQL	D. Slutz	TSR	6,2	Oct. 1990	46987
High-Performance SQL Through Low-Level System Integration	A. Borr	TSR	4,2	July 1988	13693
Improvements in TMF	T. Lemberger	TSR	1,2	June 1985	83935
Online Reorganization of Key-Sequenced Tables and Files	G. Smith	TSR	6,2	Oct. 1990	46987
Optimizing Batch Performance	T. Keefauver	TSR	5,2	Sept. 1989	28152
Overview of NonStop SQL	H. Cohen	TSR	4,2	July 1988	13693
Parallelism in NonStop SQL Release 2	M. Moore, A. Sodhi	TSR	6,2	Oct. 1990	46987
NetBatch: Managing Batch Processing on Tandem Systems	D. Wakashige	TSR	5,1	April 1989	18662
NetBatch-Plus: Structuring the Batch Environment	G. Earle, D. Wakashige	TSR	6,1	March 1990	32986
NonStop SQL: The Single Database Solution	J. Cassidy, T. Kocher	TSR	5,2	Sept. 1989	28152
NonStop SQL Data Dictionary	R. Holbrook, D. Tsou	TSR	4,2	July 1988	13693
NonStop SQL Optimizer: Basic Concepts	M. Pong	TSR	4,2	July 1988	13693
NonStop SQL Optimizer: Query Optimization and User Influence	M. Pong	TSR	4,2	July 1988	13693
NonStop SQL Reliability	C. Fenner	TSR	4,2	July 1988	13693
The NonStop SQL Release 2 Benchmark	S. Englert, J. Gray, T. Kocher, P. Shah	TSR	6,2	Oct. 1990	46987
The Outer Join in NonStop SQL	J. Vaishnav	TSR	6,2	Oct. 1990	46987
The Relational Data Base Management Solution	G. Ow	TJ	2,1	Winter 1984	83931
Tandem's NonStop SQL Benchmark	Tandem Performance Group	TSR	4,1	Feb. 1988	11078
The TRANSFER Delivery System for Distributed Applications	S.Van Pelt	TJ	2,2	Spring 1984	83932
TMF Autorollback: A New Recovery Feature	M. Pong	TSR	1,1	Feb. 1985	83934

Article title	Author(s)	Publication	Volume, Issue	Season or month and year	Part number
Manuals/Courses					
B00 Software Manuals	S. Olds	TSR	1,2	June 1985	83935
C00 Software Manuals	E. Levi	TSR	4,1	Feb. 1988	11078
New Software Courses	M. Janow	TSR	1,2	June 1985	83935
New Software Courses	J. Limper	TSR	4,1	Feb. 1988	11078
Subscription Policy for Software Manuals	T. McSweeney	TSR	2,1	Feb. 1986	83936
Tandem's New Products	C. Robinson	TSR	2,1	Feb. 1986	83936
Tandem's New Products	C. Robinson	TSR	2,2	June 1986	83937
Operating Systems					
Highlights of the B00 Software Release	K. Coughlin, R. Montevaldo	TSR	1,2	June 1985	83935
Increased Code Space	A. Jordan	TSR	1,2	June 1985	83935
Managing System Time Under GUARDIAN 90	E. Nellen	TSR	2,1	Feb. 1986	83936
New GUARDIAN 90 Time-keeping Facilities	E. Nellen	TSR	1,2	June 1985	83935
New Process-timing Features	S. Sharma	TSR	1,2	June 1985	83935
NonStop II Memory Organization and Extended Addressing	D. Thomas	TJ	1,1	Fall 1983	83930
Overview of the C00 Release	L. Marks	TSR	4,1	Feb. 1988	11078
Overview of the NonStop-UX Operating System for the Integrity S2	P. Norwood	TSR	7,1	April 1991	46988
Robustness to Crash in a Distributed Data Base: A Nonshared-memory Approach	A. Borr	TSR	1,2	June 1985	83935
The GUARDIAN Message System and How to Design for It	M. Chandra	TSR	1,1	Feb. 1985	83935
The Tandem Global Update Protocol	R. Carr	TSR	1,2	June 1985	83935
Performance and Capacity Planning					
A Performance Retrospective	P. Oleinick, P. Shah	TSR	2,3	Dec. 1986	83938
Buffering for Better Application Performance	R. Mattran	TSR	2,1	Feb. 1986	83936
Capacity Planning Concepts	R. Evans	TSR	2,3	Dec. 1986	83938
Capacity Planning With TCM	W. Highleyman	TSR	7,2	Oct. 1991	65248
C00 TMDS Performance	J. Mead	TSR	4,1	Feb. 1988	11078
Credit-authorization Benchmark for High Performance and Linear Growth	T. Chmiel, T. Houy	TSR	2,1	Feb. 1986	83936
Debugging Accelerated Programs on TNS/R Systems	D. Cressler	TSR	8,1	Spring 1992	65250
DP2 Performance	J. Enright	TSR	1,2	June 1985	83935
Estimating Host Response Time in a Tandem System	H. Horwitz	TSR	4,3	Oct. 1988	15748
FASTSORT: An External Sort Using Parallel Processing	J. Gray, M. Stewart, A. Tsukerman, S. Uren, B. Vaughan	TSR	2,3	Dec. 1986	83938
Getting Optimum Performance from Tandem Tape Systems	A. Khatri	TSR	2,3	Dec. 1986	83938
How to Set Up a Performance Data Base with MEASURE and ENFORM	M. King	TSR	2,3	Dec. 1986	83938
Improved Performance for BACKUP2 and RESTORE2	A. Khatri, M. McCline	TSR	1,2	June 1985	83935
Improving Performance on TNS/R Systems With the Accelerator	M. Blanchet	TSR	8,1	Spring 1992	65250
MEASURE: Tandem's New Performance Measurement Tool	D. Dennison	TSR	2,3	Dec. 1986	83938
Measuring DSM Event Management Performance	M. Stockton	TSR	8,1	Spring 1992	65250
Message System Performance Enhancements	D. Kinkade	TSR	2,3	Dec. 1986	83938
Message System Performance Tests	S. Uren	TSR	2,3	Dec. 1986	83938
Network Design Considerations	J. Evjen	TSR	5,2	Sept. 1989	28152
NonStop VLX Performance	J. Enright	TSR	2,3	Dec. 1986	83938
Optimizing Sequential Processing on the Tandem System	R. Welsh	TJ	2,3	Summer 1984	83933
Pathway TCP Enhancements for Application Run-Time Support	R. Vannucci	TSR	7,1	April 1991	46988
Performance Benefits of Parallel Query Execution and Mixed Workload Support in NonStop SQL Release 2	S. Englert, J. Gray	TSR	6,2	Oct. 1990	46987
Performance Considerations for Application Processes	R. Glasstone	TSR	2,3	Dec. 1986	83938
Performance Measurements of an ATM Network Application	N. Cabell, D. Mackie	TSR	2,3	Dec. 1986	83938
Predicting Response Time in On-line Transaction Processing Systems	A. Khatri	TSR	2,2	June 1986	83937

Article title	Author(s)	Publication	Volume, Issue	Season or month and year	Part number
Performance and Capacity Planning					
The 6600 and TCC6820 Communications Controllers: A Performance Comparison	P. Beadles	TSR	2,3	Dec. 1986	83938
The ENCORE Stress Test Generator for On-line Transaction Processing Applications	S. Kosinski	TJ	2,1	Winter 1984	83931
The PATHWAY TCP: Performance and Tuning	J. Vatz	TSR	1,1	Feb. 1985	83934
The Performance Characteristics of Tandem NonStop Systems	J. Day	TJ	1,1	Fall 1983	83930
Sizing Cache for Applications that Use B-series DP1 and TMF	P. Shah	TSR	2,2	June 1986	83937
Sizing the Spooler Collector Data File	H. Norman	TSR	4,1	Feb. 1988	11978
Tandem's 5200 Optical Storage Facility: Performance and Optimization Considerations	S. Coleman	TSR	5,1	April 1989	18662
Tandem's Approach to Fault Tolerance	B. Ball, W. Bartlett, S. Thompson	TSR	4,1	Feb. 1988	11078
Understanding PATHWAY Statistics	R. Wong	TJ	2,2	Spring 1984	83932
Peripherals					
5120 Tape Subsystem Recording Technology	W. Phillips	TSR	3,2	Aug. 1987	83940
An Introduction to DYNAMITE Workstation Host Integration	S. Kosinski	TSR	1,2	June 1985	83935
Data-Encoding Technology Used in the XL8 Storage Facility	D.S. Ng	TSR	2,2	June 1986	83937
Data-Window Phase-Margin Analysis	A. Painter, H. Pham, H. Thomas	TSR	2,2	June 1986	83937
Introducing the 3207 Tape Controller	S. Chandran	TSR	1,2	June 1985	83935
Peripheral Device Interfaces	J. Blakkan	TSR	3,2	Aug. 1987	83940
Plated Media Technology Used in the XL8 Storage Facility	D.S. Ng	TSR	2,2	June 1986	83937
Streaming Tape Drives	J. Blakkan	TSR	3,2	Aug. 1987	83940
The 5200 Optical Storage Facility: A Hardware Perspective	A. Patel	TSR	5,1	April 1989	18662
The 6100 Communications Subsystem: A New Architecture	R. Smith	TJ	2,1	Winter 1984	83931
The 6600 and TCC6820 Communications Controllers: A Performance Comparison	P. Beadles	TSR	2,3	Dec. 1986	83938
The DYNAMITE Workstation: An Overview	G. Smith	TSR	1,2	June 1985	83935
The Model 6VI Voice Input Option: Its Design and Implementation	B. Huggett	TJ	2,3	Summer 1984	83933
The Role of Optical Storage in Information Processing	L. Sabaroff	TSR	3,2	Aug. 1987	83940
The V8 Disc Storage Facility: Setting a New Standard for On-line Disc Storage	M. Whiteman	TSR	1,2	June 1985	83935
Processors					
Fault Tolerance in the NonStop Cyclone System	S. Chan, R. Jardine	TSR	7,1	April 1991	46988
NonStop CLX: Optimized for Distributed On-Line Transaction Processing	D. Lenoski	TSR	5,1	April 1989	18662
NonStop VLX Hardware Design	M. Brown	TSR	2,3	Dec. 1986	83938
Overview of Tandem NonStop Series/RISC Systems	L. Faby, R. Mateosian	TSR	8,1	Spring 1992	65250
The High-Performance NonStop TXP Processor	W. Bartlett, T. Houy, D. Meyer	TJ	2,1	Winter 1984	83931
The NonStop TXP Processor: A Powerful Design for On-line Transaction Processing	P. Oleinick	TJ	2,3	Summer 1984	83933
The VLX: A Design for Serviceability	J. Allen, R. Boyle	TSR	3,1	March 1987	83939
Security					
Dial-In Security Considerations	P. Grainger	TSR	7,2	Oct. 1991	65248
Distributed Protection with SAFEGUARD	T. Chou	TSR	2,2	June 1986	83937
Enhancing System Security With Safeguard	C. Gaydos	TSR	7,1	April 1991	46988
System Connectivity					
Building Open Systems Interconnection with OSI/AS and OSI/TS	R. Smith	TSR	6,1	March 1990	32986
Network Design Considerations	J. Evjen	TSR	5,2	Sept. 1989	28152
Terminal Connection Alternatives for Tandem Systems	J. Simonds	TSR	5,1	April 1989	18662
The OSI Model: Overview, Status, and Current Issues	A. Dunn	TSR	5,1	April 1989	18662

Article title	Author(s)	Publication	Volume, Issue	Season or month and year	Part number
System Management					
Configuring Tandem Disk Subsystems	S. Sitler	TSR	2,3	Dec. 1986	83938
Data Replication in Tandem's Distributed Name Service	T. Eastep	TSR	4,3	Oct. 1988	15748
Enhancements to TMDS	L. White	TSR	3,2	Aug. 1987	83940
Event Management Service Design and Implementation	H. Jordan, R. McKee, R. Schuet	TSR	4,3	Oct. 1988	15748
Introducing TMDS, Tandem's New On-line Diagnostic System	J. Troisi	TSR	1,2	June 1985	83935
Instrumenting Applications for Effective Event Management	J. Dagenais	TSR	7,2	Oct. 1991	65248
Measuring DSM Event Management Performance	M. Stockton	TSR	8,1	Spring 1992	65250
Network Statistics System	M. Miller	TSR	4,3	Oct. 1988	15748
Overview of DSM	P. Homan, B. Malizia, E. Reisner	TSR	4,3	Oct. 1988	15748
SCP and SCF: A General Purpose Implementation of the Subsystem Programmatic Interface	T. Lawson	TSR	4,3	Oct. 1988	15748
RDF: An Overview	J. Guerrero	TSR	7,2	Oct. 1991	65248
Tandem's Subsystem Programmatic Interface	G. Tom	TSR	4,3	Oct. 1988	15748
Using FOX to Move a Fault-tolerant Application	C. Breighner	TSR	1,1	Feb. 1985	83934
Using the Subsystem Programmatic Interface and Event Management Services	K. Stobie	TSR	4,3	Oct. 1988	15748
VIEWPOINT Operations Console Facility	R. Hansen, G. Stewart	TSR	4,3	Oct. 1988	15748
VIEWSYS: An On-line System-resource Monitor	D. Montgomery	TSR	1,2	June 1985	83935
Writing Rules for Automated Operations	J. Collins	TSR	7,2	Oct. 1991	65248
Utilities					
Enhancements to PS MAIL	R. Funk	TSR	3,1	March 1987	83939

Index by Product

Article title	Author(s)	Publication	Volume, Issue	Season or month and year	Part number
3207 Tape Controller					
Introducing the 3207 Tape Controller	S. Chandran	TSR	1,2	June 1985	83935
5120 Tape Subsystem					
5120 Tape Subsystem Recording Technology	W. Phillips	TSR	3,2	Aug. 1987	83940
5200 Optical Storage					
Tandem's 5200 Optical Storage Facility: Performance and Optimization Considerations	S. Coleman	TSR	5,1	April 1989	18662
The 5200 Optical Storage Facility: A Hardware Perspective	A. Patel	TSR	5,1	April 1989	18662
The Role of Optical Storage in Information Processing	L. Sabaroff	TSR	4,1	Feb. 1988	11078
6100 Communications Subsystem					
The 6100 Communications Subsystem: A New Architecture	R. Smith	TJ	2,1	Winter 1984	83931
6530 Terminal					
The Model 6VI Voice Input Option: Its Design and Implementation	B. Huggett	TJ	2,3	Summer 1984	83933
6600 and TCC6820 Communications Controllers					
The 6600 and TCC6820 Communications Controllers: A Performance Comparison	P. Beadles	TSR	2,3	Dec. 1986	83938
Ada					
Ada: Tandem's Newest Compiler and Programming Environment	R. Vnuk	TSR	3,2	Aug. 1987	83940
BASIC					
An Introduction to Tandem EXTENDED BASIC	J. Meyerson	TJ	2,2	Spring 1984	83932
C					
State-of-the-art C Compiler	E. Kit	TSR	2,2	June 1986	83937
CIS					
Customer Information Service	J. Massucco	TSR	3,1	March 1987	83939
CLX					
NonStop CLX: Optimized for Distributed On-Line Transaction Processing	D. Lenoski	TSR	5,1	April 1989	18662
COBOL85					
Tandem's New COBOL85	D. Nelson	TSR	2,1	Feb. 1986	83936
COMINT (CI)					
Writing a Command Interpreter	D. Wong	TSR	1,2	June 1985	83935
Cyclone					
Fault Tolerance in the NonStop Cyclone System	S. Chan, R. Jardine	TSR	7,1	April 1991	46988
DP1 and DP2					
A Comparison of the B00 DP1 and DP2 Disc Processes	T. Schachter	TSR	1,2	June 1985	83935
Determining FCP Conversion Time	J. Tate	TSR	2,1	Feb. 1986	83936
DP1-DP2 File Conversion: An Overview	J. Tate	TSR	2,1	Feb. 1986	83936
DP2 Highlights	K. Carlyle L. McGowan	TSR	1,2	June 1985	83935
DP2 Key-sequenced Files	T. Schachter	TSR	1,2	June 1985	83935
DP2 Performance	J. Enright	TSR	1,2	June 1985	83935
DP2's Efficient Use of Cache	T. Schachter	TSR	1,2	June 1985	83935
Sizing Cache for Applications that Use B-series DP1 and TMF	P. Shah	TSR	2,2	June 1986	83937

Article title	Author(s)	Publication	Volume, Issue	Season or month and year	Part number
DSM					
Data Replication in Tandem's Distributed Name Service	T. Eastep	TSR	4,3	Oct. 1988	15748
Event Management Service Design and Implementation	H. Jordan, R. McKee, R. Schuet	TSR	4,3	Oct. 1988	15748
Instrumenting Applications for Effective Event Management	J. Dagenais	TSR	7,2	Oct. 1991	65248
Measuring DSM Event Management Performance	M. Stockton	TSR	8,1	Spring 1992	65250
Network Statistics System	M. Miller	TSR	4,3	Oct. 1988	15748
Overview of DSM	P. Homan, B. Malizia, E. Reisner	TSR	4,3	Oct. 1988	15748
SCP and SCF: A General Purpose Implementation of the Subsystem Programmatic Interface	T. Lawson	TSR	4,3	Oct. 1988	15748
Tandem's Subsystem Programmatic Interface	G. Tom	TSR	4,3	Oct. 1988	15748
Using the Subsystem Programmatic Interface and Event Management Services	K. Stobie	TSR	4,3	Oct. 1988	15748
VIEWPOINT Operations Console Facility	R. Hansen, G. Stewart	TSR	4,3	Oct. 1988	15748
Writing Rules for Automated Operations	J. Collins	TSR	7,2	Oct. 1991	65248
DYNAMITE					
An Introduction to DYNAMITE Workstation Host Integration	S. Kosinski	TSR	1,2	June 1985	83935
The DYNAMITE Workstation: An Overview	G. Smith	TSR	1,2	June 1985	83935
ENABLE					
The ENABLE Program Generator for Multifile Applications	B. Chapman, J. Zimmerman	TSR	1,1	Feb. 1985	83934
ENCOMPASS					
The Relational Data Base Management Solution	G. Ow	TJ	2,1	Winter 1984	83931
ENCORE					
The ENCORE Stress Test Generator for On-line Transaction Processing Applications	S. Kosinski	TJ	2,1	Winter 1984	83931
ENSCRIBE					
Converting Database Files from ENSCRIBE to NonStop SQL	W. Weikel	TSR	6,1	March 1990	32986
FASTSORT					
FASTSORT: An External Sort Using Parallel Processing	J. Gray, M. Stewart, A. Tsukerman, S. Uren, B. Vaughan	TSR	2,3	Dec. 1986	83938
FOX					
Changes in FOX	N. Donde	TSR	1,2	June 1985	83935
Using FOX to Move a Fault-tolerant Application	C. Breighner	TSR	1,1	Feb. 1985	83934
FUP					
Online Reorganization of Key-Sequenced Tables and Files	G. Smith	TSR	6,2	Oct. 1990	46987
GUARDIAN 90					
B00 Software Manuals	S. Olds	TSR	1,2	June 1985	83935
C00 Software Manuals	E. Levi	TSR	4,1	Feb. 1988	11078
Highlights of the B00 Software Release	K. Coughlin, R. Montevaldo	TSR	1,2	June 1985	83935
Improved Performance for BACKUP2 and RESTORE2	A. Khatri, M. McCline	TSR	1,2	June 1985	83935
Increased Code Space	A. Jordan	TSR	1,2	June 1985	83935
Managing System Time Under GUARDIAN 90	E. Nellen	TSR	2,1	Feb. 1986	83936
Message System Performance Enhancements	D. Kinkade	TSR	2,3	Dec. 1986	83938
Message System Performance Tests	S. Uren	TSR	2,3	Dec. 1986	83938
New GUARDIAN 90 Time-keeping Facilities	E. Nellen	TSR	1,2	June 1985	83935
New Process-timing Features	S. Sharma	TSR	1,2	June 1985	83935
NonStop II Memory Organization and Extended Addressing	D. Thomas	TJ	1,1	Fall 1983	83930
Overview of the C00 Release	L. Marks	TSR	4,1	Feb. 1988	11078
Robustness to Crash in a Distributed Data Base: A Nonshared-memory Multiprocessor Approach	A. Borr	TSR	1,2	June 1985	83935
Tandem's Approach to Fault Tolerance	B. Ball, W. Bartlett, S. Thompson	TSR	4,1	Feb. 1988	11078
The GUARDIAN Message System and How to Design for It	M. Chandra	TSR	1,1	Feb. 1985	83935
The Tandem Global Update Protocol	R. Carr	TSR	1,2	June 1985	83935

Article title	Author(s)	Publication	Volume, Issue	Season or month and year	Part number
Integrity S2					
Overview of the NonStop-UX Operating System for the Integrity S2	P. Norwood	TSR	7,1	April 1991	46988
MEASURE					
How to Set Up a Performance Data Base with MEASURE and ENFORM	M. King	TSR	2,3	Dec. 1986	83938
MEASURE: Tandem's New Performance Measurement Tool	D. Dennison	TSR	2,3	Dec. 1986	83938
MULTILAN					
Introduction to MULTILAN	A. Coyle	TSR	4,1	Feb. 1988	11078
Overview of the MULTILAN Server	A. Rowe	TSR	4,1	Feb. 1988	11078
Using the MULTILAN Application Interfaces	M. Berg, A. Rowe	TSR	4,1	Feb. 1988	11078
NetBatch-Plus					
NetBatch: Managing Batch Processing on Tandem Systems	D. Wakashige	TSR	5,1	April 1989	18662
NetBatch-Plus: Structuring the Batch Environment	G. Earle, D. Wakashige	TSR	6,1	March 1990	32986
NonStop SQL					
An Overview of NonStop SQL Release 2	M. Pong	TSR	6,2	Oct. 1990	46987
Concurrency Control Aspects of Transaction Design	W. Senf	TSR	6,1	March 1990	32986
Converting Database Files from ENSCRIBE to NonStop SQL	W. Weikel	TSR	6,1	March 1990	32986
Gateways to NonStop SQL	D. Slutz	TSR	6,2	Oct. 1990	46987
High-Performance SQL Through Low-Level System Integration	A. Borr	TSR	4,2	July 1988	13693
NonStop SQL Data Dictionary	R. Holbrook, D. Tsou	TSR	4,2	July 1988	13693
NonStop SQL: The Single Database Solution	J. Cassidy, T. Kocher	TSR	5,2	Sept. 1989	28152
NonStop SQL Optimizer: Basic Concepts	M. Pong	TSR	4,2	July 1988	13693
NonStop SQL Optimizer: Query Optimization and User Influence	M. Pong	TSR	4,2	July 1988	13693
NonStop SQL Reliability	C. Fenner	TSR	4,2	July 1988	13693
Overview of NonStop SQL	H. Cohen	TSR	4,2	July 1988	13693
Parallelism in NonStop SQL Release 2	M. Moore, A. Sodhi	TSR	6,2	Oct. 1990	46987
Performance Benefits of Parallel Query Execution and Mixed Workload Support in NonStop SQL Release 2	S. Englert, J. Gray	TSR	6,2	Oct. 1990	46987
Tandem's NonStop SQL Benchmark	Tandem Performance Group	TSR	4,1	Feb. 1988	11078
The NonStop SQL Release 2 Benchmark	S. Englert, J. Gray, T. Kocher, P. Shah	TSR	6,2	Oct. 1990	46987
The Outer Join in NonStop SQL	J. Vaishnav	TSR	6,2	Oct. 1990	46987
OSI					
Building Open Systems Interconnection with OSI/AS and OSI/TS	R. Smith	TSR	6,1	March 1990	32986
The OSI Model: Overview, Status, and Current Issues	A. Dunn	TSR	5,1	April 1989	18662
PATHFINDER					
PATHFINDER—An Aid for Application Development	S. Benett	TJ	1,1	Fall 1983	83930
PATHWAY					
A New Design for the PATHWAY TCP	R. Wong	TJ	2,2	Spring 1984	83932
PATHWAY IDS: A Message-level Interface to Devices and Processes	M. Anderton M. Noonan	TSR	2,2	June 1986	83937
Pathway TCP Enhancements for Application Run-Time Support	R. Vannucci	TSR	7,1	April 1991	46988
The PATHWAY TCP: Performance and Tuning	J. Vatz	TSR	1,1	Feb. 1985	83934
Understanding PATHWAY Statistics	R. Wong	TJ	2,2	Spring 1984	83932
PS MAIL					
Enhancements to PS MAIL	R. Funk	TSR	3,1	March 1987	83939
RDF					
RDF: An Overview	J. Guerrero	TSR	7,2	Oct. 1991	65248
SAFEGUARD					
Dial-In Security Considerations	P. Grainger	TSR	7,2	Oct. 1991	65248
Distributed Protection with SAFEGUARD	T. Chou	TSR	2,2	June 1986	83937
Enhancing System Security With Safeguard	C. Gaydos	TSR	7,1	April 1991	46988

Article title	Author(s)	Publication	Volume, Issue	Season or month and year	Part number
SNAX					
An Overview of SNAX/CDF	M. Turner	TSR	5,2	Sept. 1989	28152
A SNAX Passthrough Tutorial	D. Kirk	TJ	2,2	Spring 1984	83932
SNAX/APC: Tandem's New SNA Software for Distributed Processing	B. Grantham	TSR	3,1	March 1987	83939
SNAX/HLS: An Overview	S. Saltwick	TSR	1,2	June 1985	83935
SPOOLER					
Sizing the Spooler Collector Data File	H. Norman	TSR	4,1	Feb. 1988	11078
TACL					
Debugging TACL Code	L. Palmer	TSR	4,2	July 1988	13693
TACL, Tandem's New Extensible Command Language	J. Campbell, R. Glascock	TSR	2,1	Feb. 1986	83936
TAL					
New TAL Features	C. Lu, J. Murayama	TSR	2,2	June 1986	83837
TCM					
Capacity Planning With TCM	W. Highleyman	TSR	7,2	Oct. 1991	65248
TLAM					
TLAM: A Connectivity Option for Expand	K. MacKenzie	TSR	7,1	April 1991	46988
TMDS					
C00 TMDS Performance	J. Mead	TSR	4,1	Feb. 1988	11078
Enhancements to TMDS	L. White	TSR	3,2	Aug. 1987	83940
Introducing TMDS, Tandem's New On-line Diagnostic System	J. Troisi	TSR	1,2	June 1985	83935
TMF					
Improvements in TMF	T. Lemberger	TSR	1,2	June 1985	83935
TMF and the Multi-Threaded Requester	T. Lemberger	TJ	1,1	Fall 1983	83930
TMF Autorollback: A New Recovery Feature	M. Pong	TSR	1,1	Feb. 1985	83934
TNS/R					
Debugging Accelerated Programs on TNS/R Systems	D. Cressler	TSR	8,1	Spring 1992	65250
Improving Performance on TNS/R Systems With the Accelerator	M. Blanchet	TSR	8,1	Spring 1992	65250
Overview of Tandem NonStop Series/RISC Systems	L. Faby, R. Mateosian	TSR	8,1	Spring 1992	65250
TRANSFER					
The TRANSFER Delivery System for Distributed Applications	S. Van Pelt	TJ	2,2	Spring 1984	83932
TXP					
The High-Performance NonStop TXP Processor	W. Bartlett, T. Houy, D. Meyer	TJ	2,1	Winter 1984	83931
The NonStop TXP Processor: A Powerful Design for On-line Transaction Processing	P. Oleinick	TJ	2,3	Summer 1984	83933
V8					
The V8 Disc Storage Facility: Setting a New Standard for On-line Disc Storage	M. Whiteman	TSR	1,2	June 1985	83935
VIEWSYS					
VIEWSYS: An On-line System-resource Monitor	D. Montgomery	TSR	1,2	June 1985	83935
VLX					
NonStop VLX Hardware Design	M. Brown	TSR	2,3	Dec. 1986	83938
NonStop VLX Performance	J. Enright	TSR	2,3	Dec. 1986	83938
The VLX: A Design for Serviceability	J. Allen, R. Boyle	TSR	3,1	March 1987	83939
XL8					
Data-encoding Technology Used in the XL8 Storage Facility	D.S. Ng	TSR	2,2	June 1986	83937
Plated Media Technology Used in the XL8 Storage Facility	D.S. Ng	TSR	2,2	June 1986	83937

Article title	Author(s)	Publication	Volume, Issue	Season or month and year	Part number
Miscellaneous[1]					
A Performance Retrospective	P. Oleinick	TSR	2,3	Dec. 1986	83938
Batch Processing in Online Enterprise Computing	T. Keefauver	TSR	6,2	Oct. 1990	46987
Buffering for Better Application Performance	R. Mattran	TSR	2,1	Feb. 1986	83936
Capacity Planning Concepts	R. Evans	TSR	2,3	Dec. 1986	83938
Configuring Tandem Disk Subsystems	S. Sitler	TSR	2,3	Dec. 1986	83938
Credit-authorization Benchmark for High Performance and Linear Growth	T. Chmiel, T. Houy	TSR	2,1	Feb. 1986	83936
Data-window Phase-margin Analysis	A. Painter, H. Pham, H. Thomas	TSR	2,2	June 1986	83937
Estimating Host Response Time in a Tandem System	H. Horwitz	TSR	4,3	Oct. 1988	15748
Getting Optimum Performance from Tandem Tape Systems	A. Khatri	TSR	2,3	Dec. 1986	83938
Network Design Considerations	J. Evjen	TSR	5,2	Sept. 1989	28152
New Software Courses	M. Janow	TSR	1,2	June 1985	83935
New Software Courses	J. Limper	TSR	4,1	Feb. 1988	11078
Optimizing Batch Performance	T. Keefauver	TSR	5,2	Sept. 1989	28152
Optimizing Sequential Processing on the Tandem System	R. Welsh	TJ	2,3	Summer 1984	83933
Performance Considerations for Application Processes	R. Glasstone	TSR	2,3	Dec. 1986	83938
Performance Measurements of an ATM Network Application	N. Cabell, D. Mackie	TSR	2,3	Dec. 1986	83938
Peripheral Device Interfaces	J. Blakkan	TSR	3,2	Aug. 1987	83940
Predicting Response Time in On-line Transaction Processing Systems	A. Khatri	TSR	2,2	June 1986	83937
Remote Support Strategy	J. Eddy	TSR	3,1	March 1987	83939
Streaming Tape Drives	J. Blakkan	TSR	3,2	Aug. 1987	83940
Subscription Policy for Software Manuals	T. McSweeney	TSR	2,1	Feb. 1986	83936
Tandem's New Products	C. Robinson	TSR	2,1	Feb. 1986	83936
Tandem's New Products	C. Robinson	TSR	2,2	June 1986	83937
Tandem's Software Support Plan	R. Baker, D. McEvoy	TSR	3,1	March 1987	83939
Terminal Connection Alternatives for Tandem Systems	J. Simonds	TSR	5,1	April 1989	18662
The Performance Characteristics of Tandem NonStop Systems	J. Day	TJ	1,1	Fall 1983	83930
The Role of Optical Storage in Information Processing	L. Sabaroff	TSR	3,2	Aug. 1987	83940

[1]This category is composed of articles that contain product information but are not specifically product-related.

TANDEM SYSTEMS REVIEW ORDER FORM

Use this form to request or renew a subscription, change subscription information, or order back copies.

☐ If you are a Tandem customer, you may complete Part A of this form and send it to your Tandem representative. Your representative may request that you receive an invoice for this subscription, in which case you will be notified.

☐ For other subscribers, complete Part A of the form and send it to the address below. Enclose a check or money order, payable to Tandem Computers Incorporated, for the subscription and back copies that you order. The cost is $75 for a one-year subscription and $20 for each back issue.

Part A. To be completed by the subscriber.

Subscription Information

☐ New subscription

☐ Subscription renewal

☐ Update to subscription information
Subscription number: _____

Your subscription number is in the upper right corner of the mailing label.

_____
COMPANY

_____
NAME

_____
JOB TITLE

_____
DIVISION

_____
ADDRESS

_____

_____
COUNTRY

_____
TELEPHONE NUMBER (include all codes for U.S. dialing)

Title or position:

☐ President/CEO
☐ Director/VP information services
☐ MIS/DP manager
☐ Software development manager
☐ Programmer/analyst
☐ System operator
☐ End user
☐ Other: _____

Your company's association with Tandem:

☐ Tandem customer
☐ Third-party vendor
☐ Consultant
☐ Other: _____

Back Issue Requests

Number of copies ***Tandem Systems Review***

_____ Vol. 1, No. 1, Feb. 1985
_____ Vol. 1, No. 2, June 1985
_____ Vol. 2, No. 1, Feb. 1986
_____ Vol. 2, No. 2, June 1986
_____ Vol. 2, No. 3, Dec. 1986
_____ Vol. 3, No. 1, March 1987
_____ Vol. 3, No. 2, Aug. 1987
_____ Vol. 4, No. 1, Feb. 1988
_____ Vol. 4, No. 2, July 1988
_____ Vol. 4, No. 3, Oct. 1988
_____ Vol. 5, No. 1, April 1989
_____ Vol. 5, No. 2, Sept. 1989
_____ Vol. 6, No. 1, March 1990
_____ Vol. 6, No. 2, Oct. 1990
_____ Vol. 7, No. 1, April 1991
_____ Vol. 7, No. 2, Oct. 1991
_____ Vol. 8, No. 1, Spring 1992

Tandem Journal

_____ Vol. 1, No. 1, Fall 1983
_____ Vol. 2, No. 1, Winter 1984
_____ Vol. 2, No. 2, Spring 1984
_____ Vol. 2, No. 3, Summer 1984

Tandem customers should send this form to their Tandem representative.

Other subscribers send this form to:

Tandem Computers, Incorporated
Tandem Systems Review, Loc 216-05
18922 Forge Drive
Cupertino, CA 95014-0701

Tandem employees must order their subscriptions and back issues through Courier.

Menu sequence: Marketing Information → Literature Orders → Technical Marketing Pubs → Tandem Systems Review

Part B. To be completed by the Tandem representative.

Processing This Form

Please complete this portion of the form to approve your customer's subscription. Your department will be charged $75 for each one-year subscription and $20 for each back issue. If you would like the charges to be passed on to your customer, please indicate by checking the box below. Your customer will be invoiced, and you and your sales district will receive revenue credit.

☐ **Yes, please invoice the customer for this subscription.** (Be sure to notify your customer that they will be receiving an invoice for this subscription.)

Send this completed form to:

Tandem Computers Incorporated
Tandem Systems Review, Loc 216-05
18922 Forge Drive
Cupertino, CA 95014-0701

NAME	EMPLOYEE NUMBER
TITLE	DEPARTMENT NUMBER
LOC	TELEPHONE NUMBER
CUSTOMER NUMBER	SYSTEM NUMBER
SIGNATURE	

Ordering Through Courier

You may use Tandem's online Courier system to order *Tandem Systems Review* subscriptions and back issues. An order through Courier replaces this form. The Courier menu sequence is:

Marketing Information → Literature Orders → Technical Marketing Pubs → Tandem Systems Review

TANDEM SYSTEMS REVIEW CUSTOMER SURVEY

The purpose of this questionnaire is to help the *Tandem Systems Review* staff select topics for publication. Postage is prepaid when mailed in the U.S. Customers outside the U.S. should send their replies to their nearest Tandem sales office.

1. How useful is each article in this issue?

Product Update

01 ☐ Indispensible 02 ☐ Very 03 ☐ Somewhat 04 ☐ Not at all

Overview of Tandem NonStop Series/RISC Systems

05 ☐ Indispensible 06 ☐ Very 07 ☐ Somewhat 08 ☐ Not at all

Improving Performance on TNS/R Systems With the Accelerator

09 ☐ Indispensible 10 ☐ Very 11 ☐ Somewhat 12 ☐ Not at all

Debugging Accelerated Programs on TNS/R Systems

13 ☐ Indispensible 14 ☐ Very 15 ☐ Somewhat 16 ☐ Not at all

Measuring DSM Event Management Performance

17 ☐ Indispensible 18 ☐ Very 19 ☐ Somewhat 20 ☐ Not at all

Ongoing Support: Tandem Professional Services

21 ☐ Indispensible 22 ☐ Very 23 ☐ Somewhat 24 ☐ Not at all

2. I specifically would like to see more articles on (select one):

25 ☐ Overview discussions of new products and enhancements.
26 ☐ Performance and tuning information.
27 ☐ High-level overviews on Tandem's approach to solutions.
28 ☐ Application design and customer profiles.
29 ☐ Technical discussions of product internals.
30 ☐ Other_____

3. Your title or position:

31 ☐ President, VP, Director 32 ☐ Systems analyst 33 ☐ System operator
34 ☐ MIS manager 35 ☐ Software developer 36 ☐ End user
37 ☐ Other_____

4. Your association with Tandem:

38 ☐ Tandem customer 39 ☐ Tandem employee 40 ☐ Third-party vendor 41 ☐ Consultant
42 ☐ Other_____

5. Comments

_____

_____

_____

_____
NAME

_____
COMPANY NAME

_____
ADDRESS

_____

▶ FOLD

▶ FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 482 CUPERTINO, CA. U.S.A.

POSTAGE WILL BE PAID BY ADDRESSEE

TANDEM SYSTEMS REVIEW
LOC 216-05
TANDEM COMPUTERS INCORPORATED
19333 VALLCO PARKWAY
CUPERTINO, CA 95014-9862

▶ FOLD

▶ FOLD

Tandem Computers Incorporated
19333 Vallco Parkway
Cupertino, CA 95014-2599

Part No. 65250

4/92 Printed in USA