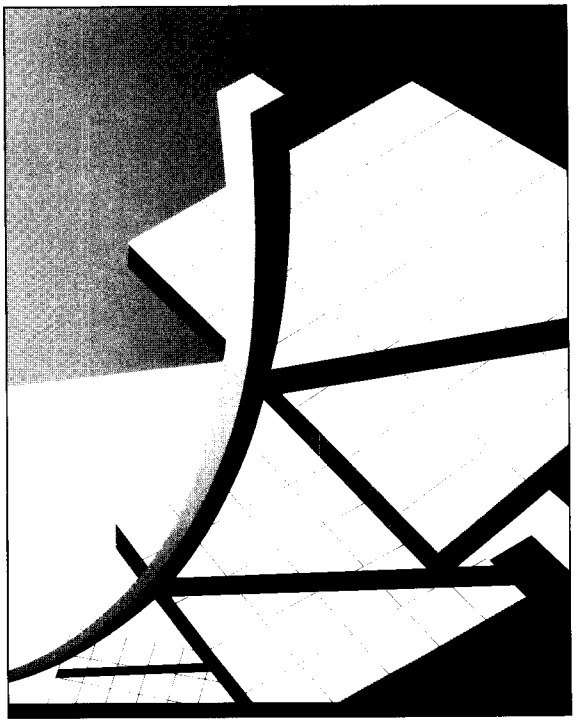
T A N D E M

SYSTEMS REVIEW

VOLUME 2, NUMBER 5 DECEMBER 1986

BRANDIFINO



A Performance Retrospective

New NonStop VLX Processor

GUARDIAN 90 Message System

MEASURE ▪ FASTSORT ▪ SNAX

Capacity Planning ▪ Tuning

Configuring Disks ▪ Tape Performance

Customer Profile: ATM Network

Volume 2, Number 3, December 1986

Editor
Ellen Marielle-Tréhoüart

Technical Advisor
Dick Thomas

Associate Editors
Wendy Osborn
Carolyn Turnbull White

Assistant Editor
Sarah Rood

Art Director/Cover Art
Stephen Stavast

Production and Layout
Claire Dolan
Jaroslav Dostal
Steve Elwood
Jim Geddes
Stephen Stavast
Janet Stevenson
David Thompson
John Tomasini

Typesetting
Barbara Cowlishaw

The *Tandem Systems Review* is published by Tandem Computers Incorporated.

Purpose: The *Tandem Systems Review* publishes technical information about Tandem software releases and products. Its purpose is to help programmer-analysts who use our computer systems to plan for, install, use, and tune Tandem products.

Subscription additions and changes: Subscriptions are free. To add names or make corrections to the distribution data base, requests within the U.S. should be sent to Tandem Computers Incorporated, *Tandem Systems Review,* 1309 South Mary Avenue, LOC 5-04, Sunnyvale, CA 94087. *Requests outside the U.S. should be sent to the local Tandem sales office.*

Comments: The editor welcomes suggestions for content and format. Please send them to the *Tandem Systems Review*, 1309 So. Mary Avenue, Sunnyvale, CA 94087.

A Performance Retrospective

Over the years, Tandem has made a substantial commitment to product performance, both for hardware and software. This article gives a retrospective on some of Tandem's performance enhancements and explains their significance.

For any computer vendor to survive, first it must deliver products that work; how fast and efficient its products are is important but not essential. In the small computer company, software product performance is usually a distant third behind functionality and quality when it comes to spending the company's very limited resources. As the company grows and matures, three factors come into play to increase the amount of effort put into improving performance: increases in available money for product performance evaluations, competitive pressures, and customer demand. All of these forces have been at work for a number of years at Tandem and have given rise to an impressive history of performance improvement.

Tandem's commitment to providing software performance enhancements for our systems is evidenced by the number of performance groups within the company. One group is chartered with the responsibilities to produce computer performance evaluation products, to provide engineering and development groups with predictive modeling and simulation expertise, and to provide performance measurement and analysis services for many parts of the company. Another provides performance benchmarking for Tandem users and first-line corporate performance consultation for the field support organization. A third group creates software prototypes and recommends performance enhancements to Software Development.

The combined investment in capital equipment alone for these organizations exceeds $9 million. Without Tandem's commitment to channel some of its resources into performance improvements, the many product performance enhancements would never have occurred. Customers have benefited greatly from Tandem's investments in performance improvements.

The following four areas illustrate the types of performance improvements that have occurred during the past few years: on-line transaction processing (OLTP), batch, disk-to-tape and tape-to-disk utilities, and sorting.

OLTP

The performance of OLTP systems is usually characterized by transaction throughput at a given response time. In this case, the throughput rates of a standard debit-credit banking transaction, also known as ET1, were examined.[1] For this transaction, throughput is defined as the rate at which transactions are processed when 95% of the transactions complete with a response time of two seconds or less. Figure 1 shows throughput versus response time for six different configurations of the same application. From left to right, the curves illustrate the product innovations' effect on performance. Figure 2 shows the incremental and cumulative improvements in throughput with each innovation.

Throughout this article *improvement factors* are used to describe performance improvements (e.g., an improvement factor of 1.8 means that the new version yields an 80% increase in performance).

The bar on the far left of Figure 2 is the performance of the NonStop II processor running the PATHWAY Terminal Control Process 1 (TCP1) and Disc Process 1 (DP1) on the A05 version of the GUARDIAN™ operating system. The first significant performance innovation was the advent of the more efficient PATHWAY Terminal Control Process 2 (TCP2). The next two bars show the incremental and cumulative improvement resulting from this innovation. This software improvement boosted the transaction throughput by a factor of 1.8.

Next came the introduction of the NonStop TXP™ processor. This hardware performance improvement increased the throughput rate by an additional factor of 2.8. The new Disc Process 2 (DP2), with a rewritten Transaction Monitoring Facility (TMF™), further boosted performance by a factor of 1.9. The enhanced B30 Message System yielded a 1.1 performance improvement in throughput.

[1]This transaction was based on the debit-credit, or ET1, transaction standard defined in "A Measure of Transaction Processing Power," *Datamation*, April 1, 1985.

Figure 1

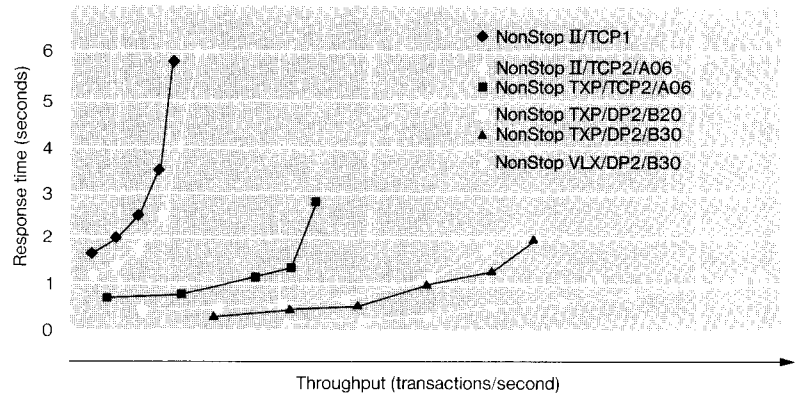


Figure 1.
OLTP throughput versus response-time curves. Shows throughput versus response time of six different configurations for a debit-credit benchmark. From left to right, the curves illustrate the product innovations' effect on performance.

Figure 2

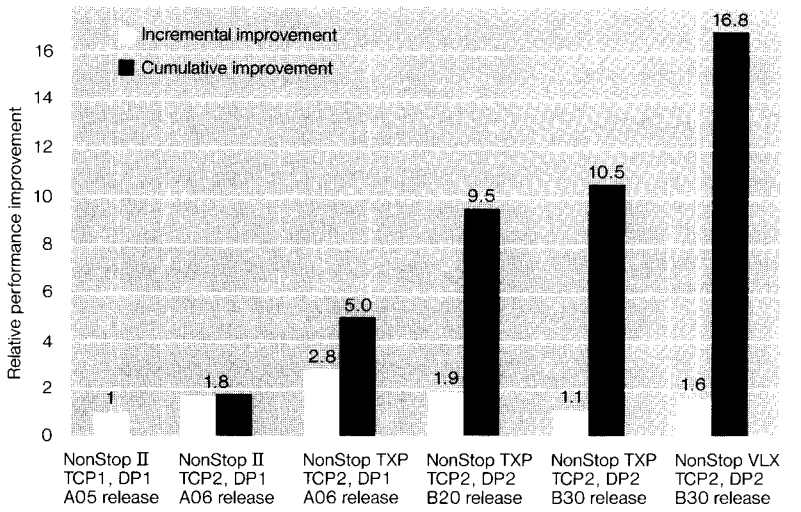


Figure 2.
OLTP relative performance improvement. From left to right the chart shows the incremental and cumulative improvements in the transaction throughput rate with each innovation.

Figure 3

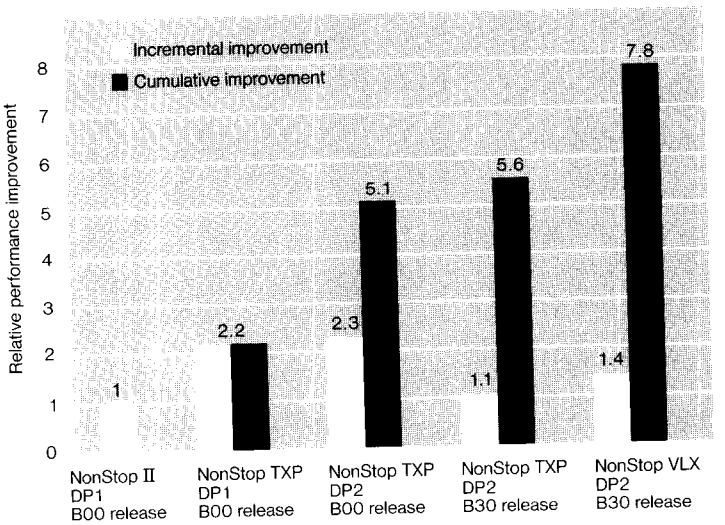


Figure 3.
Batch relative performance improvement. From left to right it shows the incremental and cumulative improvements in performance with each innovation.

Finally, the introduction of the NonStop VLX™ processor raised the throughput rate by an additional factor of 1.6.[2]

To calculate the total performance improvement since the starting point, multiply the various factors. Since the advent of TCP2, Tandem has improved the performance of OLTP by a factor of 1.8 * 2.8 * 1.9 * 1.1 * 1.6 = 16.8. If one separates the improvements into those that are hardware-based (NonStop TXP and NonStop VLX processors) and those that are software-based (TCP2, DP2, and the B30 GUARDIAN Message System), fairly balanced improvement factors of 4.5 and 3.8, respectively, result.

This analysis shows how both hardware and software improvements have contributed to Tandem's 17-fold increase in OLTP performance. In fact, if there had been only hardware improvements, the following performance improvements would have resulted. The NonStop VLX processor, capable of ten transactions per second, would only be able to process 10/3.8 = 2.6 transactions per second if Tandem had not invested in software performance improvements.

While it is clear that software has improved the throughput capacity of the system, measurements show that a reduction in instructions needed to process a transaction can account for only 1.7 of the 3.8 improvement from software; a factor of more than 2 in software improvements is unaccounted for. Some factor other than reducing the number of software instructions is at work.

Bottleneck analysis shows where the unexplained performance improvement comes from, and an examination of a simple, one-CPU, one-disk system helps to demonstrate the effect. Suppose a transaction takes 1000 ms of CPU time and 800 ms of disk time and the disk operates concurrently with the CPU. Bottleneck analysis reveals the CPU as the bottleneck in the system; it limits the system capacity to no more than 1 transaction per second (tps).

What happens to the performance of the system when the CPU speeds up? Using the hypothetical numbers, if the new CPU is 2.5 times the speed of the old one, the transaction that required 1000 ms of CPU time now needs only 400 ms. If the CPU was the bottleneck, the two systems would have a maximum throughput of 1 and 2.5 tps. However, at the rate of 2.5 tps, the system must also provide 2.5 * 800 ms = 2000 ms of disk time. But the disk can't deliver 2000 ms of service per second, so it has become the bottleneck in the system and limits the system throughput rate to no more than 1000/800 = 1.25 tps.

This example shows how bottlenecks place limits on system performance and performance improvements. The introduction of a CPU with 2.5 times the processing power of the old one results in a system throughput increase of only 1.25—half the additional CPU processing power is wasted.

A similar condition existed for the NonStop II™ processor running DP1. When the NonStop TXP processor was introduced,

[2]One should interpret these numbers with caution. Using the data from this article, throughput with a 95% response time of two seconds results in the VLX transaction throughput of 1.1 * 1.6 = 1.8 times the NonStop TXP. Public statements made by Tandem have claimed that the NonStop VLX with the new B30 GUARDIAN operating system has from 1.6 to 2.2 times the throughput capacity of the NonStop TXP. This article's data produces a relative performance ratio on the low end of the spectrum because we chose a more conservative response-time requirement in order to compare the performance of the NonStop VLX to that of the NonStop II, a much slower processor.

At the 95%, one-second response time, the performance ratio of the NonStop VLX to the NonStop TXP processor is a factor of 2. But the performance ratio of the NonStop TXP to the NonStop II processor at this response time is approximately a factor of 9. Although the NonStop TXP and NonStop VLX can be correctly compared at this response time, making a performance comparison to the NonStop II processor at this response time is not very realistic.

Also, the numbers for the NonStop II are not meant to be optimal performance. This is evidenced by the fact that the NonStop II performance with DP2 is not mentioned. The intent was to show the "leading edge" of performance over the period discussed.

the performance of the system improved, but not as much as was theoretically possible because the disk subsystem became the bottleneck in the system. Thus the software enhancements are not only a good idea, they are essential.

Batch

Batch processing has a similar history of performance enhancements. A sequential-copy benchmark that is protected by TMF was used to characterize the performance of batch.[3] The measure of performance for this benchmark is the number of records per second copied from one disk file to another. The starting point was the NonStop II processor running the B00 release of GUARDIAN with DP1. The various performance improvements appear in Figure 3. The NonStop TXP processor was introduced first, followed by DP2. Next came the faster Message System in B30 GUARDIAN and, finally, the NonStop VLX processor.

Again, the total improvement in performance is the product of the factors (2.2 * 2.3 * 1.1 * 1.4 = 7.8). If the hardware and software contributions are separated, hardware accounts for 3.1 and software for 2.5 of the gain.

BACKUP and RESTORE

The performance of the BACKUP and RESTORE utilities is characterized in terms of the data transfer rate. The investigation was limited to performance improvements from innovations other than new processors and tape drives. The base consisted of the performance of BACKUP and RESTORE on the NonStop TXP processors running DP1 using 3106 disk controllers on the GUARDIAN A06 operating system. The 5106 Tape Subsystem was used.

With the B00 release, the RESTORE utility was rewritten to incorporate an algorithm change that increased parallel processing. The reading of data from the tape was overlapped with the writing of data to the disk. This innovation is labeled DP1/B00 in Figure 4, and was responsible for improving RESTORE performance by a factor of 1.6.

[3]This benchmark was based on the sequential-copy benchmark standard also defined in the *Datamation* article previously mentioned.

Figure 4

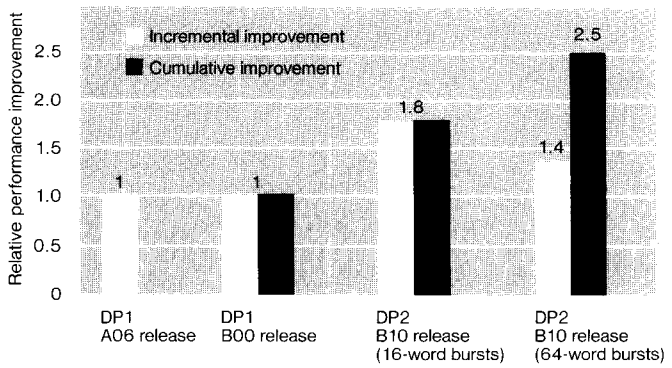
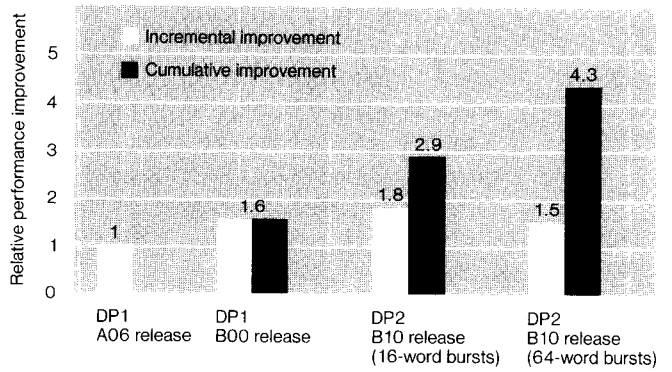


Figure 5



The new 3107 disk controller made it possible to transfer larger blocks of data than the 3106 controller (30-Kbyte blocks versus 4-Kbyte blocks), increasing the efficiency of the utilities. DP2 and the BACKUP/RESTORE software took advantage of these longer transfers, and both BACKUP and RESTORE were faster by factors of 1.8. A microcode change permitted the tape controller to move data over the channel in larger and more economical chunks, or controller burst sizes, than the standard size, i.e., 64 words instead of 16 words. A further improvement factor of 1.4 and 1.5 resulted (see the *incremental improvement factors* for DP2/B10 (b = 64) in both Figures 4 and 5).

Figure 4.
BACKUP—relative performance improvement. From left to right the chart shows the incremental and cumulative improvement in BACKUP performance with each innovation.

Figure 5.
RESTORE—relative performance improvement. From left to right it shows the incremental and cumulative improvement in RESTORE performance with each innovation.

Figure 6

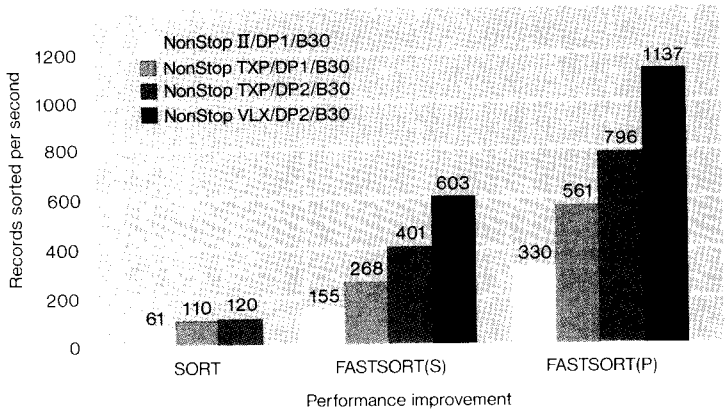


Figure 6.

SORT performance under four different hardware/software configurations. The SORT utility was rewritten initially to improve the efficiency of the serial sort algorithm, referred to as FASTSORT(S), and later to incorporate additional parallelism in the form of parallel concurrent subsorts, referred to as FASTSORT(P).

The total improvement factors for BACKUP and RESTORE are 2.5 and 4.3. Without buying any faster processors or tape drives users could simply acquire a new disk controller and move forward with new software releases, reducing the time it takes to back up and then restore a 100-Mbyte file from more than 37 minutes to just over 10 minutes.

SORT

Sorting is another task used to characterize the performance of a computer system. Performance is usually measured as the time to sort a given number of records or, inversely, the number of records sorted per second.

Tandem has paid particular attention to the SORT utility. The performance improvements shown in Figure 6 illustrate the dramatic results achieved. As in the previous figures, the sequence of performance innovations used include the NonStop II processor running DP1 and the NonStop TXP and NonStop VLX processors running DP2.

This figure differs in that the SORT algorithm itself was rewritten twice. It was rewritten initially to improve the efficiency of the serial sort algorithm, referred to as FASTSORT(S), and later to incorporate additional parallelism in the form of parallel concurrent subsorts, referred to as FASTSORT(P). FASTSORT(P) employs multiple processors to carry out the subsorts. (For a more detailed discussion see the accompanying article by Jim Gray, et al., "FASTSORT: An External Sort Using Parallel Processing.") Measurements indicate that performance is linear with the number of processors (subsorts)—two subsorts are twice as fast as one, three subsorts are three times as fast, and so on. Tandem has only begun to explore parallelism of this type (multiple processors cooperating to speed up the execution of a single task). The FASTSORT(P) data in Figure 6 is for the two subsort case only.

As shown by the SORT measurements alone, by upgrading from a NonStop II processor to the NonStop TXP and switching from DP1 to DP2, users can improve performance by about a factor of 2. However, without upgrading processors or switching to the new disk process, a factor of 2.5 increase can be obtained by employing the faster FASTSORT(S) algorithm. In fact, a factor of 5.4 increase in performance is possible simply by using FASTSORT(P).

For users presently using a NonStop II system running DP1 and the old SORT algorithm, a performance improvement factor of at least 1137/61 = 18.6 can be achieved simply by upgrading to the latest software and hardware. A factor of about 3.2 is obtained by processor upgrades, but a factor of at least 5.8 is due to software performance improvements.

Conclusion

This article describes Tandem's long-standing commitment to improving product performance. Though it is not intended to compare Tandem's compound annual performance improvement rate to other computer vendors' rates, it may well be that Tandem is second to none in delivering performance innovations.

Performance improvements take many forms, the most visible being new, faster processors. Tandem, the press, and our customers find it easy to understand and talk about faster hardware because it is tangible—it looks different. It is more difficult to relate to software—it can't be photographed and doesn't take up floor space. In fact, software performance improvements are often a well-kept secret.

Software enhancements are as much responsible for system performance improvement as new hardware, however. In one case, it was shown that the new high-performance DP2 software was necessary to use much of the added power of new processors. In another case, high-performance software employing multiple processors sped up the processing of a single task.

Over the years, Tandem has provided about a 30% software performance improvement per year, and a 65% annual price/performance improvement. As the customer's business grows, they need only install the latest software release to get more work from the same hardware. In summary, it is easy to see why Tandem has one of the best price/performance ratios in the industry.

Reference

Anon, et al. 1985. A Measure of Transaction Processing Power. *Datamation*. Vol. 31, No. 7.

Peter Oleinick manages the Performance Section in Software Development. He joined Tandem in August 1978 to work on performance evaluation. As Software Development grew, a section was created under Peter's direction to concentrate on performance modeling, measurement, and analysis. Before coming to Tandem, Peter obtained a B.S. in Electrical Engineering from the University of Michigan, and an M.S. and Ph.D. in Electrical Engineering/Computer Science from Carnegie-Mellon University.

Praful Shah joined Tandem in June 1984. Since then he has worked with the Performance Group in Software Development on performance studies related to DP2, DP1, TMF, processors, and peripherals. Before joining Tandem, he worked in a performance group for another mainframe vendor. Praful has an M.S. in Computer Science from Pennsylvania State University and a B.S. in Electrical Engineering.

NonStop VLX Hardware Design

The team that designed the NonStop VLX processor accomplished significant performance increases by providing a greater number of dedicated hardware functional units than were implemented in the NonStop TXP system. The additional functionality was made possible through the use of bipolar VLSI gate arrays with an average density of 2300 equivalent gates. This article describes some of the considerations behind the NonStop VLX hardware design and their effect on performance.

Cost, performance, reliability, maintainability, and time to market are all considerations in the design of a processor. Performance for the NonStop VLX system was significantly improved, with attendant improvements in cost, reliability, and maintainability. The resulting CPU is well balanced in that all of its attributes were improved relative to its predecessor.

The Technology

Technology defines both the machine *envelope* and its cycle time. The envelope is determined by the number of chips needed and their power, cooling, and interconnect requirements. Each of the architectural styles described below has its own influence on the number of chips and chip types.

Gate Array Technology

The design team selected the VLSI gate arrays after performing an analysis of speed, density, costs, development tools, vendor capability, and product availability, considered in terms of product performance goals and development schedule.

Technologies considered included emitter-coupled logic (ECL), complementary metal-oxide semiconductor (CMOS), transistor-transistor logic (TTL), and combinations of these.

ECL interface levels were dismissed, among other reasons, because proven ECL-rules printed circuit (PC) design tools were unavailable. A choice was made to develop the chip design tools first, then tackle the ECL PC tools for a later product. ECL internal gates, however, have a definite speed advantage over CMOS or other bipolar designs.

In considering TTL versus CMOS, the ability of TTL to drive heavy loads was a deciding factor. The power and cooling requirements of a mixed ECL internal, TTL-interface gate array were deemed manageable. The chosen array uses ECL internal logic with TTL I/O signals.

RAM Technology

Both the inherent speed of the gate array (and the wiring delays involved in their interconnection) and the speed of the supporting Static RAM (SRAM) arrays were considered in establishing the cycle time of the processor. There is more than 3/4 Mbyte of SRAM in the CPU. After assessing the product introduction date, RAM vendor predictions, technology trends, second source risks, and performance objectives, the design team decided to pursue architectural performance enhancements rather than faster cycle time. The resulting design is quite conservative; neither the gate arrays nor the RAMs are being pushed to their limits.

Given the decision not to reduce the cycle time to gain performance, the team then considered the options available in the realm of architecture.

Performance Improvements by Architecture

Architectural techniques for performance improvement might be classified as hierarchy, specialization, and replication. Each technique is discussed briefly below.

Hierarchy

A hierarchy is a layered structure. In the case of a processor's memory system, the layers are disk, main memory, cache, and the processor's registers. This list is ordered by decreasing size and access time, and increasing cost per bit.

Over a sufficiently small interval, a program does not need to access the entire data base available to it, and the portion which it does need to access may be moved into faster memory to increase performance. Movement of data from disk to main memory is controlled by software. Movement of data from main memory to cache is controlled by hardware responding to a program's memory reference requirements. Data moves from cache to the internal processor registers in response to the execution of instructions by the processor.

A hierarchical memory system exploits the statistics of a program's memory access pattern. Memory references are obtained from the cache more than 96% of the time in typical on-line transaction processing (OLTP) applications. These accesses are serviced in one cycle, as compared to ten cycles for those accesses which are not cache-resident.

Disk-resident data is also cached in main memory by the DP2 software, providing applications with fast access to frequently used disk data. The 16-Mbyte capacity of the VLX processor provides the ability to cache many disk blocks.

Specialization

Specialization is a design style that breaks a problem into small parts and devotes facilities to the solution of each part. A specialized design is optimized to do one thing well.

The high-level specialization for the NonStop VLX system is OLTP. This specialization does not detract from its performance in other applications, however. The VLX executes an extension of the same instruction set as its architectural predecessors, the TXP and NonStop II CPUs. The extensions are transparent to nonprivileged application programs. This instruction set was designed with OLTP in mind. The VLX has many specialized hardware elements, each optimized to execute its part of this instruction set.

Each of these specialized elements may operate concurrently, thus speeding the execution. The instruction pipeline, the displacement adder, and the barrel shifter are examples of specialization. They are discussed in more detail in the "Hardware Features" section.

Replication

Replication is characteristic of the NonStop™ macroarchitecture and is also evident in the microarchitecture of the NonStop VLX system. Replication allows multiple activities of a particular class to be performed simultaneously. The newly introduced dual-bank control store is an example. Two accesses are in progress at any time, effectively doubling the bandwidth of the control store memory.

Measuring the Application

Extensive measurements were made in a transaction processing benchmark to determine what architectural features would provide the most benefit. The debit-credit (ET1)[1] transaction processing benchmark simulates a debit-credit application and exercises the disk process, the PATHWAY transaction processing system, and the Transaction Monitoring Facility (TMF) in an OLTP environment. These measurements yielded instruction usage data for the NonStop TXP system, both by frequency and by percentage of total execution time. (See Table 1.) More important, perhaps, was the ability to see the usage of various machine resources in an OLTP environment. (See Table 2.)

Hardware Features

The data from which Table 1 was extracted was thoroughly studied to find out how the TXP was spending its time. Architectural enhancements were conceived, analyzed, and modeled to find the most cost-effective ways to improve on the TXP's performance.

Cache

The cache-fill routine was improved from a 23-cycle microcode routine to a 10-cycle hardware-driven interface. This reduces the cache-miss penalty on the average instruction from 1.33 to 0.41 cycles.

The cache-miss penalty (the time needed to get the data from main store if it is not in the cache) is also a function of the cache organization. Alternative cache addressing mechanisms and organizations were investigated, but none showed any consistent advantage over the way the NonStop TXP cache was organized.

[1]The benchmark was based on the debit-credit, or ET1, benchmark standard defined in "A Measure of Transaction Processing Power," *Datamation*, April 1, 1985.

Table 1.
Top ten instructions in transaction processing benchmark.[1]

By frequency of execution	By percentage of total execution time
LDI	LOAD direct, unindexed
LOAD direct, unindexed	LDI
LDXI	LWXX (L-relative)
LDD	PUSH
BAZ	BAZ
STOR	MVBX
DADD	LDD direct, unindexed
ANRI	LOAD indirect, indexed
LOAD indirect, indexed	SEND
CMPI	PCAL

[1]See the *System Description Manual*, Part no. 82507 A00, for a description of the instruction mnemonics.

Barrel Shifter

Data scaling (shifts) and byte manipulation are somewhat cumbersome operations in some architectures. The VLX has dedicated a gate array to performing these functions in a single cycle. This function reduces the average instruction time by a fraction of a cycle.

Microcode Architecture

Each instruction in the NonStop TXP system ends with a FINIS microinstruction which calculates the displacement address for the next macroinstruction. This means that the shortest TXP instruction is two clocks in length. These two-clock instructions account for a significant portion of the execution time.

The designers dedicated NonStop VLX hardware to the displacement address calculation, making the shortest instructions one clock in length. This hardware is the "displacement adder" function, which examines instructions in the pipeline and calculates their operand address. This reduces the average instruction time by one cycle.

The Interprocessor Bus (IPB) Subsystem

The IPB out queue of the NonStop TXP processor is only one packet deep. The NonStop VLX processor incorporates a 16-packet buffer to reduce blockage due to the IPB output queue. A small fraction of a cycle per instruction is thus saved.

The IPB protocol was redesigned in order to reduce expensive cabling and improve the electromagnetic interference characteristics of the bus. In the process, the IPB bandwidth was raised from 13.3 Mbytes per second to 20 Mbytes per second. This performance improvement provides for the increased bus use expected from future high-end systems. A pseudo-random poll sequencer reduces the average latency of the IPB by a factor of two, providing improved Message System response.

The FOXII™ fiber optic extension provides improved buffering and buffer management strategies to reduce the latency of intersystem traffic. In order to maintain compatibility with FOX™ on NonStop TXP or NonStop II systems, the data rate and packet structure of the optical link have not been changed.

I/O Subsystem

I/O latency and throughput are enhanced by the reconnect poll hardware. The reconnect poll sequencer generates an alert signal six cycles before the poll is complete. Certain lengthy instructions test this condition before executing, and defer starting if the condition is true. This saves the overhead of saving and restoring state to service the reconnect.

Dual-bank Control Store

The highest-speed 8Kx8 SRAM believed feasible in early 1986 had an access time of 70 ns. Since the cycle-time goal was 83.3 ns (12 MHz), there was insufficient time to propagate the control-store address and meet setup-time requirements in a single-cycle access. Faster RAM was available in 16-Kbit density, but the physical requirements would overflow the printed circuit board partitioning. This led to the use of a dual-bank overlapped control-store approach, in which two identical banks of control store are accessed on alternate cycles. This approach gives the VLX the same effective control-store bandwidth as that of the TXP with the use of slower parts.

The presence of two identical copies of control-store data provided an opportunity to correct "soft" errors in one bank. A soft error is one that can be corrected by rewriting the correct information into the memory. Soft errors are the result of the interaction of alpha particles from trace impurities in the chip package with the circuitry on the chip. Their frequency is low, but not so low as to be negligible. When an error occurs, the error address is saved and used to access the other bank. The correction takes three additional cycles, effectively operating at one-third speed. In fact, the CPU runs with only a single bank, but at only one-third speed. Software is notified of the erring address, which is rewritten with data from the good copy.

Table 2.
Comparison of NonStop VLX and NonStop TXP systems, based on an average instruction.

Characteristic	NonStop TXP	NonStop VLX
Cycles/instruction (with 100% hits)	5.03	4.03
Data reads/instruction	0.63	0.63
Data hit rate	96.31	96.31
Code reads/instruction	1.14	1.14
Code hit rate	96.95	96.95
Cache miss time (cycles)	23	10
Cache miss penalty/instruction	1.33	0.41
Page Table Cache (PTC) hit rate	99.74	99.80
PTC miss time (cycles)	40	38
PTC miss penalty/instruction	0.03	0.02
Pauses/instruction	0.12	0
Total cycles/instruction	6.66	4.46
VLX:TXP speed ratio (instruction execution)	1.49	

On-line Sparing

The logic density available in the VLSI gate arrays made it possible to implement on-line sparing in the control-store, cache, and scratchpad SRAM arrays. When a hard (uncorrectable) error is detected, the logic receiving the bad bits is commanded to take its data from the spare RAM instead of the normal one. Hard errors (chip failures) in these SRAMs are very infrequent, but the sparing was determined to be cost-effective in light of the service cost goals of the product. The spare RAM is "revived" from the data in the other control-store bank. Once the revive is complete, all operations continue at full speed.

Memory Control Unit

The main memory accesses may be pipelined, with one access buffered for delivery to the processor data path while a second access is under way in the memory array. Dedicated address registers for the block move instructions MOVW and MOVB, along with the pipeline capability, permit the MOV loops to execute in three clock cycles.

The VLX Outperforms the TXP

As a result of these architectural enhancements, the VLX executed instructions at 1.5 times the rate of the TXP. Transaction throughput relative to the NonStop TXP system will typically be greater than this. (The factor will vary depending on the application.) The greater transaction throughput was a bit of a surprise and at first seemed to be pulling a rabbit out of a hat. At a specified response time, however, some portion of the processor's capacity is not used. The faster the processor, the more of it there is to use at a specified response time.

The following bank teller analogy, from Bob Horst of Tandem's Processor-Memory Group, may provide more insight: if a response time of one minute is needed and the bank tellers take one minute per transaction, a line cannot be allowed to form behind each teller. (In this example, the tellers aren't very busy because there is no line or queue to even out the bunchy arrivals of new customers.)

If the tellers were replaced with faster ones who took only 30 seconds per transaction, throughput would double. In addition to this, an average of one person could be allowed to wait in line for each teller, making the tellers busier as well.

The double multiplier is the faster speed of the tellers times the higher utilization of each teller. If a response time were not specified, long lines would form behind the slow tellers and shorter ones would form behind the faster ones, but the utilization would be the same (100%). Quite simply, with a more powerful processor, less reserve capacity is needed to meet peak demands; this translates as fewer idle tellers.

The performance of the VLX compared to that of the TXP is

$$\frac{\text{VLX throughput}}{\text{TXP throughput}} = \frac{(1 - \text{VLX idle})}{(1 - \text{TXP idle})} * \text{VLX performance.}$$

The equation simply relates the portion of the machines that are actually used at the one-second response level to the increased hardware performance of the NonStop VLX processor. The following article, entitled "NonStop VLX Performance," provides more detail on this "double multiplier effect."

Conclusion

The NonStop VLX processor has achieved significant performance improvements over the NonStop TXP system without raising the clock speed through the use of specialized and replicated hardware functions. The conservative use of VLSI technology permitted the additional hardware to be incorporated with a reduction in board count, power consumption, and failure rate.

Reference

Anon, et al. 1985. A Measure of Transaction Processing Power. *Datamation*. Vol. 31, No. 7.

Acknowledgments

Thanks are due the entire design team of the NonStop VLX system, with special thanks to Shannon Lynch for the design and development of the SIMPLE performance monitor, and to Bob Jardine and Lino Costantino for their performance measurement experiments and analysis. Thanks also to Bob Horst for clarifying the double multiplier effect.

Mike Brown has been with Tandem for three years, working on the development of the NonStop VLX processor. He has previously held senior engineering and product planning positions, with 19 years of experience in the minicomputer business.

This article presents performance data for the NonStop VLX system. A description of the features responsible for these performance improvements is contained in the preceding article, "NonStop VLX Hardware Design."

To characterize application performance on the NonStop VLX system, a series of tests was run on two identical NonStop VLX and TXP hardware configurations running the B20 and B30 versions of the GUARDIAN 90™ operating system. Two of these applications are presented here. The first test was a sequential copy operation; the second was an on-line transaction processing (OLTP) application. This article discusses the workload for each application and describes the configuration. Test results and observations are listed at the end of each section.

Sequential Copy Benchmark

Application

To perform the sequential copy (also called the SCAN or mini-batch) application, a 100-Kbyte file was copied from one file to another residing on the same disk. This was done using a program written in TAL™, Tandem's Transaction Application Language, and a single disk extent was used for each file. A single mirrored volume contained both files. Three different versions of this application were run.

The first version used unstructured access to the file. Twenty-five 4-Kbyte blocks were read and then written.

For the two structured access versions (one with transaction protection, the other without protection) 1,000 100-byte records were read and then written. Sequential block buffering was used for reading. Writes made during the file close operation were buffered in cache. The time to close was included in the elapsed time.

Configuration

Identical hardware configurations were used for each of the tests. This configuration consisted of a two-processor NonStop system (TXP or VLX) using two 3107 disk controllers to access a single mirrored disk volume. The disk volume was configured for parallel writes. (See Figure 1.)

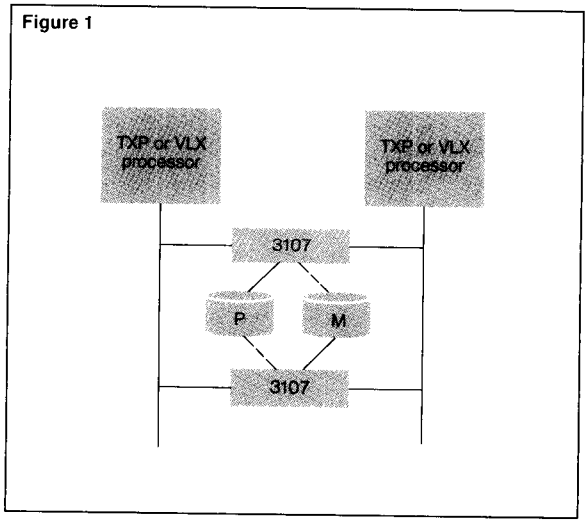


Figure 1.
Sequential copy benchmark configuration. Each processor was configured with 8 Mbytes of memory.

Table 1.
Sequential copy results (records copied per second).

	NonStop TXP/B20	NonStop VLX/B30	Improvement
Unstructured access read/write (25 4-Kbyte blocks)	581	709	22%
Structured access (1,000 100-byte records)[1]			
Read/write without TMF	149	224	50%
Read/write with TMF	115	165	43%

[1]Sequential block buffering for reads; buffered writes.

Table 2.
Sequential copy results (elapsed times in seconds).

	NonStop TXP/B20	NonStop VLX/B30	Improvement
Unstructured access read/write (25 4-Kbyte blocks)	1.72	1.41	22%
Structured access (1,000 100-byte records)[1]			
Read/write without TMF	6.27	4.47	40%
Read/write with TMF	8.72	6.06	44%

[1]Sequential block buffering for reads; buffered writes.

Figure 2

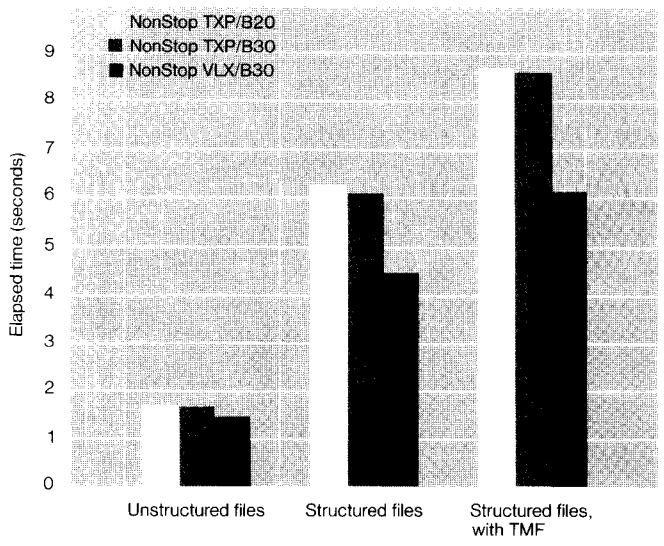


Figure 2.
Sequential copy results (elapsed time to copy a 100-Kbyte file).

Results

Tables 1 and 2 demonstrate both the number of records per second that can be copied, and the total elapsed time to copy the entire file. Figure 2 graphically compares the elapsed time required to copy the file for both the TXP and VLX processors. An improvement of 22% was measured for unstructured files. This modest improvement can be explained very simply. I/O time is the major component of elapsed time. Since no changes were made to the I/O subsystem, little improvement can be expected here.

For structured files, the elapsed time was reduced by 40%. This value approaches the raw processor speedup of 50%. When the application files were audited by TMF, the improvement increased still further to 44%. This additional improvement is due to the slightly larger CPU requirement for transaction protection.

The structured file access improvements are due to a larger portion of the elapsed time being spent in the processor. For the structured files protected by TMF, the audit trail was placed on a separate mirrored disk volume that was primaried in a different processor than the TAL program driving the application.

The high-performance XL8™ disk drive, as sold with the packaged NonStop VLX system, was not tested but could further improve these times. A processor-only comparison is presented in this article.

On-line Transaction Processing Benchmark

Application

A debit-credit banking application (known as ET1) using the full ENCOMPASS™ line of application development products was measured:[1]

- A SCREEN COBOL requester was used by the PATHWAY Terminal Control Program (TCP), which sent requests to a server written in COBOL.
- The TCPs were run with the auto restart option, and backup disk processes were run (the default). The backup processor maintained a second disk cache.

[1]This benchmark was introduced in *Datamation*, April 1985, "A Measure of Transaction Processing Power."

- All application files were mirrored and audited by TMF without audit compression.
- The ENCORE™ stress test generator was used to simulate the 800 terminals submitting transactions. It also provided response-time and transaction-rate data.
- The XRAY™ performance analysis tool was used to collect performance data.
- The ENFORM™ query language and report formatter was used to process the performance data.

The transaction flow is outlined in Figure 3.

The application data base in the example consisted of over 200 Mbytes of application data in four files. Three of the application data base files were accessed randomly. The fourth file, an entry-sequenced log, was written sequentially, one record for each transaction. A complete description of each application file is contained in Table 3. All three structured file types were used.

Configuration

Each system consisted of four NonStop TXP or VLX processors with 8 Mbytes of memory each. There were a total of four 3107 disk controllers per processor pair for the TXP system. The VLX system was configured identically, also with 3107 controllers. (This differs from the packaged NonStop VLX systems sold today.[2]) Sixteen disk drives were used to make eight mirrored volumes. The 800 simulated terminals were equally distributed on each of the systems. A diagram of the configuration is shown in Figure 4.

The processes were evenly distributed to balance the load across the system. Two disk volumes were primaried in each CPU. Additionally, two volumes had backup processes in each processor. A total of 16 TCPs (four per CPU) were used for both tests. ENCORE simulators and PATHWAY servers were evenly distributed. Three DP2 disk processes per volume (the default) were used. The READLINK caching feature of the B30 Message System was also used for our testing. (For more information on the B30 Message System, see the accompanying articles, "Performance Changes to the GUARDIAN 90 Message System," and "Message System Performance Tests.")

[2]The packaged VLX configuration replaces a pair of 3107 disk controllers (1.2 Mbyte per second transfer rate) with a pair of 3108 disk controllers (1.8 Mbyte per second transfer rate) connected to the XL4 disk drive.

Figure 3

Requester flow.

Accept 100 bytes.
Begin TMF transaction.
Send 100 bytes to server with 100-byte reply.
End TMF transaction.
Display 100 bytes.

Server flow.

Read 100 bytes from TCP.
READ Account (random, not cached).
UPDATE Account.
WRITE History (sequential, cached).
READ Teller (random, cached).
UPDATE Teller.
READ Branch (random, cached).
UPDATE Branch.
Reply 100 bytes to TCP.

Figure 3.
Transaction description.

Table 3.
OLTP benchmark file description.

File type	Name	Record size	Number of records	Notes
Key-sequenced	Account	100 bytes	2 million	Large key-sequenced file, not cached
Relative	Branch	100 bytes	200	Random access, in cache
Relative	Teller	100 bytes	800	Random access, in cache
Entry-sequenced	History	50 bytes	1 per transaction	Sequential access, in cache

Figure 4

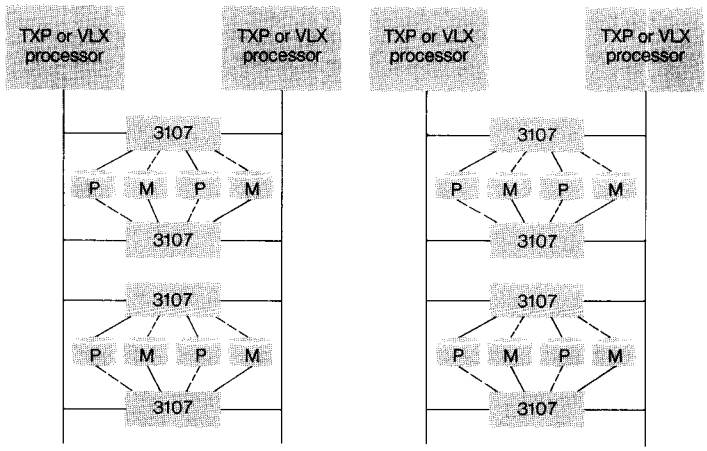


Figure 4.
OLTP benchmark configuration. Each processor was configured with 8 Mbytes of memory.

Figure 5

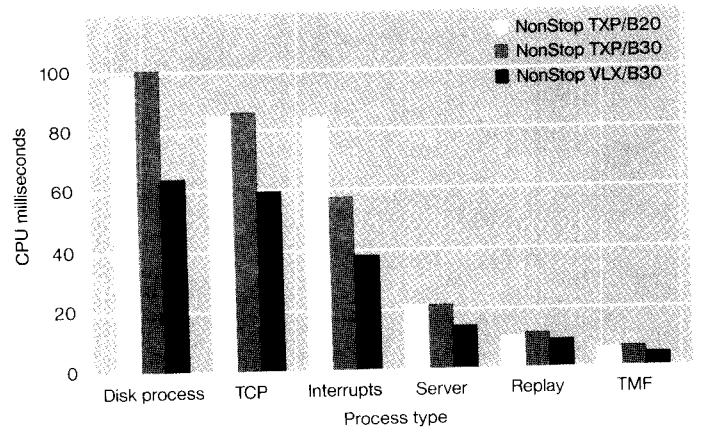


Figure 5.
OLTP benchmark results (CPU milliseconds per transaction by process type).

Results

This discussion focuses on CPU demand, throughput of transactions, and increased utilizations with identical response times.

CPU Usage. The rate at which the VLX processes transactions is 1.64 times the rate at which the TXP processes transactions. In other words, the TXP CPU time divided by 1.64 equals the VLX CPU time. Figure 5 breaks this time down by process type for both the VLX and TXP. This processor power improvement has two components: the VLX hardware improvements and the Message System software improvements.

When executing the debit-credit banking application instruction mix with system software identical to that on the TXP, the VLX processor power alone was 49%, or 1.49 times, better than that of the TXP. This was computed by comparing the CPU demand per transaction for both systems. For more information about the specific features of the processor that provides this increased level of performance, refer to Mike Brown's article, "NonStop VLX Hardware Design," or the VLX data sheet.

The B30 Message System changes shortened the instruction path to send a message by about 30%. B20 message processing represented approximately one-third of the total CPU processing time in the test. With the new Message System, the message processing portion of the CPU time was reduced to about 20% of the total. In other words, the total CPU demand for this application was reduced by 10% through changes to the Message System.

This 10% improvement (or 1.1 times increase) in CPU power multiplied by the 49% faster (or 1.49 times instruction speedup) provides an improvement of 64% (or 1.1 * 1.49 = 1.64 increase in processing power). This value is used for the computation preceding Table 4.

Reduction in Interrupts

The burden of processing interrupts has been greatly reduced on the NonStop VLX system. The changes to B30 Message System software and to the interprocessor bus (IPB) in the VLX hardware reduced interrupt time by 56%.

Throughput and Response Time

Response time and throughput improvements can be seen in the curves in Figure 6.

An important observation can be made from this figure. By comparing the increased workload that the VLX system can process at a specified response time, the NonStop VLX throughput improvement can be determined. Figure 6 shows that with a one-second response-time requirement for 95% of the transactions, the VLX running the B30 version of the operating system delivered two times the throughput of a similarly configured TXP running the B20 version of the operating system. Intuitively, this improvement seems too large given that the processing power is only 64% greater than its predecessor. However, the *double multiplier effect*, discussed later in this article, explains this unexpectedly large performance improvement.

For applications requiring a particularly stringent response time, the NonStop VLX offers a significant reduction in response time. At equal transaction rates, the VLX provided a response time of 0.46 seconds for 95% of all transactions. The NonStop TXP yielded a response time of 1.13 seconds. The reduced response time (over 50% with the VLX processor) increases both the number and the types of critical-response application solutions that can be supported on Tandem systems.

The Nature of On-line Transaction Processing

OLTP workloads usually require that a certain response-time requirement be met for individual users at terminals or workstations. The amount of work that the system can perform, while not exceeding that response-time requirement for a large percentage of the users, is the single most useful measure of system performance. Because of this response-time requirement, systems are typically run at much less than 100% processor utilization to allow the system to respond quickly to the user.

Queuing and Utilization. The OLTP-specific response-time requirement determines which system resources are available to deliver work before the alloted time has elapsed. A faster CPU allows longer queues to be processed in the same amount of time that a slower CPU can process a shorter queue. This, in turn, allows the NonStop VLX processor to operate at a higher utilization while still achieving the response-time goal. This increased utilization is possible because of a longer allowable queue length in the processor. The effects of this are characterized below.

As an example, assume the following:

- The response-time requirement is 0.5 seconds (not including such elements as communication time delays—just a host delay requirement of a 500 ms or less).
- Only a single CPU exists.
- The job uses 300 ms of CPU time to complete processing.

In a perfect system, 0.66 of the transactions could be in the queue to be processed before "transaction T" and the response-time requirement could still be met for that transaction:

$0.66 * 300 = 200$ ms elapsed time.

This would leave 300 ms of processor time remaining before the response time would exceed the half-second requirement.

Now assume that the processor is 1.64 times faster and, therefore, the service time requirement drops to 300 divided by 1.64, or 182 ms. Now, 1.75 other transactions can be queued up in front of transaction T and the processor can still meet the response-time requirement.

Figure 6

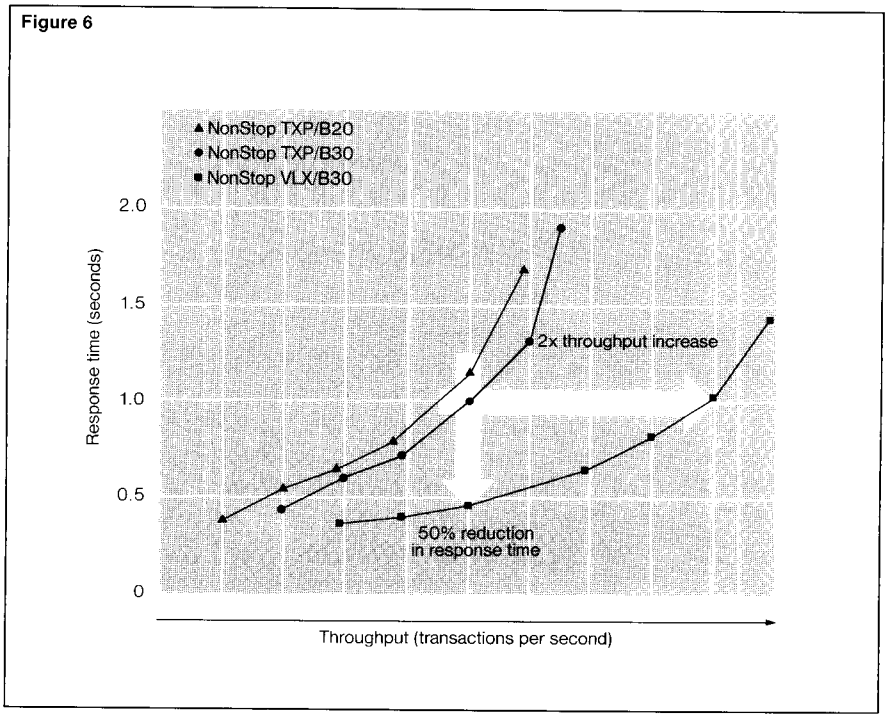


Figure 6.
OLTP response time versus throughput with TMF.

With 1.75 transactions queued up for the processor instead of 0.66, the faster processor exhibits a higher utilization than the slower processor. This higher utilization allows additional work to be performed beyond what one would expect from a "simple" 1.64 times processor speedup. This higher processor utilization has been measured at identical terminal response times (see Table 4). The equation

$$\frac{83.0\ (\text{VLX CPU utilization})}{69.7\ (\text{TXP CPU utilization})} * 1.64 = 1.95$$

describes the impact on system performance.

The Double Multiplier Effect. Not only does the faster NonStop VLX process the transaction in less time, but it also allows more of the CPU to be used. Tandem refers to this as the double multiplier effect.

Table 4.
OLTP throughput capacity increase.

	Response time[1]	Average CPU utilization
NonStop VLX/B30	1.01	83.0
NonStop TXP/B20	1.00	69.7

[1]For 95% of all transactions.

The two times throughput improvement at equivalent response times measured for NonStop VLX systems can be attributed to three major factors:

- A more efficient processor.
- Higher allowable utilization.
- More efficient software.

At a one-second response time for 95% of the transactions, the CPU BUSY rate for the TXP averaged 70% (Table 4). Because of its ability to sustain longer queue lengths without increasing the response time, the Nonstop VLX processor operated at an average of 83% while still responding within one second for 95% of the transactions. This ability to operate at higher CPU utilizations, coupled with the decreased CPU demand per transaction, provides the double multiplier improvement.

The Nature of the VLX Processor

The NonStop VLX processor was designed to perform in a demanding OLTP environment. Examples of design innovations are found in the larger page table cache and faster hardware cache fill times. Such features become more important in demanding, "real life" processing situations.

The cache fill times have been reduced from over 20 to 10 clock cycles. The cache memory itself provides a faster cycle time. A new IPB with a faster cycle time and a larger outgoing queue for messages reduces main processor involvement with message traffic. Additionally, movement of I/O buffers into the hardware cache was eliminated, thereby increasing the efficiency of the hardware cache.

Because of their size, small test applications used to benchmark applications may not demonstrate the full benefit of these features. They are usually smaller in scope than the real applications they mimic, with little or no hardware-cache-filling requirements during the test. Additionally, benchmark memory requirements are sometimes smaller than their real life counterparts.

Different types of user applications will experience different performance improvements with the NonStop VLX system. The applications deriving the greatest benefit are those that are currently processor-limited, rather than I/O-limited, although even I/O-bound applications will benefit from decreased service costs and better price/performance too. Those applications that also have stringent response-time requirements will especially benefit from the faster VLX processor. The key to determining performance improvements beyond the "raw power increase" of the NonStop VLX processor lies in increasing the processor utilization. If processor utilization cannot be increased, performance improvements may be below the range of 1.6 to 2.2.

Conclusion

As the range of workloads processed on Tandem systems expands, so must the range of benchmarks used to investigate system performance. The debit-credit benchmark model has provided an excellent measure of one type of transaction. Future benchmarks, with more varied transaction types and mixed batch/OLTP workloads, will be required to further study system performance.

The NonStop VLX processor improved batch application performance by 40% in the sequential copy tests. The raw processor power for our OLTP instruction mix improved by 50%. New system software available with the B30 release of GUARDIAN 90 added 10% to this performance improvement on both the NonStop VLX and TXP processors. Because of the double multiplier effect, OLTP benchmark performance was improved by a factor of two. Tandem delivered these performance improvements along with a reduced cost of ownership, improved reliability, availability, serviceability, and industry-leading price/performance.

Reference

Anon, et al. 1985. A Measure of Transaction Processing Power. *Datamation*. Vol. 31, No. 7.

Acknowledgments

The information presented in this article was generated from measurements performed by the Performance Analysis and Measurement Group early in 1986. This group included (but was not limited to) Pat Beadles, Nhan Chu, Praful Shah, Susan Uren, and the author. Additionally, the author would like to thank Susan Whitford for her contributions.

Jim Enright, manager of the Performance Analysis and Measurement Group in Software Development, joined Tandem in August 1983. His group has been involved in testing and analyzing the performance of new and existing Tandem software and hardware products.

Message System Performance Enhancements

For the B30 software release, large parts of the GUARDIAN 90 Message System were rewritten to improve performance. As a result of these changes, the Message System sometimes operates internally differently from the way it did before B30, and there are new SYSGEN options useful for tuning a system to peak performance. This article describes the changes in how the Message System works, for the benefit of those engaged in improving system performance. This information should also be helpful to anyone who would like to understand the Message System.

From an external viewpoint, the overall operation of the Message System is the same as before the B30 release, as described by Mala Chandra in the *Tandem Systems Review* (February 1985). The same procedures are called by Message System clients (such as the File System), producing the same results, and the fields in the link control block (LCB) used by Message System clients are the same as before. It is only the internals of the Message System, including LCB fields not used by Message System clients, that were modified to improve performance.

Two Main Changes

Two main changes were made:

- Message System code and low-level data structures were streamlined (i.e., rewritten for higher performance).
- The Message System's protocol for sending the text of a message (request) across the interprocessor bus (IPB) from the requester to the server was improved by arranging to send the data associated with a request across the bus earlier, so the server doesn't have to request the data across the bus. This change is variously called *Expedited Request Transmission* or *READLINK caching*.

Each of these changes accounts for about half of the Message System performance improvement.

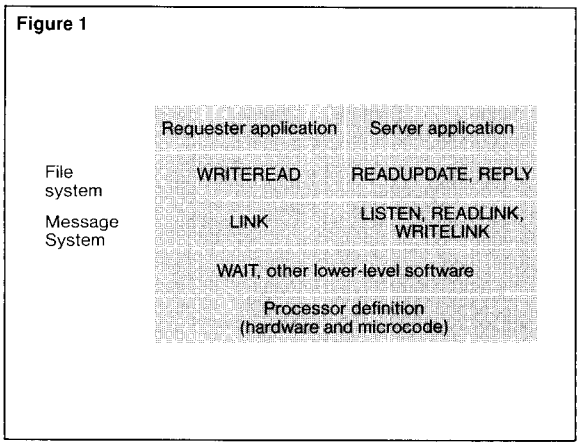
The code streamlining is the largest change in terms of the number of lines of code changed. However, it is largely an invisible change and there is nothing to tune to get the maximum benefit from this change. Accordingly, this article discusses it only briefly, after discussing other changes.

The change to the IPB protocol is a much smaller change, as measured in lines of code. However, the change is more visible and there are some SYSGEN options that you might want to adjust to tune your system for better performance or better use of memory (although the defaults should be reasonable for the great majority of installations). Accordingly, the article discusses this change in detail.

Definition and Overview of the Message System

In the Tandem system design there are many layers or levels. Components at higher levels are built on top of components at lower levels. Components at higher levels provide more complex and abstract capabilities, such as the ability to write to a file. Components at lower levels provide more concrete capabilities, such as the basic processor definition and machine instructions.

Figure 1.
The relationship between the Message System and other layers of GUARDIAN 90.



The Message System is at a level above the basic processor definition but below the File System and I/O processes. The processor provides facilities such as:

- The SEND instruction.
- The transmission or reception of a series of related bus packets.
- BUSRECEIVE interrupts.

(These facilities are described in the *System Description Manual* for NonStop systems.)

The Message System builds on the facilities supplied by the processor to provide higher-level facilities related to the sending of messages. The most important such facility is the ability of a process to send a message (request) to another process and receive a response (reply) back from the process. (See Figure 1.)

The File System, I/O processes, and other higher-level components use messages for almost all process-to-process communications, whether or not the processes are in the same CPU. Because of the large amount of process-to-process communication typical of on-line transaction processing, improvements in performance of the Message System result in significant improvements in overall system performance.

Message System Procedures

The process-to-process message protocol is based on a requester-server model. For every message, one process is the requester and one process is the server. The requester and server are usually called the *Linker* and the *Listener*. The requester (Linker) initiates the message link and calls the LINK procedure. The server (Listener) listens for requests and calls the LISTEN procedure.

Privileged Procedures

The Message System procedures (LINK, LISTEN, BREAKLINK, READLINK, and WRITELINK) are privileged procedures. In a typical system most requests are sent to privileged I/O processes such as disk processes, which call LISTEN, READLINK, and WRITELINK directly. Also, many requests are sent by privileged processes, such as the case of a primary disk process checkpointing to its backup process. Requests sent or received by privileged processes often dominate considerations of performance.

File System Procedures

Using the File System, nonprivileged processes can act as either requesters or servers. A nonprivileged requester calls WRITEREAD, which calls LINK to initiate the request, calls WAIT to wait for completion, and then calls BREAKLINK to complete the interaction. A nonprivileged server calls READUPDATE to obtain the request and calls REPLY to reply to the request. READUPDATE calls WAIT to wait for a request, calls LISTEN to get a small part of the request, and calls READLINK to get the text of the request. REPLY calls WRITELINK to reply to the request.

Message-exchange Protocol

The message protocol follows this sequence of events:

1. The Linker initiates the interaction by calling LINK. The Linker then specifies six words describing the type of request, a buffer containing request text, and the amount of request text in the buffer. Upon completion of the request, the request text will be replaced with response/reply text, and the six words describing the type of request will be replaced with status. The Message System allocates a Linker's LCB (whose address is given to the Linker) and a Listener's LCB (which is inserted into the queue of requests for the Listener). If the Listener is waiting for a request, it is woken up.
2. The Listener calls LISTEN to get the address of the LCB for its next incoming request. The LCB contains six words of data (P1 through P6) describing the type of request and the amount of additional data (request text) for the request.

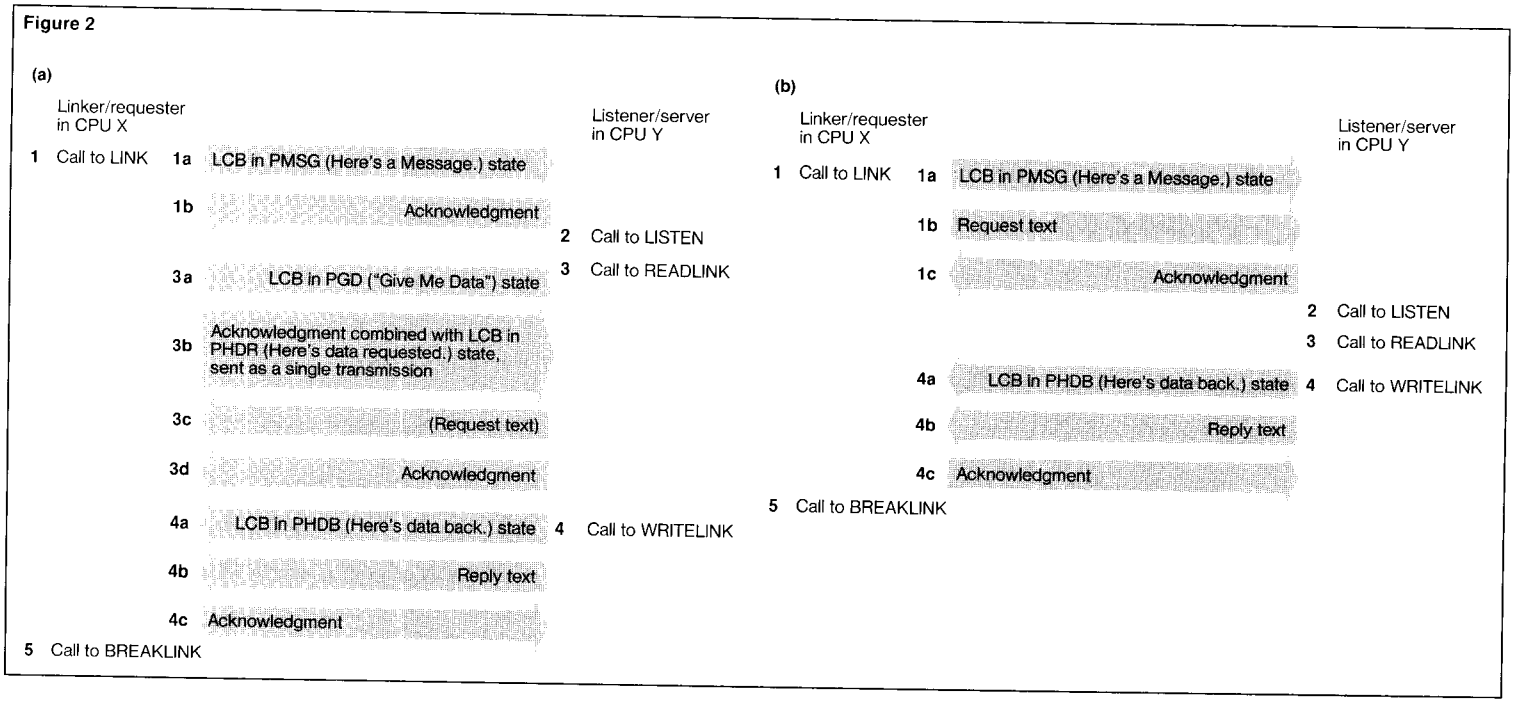


Figure 2.
IPB protocol. (a) Without READLINK caching. (b) With READLINK caching.

3. The Listener allocates enough memory to hold the request text, and calls READLINK. READLINK obtains the request text from the Linker's buffer and copies it into the Listener's buffer. Usually, the Listener calls READLINK once, very soon after being woken up to process the request. However, the Listener is allowed to call READLINK as many times as it wants and might not call READLINK at all if, for example, there is an error in the contents of P1 through P6.
4. After performing the request, the Listener calls WRITELINK to reply to the request, sending back status information and additional data (reply text), if any. The Message System copies the status information and the optional reply text back to the Linker's LCB and the Linker's buffer, marks the LCB as "done," and wakes up the Linker if the Linker is waiting for a reply. This ends the Listener's part of the interaction.
5. The Linker examines the status information and reply text, and then calls BREAKLINK to release the Linker's LCB.

This message-exchange protocol is used whether the requester and server are in the same CPU or in different CPUs. However, the Message System operations inside LINK, LISTEN, READLINK, WRITELINK, and BREAKLINK depend on whether or not the processes are in the same CPU.

If the Linker and Listener are in the same CPU, then the Message System uses ordinary data movement instructions to copy information back and forth between the Linker's and Listener's LCBs and buffers. If the Linker and the Listener are in different CPUs, then the Message System must use the IPB.

READLINK caching simplifies and speeds up the IPB protocol used when the Linker and the Listener are in different CPUs. The improvement in IPB protocol is readily apparent in Figure 2, which shows the protocols before and after the development of READLINK caching.

In Figure 2 the Linker's side of the typical interaction is shown on the left, and the Listener's side is shown on the right, with arrows representing transmissions over the bus.

As can be seen (Figure 2a), the READLINK call (step 3) takes four bus transmissions (and consequently generates four BUSRECEIVE interrupts) without READLINK caching. Also, without READLINK caching, execution of the Listener must be suspended pending arrival of the request text from the Linker's CPU. With READLINK caching (Figure 2b), all the bus transmissions for step 3 are eliminated except for transmission of the request text, which is moved from step 3c to step 1b.

Essentially, the request text is transmitted over the bus at the time of the LINK, in the expectation that the Listener will READLINK it soon. Since the request text is transmitted before the Listener has specified the place in memory to receive it, it is temporarily stored (cached) in a special pool of holding buffers (a READLINK cache buffer), to be available if the Listener asks for it soon enough. If the Listener does not ask for the text within a reasonable amount of time, the buffer may be "stolen" and used for another request.

Benefits of the New IPB Protocol

The B30 bus protocol saves CPU time:

- There is less bus activity and fewer interrupts.
- There is no need to wait for the data to arrive in the Listener's CPU. The Listener doesn't have to be suspended while waiting for the data, so the dispatcher doesn't have to switch to a different process and then back again.

A message must be reasonably short, or the costs of READLINK caching outweigh the benefits.

The new IPB protocol also reduces wait time in the server by doing more Message System processing in parallel with processing by the server process. The data for one request can be received while the server is busy processing another request. Because of this, a server can handle a larger number of requests without falling behind.

The preceding benefits can improve response time and/or throughput. Because the overall elapsed time to do a transaction has been reduced, it is possible to let the queues of pending requests for servers become a little longer while still maintaining the same response time.

Balanced against the benefits of the new protocol are the costs of using the cache:

- Physical memory for caching the data must be made available.
- There is extra data movement, from the cache to the Listener's memory.

Both costs of caching are higher for longer messages. Accordingly, GUARDIAN 90 uses READLINK caching for short requests and uses the old bus protocol for long requests. If the availability of memory were not a consideration, it would be beneficial to use READLINK caching for messages up to 3 Kbytes or so, depending on the processor. For longer messages, the additional costs in CPU time might actually decrease performance. Since the availability of memory *is* a consideration, and since most messages are *much* shorter than 3 Kbytes, SYSGEN normally sets up Message System data structures to use READLINK caching for messages up to 1536 bytes long.

Conditions for Using Cache

READLINK caching is used for short messages between different CPUs of the same system, when not disabled by SYSGEN.

As discussed, a message must be reasonably short, or the costs of READLINK caching would outweigh the benefits. The default definition of "short" is less than or equal to 1536 bytes, including File System overhead. There is a SYSGEN option to change this.

A message must also be between different CPUs of the same system, or data will not be transferred across the local bus. (The implementation does not include use of the new bus protocol from system to system using FOX.)

Finally, for cases where memory is very scarce or if for some reason READLINK caching were not beneficial, SYSGEN can be used to completely disable READLINK caching. This is done by saying that even a message of 0 bytes is a "long" message.

Data Structures Changed for READLINK Caching

This section provides a simplified description of the changes in data structures to explain how READLINK caching is implemented.

CACHEBUF

A new field (CACHEBUF) has been defined in a formerly unused word of the LCB. A Listener's LCB can be in any of the following states, as shown by values in this field:

- The LCB may have a READLINK cache buffer associated with it, ready for READLINK to use. If so, CACHEBUF points to the associated buffer, and the buffer points back to the LCB so the LCB can be updated if the buffer is "stolen" from the LCB.
- The request may have no READLINK cache buffer associated with it because READLINK caching was not used for this request. The request may have been too long, from a requester in the same CPU, or sent via FOX.
- The request may have no READLINK buffer currently associated with it because, although READLINK caching was used for the request, the READLINK cache buffer was later "stolen" from the LCB to be used for a more recently arriving request.

Cache Buffers

READLINK cache buffers are broken into two parts: a small READLINK cache header in system data space, and a large part in an absolute extended-memory segment which actually holds the request text.

A READLINK cache buffer can be in any of the following classes or states:

Unused. Initially, no memory is used for the "request text" part of READLINK cache buffers that have never been used. An unused LCB is available, if needed, to accommodate incoming requests, but the Message System will have to allocate ("lock") physical memory for the request text before bringing the buffer into use.

Receiving Text. A READLINK cache buffer is receiving text when a PMSG ("Here's a Message") LCB has been received over the bus from another CPU and the bus has been set up to receive the request text into the request text part of the READLINK cache buffer.

New. A "new" READLINK cache buffer has request text ready for a READLINK call to use, and is associated with an LCB. The Listener has not yet called READLINK to retrieve the request text. Typically, a new READLINK cache buffer is still on the incoming request queue for the Listener process.

Used. A "used" READLINK cache buffer is like a new one, except that the Listener has already called READLINK to retrieve the request text. The reason for distinguishing between new and used buffers is that, although a Listener is allowed to READLINK the text in a used buffer again, this is relatively uncommon. A used buffer is a better choice for stealing than a new buffer if a buffer is to be stolen for use with another incoming request.

Junk. A "junk" READLINK cache buffer is no longer associated with an LCB and its request text is no longer useful. Usually the buffer has been junked because the Listener completed processing for the request and replied back to the requester by calling the WRITELINK procedure. Junk buffers are the best choice for use with new incoming requests.

Handling the Pool of Buffers

When the BUSRECEIVE interrupt handler encounters the PMSG LCB, it checks to see whether or not this request will use READLINK caching; if so a READLINK cache buffer is allocated.

In choosing a buffer, the Message System uses the classes/states of READLINK cache buffers described in the preceding section, with the following rules:

- A junk buffer is the first choice, because the contents of a junk buffer are completely useless.
- A used buffer (as opposed to a new buffer) is the second choice, because it's unlikely that the request text will be linked a second time.
- An unused (not yet placed into service) buffer is the third choice.

- A new buffer (one not yet linked) is the last resort. (The count reported by PEEK as DISCARDED BEFORE USE is the number of times a new buffer is chosen.)
- A buffer that is already receiving text can't be chosen to simultaneously receive text from another CPU.

When the Message System chooses a junk buffer, a used buffer, or a new buffer, it chooses the oldest buffer of that class. For example, the oldest new buffer is the buffer that has been a new buffer for the longest time.

Figure 3

```
EXPEDITED REQUEST TRANSMISSION BUFFERS: 20        BUFFERS USED: 16
  REQUESTS CACHED: 6,896,254          DISCARDED BEFORE USE: 0.0%
  ATTEMPTS TO READ FROM CACHE: 7,015,553          SUCCESSFUL: 98.2%
  UNSUCCESSFUL: 1.8%    LENGTH > 1536b: 1.8%      TOO LATE: 0.0%
```

Figure 3.
Output from a "PEEK/ cpu 1/ EXP" command.

When an unused buffer is used for the first time, the Message System allocates real memory ("locked" memory, not pageable memory) for the request text part of the buffer. Currently, there is nothing in GUARDIAN 90 to ever undo this allocation of real memory; when a buffer is no longer needed it becomes a junk buffer. The number of buffers IN USE (as shown by PEEK) is the largest number ever needed since the CPU was last reloaded (the "high-water mark"). For example, 20 buffers "in use" times a buffer length of 1536 bytes would mean 30,720 bytes (15 pages) of physical memory were tied up for READLINK cache buffers, even if the buffers are currently empty. After the accumulation of more experience with READLINK caching, the software might be changed to periodically undo the allocation of one or more buffers. This should be done infrequently, to avoid degrading performance.

PEEK Instrumentation

PEEK has been updated to optionally display information on Expedited Request Transmission (READLINK caching). To display this information specify:

PEEK /CPU <cpu>/ EXPEDITED

(Note that *EXPEDITED* may be shortened to *EXP.*)

The four new lines of information described in this section are shown in Figure 3.

EXPEDITED REQUEST TRANSMISSION BUFFERS are the number of message cache buffers allowed in this CPU.

BUFFERS USED indicates the number of message cache buffers "in use" in this CPU. This number times the buffer length gives the amount of physical memory allocated for READLINK cache buffers. There is currently no mechanism for removing a buffer once it has been brought into use, so this number reflects the peak number of buffers needed in this CPU.

REQUESTS CACHED is the number of expedited transmissions to this CPU (the number of times a message cache buffer has been allocated). This number is the base for the following percentage.

DISCARDED BEFORE USE is the percentage of requests cached that were discarded before the server asked for the data. If this percentage and the percentage discarded before use are both high, you may want to allow a larger number of message cache buffers for this CPU. Also, you might start by using MEASURE or XRAY to determine whether one or more servers in the CPU are running slowly. If so, you should fix that problem and then recheck the use of the message cache before altering the number of message cache buffers. (See the *MEASURE User's Guide* for more information on how to determine this.)

ATTEMPTS TO READ FROM CACHE indicates the total number of times a server in this CPU asked for the data part of a request sent from another CPU. This is the base for the following percentages.

SUCCESSFUL means the percentage of attempts to read from cache that were satisfied from the cache.

UNSUCCESSFUL refers to the percentage of attempts to read from cache that were *not* satisfied from cache.

LENGTH > 1536b is the percentage of attempts to read from cache that were not satisfied from cache because the data part of the request was longer than 1536 bytes, or whatever the message cache buffer size is (the value "1536b" is replaced with the actual buffer size if your SYSGEN specifies a different size).

TOO LATE is the percentage of attempts to read from cache that were not satisfied from cache because the buffer had already been stolen to make room for a later expedited transmission. If this percentage and the percentage discarded before use are both high, better performance might be obtained by increasing the number of message cache buffers in the CPU. Alternatively, it might be interesting to try to find out which servers tend to build up a backlog of requests and help them out.

Server Behavior and READLINK Caching

READLINK caching was designed to take advantage of typical server process behavior. Specifically, READLINK caching takes advantage of the tendency of servers to READLINK their requests promptly.

Even in a heavily loaded system, with queues of unprocessed requests building up for some servers, READLINK caching is still likely to be effective for the majority of requests. This is true because of the large numbers of requests that go to high-priority processes such as I/O processes and backup processes.

A problem that can occur, particularly in a testing environment, is that a server can get "stuck" and build up a big backlog of requests before someone stops it. This causes the Message System to try to hold onto requests that will never be processed by the intended recipient. Once the maximum number of buffers allowed have been brought into use, the Message System handles the situation pretty well by discarding the oldest of the new buffers, which is likely to be the requests going to the stuck process.

Ignore READLINK caching when you do your application design. Instead, concentrate on such things as reducing the number and complexity of disk operations. *After* the application is running, it might be appropriate to make minor adjustments to READLINK caching (such as increasing or decreasing the length of the cache buffers) to better handle the application.

SYSGEN Options and Defaults

Two optional parameters added to SYSGEN allow control of READLINK caching. If the parameters are not specified, standard defaults are used.

One of the new options is:

EXPEDITED_REQUEST_BUFFERS <number>;

The number of message cache buffers allowed per CPU is designated by using the EXPEDITED_REQUEST_BUFFERS clause. Different CPUs are allowed to have different numbers of message cache buffers. This parameter is specified in either the ALLPROCESSORS paragraph or the PROCESSORS paragraph, depending on whether or not the same number of message cache buffers are to be used for all the processors in the system. The minimum number of buffers is 15, and the maximum number is 512. Also, the total amount of memory (the number of buffers times the buffer length) must not exceed 64 pages (131,072 bytes).

If the EXPEDITED_REQUEST_BUFFERS clause is not used, a default of 20 buffers is provided.

The second optional parameter is:

EXPEDITED_REQUEST_LENGTH <length>;

The message cache buffer length is specified by the EXPEDITED_REQUEST_LENGTH clause. The same value applies to all CPUs of a given system. This clause is specified in the ALLPROCESSORS paragraph. The maximum value for <length> is 3072 bytes (one-and-one-half pages of memory). The default value is 1536 bytes (three-fourths of a page of memory).

To disable the Expedited Request Transmission feature, specify <length> as 0. SYSGEN then allocates no space for message cache buffers. PEEK will show the allowed number and the used number of buffers as 0.

So far, the default values have worked well in a variety of environments. It would be reasonable to use the default values at first, and to override them only if the defaults seem to be working poorly as evidenced by PEEK statistics, or if memory is scarce.

Code Streamlining

Although READLINK caching is the focus of this article, Message System code streamlining in B30 resulted in an approximately equal amount of performance improvement, without the need for additional buffer space or SYSGEN parameters.

Faster Code

Several approaches were used to get the code to run faster while performing the same operations as before.

More Efficient Data Structures. Additional information was grouped together with the control packet reception area in a new control block called a Bus State Descriptor (BSD). Grouping the information together made it faster to find.

To help find the BSDs quickly, there is an array of pointers to BSDs, called the BSD Address Array (BSDAA). For example, the address of the BSD for CPU 5 is in BSDAA[5]. This array avoids the need for a MULTIPLY instruction that would otherwise be used to index to the fifth BSD.

Within the BSD, there is a word of flags for infrequent cases, called BSDSPECIALCASES. By simply checking the word and learning that it is zero, the code is able to bypass several rare cases all at once, without having to test for the cases one by one.

Code Rearrangement to Reduce Conditional Branching. To avoid excessive tests, code was rearranged by separating cases. For example, there was a procedure called LINKMOVE that could be called in by either READLINK or WRITELINK, if the requester and server were in the same CPU. It moved data between requester and server message buffers in either of two directions: from the server to the requester, or from the requester to the server. However, the cases for the two directions were different enough that many tests for which case was being handled were needed inside LINKMOVE. The two cases are now handled separately by subprocedures of READLINK and WRITELINK; LINKMOVE has been eliminated.

Controlled Use of Index Registers. Controlling the use of index registers was done by using USE and DROP statements for registers 6 and 7, in sequences of code that repeatedly refer to fields within the same control block. Typically, register 7 points to a BSD and register 6 points to another control block, such as an LCB.

Conclusion

This article described the internal changes to improve performance in the Message System for the B30 release of GUARDIAN 90. Most of the discussion was of READLINK caching; code streamlining was briefly outlined.

Since the externals of the Message System were not changed—not even the interfaces used by privileged programs—these changes should affect you only by making your system run faster. If memory is very scarce or if your system is extremely performance sensitive, then you might need to adjust the new SYSGEN parameters. However, the default values appear to work well for the great majority of installations.

For a discussion of measurements of the performance improvement, please see the following article by Susan Uren, "Message System Performance Tests."

References

Chandra, M. 1985. The GUARDIAN Message System and How to Design for It. *Tandem Systems Review*. Vol. 1, No. 1.

Software Release Document for Release B30 of GUARDIAN 90.

System Description Manual for NonStop Systems. Part no. 82507 A00. Tandem Computers Incorporated.

Acknowledgments

The author wishes to acknowledge the work of Richard Larson, the lead developer for these Message System improvements; Richard Carr, who prototyped READLINK caching in Frankfurt and supplied programs for measuring performance in wonderful detail; and Susan Uren, who measured the results with realistic applications and workloads. Thanks are also due to Mala Chandra for allowing the author to rework figures from her article describing the Message System, and to Leslie White, Richard Carr, and innumerable others for their helpful review comments.

Dave Kinkade joined Tandem as a software designer in the GUARDIAN OS Kernel Group in February 1984. Before joining Tandem, Dave participated in the development of several widely used software products, including a large operating system and a high-performance data base system.

Message System Performance Tests

This article presents test results comparing the performance of the B20 and B30 releases of the GUARDIAN 90 Message System, as run on both the NonStop TXP and VLX systems. The performance improvements reflect two major enhancements to the Message System: READLINK caching and streamlining of Message System code. For a detailed explanation of these enhancements, see the preceding article, "Message System Performance Enhancements."

The Enhancements

Streamlining

Streamlining the Message System code involved optimizing the low-level code. Many parts of the code were optimized to reduce the number of processor cycles needed for message transfer. The streamlining has resulted in fewer instructions executed for the same amount of work done.

READLINK Caching

The READLINK caching protocol is a performance enhancement for messages up to 3072 bytes long that are sent between processors. The smaller the message is, the better the performance gain from READLINK caching.

When a message is sent from one CPU to another, the information sent consists of two parts: the LCB (Link Control Block), which contains information about the message, and the data. The LCB part of a message records such items as the message type, the receiving process's identification number, and the length of the message.

When a message is sent under the standard protocol (i.e., without READLINK caching), the Message System uses the following sequence of events:

1. The LCB is sent to the receiver's CPU.
2. The receiver's CPU allocates an LCB to store the information received and queues the message for the receiver.
3. When it is time for the receiver to read the message, the Message System supplies the LCB part of the message.
4. The receiver asks for the data part of the message and specifies a memory location at which the data can be stored.
5. The Message System in the receiver's CPU asks for the data from the sender's CPU, and the data is transmitted over the bus from the sending processor to the receiving processor. This step causes dispatches and bus-receive interrupts in both processors.

When a message is sent using the READLINK caching protocol, the sequence of events is as follows:

1. The LCB is sent to the receiver's CPU, and the data is sent after it. When the Message System at the receiver's CPU recognizes that the incoming message is being sent via READLINK caching, a temporary message cache buffer is picked to hold the data part of the message. (The buffer is needed because the receiver is not yet awakened. It acts as a temporary residence for the data until the receiver specifies a memory location to which the data can be moved. The size of the buffer is predetermined at SYSGEN time.)

Figure 1

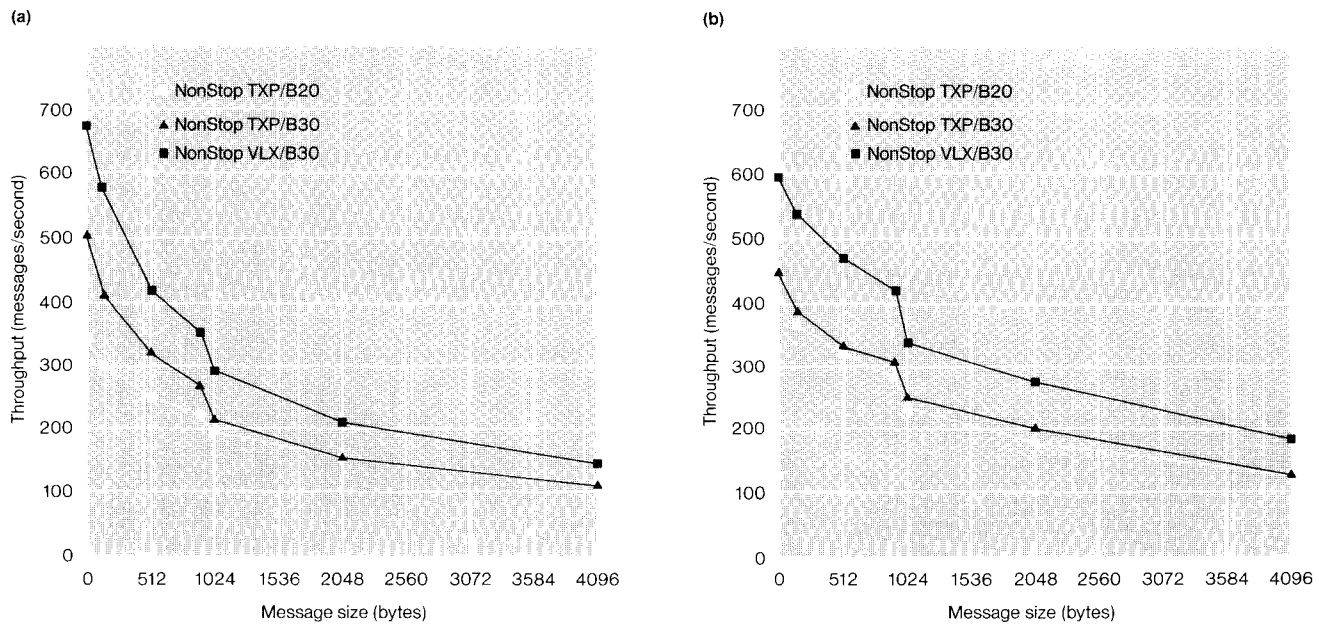


Figure 1.
Message throughput between two processes for (a) inter-CPU messages and (b) intra-CPU messages. Results are for the B20 software release (NonStop TXP processor) and the B30 release (NonStop TXP and NonStop VLX processors).

2. When both the LCB and the data have arrived, the message system in the receiver's CPU allocates an LCB for the message and wakes up the receiver process.
3. The receiver wakes up and reads the LCB part of the message, and it specifies a memory location for the data. At this point, the Message System need only copy the data from the cache buffer to memory.

In brief, several benefits are gained from using the READLINK caching protocol. Less bus activity and fewer interrupts occur because the data part of the message is already in the receiving process's CPU; also, fewer dispatches are required as there is no need to dispatch the receiver to process the request for data in step 5 of the standard protocol. Hence, fewer processor cycles are needed to transmit a message over the bus. This means service time is reduced and the processor is able to handle more requests, improving system throughput.

There are costs for using READLINK caching, however. Memory is needed to hold the message cache data, and an extra data movement is required to transfer data from the cache buffer to the memory location specified by the receiving process (see step 3 in the READLINK caching protocol). The amount of extra memory needed depends on the size and number of message cache buffers allocated with SYSGEN, but the maximum amount of memory allocated is 64 Kbytes.

Performance Tests

Test Description

A program written in Tandem's Transaction Application Language (TAL) was used to generate and send messages from a sender to a receiver using the File System procedure calls WRITEREAD, READUPDATE, and REPLY. All messages were sent WAITed, and all replies from the receiver consisted of the same amount of data as the message sent. The data was not written to disk.

All tests were conducted on a stand-alone system. The XRAY performance analysis tool was run in the background to collect measurement data; all other activities in the system were generated by the test program. Each test consisted of sending tens of thousands of messages. The elapsed time for each test was long enough so that when the test data was analyzed, beginning and ending perturbations caused by the test program could be excluded to cover program setup time, process creation time, and other software overhead. In this way, the actual message throughput between two processes could be measured.

Figure 2

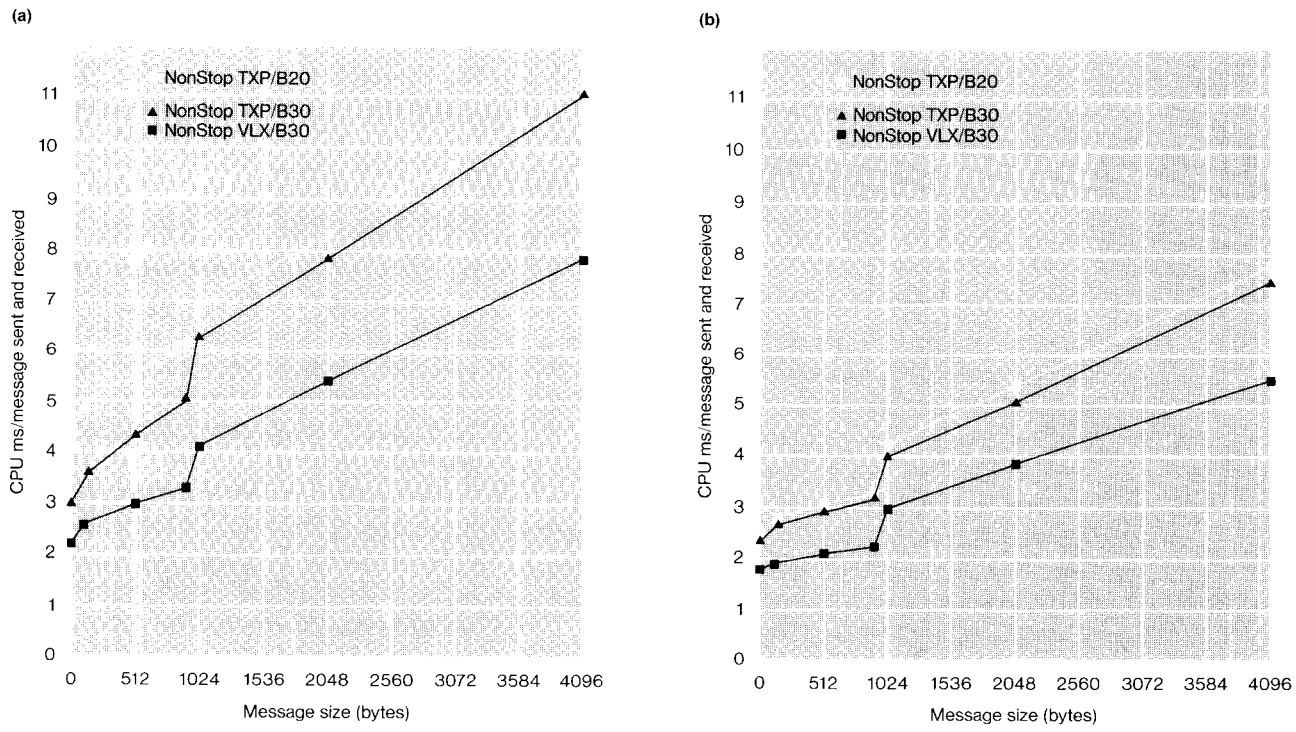


Figure 2.
CPU demand per (a) inter-CPU message and (b) intra-CPU message.

All B30 tests used the default READLINK cache size of 1536 bytes. That is, READLINK caching was used to send messages up to 1536 bytes in length, and the standard message protocol was used for larger messages. (The default size was used because, at the time the measurement tests were conducted, the SYSGEN option for the cache buffer size was not yet available.)

Definitions

In the context of this article, throughput is defined as the number of messages sent and received per second between two processes. Given the message throughput for a message of n bytes, the elapsed time to send and receive an n-byte message is equal to the reciprocal of the throughput (i.e., elapsed time = 1/throughput). CPU demand per message is defined as the CPU time in milliseconds required per message sent and received, including interrupt time.

Test Results

Figure 1 shows the throughput between processes for (a) inter-CPU and (b) intra-CPU messages. Note that these figures represent message throughput for single-threaded *process-to-process* communication; they do not represent message throughput for the processor or the system.

Figure 2a shows CPU cost per inter-CPU message sent and received, and Figure 2b shows this for intra-CPU messages. In general, the CPU cost of sending a message is about the same as that of receiving a message of the same size.

For intra-CPU messages, very few interrupts occur; inter-CPU messages, however, cause a significant number of interrupts. READLINK caching was implemented to reduce the amount of interrupt time for messages sent between processors. The test results indicate that this enhancement reduced the interrupt time for sending messages between two processors by as much as 50% for the sender and 43% for the receiver.

As Figures 1 and 2 show, significant performance gains were obtained when messages smaller than the cache size were sent. Although the tests used the default READLINK cache size, users can optimize system performance by setting this size through the SYSGEN program. (This should be done only after a detailed study of the message traffic in the application is made.)

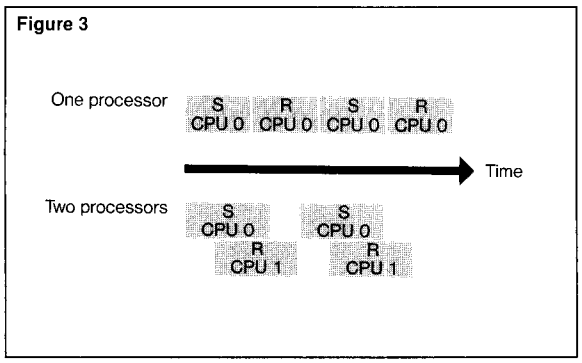


Figure 3.
Parallelism yields better throughput for inter-CPU messages up to 300 bytes long. S represents the CPU cost for the sender and R the CPU cost for the receiver. For two processors, although each unit is larger, the total elapsed time per message is less.

An unexpected result was that, in the B30 release, inter-CPU message throughput is higher than that for intra-CPU messages when small messages (up to 300 bytes) are sent. More operations can be done in parallel by two processors than can be done by one. In Figure 3, the boxes labelled S and R represent the CPU cost for sender and receiver, respectively. For two processors, although each unit is larger, parallelism increases throughput. (Note that the total CPU time it takes to complete a message is still higher for inter-CPU messages than that for intra-CPU messages.)

A small increase in CPU cost occurs for messages larger than 1024 bytes, as is evident in Figure 2. This is because, at the File System level, each process has a memory buffer of 1024 bytes allocated for use as a temporary residence for message transfers. When the message is less than 1 Kbyte, the buffer is used; otherwise, the File System allocates a buffer to hold the message. Since buffer allocation consumes processor cycles, the CPU cost for messages larger than 1 Kbyte increases. (Note that in a normal user environment, this increase is negligible compared to the cost of other system activities. The reason it was seen so clearly in these tests is that the system was doing nothing but sending and receiving messages from one process to another.)

CPU Cost Approximation

The linear formulas in Table 1 can be used to approximate CPU demand per message sent and received. For a message of size X, the resulting Y is the CPU time in milliseconds.

Increase in OLTP Throughput

As the results in Figures 1 and 2 indicate, messages under 512 bytes gained significant performance improvement with the B30 Message System. But what does this performance improvement really mean to users running on-line transaction processing (OLTP) applications?

Benchmark results have shown that 85% of the messages in an OLTP debit-credit application are small, i.e., less than 512 bytes long.[1] In Tandem systems, OLTP applications are based on requesters and servers, which communicate by messages. If the CPU cost of a message transfer is reduced, the service time for a typical transaction is less. This means that the processor can handle more requests; hence, a higher transaction throughput is possible.

To compare the performance of the B20 and B30 versions of the Message System for OLTP applications, the performance measurement team defined an *averaged CPU cost per message* to be the mean value of CPU costs for messages up to 512 bytes. This number was computed by summing up the CPU cost per message for messages of 0, 64, 128, 256, and 512 bytes and then dividing the sum by 5; i.e., the averaged CPU cost per message equalled

$$\frac{(C_0 + C_{64} + C_{128} + C_{256} + C_{512})}{5}.$$

Figure 4 shows the averaged CPU demand per message for inter-CPU and intra-CPU messages. For example, the averaged CPU demand per inter-CPU message on the NonStop TXP system is 3.73 ms for the B30 Message System, as compared to 4.88 ms for the B20 Message System. Figure 5 translates the results in Figure 4 into CPU savings.

[1]For a complete description of the debit-credit benchmark mentioned here, see the accompanying article, "NonStop VLX Performance."

Table 1.
Formulas for approximating CPU demand per message sent and received (X = message size, Y = CPU time in milliseconds).

Type of system and software release	Message size (bytes)	Formula for inter-CPU messages	Formula for intra-CPU messages
NonStop TXP system, B20 release	< 1000	Y(X) = 0.00177 X + 4.491	Y(X) = 0.00104 X + 2.491
	≥ 1000	Y(X) = 0.00171 X + 4.937	Y(X) = 0.00137 X + 2.051
NonStop TXP system, B30 release	< 1000	Y(X) = 0.00203 X + 3.277	Y(X) = 0.00104 X + 2.349
	≥ 1000	Y(X) = 0.00167 X + 4.144	Y(X) = 0.00136 X + 1.619
NonStop VLX system, B30 release	< 1000	Y(X) = 0.00126 X + 2.349	Y(X) = 0.000749 X + 1.729
	≥ 1000	Y(X) = 0.00120 X + 2.769	Y(X) = 0.00103 X + 1.290

Figure 5 shows that, on the average, the B30 release of the Message System is 1.3 times the speed of (31% faster than) the B20 release for inter-CPU message transfer. Since the Message System takes up about one-third of the processor cycles in an OLTP debit-credit environment, overall transaction throughput is improved by about 10% (31% * 0.33 = 10%).

Conclusion

The B30 Message System reduces CPU cost and speeds up the process of sending messages between processors. The READLINK caching protocol results in less bus activity and fewer interrupts during a message transfer from one processor to another. Fewer dispatches are required and, hence, fewer processor cycles are needed to transmit a message over the bus. Service time is thus reduced and the processor is able to handle more requests, improving system throughput. On an average, the performance of the Message System has improved by about 31%, which means a 10% improvement for overall system performance in a typical OLTP environment.

Acknowledgments
The author would like to thank Dave Kinkade and Rich Larson for their support throughout the measurement project. Thanks are also extended to Nhan Chu, Anil Khatri, Praful Shah, Art Sheehan, Mala Chandra, Leslie White, and Susan Whitford for reviewing the article.

Susan Uren joined Tandem in June 1984 as a member of the Performance Group in Software Development. Since then she has worked on performance evaluations of many Tandem products, such as the GUARDIAN Message System, TRANSFER, FASTSORT, and the NonStop VLX system. Currently, she is a software developer for MEASURE, Tandem's new performance measurement tool. Susan holds a B.S. in computer science from California State University at Chico.

Figure 4

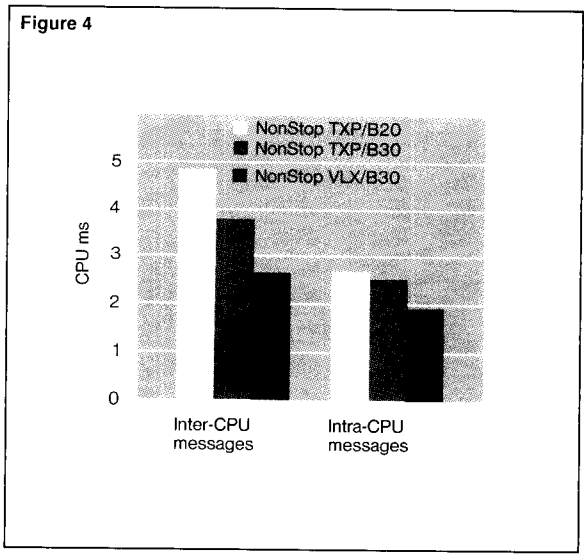


Figure 4.
CPU demand per message in an OLTP environment. The measurements represent the mean cost per message up to 512 bytes long (send and receive).

Figure 5

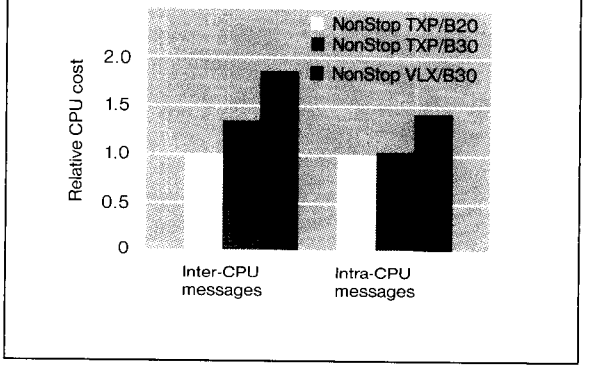


Figure 5.
Relative CPU cost in an OLTP environment. The measurements are based on the mean cost per message up to 512 bytes long (send and receive).

MEASURE, Tandem's New Performance Measurement Tool

A new performance measurement tool, MEASURE™, has been designed for the GUARDIAN 90 operating system. Its data collection and examination facilities are useful for system tuning, application optimization, and capacity planning.

The *MEASURE Reference Manual* provides detailed descriptions of the MEASCOM command language and the callable procedures. The *MEASURE User's Guide* shows a step-by-step approach to performance measurement, analysis, and system tuning. This article is intended to augment that information by providing details of MEASURE's structure and internal mechanisms. It is written for software designers who plan to use the programmatic interface.

The Need for MEASURE

MEASURE is a robust system performance measurement tool. It was designed to use minimal system resources while providing a wide range of performance metrics. It is a highly reliable, highly available subsystem.

In either a single CPU system or a multiprocessor shared-memory system, all performance counters are located in main memory. In a multicomputer system such as the Tandem NonStop system, each CPU has its own memory with counters for the local CPU. Counters located in a remote CPU's memory must be accessed with the cooperation of processes in the remote CPU. The NonStop system provides many options for system tuning and expansion that are not possible on single memory systems. Simple changes in the assignment of processes to CPUs can cause dramatic improvements in system response time. Thus a robust performance measurement and collection mechanism such as MEASURE is required.

Functions

With the cooperation of the GUARDIAN 90 operating system, MEASURE provides a variety of performance statistics for each of the several entity types it monitors. Measurable entity types include processors, disks, communication lines, network lines, terminals, other I/O devices, remote systems, FOX cluster traffic, Transaction Monitoring Facility (TMF) transactions, processes, process code ranges, logical file opens, and physical disk-file opens.

Currently, 169 different standard counters are assigned to the 13 entity types. The standard counters were chosen to provide a complete picture of an entity's performance characteristics without duplicating information that can be found by combining two or more standard counter values. In addition, three types of user-defined counters are available to measure application-specific performance parameters.

A measurement can include any combination of measurable entities. Entities are specified by entity type and entity identifying names and numbers (e.g., CPU 12, process $APPL2, and system \TANDEM). Measurements can start or stop at any time. Some entity types are transient (e.g., processes and file opens); these can be created and terminated during a measurement.

A counter record is allocated for every active entity included in a measurement. These records are written to a measurement data file at the end of a measurement or, optionally, at periodic intervals during the measurement. Counter records may be read from the data file or directly from active counter records. Up to 64 concurrent measurements are supported. When more than one measurement includes the same entity, the counter record allocated for the first measurement is shared by the second and subsequent measurements.

Components and Structure

MEASURE is composed of several processes and system library procedures:

- *MEASCOM*, the command interpreter process.
- *MEASFH*, the measurement data file-handler process.
- *MEASMON*, the subsystem monitor process.
- *MEASCTL*, the counter-record allocation process.
- *OMEASG*, a file containing low-level system-library procedures.
- *OMEASP*, a file containing the callable procedures.

Both OMEASG and OMEASP are included in the system library by the GUARDIAN 90 SYSGEN program.

Counters are "bumped" by the GUARDIAN 90 kernel, File System, disk processes, communication processes, and TMF. The MEASURE XCTR instruction is used to bump counters.

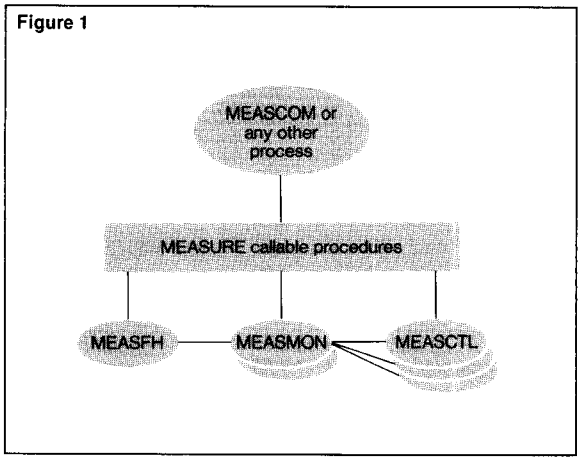
MEASCOM is the high-level command interpreter process. It provides a verb-object-attribute command language for measurement configuration, control, and status; counter-record access; and report and plot generation. Counter records can optionally be written to a structured performance data base. Multiple measurements can be controlled and accessed with a single MEASCOM process.

The callable procedures, listed in Table 1, sit between MEASCOM and the rest of the subsystem. They provide the basic subsystem functions to any calling program. All functions used by MEASCOM are available to any other process via these procedures. There are no undocumented parameters or functions. This interface is intended to serve as a foundation for future high-level performance tools.

Table 1.
MEASURE callable procedures.

Procedure	Function
MEASOPEN	Creates MEASFH process to initialize or read a data file.
MEASCLOSE	Terminates access to data file.
MEASCONFIGURE	Defines entities to be measured.
MEASCONTROL	Controls (starts/stops) a measurement.
MEASSTATUS	Returns resource usage for an active measurement.
MEASREADACTIVE	Returns a copy of an active counter record.
MEASREAD	Returns counter records from a data file.
MEASREADCONF	Returns measurement configuration and resource usage from a data file.
MEASCOUNTERBUMPINIT	Returns offset of a user-defined counter.
MEASCOUNTERBUMP	Bumps a user-defined counter.
MEASMONCONTROL	Starts or stops the MEASURE subsystem.
MEASMONSTATUS	Returns current measurement data file names.

Figure 1.
Communication paths between the MEASURE processes and callable procedures.



MEASFH is the measurement data file handler. It validates the format of a new measurement configuration, builds an index with an entry for each record in the measurement data file, builds external-format counter records from internal-format entity identifier and counter records, and provides measurement configuration information. Each MEASFH process handles one measurement data file.

A MEASCTL process runs in each processor. It allocates counter records at measurement start or transient-entity (processes and file opens) start and deallocates counter records at measurement or transient-entity stop. It returns snapshots of active counter records directly to the callable interface and writes counter records to the measurement data files.

MEASMON runs as a process pair that controls the MEASCTL processes. On subsystem start-up it creates a MEASCTL process in each local processor. It also creates a new MEASCTL process on processor reload. MEASMON keeps a copy of each measurement configuration and sends it to the MEASCTL processes on measurement start or processor reload. It disseminates control messages and gathers status information to and from all MEASCTL processes. MEASMON sits between the callable interface and the MEASCTL processes.

The communication paths between the MEASURE processes and callable procedures are illustrated in Figure 1.

Measurement Configuration

A measurement configuration consists of a set of entity descriptors. Entity descriptor fields describe an individual entity or an entity set. CPU descriptors include a CPU number; disk descriptors include a CPU number, controller number, unit number, and volume name; and process descriptors include a CPU number, process identifier number (PIN), process name, and program file name. Other entity descriptors include fields appropriate to the entity type.

Any descriptor field may have a wild card value that matches any name or number. Sets of entities can be specified with wild card values. Each descriptor has a type field, a length field, and a flag bit, which, when set, means any entities matching the descriptor should not be measured.

Descriptors are grouped into a configuration table (CONTAB) that includes a header record containing the offset to the first descriptor of each entity type in the CONTAB. The CONTAB is sent from MEASCOM via the MEASCONFIGURE procedure to the MEASFH process for validation. Any process code-range descriptors require MEASFH to generate extra records for the CONTAB and data file. MEASFH sends the validated CONTAB along with any extra code-range records to MEASMON. MEASMON stores the CONTAB in its own data space and checkpoints a copy to the backup MEASMON process. Later, at measurement start time, MEASMON sends a copy of the CONTAB to each MEASCTL process.

Measurement Start

A measurement start request contains a start time, an optional stop time, and an optional time interval. It is sent from MEASCOM via the MEASCONTROL procedure to MEASMON. MEASMON keeps an entry in its MEASTABLE for each configured measurement. These entries are indexed by measurement number and linked together on a list, ordered by start time for unstarted measurements and by stop time for running measurements. When a measurement's start time is reached, MEASMON sends a copy of the measurement's CONTAB in a start measurement message to each MEASCTL process.

Upon receipt of the measurement start message, the MEASCTL allocates a MEASTABLE entry and space for the CONTAB. MEASCTL's MEASTABLE, like MEASMON's MEASTABLE, is indexed by measurement number, but its entries are linked on a list ordered by the next copy time interval. This list is used to copy counter records to the data file at user-specified intervals for any measurement using the optional time interval.

Each counter record is accessed by a counter-record pointer, or XPTR, in the entity control block corresponding to the counter record. Examples include the PCBXPTR in the Process Control Block, the PDTXPTR in the Physical Device Table, and the NRTXPTR in the Network Routing Table. In addition to these pointers, MEASCTL allocates a counter-record identifer (CID) for each counter record. The CID entry contains a pointer to the counter record and a pointer to the XPTR. An active counter record contains a pointer to its CID entry. These pointers are illustrated in Figure 2.

Counter records are allocated in counter space, absolute segment zero, and CID entries are allocated in MEASCTL's extended segment. Entity control blocks are allocated in system data space, system-process data stacks, process file segments, and various extended segments.

At measurement start, the new CONTAB is used to find the entities to be measured. If an entity type has one or more descriptors in the CONTAB, each control block for that entity type in the CPU is examined to see if it is a member of the new CONTAB. If it is a member, it is included in the new measurement.

If the entity is not already being measured by another measurement, a CID entry and a counter record are allocated. The MEASCTL process then sets mutual exclusion on, reads the current time, initializes any queue counters, sets the XPTR, and resets mutual exclusion. The entity is now under measurement.

If the entity is already being measured by another measurement, the address of the counter record is found from the XPTR and the address of the CID entry is found in the counter record.

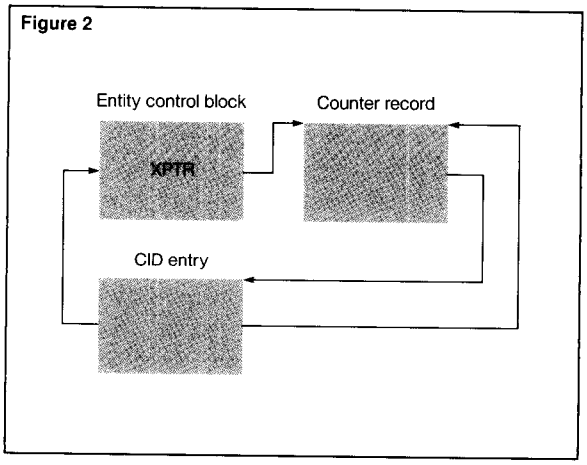


Figure 2.
Counter-record pointers, as implemented in MEASURE.

The following is done for either a new counter record or an existing counter record that is to be shared with the new measurement. The CID entry contains a four-word *sharemask* with one bit for each possible measurement number (0 through 63). The bit for the new measurement number is set in this mask. Resource usage data kept in the MEASTABLE entry is updated to show the addition of the new counter record. A new entity identifier record is created and written to the data file buffer (writes to the data file are only done when a measurement's data file buffer becomes full or the measurement is stopped). New entity identifier records contain the time that the XPTR was set and the entity identifiers (names and numbers). If the counter record was already allocated for a previous measurement, a copy of the counter record is also written to the measurement's data file buffer. Whenever a counter record is copied, several fields are added to the record. These include the allocation time, copy time, CPU number, and a field that indicates the copy type (measurement start copy, interval copy, or measurement stop copy).

Measurement Stop

A measurement stop request is triggered by a call to MEASCONTROL with a valid stop time. The stop time may be specified in the call that starts the measurement or in a subsequent call. When the stop time is reached, MEASMON sends a stop measurement request to each MEASCTL.

When a MEASCTL gets a stop request, it examines each CID entry. If the bit corresponding to the stopping measurement's number is set in the sharemask, the counter record associated with the CID entry is copied to the stopping measurement's data file buffer and the bit in the sharemask is reset. If the sharemask is all zero, the XPTR is set to zero, the counter record is deallocated, and the CID entry is put on a free list.

When all the busy CID entries have been examined, the MEASTABLE entry for the stopping measurement is deallocated.

Entity Start/Stop

The GUARDIAN 90 monitor process, the File System, the disk process, and several communication processes notify MEASCTL of process creation/deletion, file open/close, and subdevice add/delete with a simple procedure call that sends a message to the MEASCTL process in the same processor.

When MEASCTL gets an entity start message, it examines all the current measurement CONTABs to see if the new entity should be included in one or more of the currently active measurements. If the new entity is a member of an entity set specified in one of the CONTAB entity descriptors, a CID entry and counter record are allocated and initialized as described above.

When MEASCTL gets an entity stop message for an entity with a nonzero XPTR, it copies the counter record to the measurement data file buffer of each measurement indicated in the CID's sharemask. The XPTR is set to zero, the counter record is deallocated, and the CID entry is put on a free list.

Counter Maintenance

Each measurable entity type has a unique, internal counter-record format. The offset to each counter in these records is used by system procedures and processes to maintain counter values. The XCTR instruction is used to bump MEASURE counters. Its input includes the 16- or 32-bit address of a counter-record pointer, the offset of the counter within the counter record, the type of counter bump to be performed, and an optional value which may be added to a counter. XCTR accepts a variable number of input words. Counter-record pointers are located in control blocks for processes, file opens, I/O devices, remote systems, and other system entities. They are set by the MEASCTL processes when a counter record is allocated and reset to zero on deallocation. If the counter-record pointer is zero, no operation is performed. If the pointer is nonzero, it is used as an offset into absolute memory segment zero, the system counter segment. The counter offset is added to locate the start of the counter to be bumped.

Accumulating counters are 32 bits wide and can be incremented by an input value. Busy counters include a busy/idle flag and a 64-bit microsecond counter; they can be set busy or idle. Queue counters consist of the current queue length, the maximum queue length, and a 64-bit value that is used to yield the sum of the queue lengths for each microsecond since the counter was started. A queue counter can be incremented or decremented. Each of these counter types is bumped by the XCTR instruction when a measurable event occurs.

Several counters require special treatment. Process memory-usage counters are maintained with the aid of a resident table which tells how many pages are in use by each process. A system procedure updates this table whenever a memory page is allocated or deallocated. Transaction start time is saved in each terminal counter record and subtracted from the transaction end time to obtain the response time for a terminal transaction. The TERMPROCESSes, as well as all communication processes that support terminals, call a pair of system procedures to start and stop terminal response-time counters.

Process code-range measurement uses pseudorandom sampling to estimate the amount of CPU time spent in different procedures or code ranges of a process. When MEASFH validates the CONTAB, an address record is built for each code space to be measured. The address record contains the base address for the code range and is attached to the CONTAB sent to MEASMON. MEASFH also builds a code-range name record containing the ASCII name of each code range. The name record is saved in the data file. For an object file, procedure names and addresses are used.

When process code-range measurement is started for a process, MEASCTL allocates an internal counter record with one 32-bit counter for each address in the address record. A pseudorandom sampling mechanism is implemented with help from the microcode in the diagnostic data transceiver, or DDT (used for hardware diagnostics). MEASCTL sends a command to the DDT to start generating pseudorandom interrupts when the first code-range measurement is started and a second command to stop interrupts when the last one stops.

When the CPU receives a sampling interrupt from the DDT, the current process is examined to see if it is under process code-range measurement. If it is, and its current code space has an address record, a binary search is done to find the index of the address range for the current value of the interrupted process's program counter. The index is used to increment the correct counter in the associated counter record. Control is then returned to the interrupted process. Later, the MEASFH process uses the name record and the counter record to build external counter records showing code-range utilization.

User-defined Counters

One or more user-defined counters can be specified for a process. The counter descriptor includes a name, type (accumulating, busy, or queue), and optional array size for allocating a group of similar counters. User-defined counter descriptors are included in the CONTAB.

When MEASCTL starts measurement on a process with a set of user-defined counters, it allocates a counter record with the requested counters and builds an entity identifier record that includes the name and offset within the counter record for each user-defined counter. Before the first counter bump, the measured process calls the MEASCOUNTERBUMPINIT procedure with the counter name. If a measurement exists with the named counter, an offset to the counter is returned. This offset is supplied in all subsequent calls to the MEASCOUNTERBUMP procedure, which is used to bump the counter.

Up to 256 user-defined counters can be specified for each measured process. This facility provides the ability to measure parameters known only to the process that bumps the counters. In an on-line transaction processing application, user-defined counters provide the capability to count the various transaction types, measure transaction response times, and track queuing delays. Many other uses are possible also.

User-defined counters can count transaction types, measure transaction response times, and track queuing delays.

Active Counter Access

MEASCOM calls MEASREADACTIVE with an entity descriptor to return an active counter record. MEASREADACTIVE then uses the CPU number in the entity descriptor to determine which MEASCTL process can access the requested record. When MEASCTL gets the request, it uses the entity descriptor data to find the entity control block; it then uses the XPTR in the entity control block to find the counter record, and the CID index in the counter record locates the CID entry.

MEASCTL builds an entity identifier record and makes a time-stamped copy of the internal-format counter record. These are returned to the MEASREADACTIVE procedure, which converts them to an external-format counter record and returns this record to the calling process.

External-format counter records contain the counter-record allocation time, copy time, entity identifier name(s) and/or number(s), and the various counter values at copy time. Busy and queue counters are converted from their five- and six-word internal formats to a single 64-bit time value plus a one-word maximum queue-length value for queue counters.

Minimizing the cost of measurement was a primary design goal.

Data File Access

When a new measurement is started, MEASCOM calls MEASOPEN, requesting write access. If access to an existing measurement data file is desired, MEASOPEN is called with read-only access specified. MEASOPEN creates a MEASFH process and sends it the data file name. (MEASFH creates a new data file if the named file doesn't already exist.)

If write access is requested or the data file is already open (presumably for an active measurement), MEASFH opens the file via the File System and, if write access is not specified (all previous data is purged when write access is specified), it copies any data to an extended memory segment. Otherwise the file is "opened" via the memory manager as a swap file for an extended memory segment. This method provides very fast access to data file records.

On a new measurement start, after the MEASOPEN call, MEASCOM calls MEASCONFIGURE, passing a CONTAB with descriptors of the entities to be measured. MEASCONFIGURE forwards the CONTAB to the MEASFH process. After verifying the format of the CONTAB and processing any process code-range files, MEASFH writes a copy of the CONTAB to the data file and sends a copy to MEASMON.

During a measurement, the MEASCTL processes write entity identifier and counter records to the data file. An entity identifier record is written when measurement on an entity is started. If measurement is started on a counter record with any nonzero counters, a copy of the initialized counter record is written to the data file along with the entity identifier record. If the measurement was started with a copy interval, all counter records in the measurement are copied to the data file at each copy interval. When a measurement stops, all counter records are copied to the data file and each MEASCTL writes a record containing counter space-usage data for each entity type.

MEASCOM calls MEASREAD to get counter records from the data file. MEASREAD sends the request to the MEASFH process created to access the data file. A read request includes an entity descriptor and an optional target time or time window to select the desired counter record(s).

MEASFH provides access to data file records at any time during or after the measurement. Each external-format counter record contains information from the entity identifier record and one or two internal-format counter records. To provide fast access to the needed records, MEASFH builds an index with one entry for each record in the data file. Each index entry contains the entity type, counter-record allocation time, CPU number, record type (entity identifier, initial value copy, interval copy, or measurement stop copy), copy time, and record address. The index entries are sorted on entity type through copy time. These indices exhibit the following useful properties:

- They are sorted by entity type.
- All indices for a given entity (same allocation time) are located together.
- The index entries for a given entity are ordered in the following way: entity identifier record, initial value record, copy interval 1, copy interval 2, ..., copy interval *n*, measurement stop copy.

MEASFH builds the initial index when MEASOPEN is called. Thereafter, when a read request is received, MEASFH checks the data file EOF against the EOF when the index was last updated. If the two EOFs differ, MEASFH updates the index.

When a read request is received, MEASFH uses its data file index to find each entity identifier record that matches, or is a member of, the specified entity descriptor. For each matching entity identifier record, a counter record is selected by using the optional target time, time window, or the default time. Specifying a target time causes MEASFH to select the counter record copied closest to that time. When a time window is used, MEASFH selects the counter record closest to the middle of the window, but if no records are found within the window, none are returned. When the default time is used, MEASFH selects the measurement stop copy, if it is present, or the last interval copy.

The entity identifier record, initial value record (if present), and the target counter record are used to build the external-format counter record returned to the MEASREAD caller. The number of external-format counter records built depends on the measurement configuration and the entity descriptor supplied in the MEASREAD call. MEASFH uses two extended segments to build these records: one for the data file index and one for external counter records. The number of counter records returned to the caller by MEASREAD depends on the number built and the size of the caller's buffer. If the caller's buffer is too small to hold all the records, subsequent calls will fetch the remaining external counter records.

MEASURE's Performance

Minimizing the cost of measurement was a primary design goal for MEASURE. During the design phase, two guidelines were used to determine how and where to implement performance-critical functions: (1) when given a choice between using memory or CPU cycles, the designers used memory, and (2) when a function could be done at measurement start-up, shutdown, or during a measurement, they selected start-up or shutdown time.

The amount of memory currently in use for counter space can be monitored during a measurement with the MEASCOM STATUS command. The maximum amount of counter space used for a complete measurement can be obtained from the MEASCOM INFO command. These values can also be obtained by calling the MEASSTATUS and MEASREADCONF procedures.

MEASURE object files require 515 Kbytes of disk space. The disk (or tape) space used for measurement data files depends on the number of entities measured and the size of the optional copy interval. A small copy interval requires more measurement data file space than a longer copy interval or no copy interval.

The developers measured MEASURE's performance on a system having adequate memory to determine memory usage by MEASURE processes. Only the CPUs and the MEASURE processes were included in the measurement. Memory pages are 2048 bytes each. MEASCOM used 52 pages, MEASFH used 24, the primary MEASMON used 16, the backup MEASMON used 11, and each MEASCTL used 27.

Measurements have shown that MEASURE processes required from 0.02% to 1.14% of the system's CPU time, depending on the number of entities measured and number of concurrent measurements. More detail on MEASURE's performance can be found in the *MEASURE User's Guide*.

References

MEASURE Reference Manual. Part no. 82441 A00. Tandem Computers Incorporated.

MEASURE User's Guide. Part no. 82440 A00. Tandem Computers Incorporated.

Acknowledgments

The author wishes to thank Raghav Sharma for implementing MEASCTL and MEASMON, Dennis Markt for implementing MEASCOM, Joyce Lamkin for implementing the MEASURE quality assurance tests, and Heidi Kuehn for writing the *MEASURE Reference Manual* and *MEASURE User's Guide*.

Denny Dennison joined Tandem in August 1979, taking responsibility for the XRAY performance analysis tool. He rewrote XRAY for the NonStop II system and, in the last two years, has written the MEASURE product requirements and external and internal specifications and has implemented MEASFH and the MEASURE callable procedures. Before joining Tandem, Denny was a chemical quality control inspector, math teacher, and software quality assurance programmer. He has a B.S. in Math and an M.S. in Math Education.

FASTSORT: An External Sort Using Parallel Processing

The FASTSORT™ program is an external sort which uses parallel processing, large main memories, and parallel disk accesses to obtain high performance. When sorting a file of one million 100-byte records, FASTSORT is competitive with the industry leader in single-processor sorting and can outperform it by using the Tandem architecture for parallel sorting. FASTSORT is four to eight times faster than Tandem's standard SORT product. With larger files, the FASTSORT program's advantages are even more dramatic.

The speed of the FASTSORT program is proportional to the size of the input file, N, rather than the traditional Nlog(N) speed of conventional sorting products. This linearity is achieved by distributing the processor and disk load among several processors if the load exceeds the capacity of a single processor or disk. FASTSORT can sort records as quickly as it can read them. Once the file has been read, FASTSORT can produce the output as quickly as it can write the output file.

Loosely coupled multiprocessors can give linear growth in transaction throughput for on-line transaction processing. By doubling the number of processors, disks, and communications lines, Tandem systems can process twice as many transactions per second (Chmiel and Houy, 1986).

This linear growth of throughput as resources are added does not usually apply to batch transaction processing. A batch COBOL program will not run much more quickly as processors are added because the program executes on a single processor. Tandem is exploring ways to apply parallelism to processing large volumes of data. The FASTSORT program is an example of this research.

FASTSORT breaks a large sort job into several smaller ones that are done in parallel by subsort processes. These subsorts can use multiple processors, multiple channels, and multiple disks. The result is a high-performance sorting system.

This article presents the history and design of the FASTSORT program. It also explains how to estimate FASTSORT execution times on NonStop II, NonStop TXP, and NonStop VLX processors.

Evolution of FASTSORT

Many programs and products use the SORT utility on Tandem systems. User programs and batch-oriented job control files invoke it explicitly. The File Utility Program (FUP) invokes SORT to create key-sequenced files and indexes for structured files. The ENFORM query processor uses SORT to evaluate queries.

SORT

The SORT utility is a mature and functional product. It sorts records based on multiple-typed key fields, allows a user-defined collating sequence, eliminates duplicates, projects out fields, merges sorted files, and produces statistics on the sort run. It accepts input from devices, processes, and all file types. Similarly, the sorted output can be sent to any destination, although SORT does not directly produce key-sequenced or edit files.

The SORT utility's only serious flaw is performance. Originally written for the Tandem 16-bit NonStop 1+ ™ system, SORT does not take advantage of the 32-bit addressing or the parallel architecture of the newer Tandem NonStop systems. SORT runs as a single process that defaults to 34 Kbytes of memory and uses a maximum of 128 Kbytes of main memory. Consequently, its performance is not impressive for large batch applications, or for loading or reorganizing large files.

Because of the SORT utility's limitations, other sort/merge products were developed for NonStop systems. Roland Ashbaugh created SUPERSORT, and Eric Rosenberg developed and marketed QSORT. Both of these sort programs use multiple processors executing in parallel to sort large files using a technique known as parallel sorting. In addition, QSORT uses the large main memory provided by the NonStop architecture.

FASTSORT

The FASTSORT program is a compatible evolution of SORT. They share a common manual, and any program using SORT will work with FASTSORT. External improvements in FASTSORT include:

- The ability to build key-sequenced files in addition to other file types.
- Automatic selection of an efficient configuration which the user can override.
- Generation of better diagnostic messages in error cases.

Speed is the main advantage of FASTSORT. Internally, FASTSORT uses improved algorithms, extended memory, double buffering, streamlined comparison logic, streamlined structured file access, bulk I/O, and multiple processors and disks.

As a result, when sorting a file of one million 100-byte records, FASTSORT on a single processor is four times faster than SORT (eight times faster if multiple processors are used).

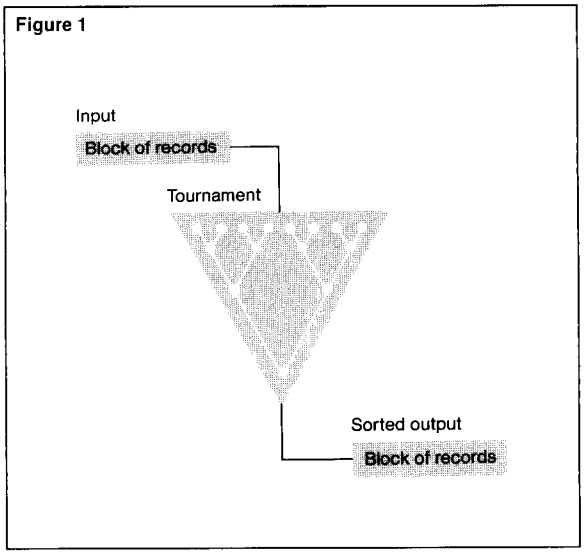


Figure 1.
The structure of a tournament. The input arrives as double-buffered blocks of records. The sorted output is produced in double-buffered blocks of records. Records move from the leaves (top) of the tournament tree to the root (bottom). The "winner" record is at the root.

How FASTSORT Works

From one or more input files, the FASTSORT program produces an output file containing the input records ordered by up to 32 key fields.

Sorting is done in passes. During the first pass FASTSORT reads the input records and adds them to a binary tournament tree arranged much like the winners in a tennis tournament. The maximum record is at the root of the tree, and the winner of each subtree is the root of that subtree. (See Figure 1.)

Initially the tournament tree is full of "maximum" null records. FASTSORT adds input records to the leaves of the tree, gradually displacing null records, which are removed at the root of the tree. The tree minimizes the number of times FASTSORT compares a record with other records. A tree of height 14 can hold 16,385 records (2^{14+1}), and the FASTSORT program compares each record with only 14 others in sorting the whole tree. If the records are 100 bytes each, such a tree occupies about 16K*100, or approximately 1.7 Mbytes.

Even with all this attention to minimizing compares, about 75% of FASTSORT's time is devoted to comparing records. This is because the compare work for a file of N records rises as Nlog(N). For N beyond 10,000, the Nlog(N) work dominates the costs of reading the input and writing the output. For more detailed information, refer to Knuth's discussion of replacement selection (Knuth, 1973).

One-pass Sorts

If the input file is less than the size of main memory, the sort can be done in one pass. As records are read in, they are added to the leaves of the tournament tree. By the time the last record is read, the records are completely sorted in the tree and ready to be written to the output file.

Currently, Tandem NonStop systems can attach 16 Mbytes of main memory per processor. Such a processor can sort large files entirely in memory. On a full system, parallel FASTSORT can apply 16 processors to the problem and sort even larger files in main memory. In this parallel one-pass approach, a distributor-collector process starts a subsort process in each CPU. The subsorts allocate sufficient memory to hold their part of the job. The distributor then reads the input stream (tape, process, or disk file) and distributes the records in round-robin fashion to the subsorts. When the distributor-collector comes to the end of the input file, it sends an end-of-file to the subsorts. The distributor-collector process now becomes a collector. It reads the output runs from the subsorts, merges (sorts) these runs into a single run, and writes the resulting run to the output stream.

The FASTSORT program reads input records and writes output records in large blocks to minimize message overhead and disk usage. Block sizes can be up to 30 Kbytes, but 16-Kbyte blocks provide most of the benefits. FASTSORT also uses double buffering; it always gets the next input block while sorting the current block. During output, it always writes the previous block to the output stream while filling the current block in memory. During the first pass, reading the input file and writing the output file is purely sequential access to the disk (almost no seeks), so parallel FASTSORT is limited by the speed at which disks can be sequentially read or written.

Multipass Sorts

One-pass main-memory sorts are the fastest, but not always the cheapest, way to sort. For larger files, or for a less memory-intensive approach, a two-pass or a multipass algorithm is appropriate.

If the file is bigger than the tournament, non-null winners are selected from the root and written out to a scratch file as new records arrive. The tournament is then recomputed to calculate the new root. The result is a "hole" in a leaf of the tournament. A new input record replaces this hole and the cycle repeats (thus the name, "replacement-selection" sort). This process produces a run of output records.

If an input record bigger than the previous winner arrives, it "breaks" the run—the new record cannot be added to the end of the current run and still keep the run sorted. In this case, the sorted tournament is written to the scratch file and a new tournament is begun with the new record. The actual implementation is a little fancier than this (Knuth).

If the file arrives sorted or almost sorted, then only one run is generated. This is the best case. If the file arrives sorted in reverse order, then each run is the size of the tournament. This is the worst case. On average, if the file is random, the average run is twice the size of the tournament (Knuth).

Merging

If only one run is produced, it is copied to the output file and the FASTSORT program is done. If multiple runs are produced, then FASTSORT merges them to form a single run. These latter passes over the data are collectively called merging. If multiple merge passes are necessary, the intermediate results are kept in the scratch file. (See Figure 2.)

During merging, a tournament is used to combine multiple runs into a single larger run. For example, suppose that the first pass generates 14 runs, and that the tournament size is ten. Only ten runs can be merged at the same time. During first merge, five runs will be merged, which reduces the number of runs to ten—one run produced from merging, and nine runs from pass one. During a third pass, these ten runs are merged into the output file.

Each pass over the data costs disk and CPU time. Therefore, it is very desirable to have at most one merge pass—an initial pass to produce runs, and a second pass to merge them into the sorted output. A sufficiently large tournament gives a two-pass sort. Surprisingly, not much memory is required for a two-pass sort. The memory requirement rises as

$$\sqrt{\frac{\text{File Size} * \text{Block Size}}{2}}.$$

Table 1 assumes a 16-Kbyte blocking factor to show the approximate memory requirements for a two-pass sort.

Memory Requirements

Table 1 shows that the file can get 10,000 times bigger and only need 100 times as much memory. Note that 10 Mbytes of main memory costs $50,000 while the disk file cost is $1 million unmirrored and $2 million mirrored (a total of 30 Gbytes for input, scratch, and output files). Memory cost is only 5% of the disk cost.

If a user selects the AUTOMATIC option, the FASTSORT program tries to allocate enough memory to give a one-pass sort if the file is small (less than 100 Kbytes) or a two-pass sort if enough memory is available. In general, it uses the equation shown in Table 1 to estimate the memory size and then adds 30% as a safety factor. The AUTOMATIC option limits itself to 50% of available main memory, while the MINTIME option uses at most 70% of the processor's available main memory.

Table 1.
Tournament size needed for a two-pass sort assuming a 16-Kbyte block size.

File size (bytes)	Tournament
1M	0.1M
10M	0.3M
100M	1M
1G	3M
10G	10M

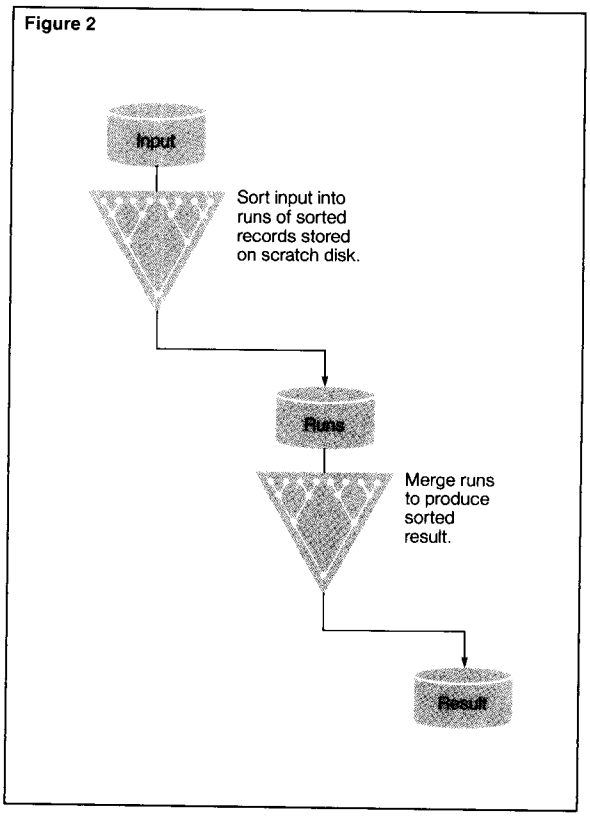


Figure 2.
A two-pass sort. The first pass over the data produces runs of sorted records stored on a scratch disk. A new run starts whenever an input record is bigger than the tournament winner. During the second pass, the runs are merged together to form a single run (the sorted output). If there are many runs, then more than one merge pass over the data may be required.

Alternatively, the user can specify how much memory FASTSORT should use instead of letting FASTSORT estimate memory requirements.

Block Size

The FASTSORT program also tries to optimize its I/O by using a large block size (16 Kbytes is the default) in reading and writing files. It also double-buffers reads and writes so that sorting overlaps I/O requests. These features enable FASTSORT to reduce I/Os by a factor of four and to eliminate half of the data moves.

By combining all these improvements, serial FASTSORT runs about four times faster than standard SORT in sorting a file of one million 100-byte records—reducing the time from 115 minutes to 29 minutes on a NonStop VLX processor. This compares to 21 minutes for the industry leader, SYNCSORT, on an IBM 4381-II.

FASTSORT can beat SYNCSORT by using parallel processing.

Figure 3

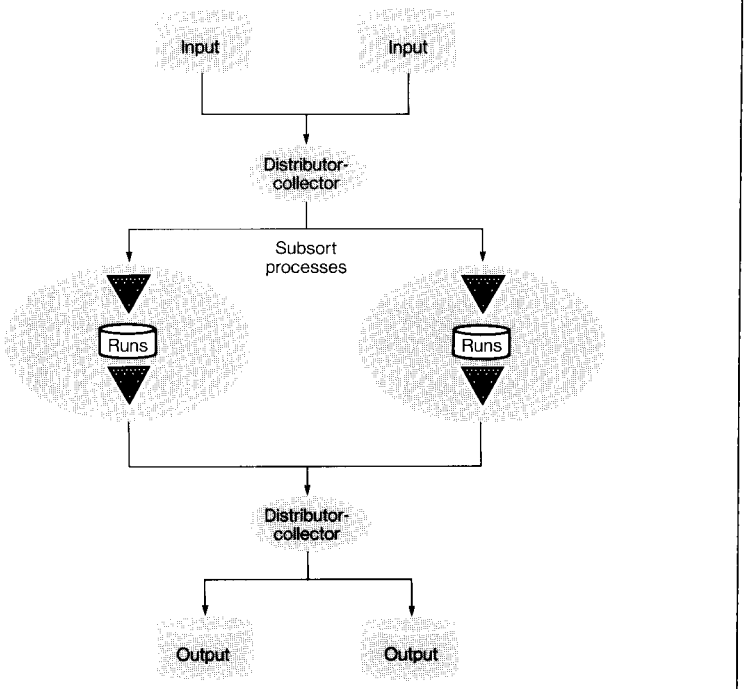


Figure 3.
The structure of a parallel sort. The distributor-collector process drives two or more subsorts, which in turn write their runs to their scratch files. When input is complete, the subsorts merge their runs and return their sorted data to the distributor-collector, which merges them into the output file.

Parallel Sort

The speed of a single processor sort is limited by the speed of one CPU and the speed of one scratch disk. Figure 3 shows how parallel sorting uses multiple subsort processes to sort parts of the input file and a distributor-collector process to allocate work to the subsort processes and merge their output runs into the final result. Parallel FASTSORT operates as follows:

1. The distributor-collector accepts user parameters and starts subsort processes. Every subsort has its own scratch file.
2. The distributor-collector reads the input file(s) and distributes records among subsort processes on a round-robin basis. Each subsort sorts its part and produces a sorted stream of records.
3. The distributor-collector merges output from the subsorts and writes the output file.

Each sort process should run in a different CPU to minimize CPU and memory contention. To minimize disk contention, each subsort should have a different scratch disk. Also, the distributor-collector should run in a lightly loaded CPU because it can be CPU-bound. Finally, each subsort should run in the CPU containing its primary scratch disk so that DYNABUS™ traffic does not increase with file size.

The FASTSORT program automatically configures parallel sorts to satisfy these configuration rules. The user can configure a three-subsort parallel sort by simply naming the scratch disks. (In the following example, the scratch disks are called $DATA1, $DATA2, and $DATA3.)

```
FROM      infile
TO        outfile
SUBSORT   $data1
SUBSORT   $data2
SUBSORT   $data3
RUN   ,   AUTOMATIC
```

Of course, the user can override the FASTSORT program's decisions by specifying the CPU, priority, memory size, block size, and other attributes of the subsort processes. The user can also prohibit use of certain CPUs or restrict use to certain CPUs.

When properly configured, parallel sorting is faster than serial sorting for the following reasons:

- The first pass is CPU-bound for tournaments containing more than 10,000 records. Parallel sorting spreads this load among multiple CPUs so that the first pass remains bound by the speed at which the distributor-collector can read the input file.
- The second (merge) pass is disk-seek–bound while merging the runs. By applying multiple disks to the merge pass, merging can run at the speed of the distributor-collector writing the output file.

Consequently, parallel sort runs as fast as the distributor-collector can read and write the input and output files. The fact that SORT does Nlog(N) work to sort a file of size N is completely hidden in the subsorts. Parallel FASTSORT is a linear time sorting algorithm, running at the rate of about 30 Kbytes per second on a NonStop II processor, 80 Kbytes per second on a NonStop TXP processor, and 110 Kbytes per second on a NonStop VLX processor.

Table 2.
The NonStop TXP system's elapsed time to sort various file sizes using single-processor sort or multiprocessor sort.

File size (bytes)	Sort time (seconds) Serial time	Parallel time	Parallel CPUs	Speedup[1]
0.1M	5E0	8E0	2	0.6
1M	2.7E1	2.8E1	2	1
10M	2.1E2	1.3E2	2	1.6
100M	2.5E3	1.3E3	3	2
1G	3.7E4	1.2E4	4	3

[1]The speedup of parallel sorting over serial sorting.

As a general rule, files less than 1 Mbyte should be sorted in one pass in memory. For such files, the parallel sort setup time dominates any savings in speed, so parallel sort is slower than serial sort for files under 0.5 Mbyte. But for files larger than 1 Mbyte, parallel sort begins to pay off. Table 2 shows the speedups from parallel sorting on a NonStop TXP processor.

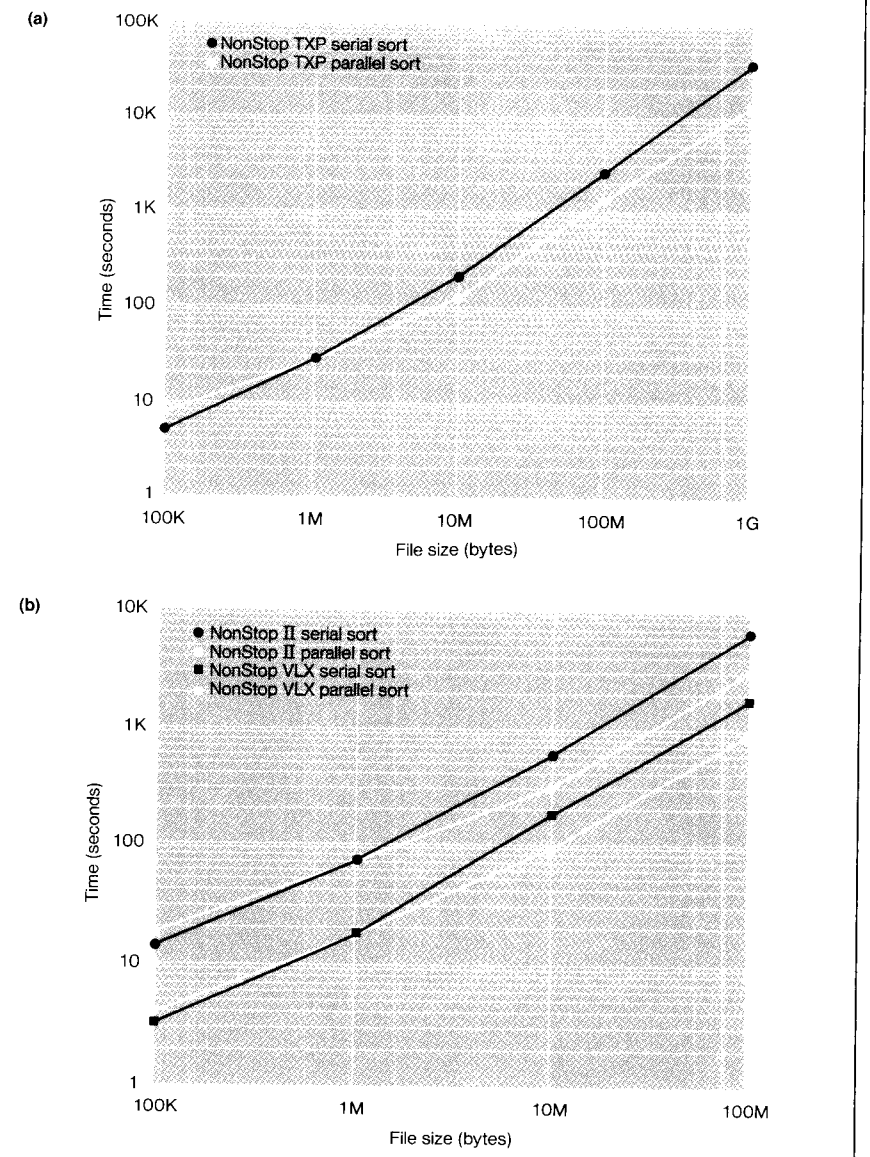
At our design point of a million 100-byte records, parallel FASTSORT gives a speedup of about two over a single-processor sort, and it outperforms SYNCSORT on 4381-II by a factor of 1.4 on the NonStop VLX processor.

FASTSORT Performance Measurements

A specific benchmark is needed to discuss sort performance. Tandem uses the sort benchmark described in *Datamation* (Anon, et al., 1985) as its standard. It is based on an entry-sequenced file of one million records. Each record is 100 bytes long and has a ten-character key. The input records are in random order.

In the following tests all disks are mirrored. The file is scaled to be one thousand, ten thousand, one hundred thousand, one million, and ten million records. For each of these file sizes, the elapsed time for the sort is measured on the best configuration for a specific processor type. For these measurements, our "best configuration" includes GUARDIAN 90, Disc Process 2 (DP2), 3108 disk controllers, and 4130 disks (XL8). Partitioned files are used in the ten million record case: six mirrored input disks, six mirrored scratch disks, and six mirrored output disks. Figure 4 gives the resulting log-log plots of elapsed times.

Figure 4



Parallel Sorting and Throughput

Table 2 shows that parallel sorting improves throughput over serial sorting by a factor of two for one million records, and a factor of three for ten million records. As the file sizes get even larger, parallel sorting becomes even more attractive. On the other hand, at the low end, below 1 Mbyte, parallel sorting is more expensive than serial sorting because the process setup time dominates. The break-even point is about 10,000 records, or about 1 Mbyte. Files smaller than this are best sorted entirely in main memory using a single processor.

Figure 4.
Plot of file size (bytes) vs. FASTSORT elapsed time (seconds) for various CPU types. (a) Parallel and serial sort for the TXP. (b) Parallel and serial sort for the NonStop II and VLX. Note that CPU speed is significant and that parallel sort is faster than serial sort for files larger than 1 Mbyte.

Figure 5.
Speed in records per second of various CPUs and methods in sorting one million records.

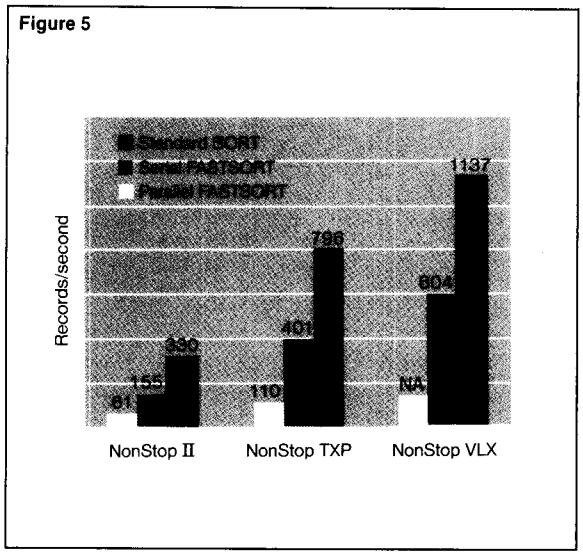
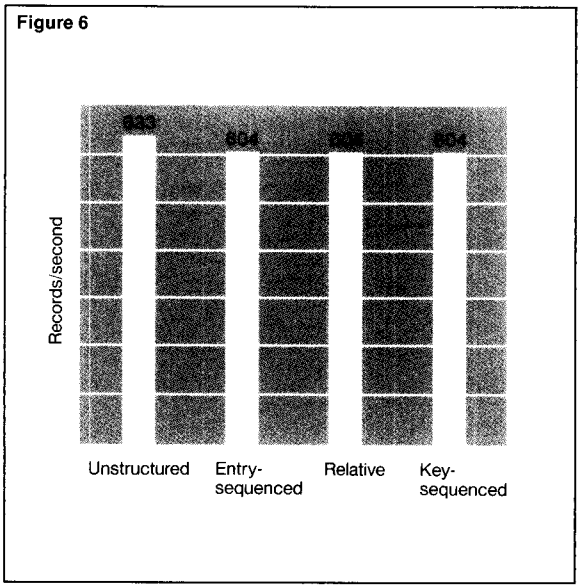


Figure 6.
Speed of FASTSORT of a million records of various file types on a NonStop VLX processor.



The time to sort "N" 100-byte records on the various processors can be roughly computed by dividing by the rate shown in Figure 5. For example, serial FASTSORT on the NonStop VLX processor would take about N/604 = 166 seconds, or approximately three minutes to sort if N = 100,000 records. Such calculations are very rough, but are helpful in giving estimates. Actual sorting speed may vary depending on the input stream, the configuration, and the concurrent workload.

Speed vs. File Type

The speed of FASTSORT varies very little with file type if records are packed densely in the file. (See Figure 6.) The 5% difference between structured and unstructured files is caused by the differences between the routines used to deblock the records. Figure 6 gives the FASTSORT rate versus file type assuming the files are dense. If the files are sparse, e.g., an almost empty relative file or a B-tree with slack, the FASTSORT program will appear to run more slowly because it is reading some useless data. For example, if the input and output files are B-trees with the typical 30% slack, FASTSORT runs about 10% slower on a per-record basis.

Speed also varies with disk process type and block size. The old disk process (DP1) supports at most a 4-Kbyte file transfer size. The new disk process (DP2) supports up to a 30-Kbyte transfer size. Large transfer sizes reduce the number of disk and message transfers. Figure 7 shows the effect of block size on sorting speed, i.e., using 16-Kbyte blocks instead of 4-Kbyte blocks improves throughput by about 30%. Using larger blocks provides very little additional savings.

Conclusion

The FASTSORT program taught one lesson quickly—when in doubt use brute force. Sophisticated algorithms gave small gains. But brute force techniques (faster processors, multiple processors, large memories, large block sizes, and double buffering) yielded high payoffs.

These techniques make FASTSORT competitive with the fastest sort products. A novel feature of the FASTSORT program is that, by using parallel processors and disks, it can sort in time proportional to the size of the input file, rather than the traditional NLog(N) time of conventional sorts. On a NonStop VLX, for example, FASTSORT can sort at the rate of 110 Kbytes per second, independent of file size.

References

Anon, et al. 1985. A Measure of Transaction Processing Power. *Datamation*. Vol. 31, No. 7.

Chmiel, T., and Houy, T. 1986. Credit-authorization Benchmark for High Performance and Linear Growth. *Tandem Systems Review*. Vol. 2, No. 1.

Knuth, D. 1973. *The Art of Computer Programming, Vol. 3*. Addison Wesley.

Rosenberg, Eric. 1984. *QSORT Reference Manual*. Dedalus Systems.

SORT and FASTSORT Manual. Part no. 82442 A00. Tandem Computers Incorporated.

Acknowledgments

John Shipman wrote the original Tandem SORT software. The product was enhanced and refined by Bob Wells. The idea for parallel sorting is not new, but was suggested by the work of Ed Ashbaugh on SUPERSORT. During the implementation, the authors benefited both from the sequential I/O improvements of DP2, and from the utilities which use these improvements provided by Joan Pearson, Janardhana Jawahar, and Charles Levine. Richard Carr gave valuable advice, showed us how to streamline the compare logic, and helped us interface to the operating system memory manager. David Hatala and Larry Watson provided the performance numbers for SYNCSORT. Kevin Coughlin, Peter Oleinick, and Art Sheehan contributed valuable review comments on this document.

Jim Gray is in the Software Development department and is currently working on enhancements to the ENCOMPASS data management system. He has worked on the design and implementation of a word processor, system dictionary, parallel sort, and a distributed SQL.

Michael Stewart joined Tandem in August of 1984. Since then he has worked as a software designer in data base utilities, specifically on FUP, DSAP/DCOM, and FASTSORT. Before coming to Tandem, he developed a set of interactive file system utilities. He has a B.S. in Mathematical Sciences from Stanford and an M.S. in Computer Science from Cal Poly SLO.

Alex Tsukerman has an M.S. in Mathematics from Kharkov University. He is currently a member of the Low-Level Database Group, involved with enhancements to ENCOMPASS, and has worked on both FASTSORT and the Spooler. Prior to joining Tandem, Alex worked in applications areas (optimization problems, resource consumption, marketing research) and as an IMS system programmer on Fast Path-related utilities.

Susan Uren joined Tandem in 1984. She contributed to this article and wrote the accompanying article, "Messenger System Performance Tests."

Bonnie Vaughan is a technical writer in the Software Publications department of Software Development. She writes manuals for languages and tools, including the *SORT and FASTSORT Manual*. Before joining Tandem, she wrote and edited a variety of software manuals. Her writing career began in journalism and public relations. She has published many newspaper and magazine articles and holds a B.A. in Journalism from San Jose State University.

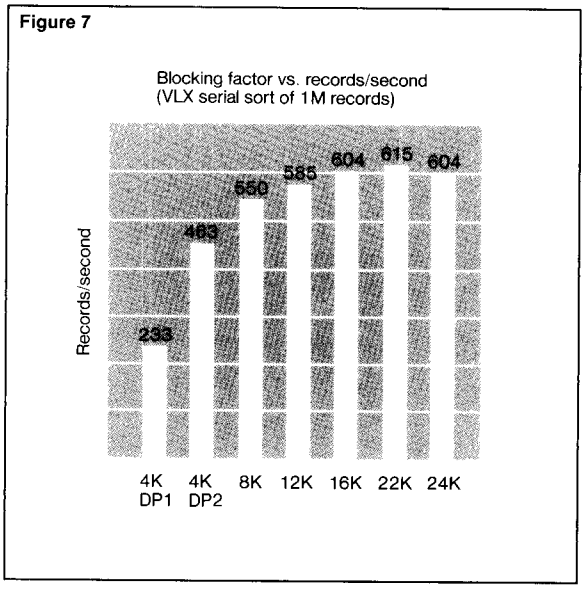


Figure 7.
Effect of block size on speed of single-processor FASTSORT of a million records on a NonStop VLX processor.

The 6600 and TCC6820 Communications Controllers: A Performance Comparison

Tandem's 6600 Cluster Controller and TCC6820 Terminal Cluster Concentrator were tested to determine which performed more efficiently. A stress test was used to saturate the line; then the CPU cost and maximum throughput were observed. The performance differences encountered are as follows:

- The throughput of the 6600 Controller is higher than that of the TCC6820 Concentrator in most cases, despite the fact that the 6600's CPU cost per message is higher than that of the TCC6820.
- The 6600 Controller performs better than the TCC6820 Concentrator in four- and eight-terminal configurations.[1] In certain instances, however, the TCC6820 outperforms the 6600 when only two terminals are configured.

In this article, the two products are briefly described, the test environment is explained, and the test results are presented.

[1]The tests described in this article were run on the A00 release of the 6600 Intelligent Cluster Controller, which allows up to eight terminals to be attached to the controller. The latest release is now A10, which allows up to 12 terminals to be attached.

Product Descriptions

6600 Controller

The 6600 Cluster Controller is an intelligent communications controller that allows the remote clustering of Tandem terminals, workstations, and printers. The remote clustering is implemented with SNATERM software, which emulates an SNA PU Type 2 cluster controller. It communicates to a Tandem host through SNAX6600, part of SNAX, Tandem's fault-tolerant interface to the Systems Network Architecture, SNA.

As of the A10 version, a single 6600 can support up to 12 Tandem terminals or workstations using 6530 point-to-point protocol at speeds up to 19,200 bps.

Application programs on a Tandem host communicate to terminals attached to the 6600 as if they were locally attached Tandem 653X terminals. One SNA LU session can be configured per terminal.

Other features include:

- Coexistence on a multipoint, leased RS-232C communications circuit with IBM SNA devices, using half-duplex, flip-flop, send-receive protocols at speeds up to 19,200 bps.
- Support of 5508, 5520, 5530, 5540, and most generic DTR printers as IBM 3287 LU Type 3 devices. The printer ports can support transmission rates up to 19,200 baud.
- Connection of DYNAMITE™ workstations in current loops up to 1500 feet long, RS-232C circuits up to 50 feet long, and RS-422 circuits up to 4000 feet long.
- Connection of 653X terminals in current loops up to 1500 feet long and RS-232C circuits up to 50 feet long.
- Power-on self-test diagnostics.

TCC6820 Concentrator

The TCC6820 aids multiuser data communications by providing a simple communications link between Tandem systems and up to eight strings of Tandem terminals and/or two serial printers. A maximum of 64 devices can be connected to a single line.

The TCC6820 is simply a broadcast box managed by the AM6520 access method. AM6520 polls one terminal at a time, and only that terminal responds with an acknowledgment of data or no data.

Additional features include:

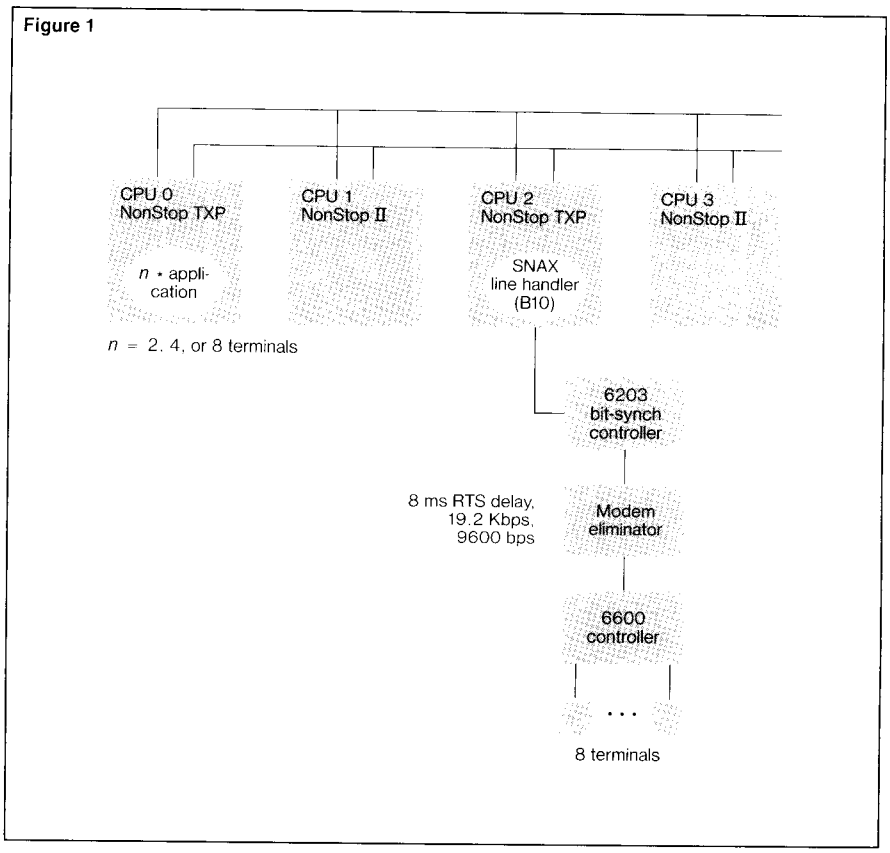
- Sixteen data transmission speeds from 50 bps to 19,200 bps.
- Connection of devices up to 1500 feet (454 meters) away.
- Cascading of additional TCC6820s to increase the number of ports available or to increase the distance to remote terminal clusters.
- Simplification of installation and testing through a built-in test mode.
- Plug-in installation.

The Differences

As the 6600 supports the controlling intelligence for devices connected to it, it is a true cluster controller. None of the devices attached to it needs to provide protocol support to the communication access method; it provides responses to the host for all connected terminals. With the TCC6820 Concentrator, however, the host (using AM6520) must poll each of the devices connected to the communication line individually. Thus, the 6600 Controller reduces polling overhead, most significantly when large numbers of terminals are connected.

Also, SNAX6600 protocol is more efficient than that of the AM6520 access method. By sending larger amounts of data between line turnarounds and acknowledgments, the 6600 Controller makes more efficient use of the communication resource. SNAX6600 also improves data integrity by using a 2-byte frame check sequence (FCS) that includes address and control information. This means all frames, not just those containing user data, are included in error checking. The AM6520 access method uses a longitudinal redundancy check (LRC), which checks only data.

Figure 1



Test Environment

A test program based on 6530 terminals in block mode was used. The program consisted of 20 iterations of a write operation followed by a read operation. Eight terminals were configured and enabled (started) in each test. (This is the maximum for the A00 release of the 6600 Controller and the current release of the TCC6820 Concentrator.)

The number of concurrent applications (active 6530 terminals) were two, four, or eight. Thus, although all eight terminals were being polled, two, four, or eight were actually sending and receiving data. The message sizes examined were 240 bytes (1/8 screen), 960 bytes (1/2 screen), and 1920 bytes (full screen).

6600 Controller Test

The B10 release of SNAX was run on a four-processor system (two NonStop TXP and two NonStop II processors). One 6204 bit-synchronous controller with a modem eliminator set for 9600 bps or 19.2 Kbps was connected to one 6600 Intelligent Cluster Controller. The line handler resided in processor 2 and the applications in processor 0 (see Figure 1).

Figure 1.
Hardware configuration for the 6600 Intelligent Cluster Controller tests. All processors were configured with 4 Mbytes of memory.

Figure 2

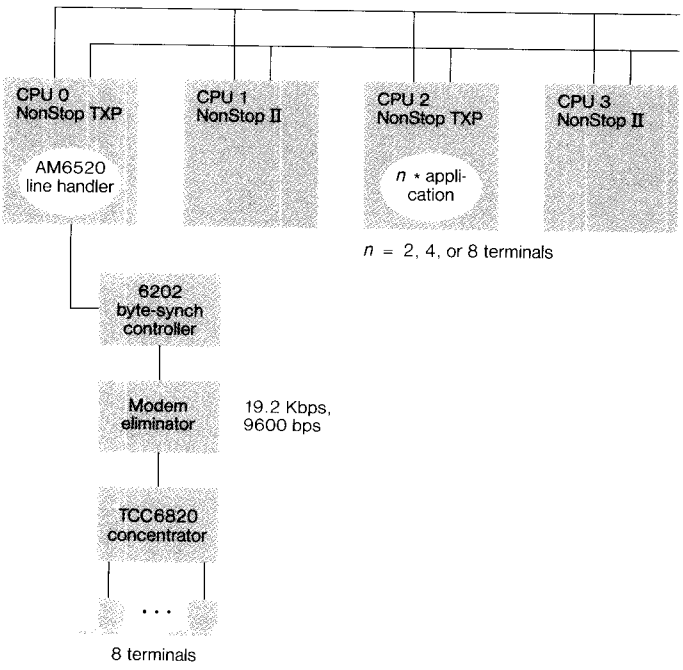


Figure 2.
Hardware configuration for the TCC6820 Terminal Cluster Concentrator tests. All processors were configured with 4 Mbytes of memory.

Figure 3

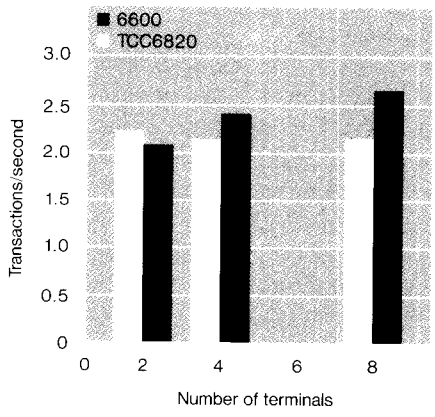


Figure 3.
Number of transactions per second processed by the 6600 and TCC6820 (1920-byte messages, 9600 bps).

TCC6820 Concentrator Test

The B10 release of the AM6520 access method was run on a four-processor system (two NonStop TXP and two NonStop II processors). One 6202 byte-synchronous controller with a modem eliminator set for 9600 bps or 19.2 Kbps was connected to one TCC6820 Terminal Cluster Concentrator. The line handler resided in processor 0 and the applications in processor 2 (see Figure 2).

Parameter Changes

SNAX. To test the performance of the SNAX line handler and 6600 Cluster Controller, several parameter changes were needed during the course of testing.

The TIMEOUT parameter for the SNAX line handler was changed from the default of 3 minutes and 20 seconds to 10 seconds. This was done to allow only a 10-second delay in SNAX line handler processing if no reply was received to the receiver ready. Because the 6600 Cluster Controller has a timeout default of one minute, the controller would have gone idle before SNAX could have sent the next poll if the default had been used.

The polling interval (POLLINT) was changed from the default of one second to 250 ms to decrease the amount of time the SNAX process waited between transmissions of the poll list when no I-frames were ready for transmission.

The RTS-CTS (request to send-clear to send) on the modem eliminator has a default of 0 ms. This was changed to 8 ms to slow down the NonStop TXP processor to allow the 6600 Controller to send its receive/response (RR) and extra pad bytes. (Without this delay, the NonStop TXP processor would have sent another RR before the 6600 had changed from transmit to receive, in half-duplex operation; then, because of this early transmission, the SNAX line handler would have waited for a response with a TIMEOUT of ten seconds.)

AM6520. The AM6520 line was configured for PACE 7. (The PACE parameter specifies the number of times one terminal can be selected to send data before it receives an acknowledgment from the host.) This was done to correspond with the SNA RR scheme. SNAX sends seven packets, then waits for an RR or acknowledgment from the controller before sending the remainder of the message.

The nonactive terminals were in conversational (ITI) mode with a COMINT command interpreter. The SLOWPOLL (time between polling inactive terminals) was set at 120 seconds. (The default SLOWPOLL is 0.)

Test Results

Throughput

In almost all cases, the 6600 Controller demonstrated higher throughput. Figure 3 shows throughput in transactions (a write followed by a read) per second; Figure 4 shows throughput in bytes per second.

Notice that the transaction rate for the TCC6820 Concentrator does not vary significantly with the number of terminals. The reason for this is that the TCC6820 individually polls each of the devices connected to the communication line, regardless of whether or not there is data. If there is data, the TCC6820 pushes it through to the terminal as quickly as possible. The only buffer is in the terminal.

With the 6600 Controller, the devices do not need to provide protocol support to the communication access method; the 6600 provides the responses to the host. This difference significantly reduces the polling overhead, which, in turn, increases throughput when a large number of terminals are connected. When fewer than four terminals are active, however, the TCC6820 could produce higher throughput. The reason for this is that the 6600 Controller uses two buffers (one in the controller and one in the terminal). This increases handling requirements.

CPU Cost

As a consequence of overhead processing for the SDLC protocol, the CPU cost per transaction for SNAX6600 is higher than that for the AM6520 access method. (Figure 5 shows CPU milliseconds per transaction, and Figure 6 shows the line handler process CPU BUSY rate.) The following is a description of the overhead processing for SNAX6600 and AM6520.

Figure 4

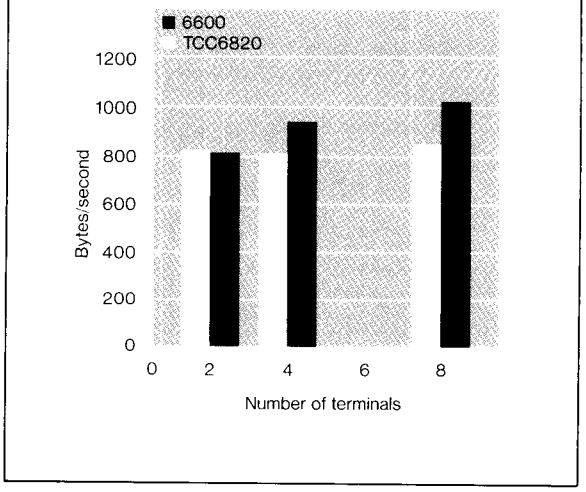


Figure 4.
Bytes per second processed by the 6600 and TCC6820 (1920-byte messages, 9600 bps).

Figure 5

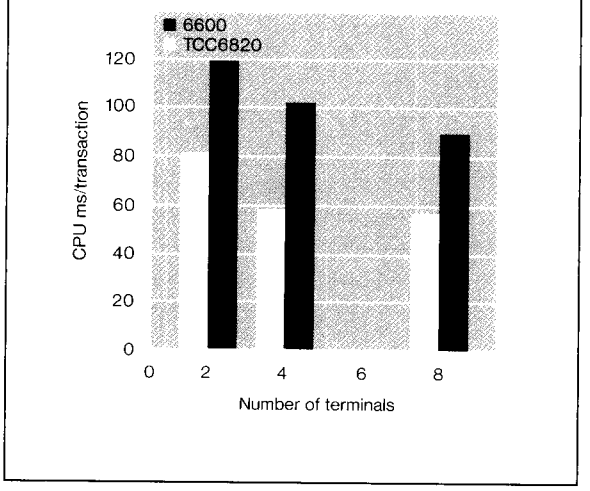


Figure 5.
CPU milliseconds per transaction required for the 6600 and TCC6820 to process 1920-byte messages at 9600 bps.

Figure 6

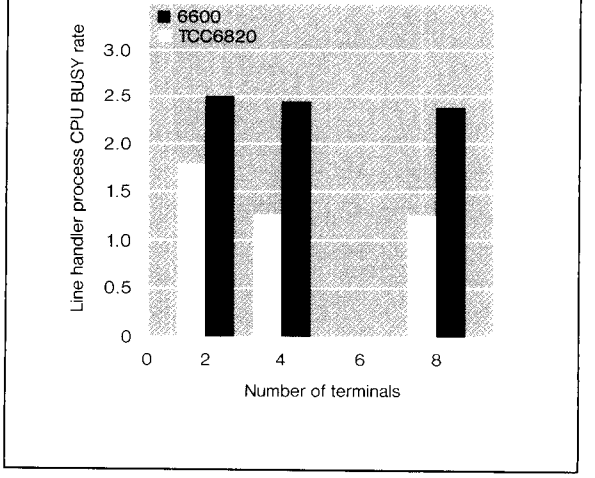


Figure 6.
Line handler process CPU BUSY rates produced by the 6600 and TCC6820 (1920-byte messages, 9600 bps).

Figure 7

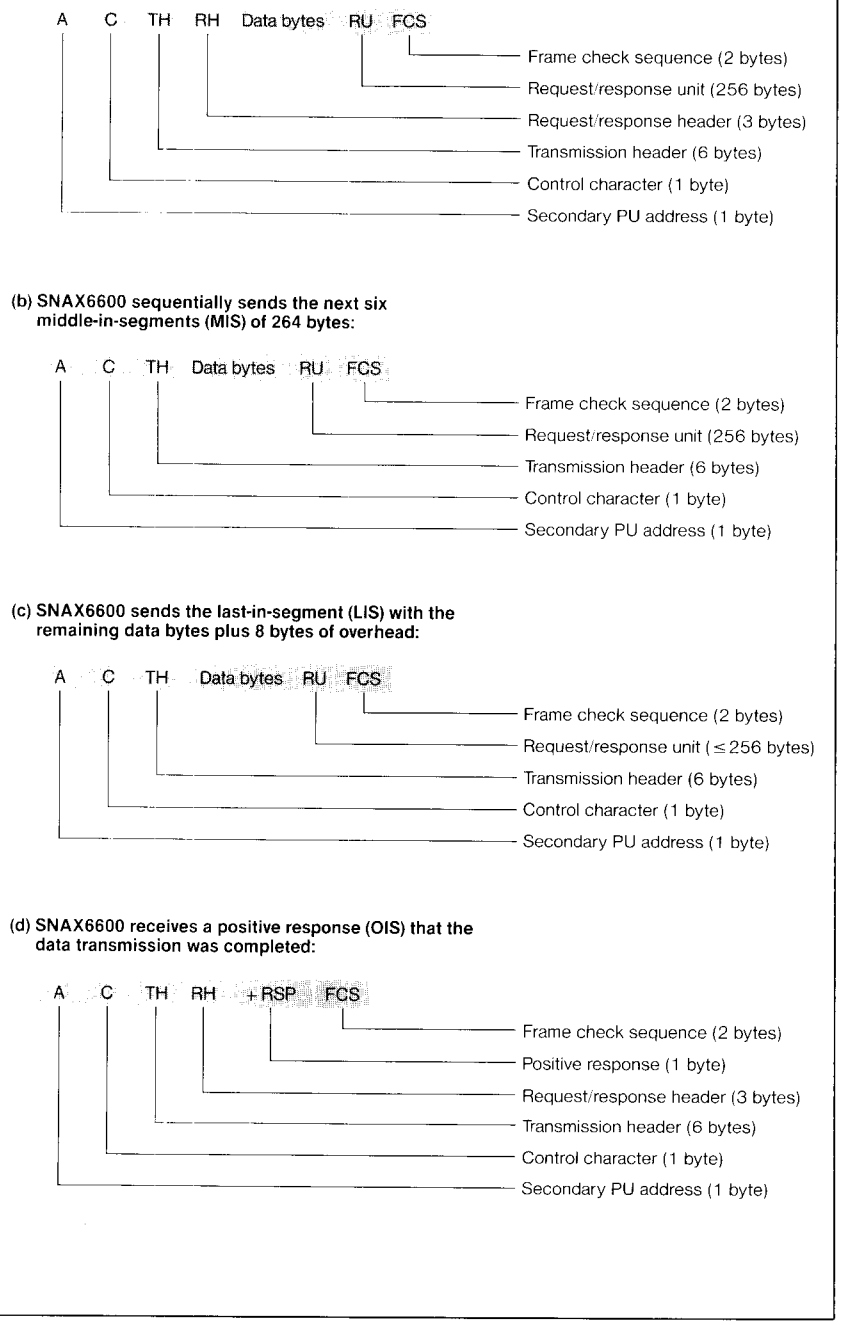


Figure 7.
The format used by SNAX6600 to segment the write to the terminal into eight packets.

Overhead Processing for SNAX6600. The application issued a 1920-byte write to the terminal, and SNAX segmented the message into eight packets. The format is shown in Figure 7. The following is an explanation of the SDLC protocol overhead, as represented in Figure 7.

A represents the secondary PU address, which identifies the physical unit sending the frame.

C represents the SDLC control character. The control features provided include the poll/final bit, frame sequence numbers, and the frame format. The *poll/final (P/F) bit* is used to control the directional flow of SDLC frames on the data link. Its presence signals the receiving station that it may now send frames.

TH represents the transmission header, an SNA header format created and interpreted by the path-control elements of network addressable units (NAU). It specifies the desired physical manipulation of the message (blocking and segmenting) and controls the physical routing of the message to the correct series of links toward its destination.

Any of three *frame formats* can be specified in the transmission header:

- *Unnumbered format (U-frames)*, also known as nonsequential frames (NSF), is used to initialize and control the response mode of secondary PUs, report procedural errors, and transmit unsequenced data.
- *Information transfer format (I-frames)* is used to hold sequence-numbered information fields (I-fields) and to confirm the error-free receipt of other I-frames.
- *Supervisory format (S-frames)* is used to confirm the error-free receipt of sequence-numbered I-frames and to convey the ready or busy condition of a PU.

RH represents the request/response header, an SNA header containing data flow control (DFC) and transmission control (TC) indicators that apply to the transmission of a single request/response unit (RU).

RU represents the request/response unit, the fundamental portion of an SNA message to which the various SNA headers are added. For message units that transfer data characters, the RU is the data portion of the SNA message.

FCS represents the frame sequence numbers, next receive (Nr) and next send (Ns). They are used by the SDLC protocol to guarantee the proper transmission and receipt of I-frames, given that link error recovery is possible. Ns is used in the control character of an I-frame, and Nr, in the control character of the S-frame.

Overhead Processing for the AM6520. When AM6520 byte-synchronous protocol is used, the character overhead is much lower. The application issued a 1920-byte write to the terminal, which AM6520 broke into eight 256-byte packets, including overhead and data. The protocol overhead is illustrated in Figure 8.

Before sending data, the AM6520 access method sends a fast select to the appropriate terminal (Figure 8a). It then waits for the acknowledgment from the terminal before sending the data.

For outbound data (Figure 8b), AM6520 expects a single buffer containing the textual part of a screen, escape character sequences, or both text and escape sequences. AM6520 adds a start of text (STX) at the beginning of the buffer and an end of text (ETX) after the last text character. Write operations to the selected terminal are split into 256-byte blocks for transmission.

AM6520 polls the terminal poll list again (Figure 8c).

For inbound data (Figure 8d), the buffer passed back to the application process contains the complete data message starting with the first text character received from the subdevice. Communication characters are removed. Although a Tandem 653X terminal sends input data in 256-byte blocks, AM6520 passes a single buffer back to the application containing the data portions of all blocks in a screen transfer.

The following is a description of some of the AM6520 protocol overhead:

- *Q, or fast select*, is generated when an application process issues a call to the WRITE-READ procedure.
- *ENQ, or enquiry byte*, is a priority request for a response.
- *ACK* is a positive acknowledgment from the device, returned in response to each block. *NAK* is a negative acknowledgment.
- *DEL or DELE* is a padding character.
- *P, or polling*, is initiated when an application issues a read to a device.
- *LRC, or longitudinal redundancy check*, is a 1-byte error check.

Figure 8.

AM6520 Protocol Overhead. (The SYN characters supplied by the driver are not shown.)

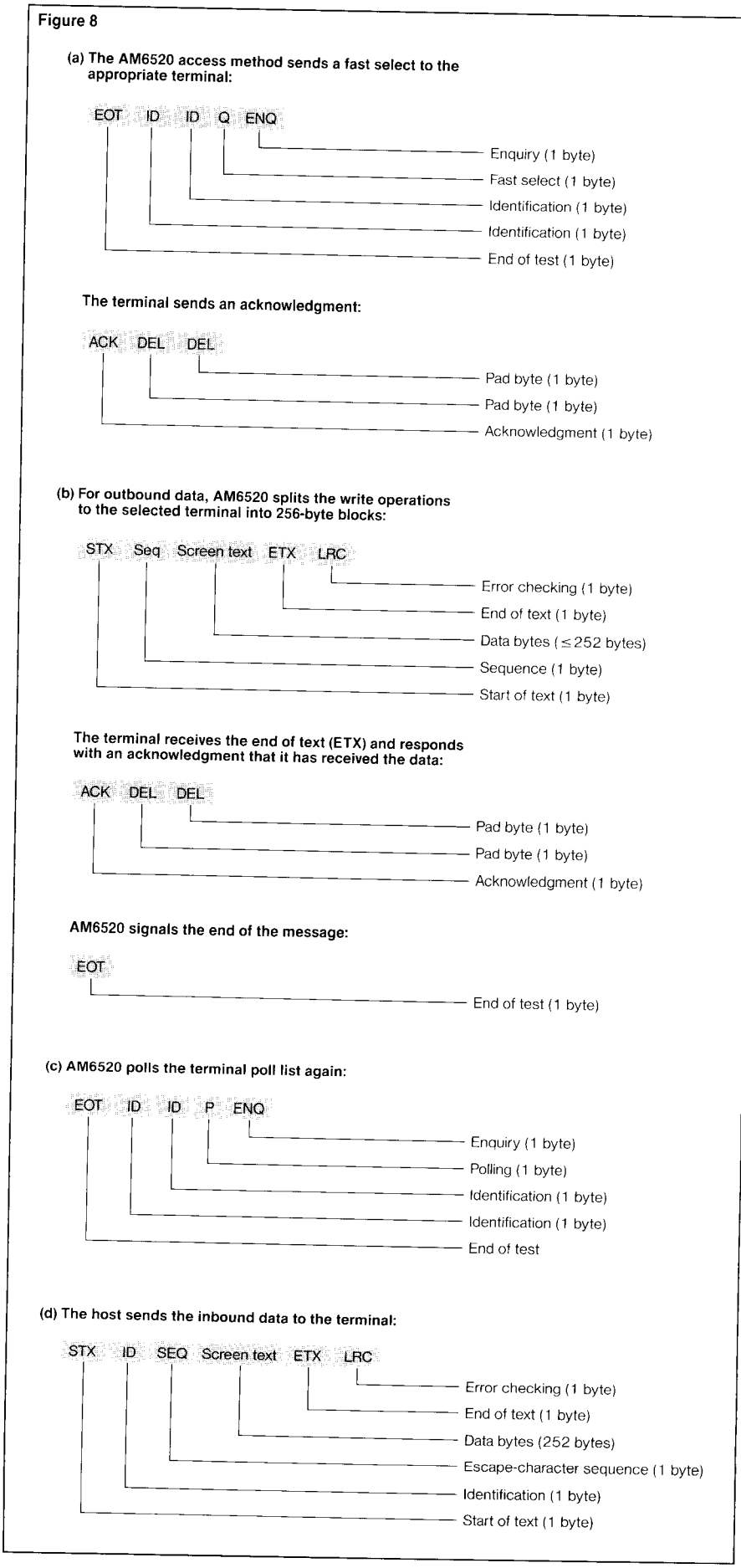
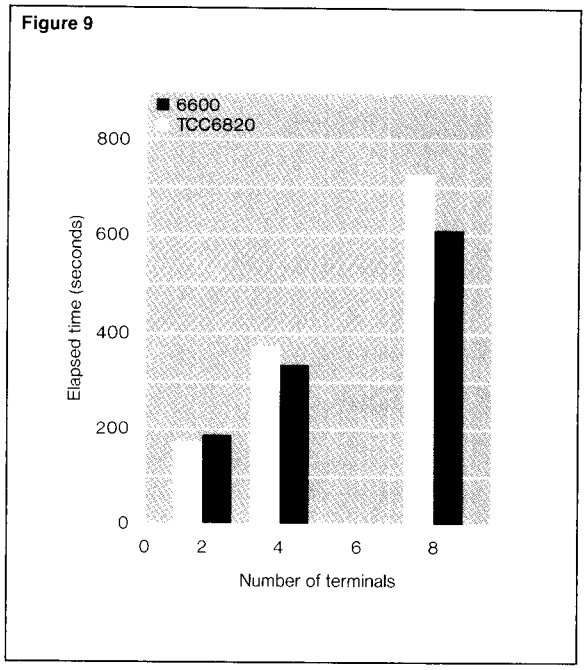


Figure 9.
Elapsed times for the 6600 and TCC6820 to process 1920-byte messages at 9600 bps.



Elapsed Time

To measure elapsed time, timestamps, buffered in memory, were placed in the application, one at the beginning of the first write and one at the end of the last read.

Figure 9 shows the elapsed times for the 6600 Controller and TCC6820 Concentrator. The difference in the elapsed time for four or more terminals is accounted for by the polling algorithms. Because of the "round robin" effect, the AM6520 access method must poll each device and wait for an acknowledgment before continuing down the list, whereas SNAX6600 uses first-come, first-served order, and polls only the controller.

Equations Used

The transaction rate was calculated by dividing the total number of transactions by the elapsed time (see the elapsed time calculation below). For example, for SNAX6600, two terminals, a message size of 1920 bytes, a rate of 9600 bps, and a completion of 20 transactions (trans),

$$2 \text{ terminals} * 20 \text{ trans} = \frac{40 \text{ trans}}{193 \text{ secs}},$$

or 0.207 transactions per second.

CPU cost per transaction was calculated by dividing the SNAX line handler CPU BUSY rate in milliseconds for the entire test by the number of transactions per second. For example, for SNAX6600, two terminals, a message size of 1920 bytes, a rate of 9600 bps, and a completion of 20 transactions,

$$\frac{24.8 \text{ CPU ms}}{0.207 \text{ trans/sec}} = 119.8 \text{ CPU ms/trans.}$$

The elapsed time was calculated by subtracting the timestamp of the first terminal started from the timestamp of the last terminal stopped. For example, for SNAX6600, two terminals, a 1920-byte message size, a rate of 9600 bps, and a completion of 20 transactions,

12:19:37.21 (time last terminal stopped)
- 12:16:19.37 (time first terminal started)
3:13.40 or 193 secs elapsed time.

Acknowledgments
The author would like to thank David Yang and Dick Van Praag of Large Systems Marketing Support for application support and hardware support, respectively.

Pat Beadles joined Tandem's Software Development Group in April 1984. She is currently developing a DP2 course in Software Education. Before joining Tandem, Pat spent three years as an analyst supporting data base performance and transaction accounting software for another mainframe.

To the computer industry, capacity planning means the process of predicting future data processing needs. Its goal is to ensure the availability of necessary data processing resources when and where they are needed. When used as a continuous process, it is an effective tool for an organization's tactical and strategic planning.

A comprehensive discussion of capacity planning and its relationship to capacity management, including performance tuning, is beyond the scope of this article. This piece discusses capacity planning and why it should be done. Intended for those individuals unfamiliar with capacity planning, it introduces the subject and describes some of the benefits derived from conducting capacity planning in an ongoing fashion. It also gives a general description of a model, general guidelines for conducting a successful study, suggestions for what can go wrong, and tips for presenting the results to management.

What Is Capacity Planning?

Almost everyone experiences capacity planning on a daily basis. Rush hour traffic jams are an example of demand outstripping capacity. The stop-and-go traffic caused by too many cars for the capacity of the highway is similar to what happens in a system when the demand exceeds the available capacity of the CPU; bottlenecks caused by contention for resources cause individual jobs and/or transactions to experience stop-and-go situations. The delay in getting to work is equivalent to the increased response time users experience when demand exceeds available capacity.

Scope

Capacity planning deals with a broad range of activities. For example, capacity planners are concerned with both network and host capacities.[1] They are interested not only in the growth of existing applications, but in new application development, business issues such as new market penetration or divestment, and changing strategic decisions. Any of these things may have an impact on the need for, or redistribution of, computing resources. Therefore, capacity planning can be viewed as a decision support mechanism for management. While its methods are technical, its point of view is that of business planning.

[1]Changes in the host hardware and/or software configuration can have dramatic effects on the response times that users receive. Conversely, the number of new terminals, the manner of attachment, location within the network, and type of work being done on them can have a serious impact on both users and the available capacity within the host.

Getting Started

The development of an effective capacity planning program requires an in-depth study of current resource utilization and workload characterization. From this study, the three or four applications that account for 80% to 90% of the workload become apparent. Also, by learning about future business growth plans and new or enhanced applications under development, the capacity planner knows which applications are likely to grow the most and is able to calculate the resources needed in the future. Steps can then be taken to ensure that resources are available to maintain established service levels.

These growth projections are used in many ways. Once an organization knows how and where its data processing resources are consumed, it can implement a chargeback system. A chargeback system can be used to distribute costs for the data center across departments within the organization or to sell time to other companies on a time-sharing basis. Growth projections can also be useful in the justification of new equipment.

Terminology

Total Capacity vs. Available Capacity. When an individual CPU is running at 100%, it is running at "total capacity." "Available capacity" is that available for use by a CPU before queuing delays inhibit processing and throughput. While this varies with the workload mixture, available capacity is usually between 70% and 80% of a CPU's total capacity.

Capacity Planning vs. Tuning. A system must be well tuned prior to conducting a capacity planning study. Whereas performance tuning is aimed at the technical staff and is done on a day-to-day basis, capacity planning is aimed at management, is expressed in business terms, and predicts various levels of resource utilization for the future (i.e., six months, one year, etc., in the future).

Models. The performance characteristics of a system can be represented mathematically as a model. The use of models, simple or complex, is an efficient means of estimating how changing the characteristics of the system (via the model) will affect various components of the system such as CPU Busy, Response Time, etc.

There are two primary capacity planning methodologies: regression analysis and extrapolation (RAE), and modeling. In the RAE type of analysis, historical data for the last two years or so is collected and plotted using regression analysis. Within confidence limits, the regression line produced by this method is then extended into the future on the assumption that, in general, growth will continue along the regression line. This line can then be adjusted based on assumptions about future growth.

With the modeling method of analysis, current performance data is collected to establish the current levels of resource utilization and available system capacity. A model is built and validated to the measurement sample. Data on resource utilization is collected by talking to application developers, strategic planners, and the organization's management. This data is then placed into the model and predictions are made as to when and where bottlenecks are most likely to occur. Proposed hardware, software, or configuration changes can also be added to a baseline model to determine the effects of these changes.

This article discusses the modeling method since it is more accurate in today's constantly changing data processing environment.

Model Types

The complexity of the model used in capacity planning depends on the study's objective. If its objective is to determine the overall capacity of a system, the model can be fairly simple.

A *baseline model* is the first model built and reflects the current environment. Projected changes to the system use this model as their base.

Predictive models are used to predict where resource bottlenecks will occur. They are also used to determine the effects of any proposed changes to the system. When the projections are added to the baseline model, it becomes a "predictive model."

However, models used for detailed I/O subsystem analysis are an advanced form of modeling and can greatly increase the complexity of a model. An I/O subsystem model is needed only about 5% of the time and is used primarily for very specialized studies.

Conducting a Capacity Planning Study

Who Should Conduct It

Where there is a critical dependence on computing resource availability and performance, some form of capacity planning should be going on. This can be as critical in small shops as in large organizations. How formalized the process is often depends on the size of the organization.

When possible, the capacity planning function is a staff function reporting to the Director of Management Information Systems or higher. This helps to ensure the cooperation necessary when capacity planning crosses organizational boundaries.

Large organizations should have formal charters for capacity planning functions with one or more full-time people dedicated to them. (A charter is a statement of purpose endorsed by management.) Smaller shops may have someone in the operations or systems area doing informal capacity planning on a part-time basis.

Because of its size, a smaller shop has fewer available resources and possibilities for expansion. Also, the smaller shop is likely to experience more rapid growth than the larger organization. A small shop can, in most cases, take advantage of capacity planning help from the CPU vendor. This help is usually in the form of guidelines for setting up the capacity planning process based on the shop's needs and may include the vendor doing a study for the customer.

When to Conduct It

The data collection for capacity planning should be an ongoing process. Current capacity figures should be added to the capacity planning projections monthly and "actual vs. projected" charts should be produced quarterly for management review. At least semiannually, a new baseline model should be built and new projections obtained and input to the model. Major changes to the existing configuration or the introduction of a new application which is a large resource consumer also necessitates a new baseline model and corresponding projections. If the data that the projections are based on changes significantly due to economic conditions or changes in management direction, then it is time to prepare a new baseline model and adjust the projections accordingly.

How to Conduct It

The methods used and the assumptions made when doing a capacity planning study have a direct bearing on the accuracy of the results. Even "quick and dirty" studies must consider certain steps to follow.

Senior management support and a well-tuned system are the two primary prerequisites for conducting a study.

Senior management support is necessary because the capacity planner conducting the study interfaces with different managers and departments. This support aids in getting the information and support needed from other managers. Diplomacy is important, as are assurances to the managers that they can review the final document before publication.

Senior management support and a well-tuned system are the two primary prerequisites for conducting a study.

If the system being modeled is not well tuned, the projections vary significantly from actuality. The capacity planner should know enough about performance tuning to ensure that the system is well tuned. If there is some question about this, it may be necessary to consult with the people in charge of system tuning to see if improvements can be made. If necessary, the vendor can provide valuable support in this area.

This preliminary tuning should be done with a minimum expenditure of funds on additional hardware or software. The results of the capacity planning study determine what, if anything, is needed and aid in providing justification for the recommendation.

A successful capacity planning study requires a definite plan. The following steps are typical, though there may be variations dictated by the nature of the study or company policy. The time needed to complete each step may also vary.

Management Approval. Getting management approval for a study is essential to the success of the study. With upper management support, the people contacted are more likely to realize the importance of the project and provide the level of support and information needed. If they know that senior management supports the project they are more careful to ensure the accuracy of the information given to the analyst.

Establishing Objectives. Everyone involved must be aware of the objectives and of the importance of the outcome to the future of the organization. Answering the following questions helps to identify the objectives:

- What is our current capacity?
- Based on our projected rate of growth, when will we run out of capacity?
- What are our options and the associated costs?

Collecting Data. Data must be collected on a continuous basis for performance tuning, problem resolution, job accounting, and capacity planning. By having this data available at the start of a capacity planning study, the analyst can save time and ensure that the data selected is representative of "normal" operating conditions.

Analyzing Data. To effectively analyze data, the capacity planner must first determine the "typical" workload level to plan for (i.e., the peak hour of the peak day of the year, the peak hour of an average day of the year, etc.). This information determines the time frame and characteristics of the period selected for modeling.

The data collected is analyzed to verify the following:

- The criteria established by the questions above are met.
- There is nothing abnormal in the system that might skew the data.

The various workloads in the system (batch, on-line, etc.) broken down by major application and the resources consumed by each need to be determined. Accurate characterization of the workload mix in the system during the time frame being modeled is critical to the accuracy of the results.

Building a Model. After completion of the previous steps, a model is built using performance data from the "typical" time period selected in the previous step.

A basic model generally has two major sections: a description section containing information about the operating environment (CPU type, disk and tape types) and the major applications that are in the system, and a resource-consumption section.

These applications are known as workloads. There is usually one workload for each major application in the study and additional workloads for batch, on-line, and an "other" category to allocate the rest of the resource consumption.

The resource consumption section of the model is where the resource utilization of each workload is distributed across each of the various devices used by that workload.

Each disk and/or tape device, as well as the CPU, is known as a server. Every workload has some of its total resource consumption allocated to the CPU server. The rest of the resource utilization is distributed only among servers used by that workload.

The resource allocated to each server is the amount of time spent at each server. This information comes from the measurement report analyzed previously and is usually in milliseconds. This value is known as "service time" or "demand."

The model constructed from the current system and its current workloads is the baseline model.

Validating the Model. After the baseline model is built, it is run and the output compared to the corresponding measurement data. The utilizations, queue lengths, throughputs, etc., calculated by the model should match the measured data from which the model was built to within 5%. The key indicators are the utilization percentages of the CPUs, I/O activity, and I/O devices of concern. If the model doesn't match to within 5% or less, a check of the input data is made. This check shows whether or not the various workloads were determined correctly in the first place, and if the total resource consumption was accurately allocated to each workload.

Other items of interest are queue times, queue lengths, memory utilization, and throughput, if this data is available.

Collecting Growth Projections. GIGO (garbage in/garbage out) is a frequently used acronym in capacity planning. When interviewing managers for growth predictions for the upcoming year, the capacity planner must stress that the accuracy of the model's projections is only as good as the input data. It is also important to remember that growth can be either upward or downward. Downward growth can result in more capacity in the future than is available now. To collect the information needed for growth projections, start with the following questions:

- Will there be more, fewer, or the same number of employees next year?
- Will people be doing the same type of work they are doing now? If not, how will the workload change?

Senior management should be able to provide insight into how economic projections will affect the company's growth plans over the next year. For industry growth projections, check trade publications.

What-if Analysis. Once the growth projections are collected, they can be broken down by application, converted to milliseconds of service time, and added into the baseline model. This provides the same results as asking "What if system resource consumption grows by this amount?"

When applying the growth projections to a model, allow for projections being high or low. A good way to do this is to add 10% or 15% to the projections and rerun the model. Next, subtract 10% or 15% from the projections and run the model. Plot all three projections on the same graph. When this is done, available capacity shortfalls can be evaluated based on the possibility that the projections were high, low, or right on target.

"Reality Checks" of Analysis Results. When both the runs and preliminary charts are made, a "reality check" is done to see if the projections look reasonable. If they don't, then the projections and their distribution across the various workloads needs reevaluating. This is perhaps the most difficult aspect of capacity planning, for it requires sound and seasoned judgment.

Evaluating Alternatives and Costs. After the projections are accepted as reasonable, alternatives are examined. If the growth projections were negative, there will be available capacity. On the other hand, if the projections were positive, bottlenecks in the system will have appeared.

Various alternatives can now be evaluated to determine what can be done to minimize or eliminate the effect of the additional workload caused by the projected growth. Alternatives to consider include:

- Doing nothing.
- Adding a CPU, disks, controllers, etc.
- Adding more memory.
- Making scheduling or operations changes.
- Redistributing the hardware and/or the data base.

Total costs must also be considered when proposing a plan of action to management. Too often a capacity planner makes the mistake of recommending a solution that includes only the cost of the hardware. Take care to include costs such as heating/cooling, power, and other physical site changes that may be needed. Also consider the need for additional personnel in the operations and/or scheduling areas to support any additional equipment. And finally, can the new equipment fit in the existing computer room and still meet any cable length restrictions?

The Report. The most important step in the capacity planning process is the preparation of the report, but it is begun only after the following steps are completed:

- All data is collected, validated, placed into a model, and evaluated.
- Growth projections are added to the model and it is run.
- All options are considered and decided on.

While preparing the report it is important to remember that a poorly prepared report can kill an excellent study.

Care must be taken to ensure that the results of the study are presented to management in terms related to business functions and not in "computerese." One of the most important abilities of the capacity planner is that of translating technical computer terms into business-related measures that management understands. Rather than talking about "resource consumption," examples of it are used; i.e., how do the number of passenger miles flown or the number of ATM transactions in a day relate to the available CPU capacity.

Management is not interested in detail information; a general outline of the steps for collecting and validating the data and growth projections is sufficient. Management needs to know what the options are, why a particular course of action is recommended, the costs and time involved, and most importantly, how this will improve the overall efficiency and profitability of the company.

Make sure that all assumptions in the study are clearly presented in the report to management. If inaccuracies occur later between actual and projected changes, these assumptions should be reevaluated.

When considering various alternatives and the effects of each on the data processing environment, be sure to include the option of "doing nothing" and its effect.

Also allow for latent demand. Latent demand is work that needs doing, but the resources to do it are lacking. An example is when logon is denied to a user because the system's maximum capacity for logged-on users has been reached. As more resources become available and the number of users allowed on the system is increased, new users may use up the additional capacity that was just gained. The system is out of capacity again and the capacity planner is forced to return to management and request more resources. This undersizing of needed capacity damages the capacity planner's credibility. Lines of communication should be established early with the user community, and users should be encouraged to report any inability to access the system. This provides a barometer of latent demand.

Presenting the Results to Management. After the report is prepared, it is presented to management. Just as a poor report can kill a good study, a poor presentation of that report to management casts doubt on the validity of the study and the credibility of the capacity planning group. Be prepared and sure of the facts. Credibility is the most important asset of the capacity planning group as well as the hardest to achieve. It comes only after many studies in which the group's predictions and analyses of the issues are proved correct. One bad study can lower credibility more than many good ones can improve it.

Presenting the report to management gives the capacity planner the opportunity to state his or her case in person and to answer questions. The more supporting information the capacity planner has, the better the case made. Detailed supporting documentation is needed on hand for answering technical questions that may arise. Color foils and/or slides aid management's understanding of the study as well as enhance the professional image of the capacity planning group.

Validating Projected vs. Actual Growth. The study does not end when management reaches a decision concerning the proposed options. The analyst continues to track actual versus projected growth as the data becomes available (usually monthly) and presents this information to management.

If a wide discrepancy starts to develop, the capacity planner determines why and adjusts the model accordingly. Only by this constant refining of the model is the capacity planner assured of the utmost accuracy on future projections. Management should always be informed when changes are made. They are also given reasons for the variance between the actual and the projected data.

This feedback loop is equally important for the departments supplying projections. They must be aware of the effects of any discrepancies introduced by them if they are to fine tune their own planning processes. Accuracy must be emphasized.

Pitfalls and/or Reasons for Inaccuracies in Studies. There are a number of things to watch out for when conducting a capacity planning study. Any one can cause problems and render a study useless. Some of the major ones are listed below:

- The system is poorly tuned to start with.
- The system workload characteristics are misunderstood.
- The objectives of the study are misunderstood.
- The baseline model is inaccurately validated.
- Management support is lacking.
- Nontypical data is chosen to build the model.
- Growth predictions are inaccurate.
- The needed data is unavailable.

Conclusion

Much of an organization's cost savings today comes from effective capacity planning. Money saved by deferring installation of one piece of equipment or by ordering only as much as is needed can exceed a million dollars. There are also the productivity improvements experienced by the system's users. With hardware costs decreasing and personnel costs increasing, organizations must make users as productive as possible. Good capacity planning assures that sufficient resources are available when people need them the most.

Increasingly, senior management is coming to realize that the lifeblood of an organization is the information contained in the data processing system. Because of this, data processing is treated as an important resource to the company rather than an overhead function.

References

Cooper, J.C. 1980. A Capacity Planning Methodology. *IBM Systems Journal*. Vol. 19, No. 1.

Lipner, Dr. L.D. Capacity Planning Methodology. *Auerbach Information Management*. Series 45-01-07.

_____. 1984. Five Risks in Information Center Capacity Planning. BGS Systems Inc.

Watson, L., and Madsen, K. 1984. Capacity Planning for Tandem Computer Systems. *Tandem Application Monograph Series*. Part no. 83904. Tandem Computers Incorporated.

Acknowledgment

The author would like to thank Karna Thulin for providing valuable support and expertise during the development of this article.

Richard W. Evans was a senior staff analyst in the PSG Capacity Planning Group. He joined Tandem in March 1985 after spending 15 years doing capacity planning and performance analysis on IBM systems. He currently works for a Tandem customer.

How to Set Up a Performance Data Base with MEASURE and ENFORM

A performance data base is a structured data base of system performance information. It is used for capacity planning and the tuning and balancing of system resources. The data is measured and collected with a performance measurement tool, such as MEASURE. Other software, such as the File Utility Program (FUP), PATHWAY transaction processing system, Communications Management Interface (CMI), and Communications Utility Program (CUP), can provide additional information.

The detailed data from these sources is condensed to form the performance data base. Only those items necessary to analyze the performance of the system/network over a period of time are stored. Currently, on Tandem systems, this can be done with the ENFORM query language and report generator.

This article provides a brief introduction to using MEASURE and ENFORM together and then describes one way to set up a performance data base and produce reports. It includes sample OBEY files of MEASURE commands and ENFORM queries that streamline the process. To benefit fully from the article, readers should have a basic understanding of which performance metrics are significant for Tandem systems and how Tandem system performance is analyzed.

Using MEASURE and ENFORM Together

MEASURE collects performance data on all of the major hardware elements in typical on-line transaction processing (OLTP) applications, such as the processors, disk drives, and lines to the terminals. It also collects information on all the major software elements, such as the system software for disk drives and lines and the application requesters and servers. Finally, it collects information on such elements as files and supporting software (for example, PATHMON, the PATHWAY Monitor).

Users may not need all of the many items MEASURE collects for a given performance analysis task. To condense the data and produce the specific reports they need they can use ENFORM.

The use of ENFORM with MEASURE generally follows a standard scenario. Typically, the performance analyst sets up a MEASURE measurement. Then the data generated by MEASURE is moved to structured files and fed into ENFORM queries for data reduction and analysis. If the data is to be used for historical purposes, ENFORM can be used again to condense the data for long-term storage. For example, 24 hourly reports can be condensed into three 8-hour shift reports containing their minimum, maximum, and average measurements. This data can then be summed up into monthly minimum, maximum, and average measurements.

ENFORM can also make the data more useful by computing the results of various formulas on specific data items. For instance it can compute the ratio of the MEASURE Process Entity report items ATIME READY and ATIME BUSY so that the effect of priority queuing can be analyzed.

With ENFORM, users can create a variety of reports on the performance data base as well as correlate the performance data with that of other data bases. One highly useful report signals threshold violations for key performance indicators (Figure 1); another shows the CPU utilization trend (Figure 2).

For a complete description of how to use ENFORM with MEASURE, see the *MEASURE User's Guide*, *MEASURE Reference Manual*, and *ENFORM Reference Manual*.

Setting Up the Data Base

The following is one example of how to set up a performance data base with MEASURE and ENFORM. The OBEY and ENFORM query files mentioned follow in the next section of the article.

Define Performance Indicators

Start by defining the key performance indicators for the system and setting optimum goals for them. These indicators should show how much of each system resource is being used and how much is left. They should also aid in predicting resource usage trends, including which of the resources are most likely to become bottlenecks.

The resource usage and queue lengths for CPUs, disks, and lines as well as system response time, turnaround time, and transaction throughput are excellent indicators of these system characteristics. With MEASURE, these indicators are easily accessible, as is information related to applications, such as file activity and interprocess message traffic.

Set Up MEASURE Commands in OBEY Files

A sample MEASURE data-collection configuration, set up in a MEASCOM OBEY file, appears in the next section. This configuration is suitable for a generic OLTP application.[1] The *MEASURE User's Guide* explains how to implement the counters in the application programs.

A sample OBEY file containing MEASURE commands to convert raw data files into structured data files is also shown.

[1]Information from other sources, such as the Communications Utility Program (CUP) and the PATHWAY transaction processing system may also be useful. ENFORM can be used to integrate this information into the data base as well.

Figure 1

THRESHOLD REPORT

SYSTEM: BANCARD
DATE: 05/08/86
SHIFT N
OS VERSION: B30

CPU #	TIME OF DAY	MAX CPU QLEN	DELT CPU QLEN	MAX CPU SWAP	MAX CPU BUSY	DELT CPU BUSY	MAX INTR BUSY	AVG CPU QLEN	AVG CPU SWAP	AVG CPU BUSY	AVG CPU INTR
0	11:00	2.0	1.3	.2	55.4	10.5	8.5	1.2	.1	42.6	6.3
0	13:00	3.4	2.8	1.4	65.1	15.3	10.2	1.5	.3	53.3	8.6
0	14:00	4.6	2.7	3.1	60.2	17.9	13.8	2.6	1.6	52.4	11.5
0	15:00	7.1	4.1	1.2	75.1	24.9	15.5	2.6	.6	63.6	12.9
0	17:00	9.3	1.1	7.2	85.3	23.0	19.2	3.8	1.8	73.2	14.4

Figure 2

CPU TREND REPORT

DATES: JAN 1 TO APR 2

SYSTEM NAME	LOADID DTDDDHH	CPU NUM	CPU TYPE	OS VERS	MAX CPU QLEN	DELT CPU QLEN	MAX CPU SWAP	MAX CPU BUSY	DELT CPU BUSY
\POS	DT00111	01	02	B30	1.5	.3	.2	35.4	5.5
\POS	DT00211	01	02	B30	2.0	1.3	.2	55.4	10.5
\POS	DT00310	01	02	B30	4.6	2.7	3.1	60.2	17.9
\POS	DT00411	00	02	B30	3.4	2.8	1.4	65.1	15.3
	.								
	.								
	.								
\POS	DT06011	00	02	B30	7.1	4.1	1.2	75.1	24.9
	.								
	.								
	.								
\POS	DT09211	01	02	B30	9.3	1.1	7.2	85.3	23.0

Figure 1.
A sample threshold report created with the ENFORM query language and report generator from MEASURE performance data. This report signals threshold violations for key performance indicators. The ENFORM query file to produce the report appears in the last section of the article.

Figure 2.
A sample CPU utilization trend report created with ENFORM from MEASURE data. For the hour of each day with the largest maximum average CPU queue length, the following are listed: the maximum average CPU queue length (MAX CPU QLEN), the difference between the maximum and minimum average CPU queue lengths (DELT CPU QLEN), and the swap rate (MAX CPU SWAP), maximum CPU BUSY rate (MAX CPU BUSY), and delta CPU BUSY rate (DELT CPU BUSY) between the busiest processor and the least busy processor.

Figure 3

```
(Log on with normal user group ID.)
VOLUME $VOL.CPLNWEEK                  ! Sets subvolume.
                                      !
ADD CPU *                             ! Measures all CPUs.
ADD DISC *                            ! Measures all disks.
ADD LINE *                            ! Measures all lines.
ADD PROCESS SYSTEM-PROCESSES          ! Measures all system procedures.
ADD USERDEF $PROGS.SERVER1.*          ! Adds processes that have
ADD COUNTER RESP-TIME,                !   response time and...
  PROCESS $PROGS.SERVER1.*,           !
  QUEUE                               !
                                      !
ADD COUNTER THROUGHPUT,               !   throughput counters in
  PROCESS $PROGS.SERVER1.*,           !   them.
  ACCUM                               !
                                      !
START MEASUREMENT CPLNDAY<n>,         ! Starts the measurement for
                                      !     day <n> of the week.
  FROM 00:05 FOR 24 HOURS,            !   Starts at midnight for
                                      !     24 hours.
  INTERVAL 1 HOURS                    !   Writes data every hour.
```

Figure 4

```
VOLUME $VOL.CPLNWEEK                  ! Sets volume for week file.
ADD FILE CPLNDAY<n-1>                 ! Lists yesterday's data.
SET REPORT FORMAT STRUCTURED          ! Outputs structured files.
LIST CPU *, LOADID DT<jd>00, FROM 00:00, FOR 1 HOURS
LIST DISC *, LOADID DT<jd>00, FROM 00:00, FOR 1 HOURS
LIST LINE *, LOADID DT<jd>00, FROM 00:00, FOR 1 HOURS
LIST PROCESS *, LOADID DT<jd>00, FROM 00:00, FOR 1 HOURS
LIST USERDEF *, LOADID DT<jd>00, FROM 00:00, FOR 1 HOURS
LIST CPU *, LOADID DT<jd>01, FROM 01:00, FOR 1 HOURS
LIST DISC *, LOADID DT<jd>01, FROM 01:00, FOR 1 HOURS
          .
          .
          .
LIST CPU *, LOADID DT<jd>23, FROM 23:00, FOR 1 HOURS
LIST DISC *, LOADID DT<jd>23, FROM 23:00, FOR 1 HOURS
LIST LINE *, LOADID DT<jd>23, FROM 23:00, FOR 1 HOURS
LIST PROCESS *, LOADID DT<jd>23, FROM 23:00, FOR 1 HOURS
LIST USERDEF *, LOADID DT<jd>23, FROM 23:00, FOR 1 HOURS
```

Figure 3. *A MEASCOM input file containing a sample MEASURE configuration.*

Figure 4. *An OBEY file that converts raw data files to structured data files.*

Set Up ENFORM Queries

Make a set of ENFORM query files. These files will manage the data condensation, analysis, and reporting. Typically, they will handle the following:

- Daily reports, containing threshold alarms.
- Weekly or monthly reduction of daily files.
- Monthly/yearly trend reports of resource consumption and system health.

As mentioned earlier, sample threshold and CPU trend reports are illustrated in Figures 1 and 2. Sample ENFORM query files follow the MEASURE OBEY files in the next section.

Sample OBEY and ENFORM Query Files

These files can be used to streamline the data collection, condensation, and reporting process for performance analysis.

Start Up the MEASURE Subsystem

Begin by starting up the subsystem:

LOGON <Super.operator,password>[2]

START MEASSUBSYS

Start Up the MEASURE Configuration

Once the MEASURE data collection processes have been started, the data collection configuration must be installed in the MEASCTL processes. The MEASCOM input file in Figure 3 contains a sample configuration and starts the data collection for the next 24 hours. (The collection automatically stops at the end of the 24-hour period.)

After this file has been obeyed by MEASCOM, wait 24 hours, change <n> to <$n+1$>, and obey it again (<n> is modulo 7).

Convert Raw Data to Structured Data File

Use MEASCOM nightly to convert raw data files to structured data files. The OBEY file in Figure 4 contains operations performed in MEASCOM. Change the data file name to CPLNDAY<n>, where <n> is the day of the week; the start-up OBEY file is an example of this. (Note that <jd> is the Julian date of the MEASURE data.)

[2]Angle brackets (< >) in the examples represent user-supplied variable entries.

Condense the Data for Each Hour of the Day

The query in Figure 5 condenses the data set to a much smaller subset by selecting only those items in the CPU report that are of interest to the capacity planner. This step begins to make the data set more manageable. This report is for the CPU resource only; use the same technique for the rest of the resource reports before the next set of ENFORM queries.

Note: The ENFORM queries presented in this article are examples only; they may not work on all systems. Read the *ENFORM User's Guide*, section 3, before attempting to use ENFORM to analyze MEASURE data.

Condense Hourly Items into Shift Items

The query in Figure 6 further condenses the data from hourly items into shift items by approximately an 8-to-1 reduction. This example is for CPU performance; set up similar queries for the other resources.

When the query is complete, copy the results of the data reduction into the end of the current month's summary file for that shift:

```
FUP
copy cplnmnth.cput,cplnmnth.cpushft<n>
```

Run the query and copy the results to the summary file as many times as there are shifts, changing the BEGIN^SHIFT and END^SHIFT parameters and the CPUSHFT<*n*> file name each time.

Figure 5.

An ENFORM query that condenses the performance data set to a more usable subset. (This query condenses CPU data; use similar ones for the other resources.)

Figure 6.

A query that further condenses the data from hourly items into shift items (approximately an 8-to-1 reduction).

Figure 5

```
?DICTIONARY
?VOLUME CPLNWEEK
OPEN CPU,CPUT;                          ! Condenses the MEASURE data into
                                        ! fewer data files containing
FIND CPUT (                             ! pertinent data only.
BY SYSTEM-NAME
BY LOADID
BY CPU-NUM
OS-VERSION
CPU-TYPE

QLEN     : =    CPU-QTIME/DELTA-TIME
MQLEN    : =    MEM-QTIME/DELTA-TIME
SWAP     : =    SWAPS/DELTA-TIME
BUSY     : =    CPU-BUSY-TIME/(DELTA-TIME/100)
IBUSY    : =    INTR-BUSY-TIME/(DELTA-TIME/100)
SBUSY    : =    SEND-BUSY-TIME/(DELTA-TIME/100)
);
```

Figure 6

```
?VOLUME CPLNWEEK
?DICTIONARY
PARAM       Begin^shift                 ! Brackets the data into shifts.
            End^shift                   !
            shift;                      ! Specifies the shift number.

OPEN CPUT,CPLNMNTH.CPUT;                ! Condenses the MEASURE data into
                                        ! fewer data files containing
FIND CPLNMNTH.CPUT (                    ! pertinent data only.
BY SYSTEM-NAME
BY CPU-NUM
BY LOADID
OS-VERSION
CPU-TYPE
SHIFT

QLEN-AVG-D:  = AVG ((QLEN) OVER CPU-NUM)
QLEN-MAX-D:  = MAX ((QLEN) OVER CPU-NUM)
QLEN-MIN-D:  = MIN ((QLEN) OVER CPU-NUM)

MQLEN-AVG-D:  = AVG ((MQLEN) OVER CPU-NUM)
MQLEN-MAX-D:  = MAX ((MQLEN) OVER CPU-NUM)

SWAP-AVG-D:  = AVG ((SWAP) OVER CPU-NUM)
SWAP-MAX-D:  = MAX ((SWAP) OVER CPU-NUM)

BUSY-AVG-D:  = AVG ((BUSY) OVER CPU-NUM)
BUSY-MAX-D:  = MAX ((BUSY) OVER CPU-NUM)
BUSY-MIN-D:  = MIN ((BUSY) OVER CPU-NUM)

IBUSY-AVG-D:  = AVG ((IBUSY) OVER CPU-NUM)
IBUSY-MAX-D:  = MAX ((IBUSY) OVER CPU-NUM)

SBUSY-AVG-D:  = AVG ((SBUSY) OVER CPU-NUM)
SBUSY-MAX-D:  = MAX ((SBUSY) OVER CPU-NUM)

);
Where ( From^time > ( Begin^shift - 1 ))
                AND
      ( End^time  < ( End^shift + 1 ))
```

Figure 7

```
PQLEN-AVG-D:  = AVG ((RECV-QTIME/DELTA-TIME)
                     OVER PROCESS-NAME)
                     WHERE (RECV-QTIME/DELTA-TIME) > .3
PQLEN-MAX-D:  = MAX ((RECV-QTIME/DELTA-TIME)
                     OVER PROCESS-NAME)
                     WHERE (RECV-QTIME/DELTA-TIME) > .3
VSEM-MAX-D:   = MAX ((VSEMS/DELTA-TIME)
                     OVER PROCESS-NAME)
                     WHERE (RECV-QTIME/DELTA-TIME) > .3
          .
          .
          .
READY-BUSY-AVG-D: = AVG ((READY-TIME/CPU-BUSY-TIME)
                         OVER PROCESS-NAME)
                         WHERE (CPU-BUSY-TIME/(DELTA-TIME/100)) > .3
);
```

Figure 8

```
?DICTIONARY
?VOLUME CPLNMNTH
OPEN CPUT,CPUSHFTN;                ! Condenses the MEASURE data into
                                   ! fewer data files containing
FIND CPLNYEAR,CPUT (               ! pertinent performance data only.
BY SYSTEM-NAME
BY CPU-NUM
BY LOADID
OS-VERSION
CPU-TYPE
SHIFT
QLEN-AVG-M:   = AVG ((QLEN-AVG-D) OVER CPU-NUM)
QLEN-MAX-M:   = MAX ((QLEN-MAX-D) OVER CPU-NUM)
BUSY-DELT-M:  = MAX ((BUSY-DELT-D) OVER CPU-NUM)
MQLEN-AVG-M:  = AVG ((MQLEN-AVG-D) OVER CPU-NUM)
MQLEN-MAX-M:  = MAX ((MQLEN-MAX-D) OVER CPU-NUM)
SWAP-AVG-M:   = AVG ((SWAPS-AVG-D) OVER CPU-NUM)
SWAP-MAX-M:   = MAX ((SWAPS-MAX-D) OVER CPU-NUM)
BUSY-AVG-M:   = AVG ((BUSY-AVG-D) OVER CPU-NUM)
BUSY-MAX-M:   = MAX ((BUSY-MAX-D) OVER CPU-NUM)
QLEN-DELT-M:  = MAX ((QLEN-DELT-D) OVER CPU-NUM)
IBUSY-AVG-M:  = AVG ((IBUSY-AVG-D) OVER CPU-NUM)
IBUSY-MAX-M:  = MAX ((IBUSY-MAX-D) OVER CPU-NUM)
SBUSY-AVG-M:  = AVG ((SBUSY-AVG-D) OVER CPU-NUM)
SBUSY-MAX-M:  = MAX ((SBUSY-MAX-D) OVER CPU-NUM)
);
```

Figure 7.
A query that condenses data from the PROCESS data file by restricting it to minimum activity level. Use similar queries to condense the data for the disk, line, and USERDEF reports.

Figure 8.
A query that produces the monthly average data for a resource. This example is for the CPU resource, by shift.

Summarize PROCESS Data

The query in Figure 7 resembles the CPU query except that the data is derived from the PROCESS data file (created by MEASURE for input to ENFORM). Note that the number of records is reduced by restricting the data to minimum activity level.

Use similar queries to condense the data for the disk, line, and USERDEF reports. For disks, key items include DISC BUSY, REQ QLEN, SEEK BUSY, CACHE HITS, and CACHE MISSES. WRITE BUSY and BYTE RATE suffice as report line items. For USERDEF, see the documentation on user-defined counters in the *MEASURE User's Manual*.

After all data in the weekly subvolume has been condensed, the data files in that subvolume can be purged.

Condense Weekly Shift Summaries into Monthly Ones

For long-range trend analysis, summaries of monthly averages are useful. The query in Figure 8 produces the monthly average data for the various resources that comprise the system. This example is for the CPU resource, by shift.

After the data has been condensed, copy the results into the end of the current year's summary file for that shift and do the same for the process, disk, line, and USERDEF reports.

Produce Special Threshold Reports

Threshold reports can highlight potential or real problems for prompt action. Figure 9 contains a sample OBEY file for such a report. It uses ENFORM to isolate maximum queue lengths, CPU BUSY rates, and other rates above acceptable limits. The report is illustrated in Figure 1.

Use the same query for process, disk, line, and USERDEF data. Run these reports daily.

Note: Each system has its own set of threshold items and values. The Tandem application monograph, *Capacity Planning for Tandem Computer Systems*, explains the criteria for deriving these threshold values.

Figure 9

```
?DICTIONARY
?VOLUME CPLNWEEK                    ! Threshold report OBEY file.
DECLARE
        d-FROM-TIME as TIME "H2:M2" HEADING "TIME"
        ;
OPEN CPUSHFTN;                      ! Creates a table of average
                                    ! and maximum values, one set
LIST                                ! per day, if threshold values
                                    ! are exceeded in the WHERE
                                    ! statement at the end of the
BY CPU-NUM                          ! OBEY file. Otherwise, no report
      AS I2 HEADING "CPU/#"         ! is generated.
d-FROM-TIME := MIN(TIMESTAMP-DATE(B-FROM-TIME))
          HEADING "TIME/OF/DAY"
QLEN-MAX-D
   AS F5.1 HEADING "MAX/CPU/QLEN"
QLEN-DELT := (QLEN-MAX - QLEN-MIN)
   AS F5.1 HEADING "DELT/CPU/QLEN"
SWAP-MAX-D
   AS F5.1 HEADING "MAX/CPU/SWAP"
BUSY-MAX-D
   AS F5.1 HEADING "MAX/CPU/BUSY"
BUSY-DELT := (BUSY-MAX - BUSY-MIN)
   AS F5.1 HEADING "DELT/CPU/BUSY"
IBUSY-MAX-D
   AS F5.1 HEADING "MAX/INTR/BUSY"
QLEN-AVG-D
   AS F5.1 HEADING "AVG/CPU/QLEN"
SWAP-AVG-D
   AS F5.1 HEADING "AVG/CPU/SWAP"
BUSY-AVG-D
   AS F5.1 HEADING "AVG/CPU/BUSY"
IBUSY-AVG-D
   AS F5.1 HEADING "AVG/INTR/BUSY"
Where
  (qlen-max-d > 2)    OR
  (qlen-delt > 1)     OR
  (swap-max-d > 2)    OR
  (busy-max-d > 70)   OR
  (busy-delt > 20)    OR
  (ibusy-max-d > 25) OR
TITLE
  "\NEW" SKIP 1 "                THRESHOLD REPORT"
         SKIP 1 " SYSTEM:        " SYSTEM
         SKIP 1 " DATE:          " FROM^TIME AS DATE *
         SKIP 1 " SHIFT " SHIFT
         SKIP 1 " OS VERSION:" OS-VERSION
  ;
```

Figure 9.

An OBEY file that produces the threshold report in Figure 1. It isolates maximum queue lengths, CPU BUSY rates, and other rates above acceptable limits. Use similar queries for process, disk, line, and USERDEF data.

References

ENFORM User's Guide. Part no. 82349 B00. Tandem Computers Incorporated.

MEASURE Reference Manual. Part no. 82441 A00. Tandem Computers Incorporated.

MEASURE User's Guide. Part no. 82440 A00. Tandem Computers Incorporated.

Watson, L., and Madsen, K. 1984. *Capacity Planning for Tandem Computer Systems*. Tandem Application Monograph Series. Part no. 83904. Tandem Computers Incorporated.

Acknowledgments

The author wishes to thank the entire Performance Support Group, Peter Oleinick, and Susan Uren for their valuable suggestions for the article. He would also like to thank Joyce Lamkin for her assistance with the ENFORM queries and MEASURE OBEY files.

Michael L. King is an analyst in the Performance Support Group. Before this he was a performance specialist in Tandem's Northwest Region for four years. He joined Tandem in 1979 as an analyst in the corporate Software Education Group.

Performance Considerations for Application Processes

This article is intended to help analysts investigate the performance of a production system. Many of the examples are derived from customer application field experience. Often the implementation of the system is different from the design. (Not all programmers follow the spec.) It is also not unusual for the person charged with analyzing system performance to lack detailed knowledge of the application. This is especially true if the customer is using software written by a third party. This article shows some of the techniques used on customer sites to analyze the system design and to highlight performance problems in the system implementation.

System tuning does not stop at load balancing, PATHWAY tuning, and CACHE optimization. MEASURE provides a great deal of information about application design that is not obvious unless the relationship between the entities is understood. This article presents several ways of using performance data collected by MEASURE to identify programs that are placing excessive demands on hardware resources. Guidelines are also included to assist the analyst in determining whether the problem is design-related or caused by inefficient programming.

The first phase of a performance and tuning study is usually an attempt to balance hardware usage and to tune software products (such as PATHWAY and GUARDIAN 90). The *MEASURE User's Guide* and the *PATHWAY System Management Reference Manual* describe load balancing and general tuning procedures. Specific ways of tuning the PATHWAY terminal control process (TCP) are also documented in previous issues of the *Tandem Journal* (Wong, 1984) and the *Tandem Systems Review* (Vatz, 1985).

If the response time at the terminal or the elapsed time for batch jobs are still unacceptable after basic tuning, further analysis is needed. It is often useful to identify the reason that the user is unhappy with the system performance. The nature of the complaint indicates both the type of problem and which

aspects of the system warrant further investigation. For example, problems with the "average" response time for all transaction types may well be design-related or caused by overly complex transactions. If the response time for specific transactions is causing concern, the analysis could be focused onto just those transactions. "Hiccups" in the system may be caused by batch work or unusual peak workloads.

Data Collection

To analyze software performance, the data collected must be complete enough to characterize workload patterns and to identify heavily used (or resource-hungry) programs. Generally, this will include data for all disk files and processes. MEASURE writes to the data file at the beginning and end of the measurement and at intervals specified when the measurement is started. The frequency for writing the data is known as the "collection interval." The collection interval for the type of analysis discussed in this article should be short enough to show the workload profile without placing an undue demand on disk resources. A 30-minute collection interval will usually meet these requirements.

In practice, it is not always possible to collect enough data in a single measurement because of restrictions in the amount of counter space available and the amount of disk space required to store the data file. In this case, several measurements must be taken, each concentrating on a specific aspect of the application. For example, to identify the peak time periods, only hardware devices (CPUs and disks) need be measured. Software entities such as files and processes need only be measured for a relatively short time period (one or two hours). For the reports shown in this article it is not necessary to specify a collection interval for these entities.

MEASCOM can be used to output the collected data to a set of structured files which can be analyzed using ENFORM. Besides allowing far greater flexibility in the report formats, structured files also allow the analyst to build a historical data base for capacity planning purposes.

Figure 1

```
DECLARE CPU-BUSY INTERNAL F6.6;
OPEN PROCESS;
LIST BY PROGRAM-FILE-NAME

COUNT(PROGRAM-FILE-NAME OVER PROGRAM-FILE-NAME)  HEADING
"COUNT"
AS "M<ZZZ>"

CPU-BUSY : = SUM ((CPU-BUSY-TIME) OVER PROGRAM-FILE-NAME)  NOPRINT
CPU-BUSY: = (CPU-BUSY/1000000)
AS "M<Z,ZZ9.99>" HEADING "CPU/SEC"

SUM (MESSAGES-RECEIVED OVER PROGRAM-FILE-NAME )
AS "M<ZZZ,ZZ9>" HEADING "MESSAGES/RECEIVED"

SUM (MESSAGES-SENT OVER PROGRAM-FILE-NAME )
AS "M<ZZZ,ZZ9>" HEADING "MESSAGES/SENT"

TITLE "PROCESS SUMMARY REPORT";
CLOSE PROCESS;
PROCESS SUMMARY REPORT
```

PROGRAM-FILE-NAME	COUNT	CPU SEC	MESSAGES RECEIVED	MESSAGES SENT
$HENLEY DEVOVJ GL001A	6	21.63	136	3,844
$MARLOW AAPOBJ PAGEOBJ	2	1.01	25	134
$MARLOW AAPOBJ PITMOBJ	2	1.82	90	240
$SYSTEM PRODOBJ GL50OC	6	6.07	0	666
$SYSTEM OPERATE D630OBJ	1	34.55	2,518	2,970
$TEMPLE TRANSFERTISERV	11	79.61	930	7,681
$TEMPLE TRANSFERTRECV	4	4.08	36	262
$TEMPLE TRANSFERTSCHED	2	29.29	2,615	3,310
$TEMPLE TRANSFERTWORK	4	9.52	83	1,090
$TEMPLE TRANSFERWMSERV	2	7.34	385	285

Reports on Process Data

ENFORM can be used to produce summary reports of the data collected for processes and to calculate the number of FILE I/Os and the number of CPU seconds used by individual processes for each transaction.

The Process Summary report shown in Figure 1 contains one line for each program object file measured. The COUNT column shows how many times the program was started. The last three columns show the totals of CPU-BUSY-TIME, MESSAGES-SENT, and MESSAGES-RECEIVED for the program.

Figure 1.
The Process Summary report. The number of times that the program was started is given in the column headed COUNT. The other columns show totals of CPU seconds, messages received over $RECEIVE and messages sent via the file system.

Figure 2

```
DECLARE RUN-TIME INTERNAL F6.6;
DECLARE CPU-PER-SEC INTERNAL F6.2;
DECLARE IO-PER-SEC INTERNAL F6.2;
OPEN PROCESS;
LIST BY DESC CPU-BUSY-TIME NOPRINT
PROGRAM-FILE-NAME
RUN-TIME: = (DELTA-TIME/1000000)
AS "M<Z,ZZ9.99>" HEADING "RUN TIME/(SECONDS)"
CPU-PER-SEC: = (CPU-BUSY-TIME/DELTA-TIME)
AS "M<9.99>" HEADING "CPU/UTILIZATION"
IO-PER-SEC: = (MESSAGES-SENT/DELTA-TIME)
AS "M<999>" HEADING "I.O. PER SECOND"
PRIORITY
WHERE CPU-BUSY-TIME > 60000000 PRIORITY < 200
TITLE "BUSY PROCESS REPORT";
CLOSE PROCESS;
BUSY PROCESS REPORT
```

PROGRAM-FILE-NAME		RUN TIME (SECONDS)	CPU UTILIZATION	I.O. PER SECOND	PRIORITY
$SYSTEM SYSTEM	QP	1,197.49	0.29	32	190
$SYSTEM PRODOBJ	BATCH01	1,202.32	0.19	24	185
$SYSTEM PRODOBJ	ACTREPT	1,197.49	0.16	06	185
$SYSTEM SYS30	BACKUP	1,200.48	0.08	12	131
$SYSTEM SYS30	FUP	1,207.82	0.06	03	148
$OXFORD CNSRUND	GLUT140A	361.25	0.20	21	149
$SYSTEM SYSTEM	QP	279.12	0.22	25	139

Figure 2.
The Busy Process report. This report shows the resource consumption and process priorities of processes that consumed more than 600 seconds of CPU time during the measurement. The report is sorted in descending order of CPU-BUSY-TIME in seconds. The values printed for CPU UTILIZATION and I.O. PER SECOND are resource consumption per second.

Figure 2 shows the Busy Process report. It only includes entries for processes using over 60 seconds of CPU time and is sorted in descending order of CPU-BUSY-TIME (i.e., CPU utilization in seconds). RUN TIME shows the program's duration (in seconds). CPU UTILIZATION and I/O per second show the resource consumption per second of the process. PRIORITY is self-explanatory.

The Process Activity report (see Figure 3) shows the number of I/Os and CPU seconds consumed by a process for each message read over $RECEIVE. For an on-line transaction processing (OLTP) server process, MESSAGES-RECEIVED will often be equal to the number of transactions that it has processed. MESSAGES-SENT is the number of messages sent, and will usually be equal to the number of file I/Os initiated by the process.

The FILE report of MEASCOM can show files being accessed by a particular process (see Figure 4 for syntax). If the MEASURE command "ADD FILE *" was used to start the measurement, the reports will be printed for all GUARDIAN 90 files including $RECEIVE, terminals, and processes. This data can be written to structured files, and the ENFORM query shown in Figure 5 can be used to produce a summary of the output. (This is a very expensive ENFORM query and is, therefore, best used with a small amount of data.)

These four reports provide a very detailed view of the total resource usage of any program or process. Note that the Summary report should be run against a file containing data on all of the processes that were active during a measurement period. The Activity report should concentrate on application processes, and the FILE report should be printed only for selected application processes requiring further analysis.

The Process Summary Report

The column titled COUNT on the Process Summary report (refer to Figure 1) shows the total number of times that a process based on the program object file was started. If possible, process creations should be avoided in an on-line system. Process creation is very expensive relative to the cost of processing a transaction. A high value in the count field is an indication of dynamic process creations.

The last three columns of this report (CPU SEC, MESSAGES-RECEIVED, MESSAGES-SENT) show the total resource usage of the program. These figures help identify where detailed investigation is likely to yield the most benefit. A small improvement in resource utilization of a busy process will give a large overall improvement, whereas a large improvement in a seldom-used program is unlikely to show any noticeable change.

The Busy Process Report

The Busy Process report shows the individual processes that use the most CPU time, and is most effective used in conjunction with the Process Summary report. The CPU UTILIZATION is the same value as would be obtained from MEASCOM if the report was produced with RATE ON. However, the report is sorted as if RATE OFF were set.

The value of this report is that it indicates large processes run at the wrong priority and processes that are "resource hogs." For example, in the report shown in Figure 2, the QP process at the top of the list, was running at a very high priority; it used 29% of the processor and performed 32 file I/Os per second. If this process were running alongside OLTP work, it would be likely to have a detrimental effect upon the response time at the terminal.

The Process Activity Report

The Process Activity report is most valuable for OLTP servers. Usually the number of transactions that a server has processed can be equated to its MESSAGES-RECEIVED counter. Assuming this to be true, this report shows the I/O and CPU consumption per transaction of server processes.

One factor influencing response time at the terminal is the time that the server processes take to service each request. If queuing has been eliminated from the configuration and resources are not overloaded, the service time is a function of the number of file I/Os that the server must perform. For an OLTP system, each transaction will typically need to perform between 5 and 20 file I/Os. If the values calculated are much greater than this, the actual I/O performed by the process should be checked against expected I/Os used for the sizing of the application or shown in the design documentation. For transactions requiring a very large number of I/Os, it may be necessary to revise the response-time expectations for that transaction or redesign the data base to reduce the number of disk I/Os required.

Figure 3

```
DECLARE CPU-PER-IO INTERNAL F6.6;
DECLARE CPU-PER-TRANS INTERNAL F6.6;
DECLARE IO-PER-TRANS INTERNAL F6.6;
OPEN PROCESS;
LIST BY DESC CPU-BUSY-TIME NOPRINT
PROGRAM-FILE-NAME

CPU-PER-TRANS: = (CPU-BUSY-TIME/MESSAGES-RECEIVED/1000000)
AS "M<ZZ9.9999>" HEADING "CPU PER/TRANS"

IO-PER-TRANS: = (MESSAGES-SENT/MESSAGES-RECEIVED)
AS "M<ZZZZZ9.9>" HEADING "I.O. PER/TRANS"

CPU-PER-IO: = (CPU-BUSY-TIME/MESSAGES-SENT/1000000)
AS "M<9.9999>" HEADING "CPU PER/I.O."

WHERE MESSAGES-RECEIVED > 0 AND MESSAGES-SENT > 0
TITLE "PROCESS ACTIVITY BY PROGRAM";
CLOSE PROCESS;
```

PROCESS ACTIVITY BY PROGRAM

PROGRAM-FILE-NAME	CPU PER TRANS	I/O PER TRANS	CPU PER I/O
$SYSTEM PRODOBJ GL10A	0.0924	8.7	0.0107
$SYSTEM PRODOBJ GL10B	0.0320	3.5	0.0093
$SYSTEM PRODOBJ PO15A	0.0241	2.9	0.0084
$SYSTEM PRODOBJ GL07C	0.0914	13.2	0.0069
$SYSTEM PRODOBJ PO15D	0.0736	8.2	0.0090

Figure 3.

Process Activity report. For an OLTP server the CPU-BUSY-TIME divided by MESSAGES-RECEIVED will be equal to the CPU consumption per transaction and MESSAGES-SENT divided by MESSAGES-RECEIVED will be equal to the number of file I/Os per transaction.

Figure 4

```
SET REPORT FORMAT STRUCTURED
LIST FILE * (cpu,pin),FROM hh:mm,TO hh:mm
```

Where cpu,pin is the CPU and PIN of the relevant process, and hh:mm give the start and stop times of the process.

Figure 4.

MEASCOM syntax to write file data for a specific process to a structured file.

Figure 5.
Detailed reports based on data extracted from the command in Figure 4. This report shows in great detail the resource consumption of a process.

Figure 5

```
DECLARE CPU-BUSY
INTERNAL F6.6; OPEN FILE, PROCESS

LIST BY PROCESS.CPU-NUM NOPRINT
     BY PROCESS.PIN NOPRINT
     BY FILE.OPENER-CPU NOPRINT
     BY FILE.OPENER-PIN NOPRINT
     BY PROCESS.FROM-TIMESTAMP NOPRINT
     BY FILE.FROM-TIMESTAMP NOPRINT

CPU-BUSY: = (CPU-BUSY-TIME/1000000) NOPRINT

FILE-NAME
READS SUBTOTAL OVER PROCESS.FROM-TIMESTAMP
WRITES SUBTOTAL OVER PROCESS.FROM-TIMESTAMP
DELETES-OR-WRITEREADS HEADING "DELETE OR/WRITEREAD"
                      SUBTOTAL OVER PROCESS.FROM-TIMESTAMP
INFO-CALLS HEADING "INFO/CALLS" SUBTOTAL OVER
PROCESS.FROM-TIMESTAMP

WHERE DEVICE-TYPE = 3
AND PROCESS.CPU-NUM = FILE.OPENER-CPU
AND PROCESS.PIN     = FILE.OPENER-PIN
AND PROCESS.FROM-TIMESTAMP < FILE.FROM-TIMESTAMP
AND PROCESS.TO-TIMESTAMP > FILE.TO-TIMESTAMP

AFTER CHANGE ON PROCESS.FROM-TIMESTAMP PRINT FORM SKIP
PROCESS-NAME
" CPU" PROCESS.CPU-NUM
" PIN"     PROCESS.PIN
SKIP
" PROGRAM " PROGRAM-FILE-NAME
" FROM " TIMESTAMP-TIME(
  (((PROCESS.FROM-TIMESTAMP - 211024440000000000)/ 10000) * 65536)
                      ) AS TIME *
" TO " TIMESTAMP-TIME(
  (((PROCESS.TO-TIMESTAMP - 211024440000000000)/ 10000) * 65536)
                      ) AS TIME *
SKIP
" CPU-BUSY " CPU-BUSY
" SENT "     MESSAGES-SENT
" RECVD "    MESSAGES-RECEIVED
SKIP

TITLE "PROCESS DISC FILE ACTIVITY REPORT" ;
CLOSE FILE, PROCESS;
```

PROCESS DISC FILE ACTIVITY REPORT

FILE-NAME		READS	WRITES	DELETE OR WRITEREAD	INFO CALLS
$Y186 CPU 0 PIN 37					
PROGRAM $MARLOW SWFSYS SVR05OBJ FROM 09:25:59				AM TO 09:53:37	AM
CPU-BUSY .495949 SENT 82 RECVD 2					
$MARLOW SWF	FILECOMP	7	0	0	2
$MARLOW SWF	FILESTUD	7	0	0	2
$MARLOW SWF	FILEXREF	0	0	0	2
$MARLOW SWF	FILECOUR	21	7	0	2
$MARLOW SWF	FILECLAS	1	0	0	2
$MARLOW SWF	FILEENRL	1	1	0	2
$MARLOW SWF	FILELODG	0	7	0	2
$MARLOW SWF	FILEACOM	0	7	0	2
		40	22	0	16

PROCESS DISC FILE ACTIVITY REPORT

FILE-NAME		READS	WRITES	DELETE OR WRITEREAD	INFO CALLS
$Y178 CPU 0 PIN 58					
PROGRAM $MARLOW SWFSYS SVR01OBJ FROM 09:20:01				AM TO 09:53:37	AM
CPU-BUSY 2.387662 SENT 440 RECVD 21					
$MARLOW SWF	FILECOMP	22	0	0	2
$MARLOW SWF	FILESTUD	26	0	0	2
$MARLOW SWF	FILEXREF	0	0	0	2
$MARLOW SWF	FILECOUR	84	0	0	2
$MARLOW SWF	FILECLAS	93	0	0	2
$MARLOW SWF	FILEENRL	44	0	0	2
$MARLOW SWF	FILELODG	0	0	0	2
$MARLOW SWF	FILEACOM	0	0	0	2
		269	0	0	16

Servers and the CPU. Typically, OLTP servers are I/O-intensive and do not consume a large number of CPU cycles per transaction; therefore, the CPU consumption of a server usually relates directly to the number of file I/Os that it performs. This consumption reflects the file system cost to set up the disk request, and does not include the CPU costs for the disk process, TMF, or the interrupt handlers. These items are the larger consumers of CPU resources. If the Process Activity report shows any server to have a profile that is different from the other servers in the system, an investigation of the application code should be made to ensure that it is written in an efficient manner.

The Disk Subsystem

An I/O request to a disk file can be viewed in two ways:

- As logical I/O (requests to the file system from a program).
- As physical I/O (reads or writes of data and index blocks to disk).

The FILE entity records the I/O operations performed by a user process on an explicitly opened file (logical file access). The DISCOPEN entity measures the I/O operations performed by the disk process on a specified file (physical file access). A logical I/O request can be satisfied from the "process file segment," in which case no counters will be incremented on the DISCOPEN entity and no physical I/O will take place. Alternatively, the I/O will be handled by the disk process and recorded in the DRIVER-INPUT-CALLS and DRIVER-OUTPUT-CALLS counters of the DISCOPEN entity. A proportion of these driver calls will be satisfied from cache and will also be recorded in the CACHE-HITS and CACHE-WRITE-HITS counters.

The File Summary report (Figure 6) is an ENFORM report summarizing activity on a subset of disk files opened during the measurement period. This report would usually be run against a file containing entries for all of the files accessed during the measurement, identifying any files which are being opened dynamically by processes, and highlighting the busiest files on the system. A similar report can be run against data collected for the DISCOPEN entity. (See Figure 7 on the following page.)

Figure 6

```
OPEN FILE;
LIST BY FILE-NAME
COUNT(FILE-NAME OVER FILE-NAME) HEADING "COUNT"
AS "M<ZZZ>"

SUM (READS OVER FILE-NAME) HEADING "TOTAL/READS"
AS "M<ZZZ,ZZ9>" TOTAL

SUM (WRITES OVER FILE-NAME) HEADING "TOTAL/WRITES"
AS "M<ZZZ,ZZ9>"

SUM (UPDATES-OR-REPLIES OVER FILE-NAME) HEADING "/UPDATES"
AS "M<ZZZ,ZZ9>"

SUM (DELETES-OR-WRITEREADS OVER FILE-NAME) HEADING "/DELETES"
AS "M<ZZZ,ZZ9>"

SUM (INFO-CALLS OVER FILE-NAME) HEADING "INFO/CALLS"
AS "M<ZZZ,ZZ9>"

WHERE file-name NOT contains "#" AND
(READS + WRITES + UPDATES-OR-REPLIES +
DELETES-OR-WRITEREADS + INFO-CALLS) > 0
AND DEVICE-TYPE = 3
TITLE "DISC FILE SUMMARY REPORT";
CLOSE FILE;
```

DISC FILE SUMMARY REPORT

FILE-NAME	COUNT	TOTAL READS	TOTAL WRITES	TOTAL UPDATES	TOTAL DELETES	INFO CALLS
$TEMPLE TRANDBA DISTLIST	1	1	0	0	0	
$TEMPLE TRANDBA FOLDER	1	274	115	0	79	
$TEMPLE TRANDBA IFOLDER	1	1,305	147	0	79	
$TEMPLE TRANDBA ITEMDATA	1	772	2	4	0	
$TEMPLE TRANDBA ITEMDESC	1	1,252	3	4	0	
$TEMPLE TRANDBA PROFILE	1	18	0	0	0	
$TEMPLE TRANDBA READY	1	0	4	0	0	
$TEMPLE TRANDBA RECIP	1	59	2	38	0	
$TEMPLE TRANDBA SESSION	1	1,238	3	1	0	

Figure 6.

File Summary report. The number of times that a file was opened is given by the COUNT column; the other columns show logical I/Os. Only files explicitly opened by a process are included in this report. Alternate-key files and secondary partitions are shown in the DISCOPEN Summary report (Figure 7).

Figure 7

```
OPEN DISCOPEN;
LIST BY FILE-NAME
COUNT(FILE-NAME OVER FILE-NAME) HEADING "COUNT"
AS "M<ZZZ>"
SUM (DRIVER-INPUT-CALLS OVER FILE-NAME)
HEADING "TOTAL/INPUT" AS "M<ZZZ,ZZ9>"
SUM (CACHE-HITS OVER FILE-NAME)
HEADING "CACHE/HITS" AS "M<ZZZ,ZZ9>"
SUM (DRIVER-OUTPUT-CALLS OVER FILE-NAME)
HEADING "TOTAL/WRITES" AS "M<ZZZ,ZZ9>"
SUM (BLOCK-SPLITS OVER FILE-NAME)
HEADING "TOTAL/BLOCK/SPLITS" AS "M<ZZ,ZZ9>"
SUM (REQUESTS-BLOCKED OVER FILE-NAME)
HEADING "TOTAL/STALLS" AS "M<ZZ,ZZ9>"
WHERE FILE-NAME NOT CONTAINS "#" AND
(DRIVER-INPUT-CALLS + DRIVER-OUTPUT-CALLS) > 0
TITLE "DISCOPEN SUMMARY REPORT";
CLOSE DISCOPEN;
```

DISCOPEN SUMMARY REPORT

FILE-NAME	COUNT	TOTAL INPUT	CACHE HITS	TOTAL WRITES	TOTAL BLOCK SPLITS	TOTAL STALLS
$MAIL TRANDBA ITEMDATA	1	1,788	1,348	0	0	0
$MAIL TRANDBA ITEMDESC	1	1,748	1,468	1	0	1
$MAIL TRANDBA RECIP	1	207	90	28	0	0
$TEMPLE TRANDBA DISTLIST	1	3	0	0	0	0
$TEMPLE TRANDBA FOLDER	1	1,324	1,249	210	4	1
$TEMPLE TRANDBA IFOLDER	1	4,163	3,881	194	0	0
$TEMPLE TRANDBA ITEMDATA	1	610	491	10	1	0
$TEMPLE TRANDBA ITEMDESC	1	2,030	1,844	6	0	0
$TEMPLE TRANDBA PROFILE	1	36	30	0	0	0
$TEMPLE TRANDBA READY	1	8	7	4	0	0
$TEMPLE TRANDBA READY0	1	8	7	4	0	0
$TEMPLE TRANDBA RECIP	1	62	35	12	0	0
$TEMPLE TRANDBA SESSION	1	3,727	3,681	8	1	0

Figure 7.
DISCOPEN Summary report. The DISCOPEN entity collects data on the physical I/O required to access a file including indexes, alternate-key files, and secondary partitions.

The relationship between the FILE and DISCOPEN entities is illustrated by the FILE Summary and DISCOPEN Summary reports. The files $TEMPLE.TRANDBA.ITEMDATA, ITEMDESC, and RECIP are all partitioned files with secondary partitions on $MAIL. The file $TEMPLE.TRANDBA.READY has an alternate-key file on the same subvolume called READY0. Alternate-key files and secondary partitions are not directly opened by the program, therefore they only appear on the DISCOPEN report. Note also that the figures for physical I/Os (DISCOPEN report) are greater than those for logical I/O (FILE report). This occurs because several physical I/Os are required to satisfy one logical I/O to a key-sequenced file.

As with dynamic process creations, dynamic file opens are an undesirable overhead. A program is usually designed to open all its files when it starts up, and to close them all when it terminates. However a program can be coded (either deliberately or accidentally) to open and close one or more files for each transaction. Programs which do this are likely to have a long response time and will waste system resources.

Files that are opened dynamically can be located using the COUNT column. To establish which program is responsible for the dynamic file opens, get the CPU/PIN of the openers from the FILE or DISCOPEN entity using either MEASCOM or ENFORM and locate the process or processes in the PROCESS entity.

Block Splits

The position of a new record to be inserted into a key-sequenced file is determined by the value of its primary key field. If the block where the new record is to be inserted into the file is full, a block split occurs. This means that the disk process allocates a new data block, moves part of the data from the old block into the new block, and gives the index block a pointer to the new data block. Block splits are reported on the DISCOPEN SUMMARY report in the column headed "BLOCK SPLITS." The column headed "WRITES" on the FILE SUMMARY report shows the number of new records inserted into a key-sequenced file, therefore, the higher the ratio of WRITES to BLOCK SPLITS the better. If the blocks are very full (more than 70%) or if the blocks only contain one or two records, block splits will be frequent.

If the block split rate is unacceptably high, it may be possible to decrease the rate by increasing the amount of free space, or slack, in each block. This is done using FUP LOAD with SLACK specified, or by changing file block size and reloading the file. If the volume containing the file has insufficient free space to allow this, the file will need to be partitioned over several volumes.

Occasionally, records are so large that only two or so can fit in a block. In this case it may be appropriate to just configure one record per block and partition the file. At the expense of disk space and some read degradation, this will get many disk processes involved for servicing the file, and will keep index levels down for key-sequenced structures.

FILEINFO Calls

The GUARDIAN FILEINFO call provides error and characteristic information about a file. In addition to obtaining the error code after a failed I/O, it can collect information such as the time of the last update or the end-of-file location.

Some FILEINFO parameters require a call to the disk process while others do not. This depends primarily on whether the target entity is in the file label/file control block (FCB) or in the access control block (ACB). If it is in the ACB, it is simply fetched from the process file segment. If it is in the label/FCB, a request must be issued to the disk process to service it.

The same holds true for the FILERECINFO call. The requests for those entities that are in the FCB (usually EOF or file name) appear in the report. A call to FILEINFO to fetch a GUARDIAN error code associated with an I/O operation would not entail going to the disk process since that information is retained in the process file segment.

A call to FILEINFO handled by the disk process will have a cost in CPU cycles similar to other disk file I/Os.

Design and Operational Hints

This section is essentially a checklist of some problems discovered using data collected by XRAY on customer applications. For the purpose of this article XRAY data is considered identical to that collected by MEASURE. In most cases the application programs will appear functionally correct to the user but will consume excessive resources or give a poor response time because of the way they are written.

Variable Length Transactions

To the majority of users, a predictably long response time for some transactions is preferable to one which is very variable. Unfortunately some server programs need to perform a variable number of I/Os for different transaction types. End-user response times will be more predictable if functions with very similar service times are grouped into server classes. This reduces the variability of response time and helps to avoid a potential performance problem. However, these servers are not easily identifiable from measurement data.

Figure 8

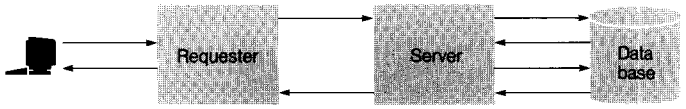


Figure 8.
Normal transaction path. Typically a transaction will only require one message to be sent from the requester to a particular server.

The problem arises when functions with long service times are mixed in the same server with functions that have short service times. Such a mapping of requesters to servers would lead to a system with erratic service times. For example, a request requiring many file accesses and known to have a mean service time of many seconds could begin just prior to another request for a single-record update, which will require less than one second. The response time for the update could be one to many seconds, depending on the arrival sequence of transaction types.

It is possible to have transactions in which the same type of request results in a variable amount of I/O. Typically, data-base searches may result in a variable number of "hits."

If this happens, erratic response time can be avoided by scanning a fixed portion of the data base and returning a message to the user. The option then is to either stop scanning or to continue with the search.

Once a server with variable service time is coded, little can be done externally to improve response time. Defining more servers of this class will not guarantee predictable response times.

Multiple Sends to a Server

The usual transaction flow is shown in Figure 8. However, it is possible to code a requester to make multiple sends to the same server for a single transaction. This will not always be obvious from the measurement data, but a server with a low number of I/Os per transaction and a high receive rate should be regarded with suspicion, especially if users complain of a poor response time for this transaction type.

TABLE LOOKUP Routines

TABLE LOOKUP routines used to replace disk I/O can be a cause of excessive CPU consumption identified from the Process Activity report (refer to Figure 2). If the table is very large, a serial search will require far more CPU cycles than a binary search and, in some cases, more than the disk read it replaces. (Note that the COBOL74 SEARCH verbs initiate a serial search. However, COBOL85 will perform a binary search for SEARCH ALL.) A high CPU consumption is not necessarily a sign of poor design or programming practices. The performance advantages gained by holding and manipulating data in memory often far outweigh any costs incurred for programming or extra memory.

SORT

The SORT utility is normally used during batch processing, although it can be used by OLTP servers. If SORT is being used by on-line work, question the data-base design and, if possible, introduce an *alternate key* to make the sort unnecessary.

Running batch programs during on-line hours is often unavoidable.

For a small number of records the TAL procedure provided by Tandem, HEAPSORT, will sort the data in the application programs data stack. This is much more efficient than using the external sort process. If a program is required to perform individual sorts, the program can create a sort process once and keep reusing it, rather than creating and deleting a new process for every sort. See the *SORT and FASTSORT Manual* for details on how to achieve this. FASTSORT, which is significantly faster than SORT, is discussed in an accompanying article by Jim Gray, et al., "FASTSORT: An External Sort Using Parallel Processing."

PATHMON

In a stable PATHWAY environment, the PATHMON process rarely becomes active. If the PATHMON process is busy (check the CPU-BUSY-TIME counter of the Process Summary report), verify that dynamic servers are not being excessively used, PATHWAY STATS are not being printed too often, or that some other operation using PATHMON is not being done repeatedly. On one customer's system, PATHMON was using 50% of a processor because of a background process that continuously requested SERVER STATUS information to check for PENDING SERVERS.

Continuously Executing Processes

Most processes in an OLTP environment are *event-driven*. This means that they wait for work by reading from $RECEIVE or from a terminal. However, a process may be written so that it actively looks for work. Some examples of these are:

- A queue manager process that repeatedly reads a disk file, until a particular record is present.
- A communications process, which continuously "polls" devices.
- A process which constantly sends requests to system processes (such as the spooler, PATHMON, or a communications process) asking for status information.

Such processes will seriously bias performance measurements in that they will consume all spare CPU and device resources. This can so distort an XRAY or MEASURE session that tuning cures are not evident. If the processes are executing at a high priority, on-line response time will be adversely affected.

A system with a continuously executing process is usually not hard to spot, and the remedy is often quite simple. The symptoms are a high CPU utilization or a high read rate on a disk with most reads satisfied from cache even at slack times. Often hardware utilization will increase as terminal-driven workload decreases. The process causing the problem may not have a high CPU-BUSY-TIME figure, but it may have a very high value for MESSAGES-SENT.

The Spooler

The spooler housekeeping disk I/O associated with starting, printing, and purging a spooler job normally amounts to over 20 disk I/Os. These can be minimized by using level 3 spooling and by always specifying report names and printer locations.

For large jobs, such as compiles or ENFORM queries, this overhead is inconsequential, but it accounts for a high proportion of the work associated with collecting and printing a small job.

Level 3 spooling allows direct I/O to the spooler data file in a mechanism analogous to buffered writes. The interprocess communication to the collector and the associated dispatches are also saved.

If no report name is specified, the spooler will obtain the name of the report owner from the USERID file. This requires a dynamic open of the file.

In extreme cases, it may be preferable to write individualized routines to control the printers or to install low-cost printers attached to the screens. It is also possible to configure several spooler collector processes with different priorities to handle different workloads; i.e., use a high-priority process for OLTP and a lower priority process for batch work.

Interactive and Batch Programs

Most interactive programs will not have a detrimental effect on the response time of transaction processing work, provided that they execute at the appropriate priorities. However, some operations such as an ENFORM request, PUP LISTFREE, BACKUP, RESTORE, FUP LOAD/DUP/COPY, FUP INFO STAT, and some Tandem Application Command Language (TACL™) programs can be resource-intensive and should only be allowed to execute when resources are available. This means that they should execute at a low priority and if possible outside of the peak periods.

The Busy Process report (refer to Figure 2) highlights the programs which use the most CPU time. The report shows resource consumption in terms of CPU utilization and file I/Os per second of all processes which have used more than five minutes of CPU time during the measurement period. The report excludes most system processes by selecting processes with a priority of less than 200.

Running batch programs during on-line hours, although undesirable for performance reasons, is often unavoidable. A batch job running at a low priority will be using high-priority resources (such as disk processes). This will have an effect on processes running on the same CPU. Batch jobs started from a terminal with a high-priority COMINT or TACL will have a high priority unless the priority is specified, a $CMON process overrides GUARDIAN 90. In many installations at least one terminal (normally in the computer room) will have a high-priority command interpreter. This terminal should not be used for running batch or heavy interactive work.

If a batch job uses data from only one disk, its effect on the rest of the system can be minimized by running it in the same processor as the primary CPU of the disk it is using. It is not normally advisable to dedicate a CPU to batch or development work unless other resources such as disks and the spooler are running in same CPU.

If the batch job uses TMF, consider the number of TMF transactions within the job. If possible, several file updates should be batched within the same TMF transaction. Attention should also be given to the number of record locks that will be held concurrently in each DISCPROCESS.

Sequential Block Buffering

Sequential block buffering (SBB) improves the efficiency of reading structured files sequentially by allowing the record-deblocking buffer to be in the application process file segment. This eliminates the request to the disk process to retrieve each record in the block. Instead, a request retrieves an entire block of records. If a file is being read using SBB, the MESSAGES-SENT counter of the PROCESS entity and the DRIVER-INPUT-CALLS counter of the DISC-OPEN entity will only be incremented for each block. However, the READS counter of the file entity will be incremented for each record read.

Table 1.
Some trouble-shooting guidelines.

Symptoms	Where to look	Problem	Possible solution(s)
Excessive process creations	Process Summary report	Programming bug, causing process to terminate	Correct error
		"Dynamic servers" being overused	Change PATHWAY configuration to allow additional static servers
		Process starting new processes	If possible, modify program to keep the new process active
Large number of file I/Os per transaction	Process Activity report	Complex transaction	Educate users to expect a long response time or redesign the program
		"Batch" type enquiry	Break transaction down into smaller units
		Programming error	Check the actual number of I/Os against the expected number of I/Os and, if different, correct the program
High CPU consumption per file I/O	Process Activity report	Serial scan of large tables	Modify program to use a binary search
		Sensible use "in memory" tables	None needed
Low number of I/Os per transaction	Process Activity report	Small transaction	None
		Requester sending many messages per transaction	Redesign requester program
High CPU consumption per transaction	Process Activity report	Serial scan of large tables	Modify the program to use a binary search
		Complex transaction	Educate users to expect a long response time or simplify the program
		Programming error	Correct program
High CPU utilization, high I/O per second and a high priority	Busy Process report	Batch job run at the wrong priority	Control priorities of batch work
High CPU utilization or high I/O per second	Busy Process report	Continuously executing processes	If possible, redesign the program to become "event-driven" or introduce a DELAY into the idle loop
Large number of file opens	File Summary report	Programming error	Correct program
		Dynamic process creations	Correct program
Large number of block splits	DISCOPEN Summary report	Blocks too full to allow inserts	Reload the file using FUP, specifying a value of at least 30% for "SLACK"

A potential problem with SBB is that it cannot detect record or key locks. If the file is opened for shared access, it is possible for other processes to update records in the block after it has been read into the process file segment. In this case, data in a block may not be absolutely up-to-date. This will not be a problem for most enquiry-type transactions because the data will be displayed almost instantly and could change at any time. However, if the data is to be updated, the program will need to be coded with this in mind.

If a file is read using an alternate key, the full benefits of SBB are not available. This is because only the alternate-key file can be read via SBB; the data file will be read randomly.

The performance and concurrency issues of SBB are discussed in detail in a previous issue of the *Tandem Systems Review* (Mattran, 1986).

Workload Patterns

Most OLTP systems are sized on known or planned transaction rates for the peak period. It is important to know if the transaction rates and workload patterns are consistent with those used to size the system. A report of the transactions from MEASURE data is not directly possible because MEASURE has no concept of a transaction. However, it is possible to obtain a report of the number of sends to a particular server, which in many cases will map onto the transaction rates.

Earlier in this article, a 30-minute collection interval was suggested. This allows the peak periods of the day to be identified and analyzed separately. It is not enough to measure just one day and expect that day to be representative. Measurements should be collected daily for several weeks (or even months) so that workload patterns and trends can be identified. The data collected can also be used to validate any sizings that were performed for the application. If a large discrepancy is found, further investigation may discover some programming errors that were not visible to conventional system testing. If the workload pattern is not as expected, there are two possibilities: the system is not being used as intended (users may benefit from more training), or the data provided for the sizing may be inaccurate.

Conclusion

Data collected by MEASURE can be used to analyze the performance and resource consumption of an application allowing the identification and correction of application-related bottlenecks, design flaws, and programmming errors. Table 1 contains some trouble-shooting guidelines.

A note of caution: There is a point in granular analysis where the returns will certainly be less than efforts expended to perform the analysis. Recognizing this point is important. MEASURE will provide a wealth of data. A risk, however, is that the appeal of analysis and collection may become an end in itself. Macroanalysis yields macro results. Microanalysis yields micro results.

References

Mattran, R. 1986. Buffering for Better Application Performance. *Tandem Systems Review*. Vol. 2, No. 1. Tandem Computers Incorporated.

MEASURE Reference Manual. Part no. 82441. Tandem Computers Incorporated.

MEASURE User's Guide. Part no. 82440. Tandem Computers Incorporated.

PATHWAY System Management Reference Manual. Part no. 82365. Tandem Computers Incorporated.

Vatz, J. 1985. The PATHWAY TCP: Performance and Tuning. *Tandem Systems Review*. Vol. 1, No. 1. Tandem Computers Incorporated.

Wong, R. 1984. Understanding PATHWAY Statistics. *Tandem Systems Review*. Vol. 2, No. 2. Tandem Computers Incorporated.

Acknowledgments

Thanks to Dick Thomas for his advice on the article structure and technical content, and to Art Sheehan, Scott Sitler, Wouter Senf, and Tuomo Stauffer for their technical input.

Ray Glasstone joined Tandem in April 1983. Since then he has analyzed the performance of many customer applications and taught performance and tuning courses to Tandem's staff and customers. Before joining Tandem, Ray was responsible for capacity planning and performance tuning of a large IBM CICS network.

Configuring Tandem Disk Subsystems

In most systems composed of Tandem's current hardware and software, mirrored disk volumes should be configured for parallel writes. When done correctly, this configuration ensures optimum performance and fault tolerance.

This article provides a practical guide to configuring disk subsystems. It first explains how various disk subsystems can be configured, with the aid of configuration diagrams and system generation (SYSGEN) configuration entries. Next, it presents some new approaches to fault tolerance. The examples used are becoming increasingly more common with Tandem's V8™ and XL8 disk drives. Finally, it discusses the results of a comprehensive set of on-line transaction processing (OLTP) tests and some new rules of thumb for configuring disks. The test results show the impact of various disk configurations on system performance.

To benefit fully from the discussion, readers should possess a general understanding of the following:

- Tandem system architecture.
- Access paths and how these paths are defined with SYSGEN.
- Tandem system performance.

Configuration Options

How does one best configure mirrored disk volumes on a Tandem system? The decision is not always straightforward.

Many NonStop 1+ systems were configured with two mirrored volumes per processor and disk controller pair. Usually, one mirrored volume was "primaried" in one processor through one controller and the other volume was primaried through the other processor and controller. Each volume was configured for serial writes, as they yielded optimum performance. The advent of the more powerful NonStop II, NonStop TXP, and NonStop VLX processors and Disc Process 2 (DP2) has since made this configuration unnecessary. Now, a parallel write, more fault-tolerant configuration yields the best performance and data integrity.

Also, while some systems have a dedicated controller pair per mirrored volume (making configuration choices fairly easy), more typically, systems have a controller pair supporting more than one mirrored volume. The architecture of the high-performance V8 and XL8 disk drives, for example, is such that a maximum of four controllers per disk cabinet can be connected. If the cabinet is equipped with eight spindles, the decision to configure for serial or parallel writes becomes significant.

Closely coupled with this decision is the determination of the correct ratio of disk volumes per processor. Will a processor be over-utilized if more than one disk volume is primaried in it? Is over-utilization a risk only on the NonStop II processor, or is it also a risk on the NonStop TXP processor? These issues are discussed in detail below.

Regardless of the application and system, users should configure their disk subsystems carefully. Some of the examples that follow illustrate that, in some instances, there is only one right way to configure the disk subsystem; in one example, however, the best way to configure is not obvious. Clearly, the most important aspect of disk configuration is the analysis of application and system characteristics that must be made before any kind of reconfiguration or upgrade is undertaken.

Software and Hardware Studied

Two disk configurations are emphasized in this study: those for serial writes and parallel writes. Their performance is compared on the following systems:

- The NonStop 1+ processor running Disc Process 1 (DP1).
- The NonStop II and NonStop TXP processors running DP2.[1]

All the examples assume that only those disks illustrated are configured and all volumes are mirrored.

Basic Configurations

There are four distinct access paths to each unit of a mirrored volume, and both units of a mirrored volume are controlled either by one disk process (DP1) or by a group of disk processes all in the same processor (DP2). This requires that both units of a mirrored volume be controlled by the same processor. Figure 1 illustrates all possible access paths to a mirrored volume.

In this simple example, there are two basic ways of configuring access to the mirrored volume: either serial or parallel writes. Figure 2 illustrates the serial write configuration. With this type of configuration, only one of the disk controllers (DISCA in this example) is used. When a mirrored volume is configured for serial writes, both halves of the

Figure 1

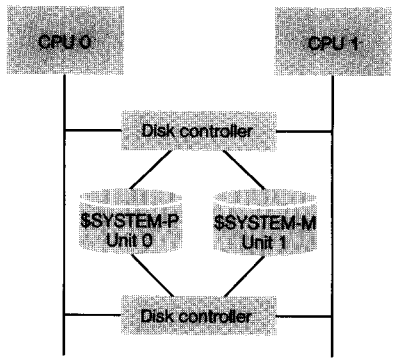
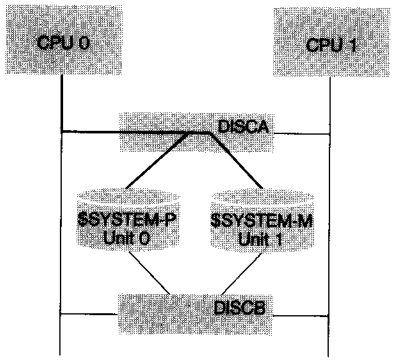


Figure 1.
Access paths to a mirrored volume.

Figure 2

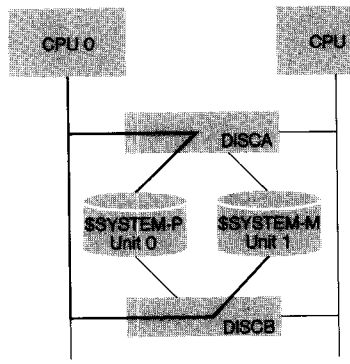



```
CONTROLLERS:  DISCA 3107 0,1   %10 CONTROLLER_DISC_PROCESS_2;
              DISCB 3107 0,1 %110 CONTROLLER_DISC_PROCESS_2;

PERIPHERALS:  $SYSTEM DISCA-DISCB.0,DISCA-DISCB.1 4110
              NUMDISCPROCESSES 3;


:PUP LISTDEV DISC
  4 $SYSTEM-P  *   00,007  %10   01,007  %10    3   8   4096   DP2
    $SYSTEM-B      00,007  %110  01,007  %110
    $SYSTEM-M  *   00,007  %11   01,007  %11
    $SYSTEM-MB     00,007  %111  01,007  %111
```

Figure 2.
Serial writes to a mirrored volume. This configuration is not recommended because it wastes a resource, provides less protection from hardware failures, and does not take advantage of the improved performance of parallel writes.

[1]Tests on DP1 were also performed on the NonStop II and NonStop TXP processors, although, for brevity, the results are not presented here. The system running DP2 clearly outperformed the identically configured system running DP1. The two systems exhibited nearly the same *relative* performance from one disk configuration to the other; thus, for disk configurations, what holds true for a system running DP2 generally holds true for a system running DP1.

Figure 3




```
CONTROLLERS:  DISCA 3107 0,1   %10 CONTROLLER_DISC_PROCESS_2;
              DISCB 3107 0,1 %110 CONTROLLER_DISC_PROCESS_2;

PERIPHERALS:  $SYSTEM DISCA-DISCB.0,DISCB-DISCA.1 4110
              NUMDISCPROCESSES 3;


:PUP LISTDEV DISC
  4 $SYSTEM-P  *  00,007  %10   01,007  %10    3   8   4096   DP2
    $SYSTEM-B     00,007  %110  01,007  %110
    $SYSTEM-M  *  00,007  %111  01,007  %111
    $SYSTEM-MB    00,007  %11   01,007  %11
```

Figure 3.
Parallel writes to a mirrored volume. This configuration is recommended for its full resource usage, improved fault tolerance, and performance advantages.

Figure 4

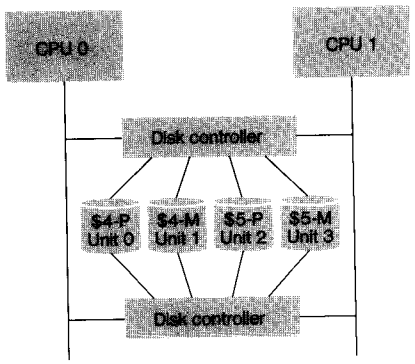


Figure 4.
A common disk configuration for two mirrored volumes.

mirror use the same controller for data transfers. (In this article, it is referred to as the *preferred controller*. DISCB is referred to as the *nonpreferred controller* and is not used unless a path switch is made to it.)

In Figure 3, DISCA is the preferred controller for unit 0 and DISCB is the preferred controller for unit 1. Because each half of the mirror is accessed via a separate controller, thus improving fault tolerance, this is the best way to configure a single mirrored volume connected to a pair of disk controllers.

Besides improving fault tolerance through the use of separate controllers, this configuration has performance advantages. Writes to mirrored volumes require two separate operations, i.e., a write operation for each of the two units. When data is written to a volume configured for serial writes, the first data transfer to one half of the mirror must complete before the second can begin. With the configuration in Figure 3, the data transfer for both writes starts (almost simultaneously) through each of the two controllers, thus the term *parallel write*. This method reduces the elapsed time required for both write operations by approximately the amount of time required to perform one data transfer.

With the introduction of DP2, parallel reads became possible. To take advantage of this new feature, each unit of a mirrored volume must be configured through a separate controller, in the same way parallel writes are configured. In fact, on a system running DP2, configuring for parallel writes automatically makes parallel reads possible. (While parallel writes are possible with either DP1 or DP2, parallel reads are possible only with DP2.)

It should now be clear why the mirrored volume in Figure 1 should be configured as in Figure 3. The configuration in Figure 2 is unacceptable because:

- Unless there is a path switch, controller DISCB will never be used, wasting a valuable system resource.
- Serial writes provide less protection from the consequences of hardware failures.
- Parallel writes are not possible.
- Parallel reads are not possible.

Two Mirrored Volumes

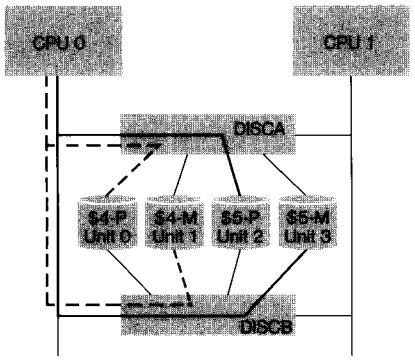
A configuration having more than one mirrored volume between a controller pair is more complicated. Figure 4 represents a configuration in which a pair of mirrored volumes, logical devices 4 and 5, share the same disk controller pair. The volumes could be configured for serial or parallel writes. Figure 5 illustrates the SYSGEN for parallel writes.

The configurations in Figures 3 and 5 are similar in that each half of the mirrored volume has each of its two units accessed via a different controller. Fault tolerance is improved with this configuration. Furthermore, this configuration allows parallel writes on both DP1 and DP2 and parallel reads on DP2. Notice that both volumes are primaried in the same processor. This means that the disk processes for the two logical volumes compete for use of the same processor and disk controller pair.

Figure 6 illustrates the access paths to two logical volumes configured in a serial write SYSGEN. Compare this configuration with the one in Figure 2. The serial write SYSGEN allows one logical volume to use one processor and controller for its access paths while the other logical volume uses another processor and controller for its access paths. Thus, this configuration avoids processor and disk-controller contention at the expense of parallel writes (DP1 and DP2) and parallel reads (DP2).

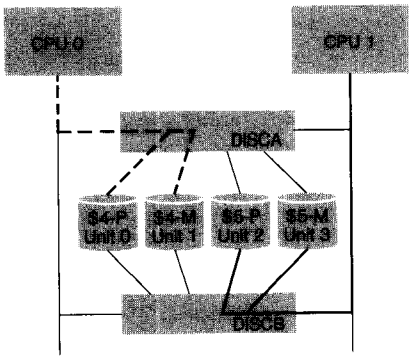
There are trade-offs to consider when disks are configured as in Figure 4. Should they be configured (at the expense of fault tolerance and parallel writes and reads) for serial writes so that potential processor and controller contention is eliminated? Or, in the interest of fault tolerance and possibly performance, should they be configured for parallel writes?

Figure 5




```
CONTROLLERS: DISCA 3107 0,1  %10 CONTROLLER_DISC_PROCESS_2;
             DISCB 3107 0,1 %110 CONTROLLER_DISC_PROCESS_2;

PERIPHERALS: $SYSTEM DISCA-DISCB.0,DISCB-DISCA.1 4110
             NUMDISCPROCESSES 3;
             $SIGN    DISCA-DISCB.2,DISCB-DISCA.3 4110
             NUMDISCPROCESSES 3;


:PUP LISTDEV DISC
  4 $SYSTEM-P  *  00,007  %10   01,007  %10    3  8  4096  DP2
    $SYSTEM-B     00,007  %110  01,007  %110
    $SYSTEM-M  *  00,007  %111  01,007  %111
    $SYSTEM-MB    00,007  %11   01,007  %11
  5 $SIGN-P    *  00,008  %12   01,008  %12    3  8  4096  DP2
    $SIGN-B       00,008  %112  01,008  %112
    $SIGN-M    *  00,008  %113  01,008  %113
    $SIGN-MB      00,008  %13   01,008  %13
```

Figure 6

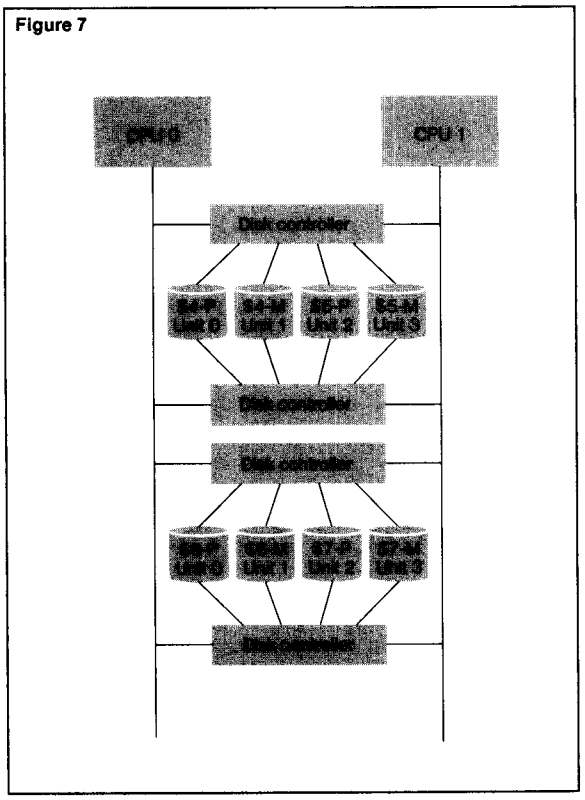



```
CONTROLLERS: DISCA 3107 0,1  %10 CONTROLLER_DISC_PROCESS_2;
             DISCB 3107 1,0 %110 CONTROLLER_DISC_PROCESS_2;

PERIPHERALS: $SYSTEM DISCA-DISCB.0,DISCA-DISCB.1 4110
             NUMDISCPROCESSES 3;
             $SIGN    DISCB-DISCA.2,DISCB-DISCA.3 4110
             NUMDISCPROCESSES 3;


:PUP LISTDEV DISC
  4 $SYSTEM-P  *  00,007  %10   01,007  %10    3  8  4096  DP2
    $SYSTEM-B     00,007  %110  01,007  %110
    $SYSTEM-M  *  00,007  %11   01,007  %11
    $SYSTEM-MB    00,007  %111  01,007  %111
  5 $SIGN-P    *  01,008  %112  00,008  %112   3  8  4096  DP2
    $SIGN-B       01,008  %12   00,008  %12
    $SIGN-M    *  01,008  %113  00,008  %113
    $SIGN-MB      01,008  %13   00,008  %13
```

Figure 5.
The SYSGEN for parallel writes for the configuration in Figure 4 (recommended).

Figure 6.
The SYSGEN for serial writes (not recommended).

Figure 7.
Four logical volumes sharing the same processor pair.



Four Logical Volumes Sharing a Processor Pair

Figure 7 represents an extension of the configuration in Figure 4. Though these two configurations are very similar, the best way to configure the disks in Figure 7 is somewhat more obvious, as will be seen in a moment.

This configuration represents what may well become a common configuration when a V8 or XL8 disk drive is connected to a processor pair. As mentioned earlier, a V8 or XL8 cabinet can be configured with a maximum of four controllers, making a two-to-one ratio of disk units to disk controllers. This results in a single disk controller being shared by at least two units and, when mirrored volumes are configured, a two-to-one ratio of logical volumes to processors.

Of course, a V8 or XL8 drive could be connected to only two controllers, but this configuration implies low levels of physical disk-file activity. Since this article primarily addresses high performance, the two-controller configuration is not discussed further here.

To gain optimum performance with the configuration in Figure 7, all system components should be utilized equally. This means that each processor should be the primary for two volumes and each of the four disk controllers should be configured as a preferred controller. Also, in the interest of fault tolerance, parallel writes should be configured.

Primarying all four of the mirrored volumes in the same processor is possible but undesirable because of the likelihood that the processor would be over-utilized. It is not necessary to primary all four volumes in the same processor in order to have parallel writes; conversely, if each processor is the primary for two volumes, the system is not restricted to serial writes.

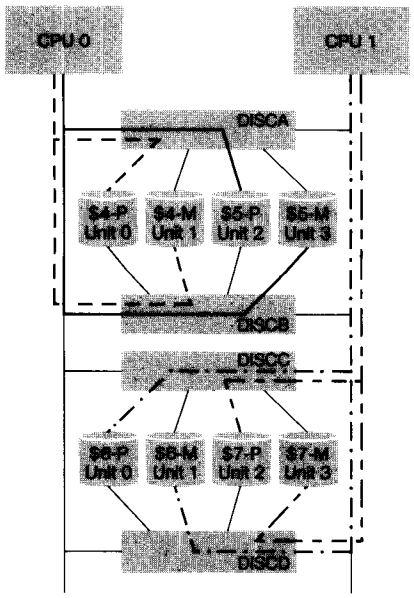
Figure 8 illustrates proper hardware utilization for this configuration: each processor is the primary for two volumes. This configuration also provides acceptable fault tolerance because parallel writes are configured. The configuration is typical in systems having four mirrored volumes connected to a pair of processors through four controllers.

A More Thorough Approach to Fault Tolerance

Although the configuration in Figure 8 provides optimum performance and *acceptable* fault tolerance, it can be enhanced slightly to improve fault tolerance without compromising performance. Figure 9 illustrates the enhanced version and is strongly recommended.

The configuration in Figure 9 has significant advantages over that in Figure 8. Because all disk units configured between a controller pair share a common pair of control (daisy chain) cables, a direct physical connection exists between these units. When disks are configured so that each half of a mirror is connected to a different controller pair, the control cables are not shared by both halves of the mirror, thus eliminating the direct physical connection. The configuration in Figure 9 eliminates physical connection between the primary and mirror halves of a logical volume all the way back to the processor's I/O channel. This is because each half of the logical volume is configured between a different controller pair.

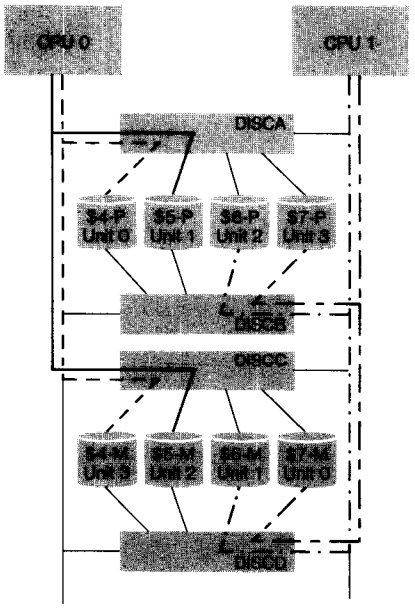
Figure 8




```
CONTROLLERS:  DISCA 3107 0,1   %10 CONTROLLER_DISC_PROCESS_2;
              DISCB 3107 0,1 %110 CONTROLLER_DISC_PROCESS_2;
              DISCC 3107 1,0 %210 CONTROLLER_DISC_PROCESS_2;
              DISCD 3107 1,0 %310 CONTROLLER_DISC_PROCESS_2;

PERIPHERALS:  $SYSTEM DISCA-DISCB.0,DISCB-DISCA.1 4110
              NUMDISCPROCESSES 3;
              $SIGN     DISCA-DISCB.2,DISCB-DISCA.3 4110
              NUMDISCPROCESSES 3;
              $BIG1     DISCC-DISCD.0,DISCD-DISCC.1 4110
              NUMDISCPROCESSES 3;
              $SML1     DISCC-DISCD.2,DISCD-DISCC.3 4110
              NUMDISCPROCESSES 3;
```

```
:PUP LISTDEV DISC
  4 $SYSTEM-P   *  00,007  %10   01,007  %10   3  8  4096  DP2
    $SYSTEM-B      00,007  %110  01,007  %110
    $SYSTEM-M   *  00,007  %111  01,007  %111
    $SYSTEM-MB     00,007  %11   01,007  %11
  5 $SIGN-P     *  00,008  %12   01,008  %12   3  8  4096  DP2
    $SIGN-B        00,008  %112  01,008  %112
    $SIGN-M     *  00,008  %113  01,008  %113
    $SIGN-MB       00,008  %13   01,008  %13
  6 $BIG1-P     *  01,009  %210  00,009  %210  3  8  4096  DP2
    $BIG1-B        01,009  %310  00,009  %310
    $BIG1-M     *  01,009  %311  00,009  %311
    $BIG1-MB       01,009  %211  00,009  %211
  7 $SML1-P     *  01,010  %212  00,010  %212  3  8  4096  DP2
    $SML1-B        01,010  %312  00,010  %312
    $SML1-M     *  01,010  %313  00,010  %313
    $SML1-MB       01,010  %213  00,010  %213
```

Figure 9




```
CONTROLLERS:  DISCA 3107 0,1   %10 CONTROLLER_DISC_PROCESS_2;
              DISCB 3107 1,0 %110 CONTROLLER_DISC_PROCESS_2;
              DISCC 3107 0,1 %210 CONTROLLER_DISC_PROCESS_2;
              DISCD 3107 1,0 %310 CONTROLLER_DISC_PROCESS_2;

PERIPHERALS:  $SYSTEM DISCA-DISCB.0,DISCC-DISCD.3 4110
              NUMDISCPROCESSES 3;
              $SIGN     DISCA-DISCB.1,DISCC-DISCD.2 4110
              NUMDISCPROCESSES 3;
              $BIG1     DISCB-DISCA.2,DISCD-DISCC.1 4110
              NUMDISCPROCESSES 3;
              $SML1     DISCB-DISCA.3,DISCD-DISCC.0 4110
              NUMDISCPROCESSES 3;
```

```
:PUP LISTDEV DISC
  4 $SYSTEM-P   *  00,007  %10   01,007  %10   3  8  4096  DP2
    $SYSTEM-B      00,007  %110  01,007  %110
    $SYSTEM-M   *  00,007  %213  01,007  %213
    $SYSTEM-MB     00,007  %313  01,007  %313
  5 $SIGN-P     *  00,008  %11   01,008  %11   3  8  4096  DP2
    $SIGN-B        00,008  %111  01,008  %111
    $SIGN-M     *  00,008  %212  01,008  %212
    $SIGN-MB       00,008  %312  01,008  %312
  6 $BIG1-P     *  01,009  %112  00,009  %112  3  8  4096  DP2
    $BIG1-B        01,009  %12   00,009  %12
    $BIG1-M     *  01,009  %311  00,009  %311
    $BIG1-MB       01,009  %211  00,009  %211
  7 $SML1-P     *  01,010  %113  00,010  %113  3  8  4096  DP2
    $SML1-B        01,010  %13   00,010  %13
    $SML1-M     *  01,010  %310  00,010  %310
    $SML1-MB       01,010  %210  00,010  %210
```

Figure 8.
Balanced hardware utilization for the configuration in Figure 7. Using parallel writes, this configuration provides acceptable, but not optimum, fault tolerance.

Figure 9.
The best configuration for performance and fault tolerance. In addition to providing parallel writes and balanced hardware utilization, it protects against dual controller failure.

Table 1.
Advantages and disadvantages of the configurations illustrated.

	Figure					
	2	3	5	6	8	9
Connectivity						
Number of paths per processor per logical volume	4	4	4	4	4	4
Fault tolerance						
Protection from:						
Single-processor failure	Yes	Yes	Yes	Yes	Yes	Yes
Channel failure	Yes	Yes	Yes	Yes	Yes	Yes
Controller failure	Yes	Yes	Yes	Yes	Yes	Yes
Control cable failure	No	No	No	No	No	Yes
Access of mirrored halves via different controllers	No	Yes	Yes	No	Yes	Yes
Protection from dual controller failures	No	No	No	No	No	Yes
Performance						
Disk processes in different processors	—	—	No	Yes	Yes	Yes
Parallel writes	No	Yes	Yes	No	Yes	Yes
Parallel reads	No	Yes	Yes	No	Yes	Yes

In the unlikely event that controllers DISCA and DISCB fail simultaneously, if the disks were configured as in Figure 8, the data on logical devices 6 and 7 would become inaccessible. The configuration in Figure 9, however, provides protection against dual controller failure. The best aspect of this configuration is that it costs no more to configure than does the one in Figure 8 because no additional hardware is required.

The advantages of the configuration in Figure 9 should be clear. It allows for parallel writes and balanced hardware utilization, as the configuration in Figure 8 does, while at the same time offering fault tolerance capabilities that Figure 8 does not. This configuration should be used whenever possible.

It is not the intent of this article to illustrate all possible disk configurations or component failure conditions, however. For example, what if the configuration illustrated in Figures 8 and 9 were to be used between four processors instead of two? Some system loads might require this in order to avoid over-utilizing the processors. The four-processor/four-controller configuration could end up looking like the one illustrated in either Figure 10 or 11. The configuration in Figure 11 offers the same fault tolerance as the configuration in Figure 8, but does not offer the degree of fault tolerance provided by the configuration in Figure 9.

It is important to note that the configurations in Figures 8 and 9 react differently to certain component failures. Configurations must be clearly understood in order to predict path-switching scenarios and the impact they might have on overall system performance. Each system must be reviewed individually and all trade-offs and compromises taken into consideration before a final configuration decision is made.

What Have We Learned So Far?

There is a right way and a wrong way to configure a disk subsystem. In some instances, performance studies need not be done to determine which configuration is best. The disk subsystem illustrated in Figure 1 can be correctly configured one way only, as it is done in Figure 3. Before disks are configured, *diagram the configuration*, including all primary preferred access paths, so that potential errors such as those illustrated in Figure 2 can be detected.

The configuration in Figure 7 is one that will become more common with V8 and XL8 disk drive installations, particularly for the NonStop VLX system. The configuration illustrated in Figure 8 might seem to be the best one for this type of disk subsystem because it allows for balanced processor and disk controller utilization. It should be reevaluated, however, in light of the advantage the configuration in Figure 9 has over it. The improved fault tolerance ensured by the latter configuration makes it the obvious choice.

When the disks in Figure 4 are configured, a potential trade-off exists between performance and fault tolerance. To quantify the trade-off, the performance of this configuration was tested with both the serial and parallel write configurations. (The results are presented in the following section.)

Table 1 summarizes the advantages and disadvantages of the configurations illustrated in Figures 2, 3, 5, 6, 8, and 9.

Performance Tests

Application

A model of a simple banking application was used to compare the performance of the system configurations. In the benchmark, 1200 simulated bank tellers operated from 120 branches supporting 1.2 million accounts. All account activity was randomly generated. The system environment was strictly controlled and all hardware resources were evenly utilized.

Files. The application consists of three key-sequenced files: the account file of 1.2 million records, the teller file of 1200 records, and the branch file of 120 records. The history file is unstructured. Table 2 describes the files.

In this benchmark, two other files were included in the application: the data file for the XRAY performance analysis tool and a file called Ustatfil. Each simulated terminal wrote transaction performance data to Ustatfil as the tests were run. Then, upon test completion, summary report programs were run against Ustatfil to calculate throughput and response times.

Transaction. In this application, a transaction consists of reading one record each from the account, branch, and teller files; updating the records in the three files; and writing to a transaction history file. Thus, the transaction performs three reads and four writes.

The operations are performed as follows:

- Read 100 bytes from terminal.
- Send transaction to server.
- Read user account information (100 bytes from account file).
- Read associated branch information (100 bytes from branch file).
- Read associated teller information (100 bytes from teller file).
- Rewrite branch file record.
- Rewrite teller file record.
- Rewrite account file record.
- Perform ten compares, ten moves, five adds, and five subtracts to account-related variables in the server.
- Write history log file to record transaction.
- Return from server to requester.
- Perform ten compares, ten moves, five adds, and five subtracts to account-related variables in the requester.
- Write 200 bytes to simulated terminal.

Table 2.
Files used in the model application.

File name	File type	Number of records	Record size (bytes)	Key length (bytes)	End of file
Account	Key-sequenced	1,200,000	100	10	126,582,784
Branch	Key-sequenced	120	100	6	16,896
Teller	Key-sequenced	1,200	100	6	159,744
History	Unstructured	1 per transaction	50	—	Varied

Table 3.
File placement for the tests.

Volume	File					
	Account	Branch	Teller	History	Ustatfil	XRAY data
$SYS	X	X	X			
$SIGN	X	X	X			
$BIG1		X		X	X	X
$SML1	X	X	X			

Table 4.
Memory and disk cache size for the tests.

Processor	Operating System	Memory size per processor	Cache size[1]
NonStop 1+	E08/DP1	576 pages	128 pages
NonStop II	B30/DP2	4 Mbytes	512 = 8 1024 = 50 2048 = 6 4096 = 200
NonStop TXP	B30/DP2	4 Mbytes	512 = 8 1024 = 50 2048 = 6 4096 = 200

[1]Cache size per processor for the NonStop 1+ processor; cache size per *volume* for the NonStop II and NonStop TXP processors.

Configuration. The following configuration was used for all testing on the NonStop II and NonStop TXP systems. The PATHWAY transaction processing system was configured with 120 terminals, 4 NonStop Terminal Control Processes (TCPs), and 52 servers. The simulator that generated the random transaction stream was implemented in Tandem's Transaction Application Language (TAL) and was configured for the PATHWAY system.

The 120 simulated PATHWAY terminals were run with varying arrival rates, or think times. (Think time is a fixed period of time, specified at run time, during which the simulator waits after learning that the previous transaction has completed before sending the next transaction.)

The servers were written in COBOL. The requesters used the PATHWAY Terminal Control Process 2 (TCP2) and were written in SCREEN COBOL. The servers were not run as process pairs, and the Transaction Monitoring Facility (TMF) was not configured.

Because of the limitations of the NonStop 1+ system, a scaled-down version of the test was run on it.

File placement for all tests is shown in Table 3.

Test Configurations

Figures 10 and 11 illustrate the serial and parallel write configurations used in the tests.

All hardware resources were evenly utilized; for example, processor utilization was balanced to within 4%. Disk utilization was very evenly balanced. In the serial write configuration, each processor had one primary disk process and was a primary processor for one of the four TCPs. Servers were moved as necessary to fine tune the system. In the parallel write tests, two processors had two disks primaried in them. The two processors that were not a primary for a disk process had two TCPs. Again, the servers were moved for fine tuning. Page faults did not occur during any of the tests.

Memory size and disk cache were configured as in Table 4. Note that disk cache on the NonStop II and NonStop TXP processors was configured according to data-base file block size. Block sizes for the application were 4096 bytes for the account file, 512 bytes for the branch file, and 1024 bytes for the teller file.

Figure 10

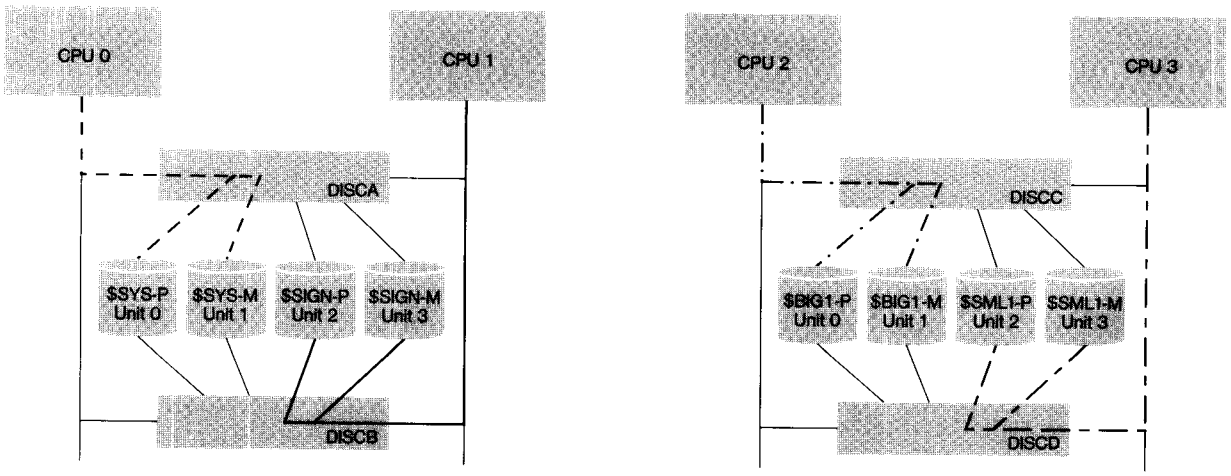


Figure 11.

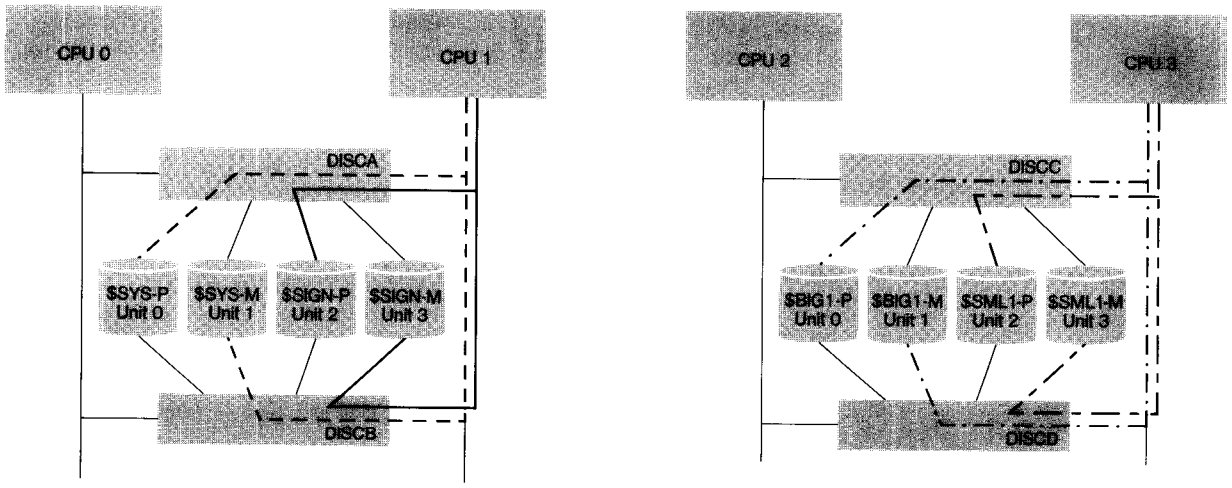
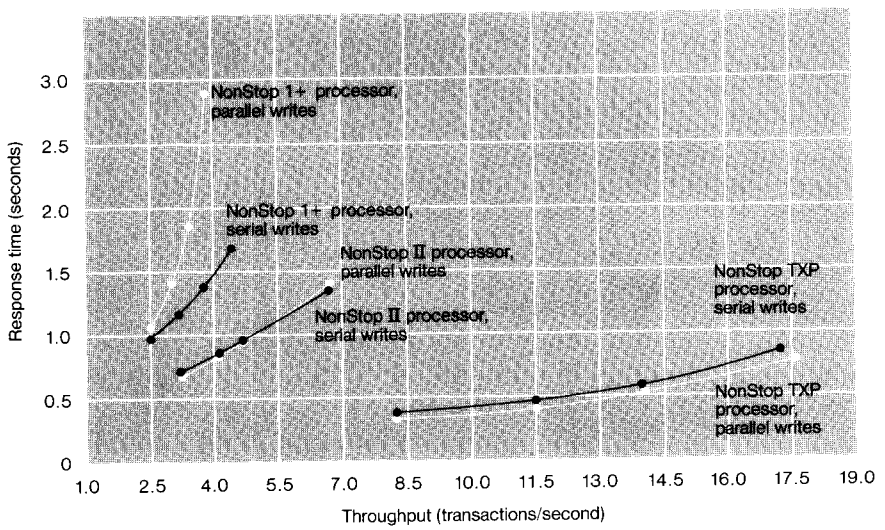


Figure 10.
The serial write configuration used in the tests.

Figure 11.
The parallel write configuration used in the tests.

Figure 12.
Response time versus throughput for serial and parallel writes on the NonStop 1+, NonStop II, and NonStop TXP processors. For the NonStop II and TXP processors, parallel and serial writes yield nearly equal system performance.

Figure 12



Results

The serial write configuration far outperformed the parallel write configuration on the NonStop 1+ processor. Throughput increased and response time decreased for serial writes.

On the NonStop II system, performance of the serial and parallel write configurations was nearly identical. As long as all hardware resources are evenly utilized, the test results indicate that system performance for serial writes and parallel writes is the same in the model OLTP environment.

The test results for the NonStop TXP system indicate slightly improved system performance with parallel writes. Though the performance improvements were modest, the parallel write configuration consistently outperformed the serial write configuration at all system loads in the model OLTP environment.

Figure 12 and Table 5 summarize the test results. Figure 12 shows response time versus throughput; Table 5 lists the test results by configuration and system load.

Some New Rules of Thumb

Only the NonStop 1+ system showed significant performance improvements when serial rather than parallel writes were configured.[2]

For a NonStop II system, parallel writes should be configured since the test results show nearly identical performance regardless of configuration. Parallel writes on the NonStop II system come at no cost to performance and provide superior fault tolerance.

The results for the NonStop TXP system are similar to those obtained for the NonStop II system, except that the performance advantage of DP2 emerges more clearly.

[2]This does not mean that the serial write configuration is recommended for the NonStop 1+ system. This configuration's reduced level of fault tolerance must be taken into consideration. The parallel write configuration is safer and should probably be used in the interest of data integrity. It also conserves system data space.

Regardless of system type, if disk activity is minimal, parallel writes are the sensible choice.

The effect of the configurations on batch work was not investigated. Also, note that a *model* application was tested: a real system environment must be thoroughly analyzed before the appropriate configuration can be selected for it.

The appropriate disk configuration for a system is not always as easily determined as those for the systems illustrated. For example, suppose that four mirrored volumes shared a single disk controller pair (unlike the example having two pairs of controllers). Would it be best to configure for parallel writes and have all four volumes primaried out of the same processor, or would it be best to configure for serial writes and primary two volumes out of each of the two processors? If, indeed, four busy volumes shared the same controller pair, the best solution would be to add another pair of disk controllers and configure the disk with the configuration in Figure 9. Again, it is very important to diagram the configuration when new disks are added to an existing system or when a system is being reconfigured.

Finally, this article does not address how paths are switched during certain failure conditions. It is imperative that system managers be able to recognize component failures quickly and redirect paths as necessary to maintain the desired system performance.

Acknowledgments

The author would like to thank the following people: Jack Mauger, manager of the Performance Support Group, for the opportunity to make this study; Jim Meyerson and Richard Vnuk for their description of the model application; Roberta Henderson for creating Table 1; the reviewers of the article—especially Art Sheehan, for the wisdom and thoroughness of his comments; and Dick Thomas for his assistance with the later revisions.

Scott Sitler, a senior staff analyst, joined Tandem six years ago. Originally, Scott was a member of Corporate Software Education, where he taught System Resource Management for the NonStop and NonStop II systems and a course on the XRAY performance analysis tool. He also developed the two-week System Resource Management II course, which emphasized system performance and XRAY. Two years ago Scott joined the Performance Support Group as a benchmark coordinator. He works on customer benchmarks, new product performance testing, and various other special projects.

Table 5.
Tests results for the NonStop 1+, NonStop II, and NonStop TXP processors.

Configuration	Transactions per second	Average response time (seconds)	90% response time (seconds)	Average CPU utilization (percent)	Terminal think time (seconds)
NonStop 1+ processor running DP1					
Serial writes	4.46	1.69	2.62	76.9	7
	3.80	1.34	2.03	65.2	9
	3.25	1.15	1.70	55.5	11
	2.51	0.97	1.35	42.7	15
Parallel writes	3.95	2.88	4.85	80.8	7
	3.64	1.86	3.12	71.2	9
	3.21	1.42	2.37	60.3	11
	2.48	1.06	1.65	44.8	15
NonStop II processor running DP2					
Serial writes	6.78	1.38	2.29	76.8	16
	5.22	0.90	1.36	58.6	22
	4.20	0.77	1.10	46.5	28
	3.29	0.69	0.93	36.5	36
Parallel writes	6.77	1.44	2.45	75.0	16
	5.20	0.89	1.37	57.9	22
	4.17	0.73	1.08	46.0	28
	3.28	0.66	0.91	36.0	36
NonStop TXP processor running DP2					
Serial writes	17.30	0.85	1.27	77.6	6
	13.96	0.55	0.80	61.4	8
	11.48	0.47	0.63	49.9	10
	8.32	0.39	0.50	35.7	14
Parallel writes	17.55	0.77	1.19	76.3	6
	13.99	0.54	0.74	60.4	8
	11.50	0.41	0.56	48.5	10
	8.34	0.35	0.45	35.1	14

Getting Optimum Performance from Tandem Tape Systems

Tape performance on Tandem systems has improved dramatically in the last two years. Users can now back up their disk files to tape 2.4 times faster than they could with A06 software, and they can restore tape files to disk 4.3 times faster. These improvements have been achieved through:

- Improved hardware (the 5106 and 5130 tape drives, 3206 and 3208 tape controllers, 3107 disk controller, and NonStop TXP processor).
- Improved software (Disc Process 2, or DP2, and improved versions of the BACKUP and RESTORE utilities).
- Improved microcode for the 3206 controller.

The purpose of this article is to help Tandem users get the best performance from their current tape systems. It presents techniques to effectively use tape hardware and software. Performance metrics for BACKUP and RESTORE are presented to illustrate the effect of these techniques. (Note that this article is not intended to be a repository of performance information for different tape applications such as backup, restoration, and Transaction Monitoring Facility, or TMF, on-line dumps.)

Performance Metric

It is difficult to choose a good measure of tape performance. Tape is used in various applications, usually in conjunction with other media, such as disks. On Tandem systems, tape is used to back up disk files, restore backed up files to disk, perform TMF on-line dumps, dump TMF audit trails, and perform a variety of other tasks. The fact that it is used with other media and performs such a variety of tasks presents two problems for quantifying tape performance.

First, the performance of the tape system is tied to the performance of the disk system. A low-performance disk system degrades the performance of even the best available tape system. Although the rate at which the data can be written from a process to tape is important, users are more interested in the performance of their applications.

Second, tape system performance varies with the workload. Backup and restoration are typically run on idle machines. On the other hand, TMF on-line dumps are usually run concurrently with transaction processing workloads. Thus, the performance of a tape system in one application might not be a good indicator of its effectiveness in others.

The metrics used in this article to quantify tape performance are:

- *Throughput*, or the rate (in Kbytes per second) at which disk data can be backed up and restored. Throughput is the natural choice for measuring performance because it determines the time required to back up or restore a disk volume.

- *Processor usage* per Kbyte of data backed up and restored. Processor usage is important when applications contend for processor time. The less processor time used, the better the tape system performs in the presence of other work.

BACKUP and RESTORE are useful to examine because they are used frequently and are seldom affected by disk performance. Also, the stand-alone test results for these utilities are very similar to actual production performance, as users normally run BACKUP and RESTORE at times when the system is lightly loaded.

This article presents techniques to improve the performance of a wide variety of tape applications. These techniques improve tape performance by reducing the time required to (1) process the tape read/write requests and (2) transfer the data from the process requesting tape I/O to the GUARDIAN tape process. BACKUP and RESTORE throughput and processor usage are used to illustrate the effect of these techniques in certain cases, i.e., when the requesters are BACKUP and RESTORE processes.

Components of Tape Processing

Techniques to improve tape performance are aimed at eliminating bottlenecks in the software and hardware components of the tape system. Knowing how data is written to and read from the tape helps in understanding why these techniques work. The following is a simplified explanation of the steps involved in executing a tape write request:

1. *Initialization*. The GUARDIAN tape process controlling the tape device receives a message to write a tape block. The tape process calls a GUARDIAN Message System procedure to move the data record from the buffer of the requesting process to its own buffer (Chandra, 1985). The time required to get the data is proportional to the size of the tape record.
2. *Channel transfer*. The tape process issues a command to the tape controller to write the data record. The tape controller then transfers the data record from the tape process buffer to its own buffer. The data is transferred over the I/O channel in small packets (typically 16 words) called *bursts*. After receiving each burst, the tape controller waits for a small duration, called the *hold-off time*, before requesting the next burst. The time required for channel transfer is a function of the size of the data record, the hold-off time, and the burst length.
3. *Tape write*. After the entire data record has been transferred to its buffer, the tape controller issues a write command to the formatter. The formatter writes the data record to tape. The time required for the formatter to write the data record depends upon the hardware characteristics of the drive (such as tape speed), the data record size, and the tape recording density.
4. *Completion*. The controller notifies the tape process that the data record has been written to tape. The tape process then returns the completion status to the requester.

A request to read data from tape works similarly. The tape process issues a command to the tape controller to read the next record, causing the formatter to read the record from the tape into the controller's buffer. From the controller, the data record is transferred to the tape process buffer in bursts over the I/O channel. The tape process then completes the read operation by moving the data record to the requester using a GUARDIAN Message System procedure.

The time required to execute steps 1 through 4 above determines tape system performance. For example, if these steps take 56 ms for a 28-Kbyte tape record, the tape system could deliver a throughput of 500 Kbytes per second. If a faster tape drive reduced this time to 40 ms, the resulting tape throughput would be 700 Kbytes per second.

Hence, the performance of a tape system can be improved by reducing the time required to execute these steps. The following sections present ways to achieve this reduction.

Requester Bottlenecks

Several factors can be responsible for unsatisfactory tape performance. Before investing in new tape hardware, determine if factors other than the tape system are responsible for low tape throughput. Some of these factors might

Figure 1.
Effect of file size on throughput. The volume mode of BACKUP and RESTORE improves throughput.

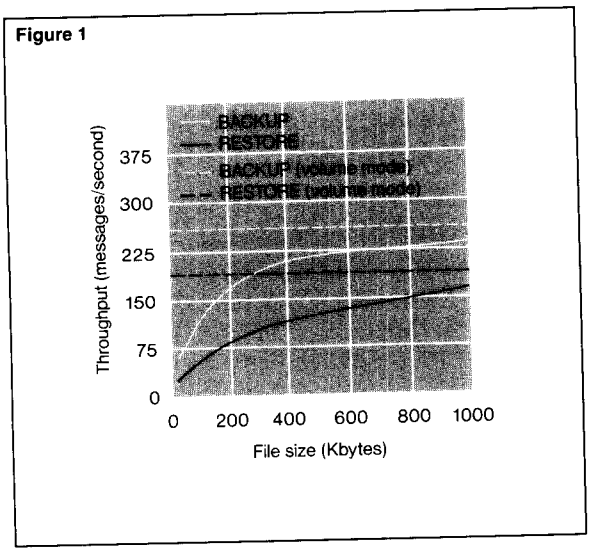
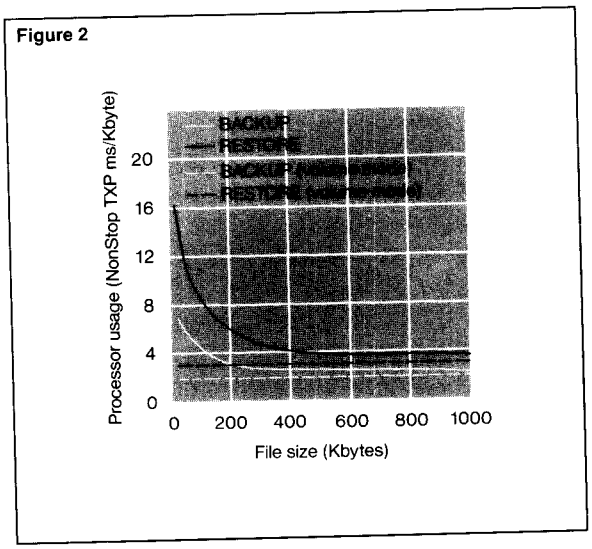


Figure 2.
Effect of file size on processor usage. The volume mode of BACKUP and RESTORE decreases processor usage.



be slower processors such as the NonStop 1+ and NonStop II, lower performance disk controllers such as the 3106, and old software. (Simply switching from DP1 to DP2 can improve performance substantially.)

The tape system can write the data only as fast as the application provides data to it. Backing up and restoring small files is an example in which throughput is largely independent of tape system performance. Figure 1 charts BACKUP and RESTORE throughput as a function of the average file size, both in standard mode and using the VOLUME option. (The VOLUME option backs up an entire volume at a time.)

As the figure shows, for an average file size of 32 Kbytes, BACKUP operates at 65 Kbytes per second and RESTORE at 22 Kbytes per second if run in standard mode. BACKUP and RESTORE are slow because of overhead associated with opening, closing, and creating disk files.

Because the VOLUME option allows BACKUP and RESTORE to bypass File System processing, it can provide very high throughput, as shown in Figure 1. For an average file size of 32 Kbytes, using the VOLUME option improves BACKUP throughput by a factor of 4 and RESTORE throughput by a factor of 8. In Figure 2, the VOLUME option is shown to reduce processor usage by a factor of 3 for BACKUP and a factor of 5 for RESTORE.

One limitation of using the VOLUME option is that it does not allow the backup and restoration of individual files, but only an entire volume. This is not a problem if volumes are backed up frequently but restored infrequently. To restore less than the entire volume, the image of the entire volume must be restored to a scratch disk and the necessary files transferred to the desired location. Here the inconvenience of retrieving particular files is outweighed by the time saved for backups. (RESTORE performance is not a big problem in this case; a 240-Mbyte disk can be restored in 22 minutes.)

Note that if only a few small files need to be backed up and restored frequently, the VOLUME option may not be a suitable alternative. A faster tape system would not improve throughput here either, because the requester processes (BACKUP and RESTORE) are responsible for low throughput.

The next sections investigate ways to improve tape performance when the tape system limits throughput.

Tape Recording Density and Data Record Size

Recording Density

The 5106 tape drive can write data in three densities. In the NRZI format, the data is recorded at 800 bits per inch (bpi) on the tape; the corresponding bit densities for PE and GCR formats are 1600 bpi and 6250 bpi, respectively. Since the tape speed is constant (125 inches per second) in all these modes, writing data in NRZI format takes seven times longer than writing it in GCR format. Hence,

using higher density usually improves performance. Changing from PE to GCR format can double throughput in some instances.

To use GCR format, the Tandem system must have the proper hardware: NonStop TXP or VLX processors, 5106 or 5130 tape drives, and the 3107 disk controller. To specify GCR format, set the density option on the drive to GCR or use the density option of BACKUP.

Higher recording densities, in addition to increasing throughput, result in better utilization of the tape itself. The GCR format allows the same tape to store *up to* four times the data it can store with PE format and seven times the data it can store with NRZI.

Figure 3

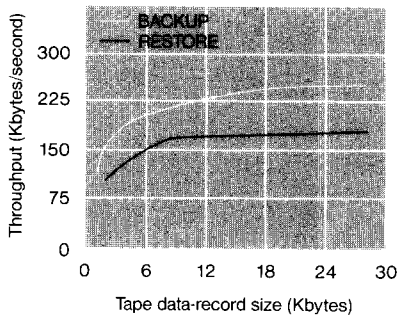


Figure 3.
Effect of tape data-record size on throughput. Larger tape blocks improve throughput.

Figure 4

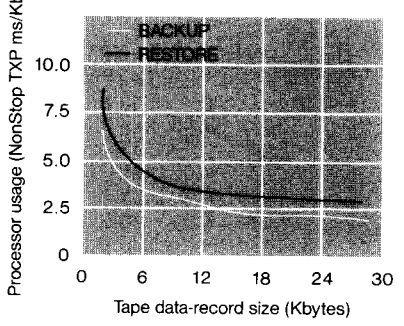


Figure 4.
Effect of tape data-record size on processor usage. Larger tape blocks reduce processor usage.

Tape Record Size

BACKUP writes data to tape as data records. It allows users to specify a tape data-record size between 2 Kbytes and 30 Kbytes (in increments of 2 Kbytes). Records are separated by interrecord gaps. RESTORE reads these tape records to construct the disk file.

Figure 3 shows BACKUP and RESTORE throughput as a function of the tape record size. Changing the tape record size from 2 Kbytes to 28 Kbytes doubles throughput. Figure 4 shows that this change reduces the processor usage by a factor of 3.

BACKUP uses a default record size of 8 Kbytes for NonStop systems (NonStop II, NonStop TXP, and NonStop VLX processors) and 2 Kbytes for NonStop 1+ systems. The record size used by BACKUP can be specified with the BLOCKSIZE parameter. RESTORE uses the block size recorded on the tape. Backing up with larger block sizes can increase throughput substantially.

Why does the record size affect the throughput and the processor usage so significantly? The following are a few reasons.

Larger Messages. Larger messages are faster and more efficient. The size of messages used by BACKUP, RESTORE, the disk process, and the tape process to transfer data to each other is equal to the tape record size. Since larger messages take proportionately less time and fewer processor cycles to transfer the same amount of data, the *initialization* of tape write and the *completion* of tape read take less time. This reduces the time required to process the tape I/O and increases throughput.

Faster Tape Write/Read. The 5106 tape drive stops after it reads or writes every record and takes about 4 ms to start again. Recording at GCR density, the 5106 takes about 6.6 ms to write a 2-Kbyte tape record (including the start/stop times). The corresponding time for a 28-Kbyte tape record is 40.7 ms. Hence the tape write time per Kbyte is 3.3 ms for the 2-Kbyte record and 1.5 ms for the 28-Kbyte record. Thus larger tape record sizes cause the *tape write* time to decrease by a factor of 2, increasing throughput. Tape reads improve for similar reasons.

More Efficient Disk Access. BACKUP and RESTORE use the tape record size as the data block size for accessing the disk. Larger tape records make each disk access more efficient by reducing the average disk latency time to

Figure 5.
Effect of tape data-record size on tape capacity. Larger tape records increase storage capacity.

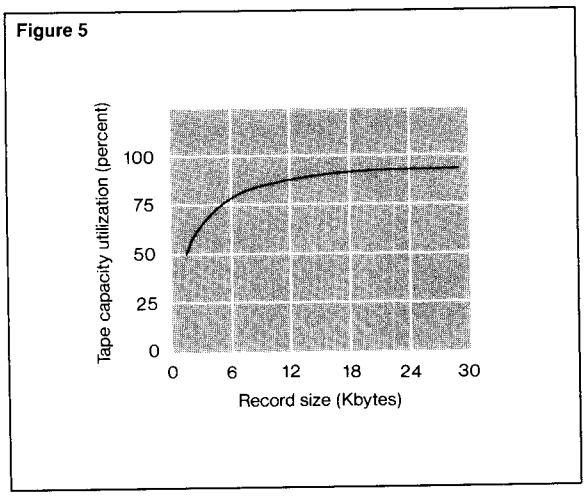
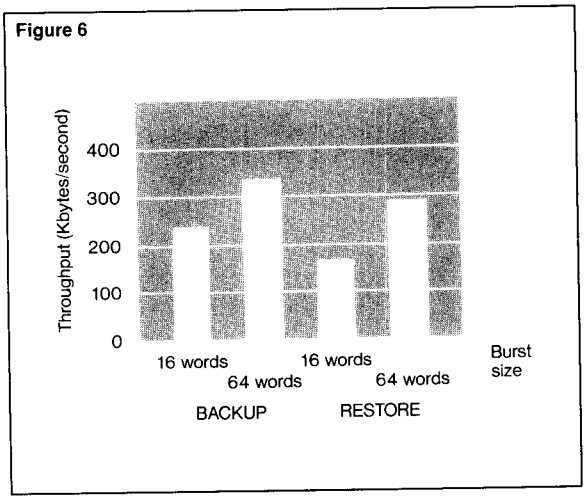


Figure 6.
Effect of channel burst size on throughput. Larger channel bursts increase throughput.



access the same amount of data (Khatri and McCline, 1985). Increased disk throughput also causes the tape throughput to go up, because the disk is really the bottleneck for smaller block sizes.

Large tape records, in addition to increasing throughput, allow the same tape to store more data. Figure 5 shows the capacity of a GCR tape as a function of the record size. Increasing the tape record size from 2 Kbytes to 28 Kbytes almost doubles tape capacity. This is because, for 2-Kbyte tape records, the tape length required to store the data (0.33 inches) is almost equal to the interrecord gap (0.3 inches). Hence half the tape contains gaps and the other half contains data. For 28-Kbyte tape records, the interrecord gaps are negligible compared to the amount of tape the records use. Thus, the medium is almost fully utilized.

Channel Burst Size

Channel transfer time is a significant component of the time required to write a tape record. As described earlier, data transfer over the I/O channel occurs in bursts separated by a hold-off period. This is necessary because multiple devices share the I/O channel; the hold-off period allows other devices to access the channel between bursts of tape data.

Transferring data over the channel in bursts takes substantially longer than continuously transferring the same data. Also, a small burst size increases the time required to process each tape I/O and degrades throughput. One way to minimize this problem is to transfer data over the channel in larger bursts. The default burst size for the 3206 controller is 16 words. This can be increased to 64 words by using the optional 3206 controller microcode available with the B-series software releases.

Figure 6 compares throughput obtained with 64-word bursts and 16-word bursts. As shown, 64-word bursts improve BACKUP performance 1.4 times and RESTORE performance 1.7 times.

Larger channel bursts may cause problems on heavily loaded channels with attached controllers that are not fully buffered. This is because larger bursts lock out other devices sharing the same channel for longer durations. On partially buffered controllers, this additional delay may cause data overruns and underruns. Before using larger bursts, use the GUARDIAN STRESS program (run with INSTALL or SYSGEN) to determine if these bursts would cause the system to malfunction. Larger bursts should only be used if the STRESS analysis indicates that it is safe to use them. (Note that larger bursts do not cause the system to malfunction if the channels connected to the 5106 controller using these bursts are connected to fully buffered controllers only. See the *System Management Manual*.)

Configuration Considerations

BACKUP and RESTORE operations involve three processes: the disk process, the tape process, and the BACKUP or RESTORE process. Users have very little control over the location of the disk and tape processes: they must run in the processors to which the disk and tape are connected. The BACKUP and RESTORE processes can be placed in any processor, however, using the CPU parameter in

the RUN command. This section examines how the locations of these processes affect throughput and processor usage.

Since there are three processes, the following configurations are possible:

1. The disk, tape, and application (i.e., BACKUP or RESTORE) processes are all in the same processor. (Note that it might not be possible to use this configuration if the disk and tape are not connected to the same processor.)
2. The disk, tape, and application processes are all in different processors.
3. The disk and tape processes are in different processors. The application process is in the same processor as the tape process.
4. The disk and tape processes are in different processors. The application process is in the same processor as the disk process.
5. The disk and tape processes are in the same processor. The application process is in a different processor.

Why does the relative location of processes affect throughput and processor usage?

First, all the processes involved in BACKUP and RESTORE operations communicate with each other via messages. The time and the processor cycles required to send these messages depend on the location of the sending and receiving processes: messages to a process in the same processor usually require less time and processor cycles than messages to a process in a different processor. The time required to send messages affects the throughput; the processor cycles used affect the processor usage.

Second, only one process can run in a processor at any time. Hence, when more processes are located in the same processor, they take more time to perform the same operations because of processor contention. For example, BACKUP or RESTORE performs poorly in the first configuration listed above because contention degrades throughput even though messages are cheaper and faster.

Figure 7 compares BACKUP throughput for the five configurations, and Figure 8 compares the corresponding processor usage. Configuration 3 (BACKUP and tape processes in the same processor) yields the maximum throughput. This is because it reduces the time required by the BACKUP process to send a message to the tape process, increasing tape throughput.

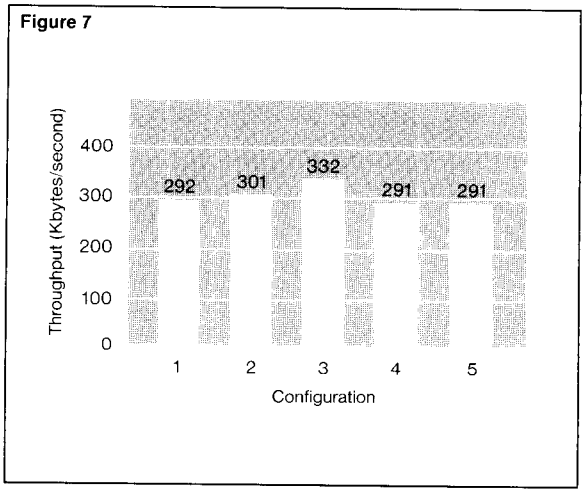


Figure 7.
Effect of configuration on BACKUP throughput. To maximize throughput, the tape process should be in the same processor as BACKUP and the disk process in a different processor.

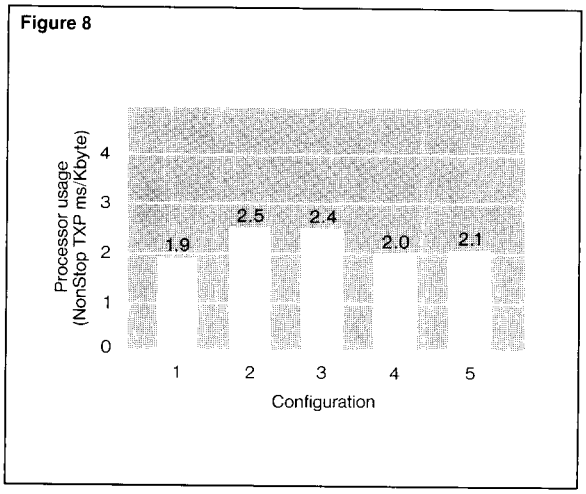


Figure 8.
Effect of configuration on BACKUP processor usage. Processor usage is minimized when the disk, tape, and BACKUP processes run in the same processor.

Configuration 1, although the option with the lowest processor usage, yields lower throughput because of increased processor contention. Configuration 2 is the most expensive in terms of processor cycles but has the minimum contention, hence does better than 1 and 5. Configuration 4 results in lower throughput than 2 and 3 because the BACKUP process takes more time to send messages to the tape process than in 3 and there is more processor contention in the disk process CPU than in 2. Configuration 5 results in the lowest throughput because messages between the disk process and BACKUP take more time.

Configuration 3 is the best option because it results in the highest throughput and uses a reasonable number of processor cycles.

Figure 9.
Effect of configuration on RESTORE throughput. RESTORE throughput is maximized by having the disk process in the same processor as the RESTORE process and the tape process in a different processor.

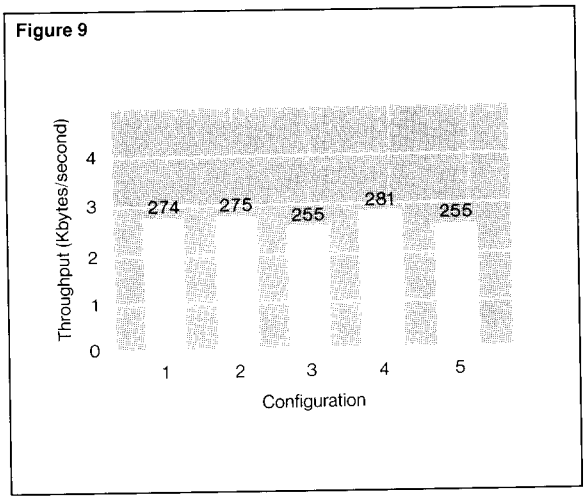
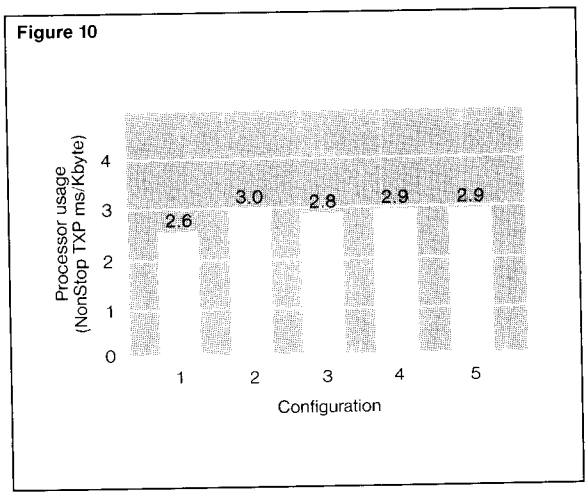


Figure 10.
Effect of configuration on RESTORE processor usage. Processor usage is minimized when the disk, tape, and RESTORE processes run in the same processor.



Figures 9 and 10 compare RESTORE throughput and processor usage for the configurations mentioned above. Note that configuration 3, which offers the best BACKUP performance, results in the lowest RESTORE throughput. This is because RESTORE has only one NOWAIT write outstanding against the disk file at any time, causing the disk write throughput to limit the RESTORE data rate (Khatri and McCline, 1984). Placing the RESTORE process in the same processor as the tape process causes messages from the RESTORE process to the disk process to take longer, resulting in still lower disk throughput. This also decreases the RESTORE data rate.

Throughput for configuration 5 is also low because the RESTORE process is not in the same CPU as the disk process and there is contention in the disk process CPU. Note that the data rates and processor usage for the other configurations are very similar to each other.

Clearly, for optimum throughput, the tape process should not be in the same processor as the RESTORE or disk process. All other configurations yield similar throughput, although keeping the RESTORE process in the same processor as the disk process maximizes throughput.

Future Performance

Tandem is committed to improving tape performance. Planned changes to the GUARDIAN 90 operating system and the tape hardware will push tape performance to new heights in future software releases.

Conclusions

Several factors could be responsible for unsatisfactory tape performance. Before investing in new tape hardware, determine if the tape system is really at fault. If the tape system is responsible for the poor throughput of the application, upgrading the tape hardware and following these suggestions will improve the performance of the tape system:

- Use the GCR recording format to increase tape throughput and storage capacity.
- Use the largest possible tape record size. Larger records result in higher throughput and consume fewer processor cycles.
- For a 5106 tape drive, use STRESS to determine if a 64-word burst size is safe for the channel configuration. If it is, use the optional microcode to improve tape performance.
- Experiment with the placement of application processes. When using BACKUP, keep the BACKUP and tape processes in the same processor. When using RESTORE, keep the RESTORE and disk processes in the same processor.

References

Chandra, M. 1985. The GUARDIAN Message System and How to Design for It. *Tandem Systems Review*. Vol. 1, No. 1. Part no. 83934. Tandem Computers Incorporated.

Khatri, A., and McCline, M. 1985. Improved Performance for BACKUP2 and RESTORE2. *Tandem Systems Review*. Vol. 1, No. 2. Part no. 83935. Tandem Computers Incorporated.

Anil Khatri joined Tandem in November 1983 after obtaining a Masters in Computer Science from the University of Maryland, College Park. After modeling Tandem hardware and software subsystems for two years, he is now developing capacity planning products for NonStop systems.

Performance Measurements of an ATM Network Application

This article is about Wells Fargo Bank's Retail Delivery System (RDS), with emphasis on a major on-line transaction processing (OLTP) benchmark done cooperatively by Tandem and Wells Fargo. It is divided into four sections.

The first section is an overview, which includes a summary of the entire article.

The second section describes Wells Fargo Bank's approach to retail banking today in the context of deregulation and California's banking environment.

The third section describes RDS, Wells Fargo's retail transaction processing on Tandem NonStop systems and establishes why the benchmark results are important to Wells Fargo.

The fourth section contains in-depth technical information on a series of benchmarks performed jointly by Tandem and Wells Fargo, including a description of the software and hardware environment and details on the results obtained.

Overview

Goals

As deregulation changes the traditional way that banks have generated revenue, competition in retail banking has become intense, especially among the five largest California banks. Wells Fargo Bank, already a leader in the size, account penetration, and availability of its own ATM network, has established a formidable plan for maintaining its position into the 1990s. Two key goals of this plan are rapid growth without interruption of services and the lowest cost per transaction in the industry. To help achieve the goals, a project was initiated to develop RDS. RDS is an application environment running on Tandem NonStop systems for performing retail transaction processing. Currently, RDS supports ATMs and shared debit-card networks. RDS makes use of Tandem's standard products whenever possible, thus avoiding some of the costs of building and maintaining in-house expertise in favor of that provided by Tandem.

Benchmarks

Tandem recognized the opportunity to enhance the performance of its products by using the classic OLTP environment that RDS represented. From the latter part of 1985 to the summer of 1986, several OLTP performance benchmarks were carried out as part of a joint project between Wells Fargo and Tandem. The purpose of these benchmarks was to test and measure the new RDS software developed to run the Express Stop ATM network. The results demonstrated that a full-function retail banking application can achieve excellent response time and high throughput, even at high levels of CPU utilization. The tests also revealed significant performance improvements in the B30 release of the GUARDIAN 90 operating system and the new NonStop VLX processor over the B10 and B20 releases and the NonStop TXP system.

Figure 1

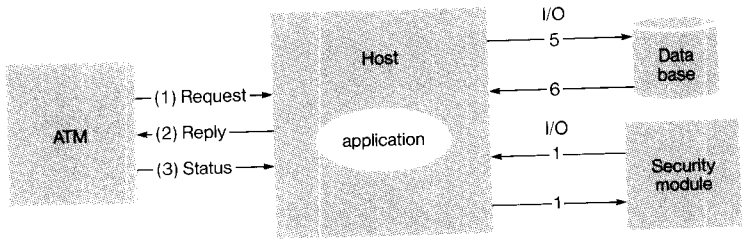


Figure 1.
Benchmark transaction. The RDS ATM transaction comprises three steps: (1) a message from the ATM identifying the customer and the transaction requested, (2) a response from the application indicating whether or not the transaction has been authorized, and (3) a status message from the ATM which informs the application of the transaction's outcome. A typical RDS ATM transaction requires 11 disk I/Os, as well as 2 I/Os to the security module to handle personal identification number (PIN) authorization.

The benchmarks first used an application prototype and later the real application as it was designed for production. Both the prototype and real application implemented a three-step ATM transaction (Figure 1). A mix of transaction types was used, the most common being a simple cash withdrawal.

The application benchmark results, summarized in Table 1, are significant in that the transaction measured is used today by a major bank to perform real business functions. The RDS transaction provides at least twice the functionality of the more commonly measured "debit-credit" transaction (also known as ET1).[1] This stems from the fact that RDS requires an average of 11 I/Os to the debit-credit benchmark's 7, uses a more complex communications protocol, and implements a three-step versus a two-step transaction.[2]

[1]The "debit-credit" transaction refers to the transaction standard defined in "A Measure of Transaction Processing Power," *Datamation*, April 1, 1985.

[2]The number of logical file I/Os for the debit-credit transaction was based on the number reported by the performance group at Tandem responsible for the debit-credit benchmarks.

The RDS benchmark was performed at a facility built to model an ATM network environment. The results were calculated from measurements of real applications configured in a production-like environment, rather than from modeling or extrapolation. To accomplish this, the application on the system under test communicated with ATM simulators running on a remote terminal emulator system using 100 communications lines and modem eliminators. The ATM simulator, designed to simulate the IBM 3624 ATM used in the current Wells Fargo network, was so effective that it functioned without modification when tested against the current production Express Stop application software.

Another distinguishing factor of the RDS application was the use of a large number of standard Tandem software products:

- GUARDIAN 90 operating system, B30 software release.
- GUARDIAN Disc Process 2 (DP2).
- Transaction Monitoring Facility (TMF).
- PATHWAY Terminal Control Process 2 (TCP2).
- PATHWAY Intelligent Device Support (IDS).
- EXPAND™ Fiber Optic Extension (FOX).
- SNA Access Method (SNALU).
- SNA High Level Support (HLS).

Products with special modifications to improve throughput or response time were avoided in favor of the supportability of off-the-shelf subsystems and tools supplied by Tandem.

Results

An objective of the benchmark was to determine how high the CPU utilization could be driven while maintaining response-time goals. In previous tests of host data-base applications, the maximum steady-state utilization that had been achieved before queuing resulted in degraded response time was in the 60%–70% range. This utilization limit, or "Q factor," made unusable a sizable portion of the overall CPU capacity (Blake, 1979). However, during the benchmark, it was discovered that a properly configured NonStop TXP or NonStop VLX system with

DP2 could run well above 70% CPU utilization before experiencing performance degradation. Due to improvements in the GUARDIAN 90 operating system, this upper limit was extended to more than 85% with NonStop TXP processors and greater than 95% with the NonStop VLX processor.

As of the B30 release of GUARDIAN 90, an improvement in the Message System interprocess message protocol reduced the CPU dispatches needed to send a message. This resulted in fewer CPU cycles per transaction and greatly reduced the system overhead portion of total processor usage.

The benchmark revealed that at higher transaction rates, CPU utilization per transaction actually decreased. Project performance analysts attribute this improvement to efficiencies of scale in the disk and communications access methods. Specifically, the disk process, DP2, performs fewer physical I/Os per transaction, and SNAX, less polling per I/O.

Table 1.
Summary of Wells Fargo RDS ATM application benchmark results.

	NonStop TXP prototype	NonStop TXP application	NonStop VLX application
Number of CPUs	6	6	4
Transaction rate per system (transactions per second)	10.8	12.0	12.0
Transaction rate per CPU (transactions per second)	1.8	2.0	3.0
Average host response time (seconds)	1.2	1.8	0.9
CPU utilization[1]	78%	95%	97%

[1]Highest CPU utilization achieved before response time degraded.

Retail Banking at Wells Fargo

The Banking Environment Now and in the Future

Banking is very different from what it was 20 years ago. As a result of deregulation, demands and pressures on banks for more products and services have increased, and their earning power is no longer defined by regulations. Financial services are now provided within the structures of large conglomerates such as Merrill Lynch, American Express, and Sears. Money market funds and insurance companies also offer investment services that have taken customers away from the traditional bank.

Competition for customers is fierce and banks must broaden their products and services to be successful. They must also be aggressive in marketing their products and services in a timely and cost-effective manner. To compete more effectively in a deregulated environment, banks must take advantage of new technological advances and continue to automate banking services. Banks need automation to reduce the cost of transactions and to serve customers quickly and efficiently.

From current projections, the U.S. banking industry is expected to change dramatically by 1990. The total number of banks will fall from 15,000 to 9600, with the reduction being in small and medium banks. Large banks (assets over $1 billion), however, will increase from 230 to 290. This consolidation will involve over half of the banks in merger activities. Point-of-sale debit cards will gain more acceptance—up to 20% of households will participate. ATMs will have a greater share of deposit and withdrawal transactions than will human tellers, and banks' major form of revenue (interest income) will decrease by almost 10% (Arthur Andersen & Co., 1983).

Wells Fargo and California Banking

As deregulation creates an environment in which the larger banks have the greatest chance of survival and success, the presence of many billion dollar banks in California makes it a particularly competitive location. At the end of 1985, California's five largest banks

Table 2.
Size and profitability of California's five largest banks, ranked by total assets (as of December 1985, except when noted).

Bank	Total assets (billions)	Assets rank[1]	Deposits rank[2]	Earnings per share, first quarter		Return on average assets,[3] 1985
				1986	1985	
Bank of America	$106.1	2	1	0.31	0.63	-0.41
Security Pacific	$ 44.8	8	5	1.11	1.00	0.61
Wells Fargo[4]	$ 23.5	12	7	2.25	1.95	0.68
First Interstate[5]	$ 20.6	14	8	1.71	1.57	0.67
Crocker[4]	$ 19.0	16	10	—	—	0.08

[1]Rank among U.S. commercial banks.

[2]Rank among U.S. commercial banks by total domestic deposits.

[3]A common measure of a bank's profitability.

[4]These figures precede the Wells Fargo-Crocker merger in June 1986.

[5]All figures for First Interstate of California except earnings per share, which are for First Interstate BanCorp.

represented 5 of the 16 largest U.S. banks (Table 2) and together owned 5 of the 11 largest proprietary ATM networks (Table 3). By domestic deposits, California banks lead the nation by an even greater margin with 5 of the 10 largest U.S. banks.

Wells Fargo has distinguished itself among the state's five largest banks not only by common measures of profitability, but also through its dramatic growth. The recent acquisition of Crocker Bank, the largest bank buyout in U.S. history, not only puts Wells Fargo close to Security Pacific in size, but increases its competitive position significantly in the southern part of the state.

Banking studies cite retail services as a key to success in the new deregulated banking world because of their role in keeping and attracting depositors. Evidence of Wells Fargo's success in this area is reflected in the phenomenal customer acceptance of its debit-card networks, particularly the Express Stop ATM network. The ATM's key role in the bank's overall strategy is shown in the size of the network relative to the size of the overall bank. With the Crocker buyout, Wells Fargo became the ninth largest bank in the United States as measured by total assets. But it operates the second largest proprietary ATM network.[3]

[3]Wells Fargo ranks as the ninth largest U.S. bank with $38.9 billion in assets as of June 30, 1986. See note 2 on Table 3 regarding the ATM network size ranking of number two.

The strongest evidence of the ATM network's popularity with Wells Fargo's customers is shown in the usage statistics: Wells Fargo announced that by the end of 1985, 66% of its demand deposit account customers (DDAs—commonly known as checking accounts) showed card activity within the month. These results are no accident, as Wells Fargo has implemented incentive plans for both tellers and branch managers that encourage non-card-using customers to use the ATMs. While the apparent conflict of interest between a teller and an ATM might seem to limit this plan, in fact it hasn't. Wells Fargo's customers use their cards mostly for after-hours banking: 70% of Express Stop's transactions are done outside banking hours (*EFT Report*, 1985).

Availability

ATM specialists and bank marketing personnel have different explanations for the success of ATM networks. Most agree, however, that availability is at least a large component of that success. Among Californians, the expectation of cash accessibility is extremely high. For cash to be accessible means that the ATM be conveniently located for consumers and that it be in a condition to dispense money. An example of this is the visible frustration of a hopeful moviegoer who drives several miles to the nearest ATM only minutes before the show is to start, discovering too late that the ATM is not operational.

Accessibility of ATMs to customers can be viewed from three perspectives: the operational state of the ATM itself as a device, the availability of the host application system, and the availability of the host from any given site. Exceptional measures have been taken by Wells Fargo to achieve high host and site availability so that the customer does not lose basic accessibility to his or her account even if a single ATM is not operational. Beyond the 24-hour operation of the network, Express Stop includes a minimum of two ATMs on separate communications lines at 80% of all its locations. This provides a level of fault-tolerant communications in addition to the built-in fault tolerance of the Tandem host. Each of the lines in a pair is attached to a separate controller, which is in turn on a separate CPU. In this way, if any single CPU, controller, modem, phone line, or ATM fails, the site still has an operational ATM.

For the year 1985, availability for all ATMs ranged between 94% and 96%, whereas the availability at any given site ranged from 98% to 99%. These figures are based on a 24-hour network operation, which means that all scheduled and unscheduled ATM maintenance, problems with phone lines or modems, and host downtime (less than 0.1%) cause an average site to be unavailable for no more than 1.7 to 3.3 hours per week. Put another way, customers who use their ATM card once a week would be turned away at a site not more than once a year. An example of what these measures mean to customers occurred in Northern California in the winter of 1985. A flood caused over 100 ATMs to be down, but only two sites lost all ATMs.

Table 3.
ATM network statistics for California's five largest banks (ranked by number of on-line ATMs).[1]

Bank	Number of on-line ATMs		U.S. rank[2]	Card base[3] (millions)	Transactions per month[4]	Percent active DDA[5]
	December 1984	December 1985				
Bank of America	1027	1330	1	3.9	13.0 (as of September 1985)	NA
Wells Fargo	719	783	3	1.3	6.3	66%
Security Pacific	516	613	7	1.4	4.7	54%
Crocker	379	415	9	1.0	3.0	53% (as of March 1986)
First Interstate	357	357	11	2.0	3.9	40%

[1]Figures that differ from *EFT Bank Network News* figures were confirmed by the respective banks.

[2]Rank by size of *proprietary* ATM network as of December 1985. Although First Interstate is listed as number two in *EFT Network News*, they confirmed that their "Day and Night Teller" network is a shared network, connected through CIRRUS and other regional switches. The second largest proprietary ATM network in the United States was actually owned by First Texas Savings. Their "MoneyMaker" network had 791 ATMs on-line as of September 1985. However, if Wells Fargo and Crocker had been combined as of that time, they would have had 1163. With the merger complete, Wells Fargo is well established in the number two spot.

[3]Number of debit cards issued for use on each bank's ATM network. The card base, or number of debit cards that are authorized to use the ATM network, can be misleading as some banks give ATM cards more freely than others. It is generally agreed that the best measure of success for the ATM network is percentage of DDA (checking) accounts with card activity (see footnote 5).

[4]Number of transactions against card base for the month of December 1985.

[5]Percentage of DDA accounts with any card activity as of December 1985. "DDA accounts with card activity" is often interpreted differently by the various banks. In all cases reported here, it indicates the percentage of active accounts in a month.

The Retail Delivery System

RDS: A Strategy for High Performance at Low Cost

In order to maintain a high level of availability while meeting the demand for increased retail services, Wells Fargo conceived of RDS. It was intended to provide:

- Data and transaction integrity.
- Continued high availability (including disaster recovery).
- Nondisruptive growth with unit cost decreases.
- Ability to introduce new products, services, and devices.

The existing Express Stop network, running on two Tandem NonStop TXP systems, did not provide the structure and flexibility to achieve reduced cost and the high volumes expected in the 1990s. Without a reliance on standard products, support would continue to require specialized skills that are difficult and expensive to obtain. New services would have to fit into the existing design. The new RDS architecture was to be a delivery system to support retail outlets. Its characteristics would include:

- Hardware and software growth without rewriting code.
- High level of vendor support.
- Application development in industry languages (e.g., COBOL).
- Streamlined network operations.
- Ability to move transaction processing resources closer to the customer in order to reduce communication costs.

Real Cost of a Transaction

If the objective of RDS were reduced to a single statement, it would be to achieve the lowest cost per transaction in the industry and yet be able to accommodate increased volume and service levels. On the surface these goals appear ambitious and even incompatible. When the *real* cost of a transaction is examined, however, a different picture emerges. The cost of a transaction is more than the cost of the processing power required to complete it. A complete "invoice" would include some hidden costs, including the costs of:

- Development (design and implementation).
- Ongoing maintenance (complexity).
- Added enhancements or new functions.
- Increased volume (or transaction types).
- Hardware to process transactions.
- Hardware and software failures.

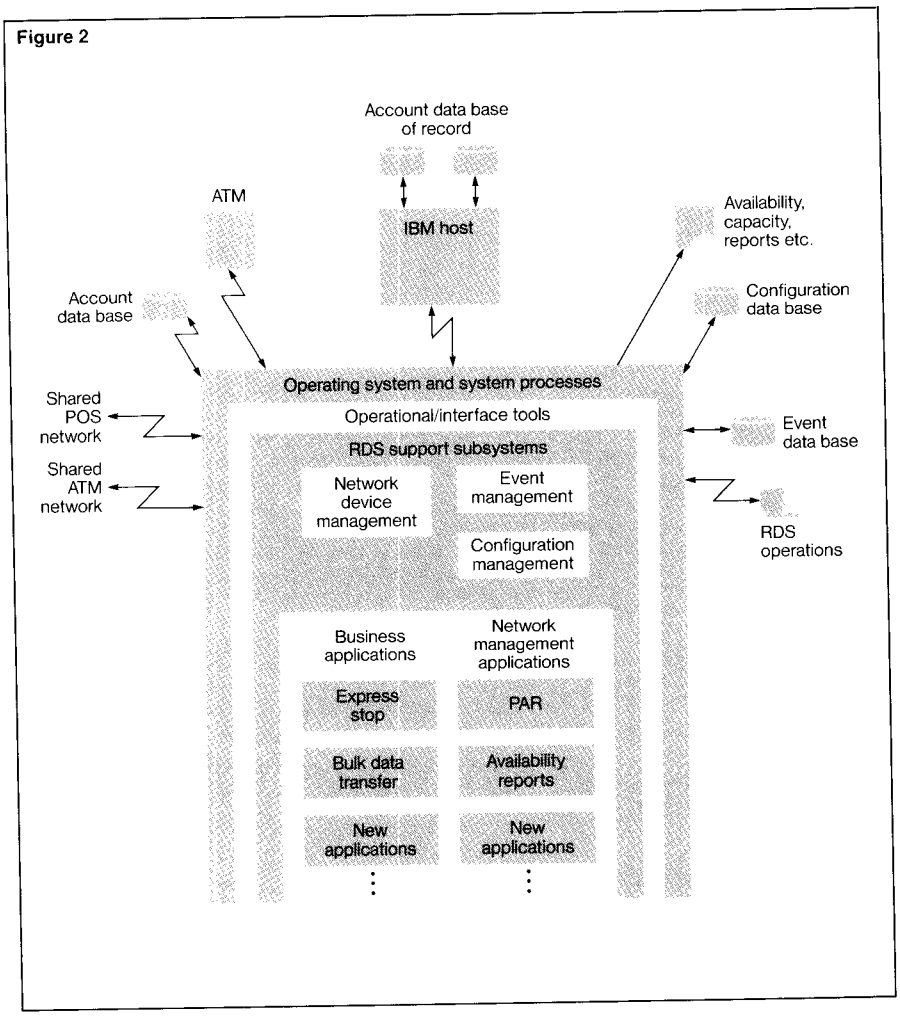


Figure 2.
RDS software architecture. Both business and network management applications use the RDS facilities and Tandem standard software to perform their tasks. Adding new applications is simplified by the layered architecture.

The selection of Tandem hardware and software and the ambitious RDS goals make more sense when viewed in the above light. The Tandem system continues to provide the high availability Express Stop is known for, and it allows for cost-effective growth by increasing capacity in manageable increments. It also provides the basis for high performance and low-cost support through use of Tandem's standard OLTP software products, and it results in basic data integrity without the need for user-written recovery routines (TMF).

RDS Architecture

RDS is a set of hardware, software, policies, and procedures to deliver retail bank products and services. The RDS architecture defines a framework for long-term growth to meet the needs of current and future applications. The architecture can be viewed as a collection of concentric frames, each providing a different set of functions. As Figure 2 shows, the outer layers are vendor-supported system processes and tools, whereas the inner two layers are customer-developed applications and support subsystems. A benefit of this modular approach is the shielding of the application developer from much of the device- and system-level programming. Additionally, vendor enhancements and performance improvements should not require change to the application programs.

RDS addresses two major functions: processing retail business transactions and managing and operating the network. A different set of modules from each RDS layer is invoked during the execution of any application. For example, a directly attached ATM connects to the RDS host through the SNAX access method of the system process layer. SNAX/HLS of the interface tools layer provides more communications services, such as session management. Transaction threads are managed by the PATHWAY IDS subsystem of the operational/interface tools layer. Device-specific characteristics, such as the formatting of ATM messages, are handled by the RDS support subsystem, network device management (NDM). The Express Stop application is then concerned mostly with updating account balances, authorizing withdrawal amounts, and logging transactions.

Other business functions will be added, including debit transactions from sources other than directly attached ATMs. For example, a transaction may be received by RDS from a travelling Wells Fargo customer accessing his or her account through an ATM on a shared network, such as MasterTeller. In this case, a different access method (ENVOY™) and interface tool (General Device Support, or GDS) will be invoked, while the same business application code, Express Stop, will perform the business function. In a similar way, point-of-sale transactions may be received from the INTERLINK network and executed by the same Express Stop application as simply another debit transaction.

Thus, other business support applications can be added with minimal impact to the RDS support subsystems, making new services a realistic and cost-effective part of Wells Fargo's strategy for growth in retail systems.

A different set of applications exists to ensure the availability of the system and devices. For example, the Problem Analysis and Resolution (PAR) subsystem notifies operators of potential problems, attempts to resolve the problem automatically, and recommends steps to be taken to close the problem. Like the business applications, this system's application relies on the outer layers, especially configuration and event management, to carry out its tasks.

For example, an ATM exists within the overall system as a collection of various software and hardware elements. From different viewpoints, it has different "aliases." A customer reporting a problem might refer to the ATM by its geographical location (e.g., the ATM on the corner of Market and Fourth). Bank personnel track ATMs by branch numbers (Accounting Units), and the phone company knows the same device by a circuit and drop number. To the systems analyst, it is related to the controller and port number, the SNA logical and physical unit number, and the PATHWAY terminal name. A problem with an ATM can cause event messages to be generated by many elements in the system. Configuration management manages the complex set of aliases and their relationships so that PAR can relate messages about a single problem when they come from different sources.

Tandem's Commitment to RDS

Because RDS represents a classic example of an OLTP application, Tandem has worked closely with Wells Fargo to judge how well the standard products perform under real-world conditions. From this experience, certain functional deficiencies and performance flaws became apparent in parts of the Tandem system. Tandem Software Development responded by making the needed enhancements to current products and by using RDS requirements to improve the design of products still being defined. These products and enhancements will be the foundation for RDS and all like OLTP applications, now, and in the future. Some of them are listed below:

- Network Systems Management (NSM) Programmatic Interface allows for integrated command and control of Tandem and user subsystems.
- Event Management System (EMS) supports high-volume event handling, safe storage of events, and filtering on events and event distribution.
- Tandem Application Command Language (TACL) includes support for both NSM and EMS, as well as data structures and PATHWAY server capability.
- GDS allows users to customize interfaces between non-SNA devices or networks and PATHWAY/IDS.
- PATHWAY includes IDS message section enhancements for support of ATMs, CONTROL 26 for prevention of data loss on a cancel to HLS, variable timeouts on ACCEPT, bit-mapped fields for ATM control, and unsolicited message handling.
- DP2 has undergone performance improvements and TMF supports an increased number of transactions per CPU.
- FOX provides for high bandwidth network communications.
- SNAX improvements reduce the cost of polling with 6204 controllers and prevent resource depletion in high stress situations. Communications network management information is also available via a new programmatic interface.
- SNAX/HLS includes a large-message option for the ATM session BIND, operations access to the application, and documented user exits.

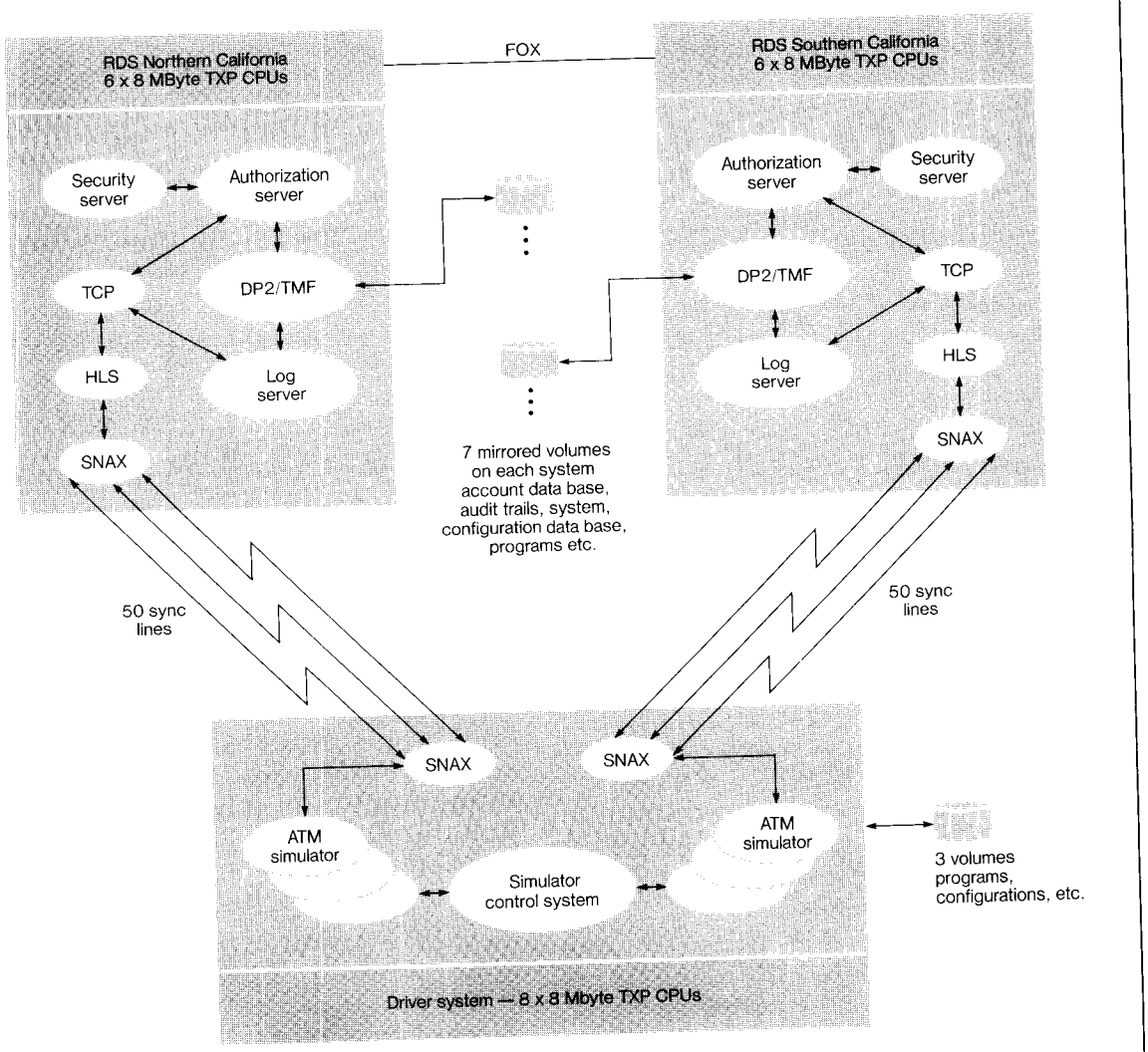
Stressing the Application and Measuring Performance

Because of the scope of RDS (24 hours, more than 1100 ATMs, all new software), Wells Fargo saw the requirement for a facility that would allow real-world stressing and measurements. Tandem saw the opportunity to evaluate and potentially improve the performance of its products in a classic yet stressful OLTP environment. Thus, a joint Tandem and Wells Fargo project, code-named Hold-up, was created.

Figure 3.

RDS benchmark configuration. The benchmark systems used up to 100 physical lines and modems which resulted in an ATM configuration larger than the production Express Stop before the Crocker buyout.

Figure 3



Hold-up consisted of a large hardware facility and analysts from both Tandem and Wells Fargo. The charter of the project team was to ensure that RDS achieved the highest possible level of performance. This would be accomplished by measuring and evaluating the RDS application under various stress conditions, recommending changes to Wells Fargo's RDS developers, and forwarding vendor product problems to Tandem's Software Development. Hold-up was built on the premise that the measure of a system's performance is not limited to throughput or response time, but can be defined as meeting the following requirements:

- Transaction and data integrity.
- Required throughput.
- Required response time.
- Required functionality.
- Easy maintenance and enhancement.
- Flexibility for expandability and deployment.
- Required availability.
- Time-critical recoveries and starts.

With this expanded definition of performance, Hold-up personnel established a number of tests to be run, including product integration tests, volume tests, start-up tests, and failure tests. To achieve throughput and response-time goals, a series of benchmarks were performed. A detailed description of the benchmarks and their results follow.

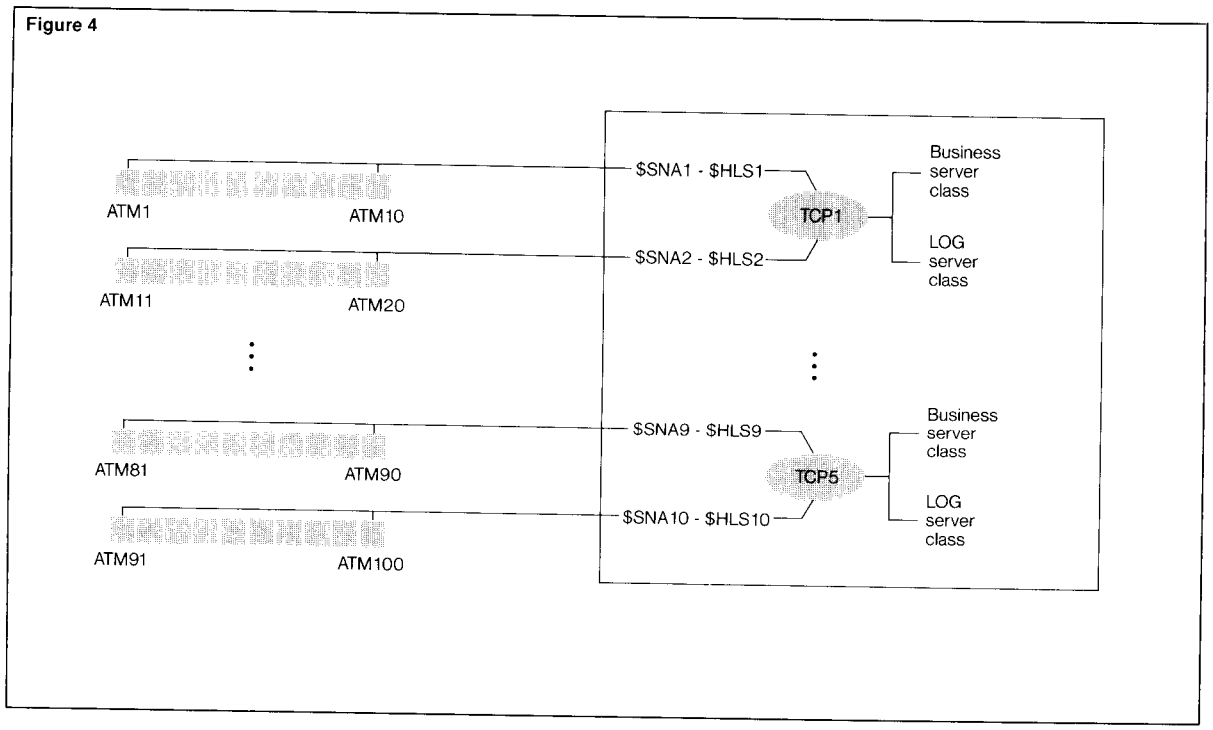


Figure 4.
Mapping of PATHWAY, HLS, and SNAX.

The RDS Benchmarks

The Benchmark Configuration

The Hold-up facility was set up to mirror the Express Stop ATM network envisioned over the next few years. The number of ATMs, for example, was set 25% higher than existed at Hold-up's inception (before Crocker's network was merged). For the prototype measurements, two host nodes were used to drive 1000 simulated ATMs. Most of the application benchmark runs used only one node (500 ATMs), as the difference in CPU time per transaction was limited to an easily identified amount attributable to network transactions. Figure 3 describes the configuration for the NonStop TXP application benchmark. The NonStop VLX system measurements were run on a four-processor single-node configuration which replicated the peripheral subsystems of the NonStop TXP system configuration to the extent possible.

Some of the relevant communications characteristics were as follows:

- Line speed was 2400 bps, full duplex.
- Average addresses on the poll list were 10.
- Poll interval was 2 seconds.
- The SNA primary and secondary response protocol was definite response mode.
- Average bytes per message: transaction request was 57, transaction reply was 167, ATM status was 30. Each message was followed by a 9-byte SNA response.

The driver, or remote terminal emulator, was an eight-processor NonStop TXP system connected by 100 lines with 2400-bps modem eliminators. The entire system was dedicated to simulating the behavior of 1000 IBM 3624 SNA ATMs (500 for single-host node tests). Each host system ran 50 SNAX line handlers, one for each of the multidrop ATM lines. Each line supported an average of ten ATMs, though the mix reflected the real Express Stop ATM network and ranged from 8 to 15.

The decision of how many ATMs to have on a line was made depending on the transaction submission rate for a given set of ATMs. Since each ATM is a single SNA physical unit, or PU (the 3624 does not support multiple logical units per PU), supporting more than 13 ATMs on a line was found to create response-time delays due to the length of the poll list and the time to service other ATMs.

For ease of configuration a one-to-one correspondence was made between a SNAX line and an HLS process (Figure 4). Had the need arisen, the number of HLS processes could have been reduced to decrease the memory usage and the number of process control blocks as HLS can easily handle more than ten terminals per process.

Figure 5

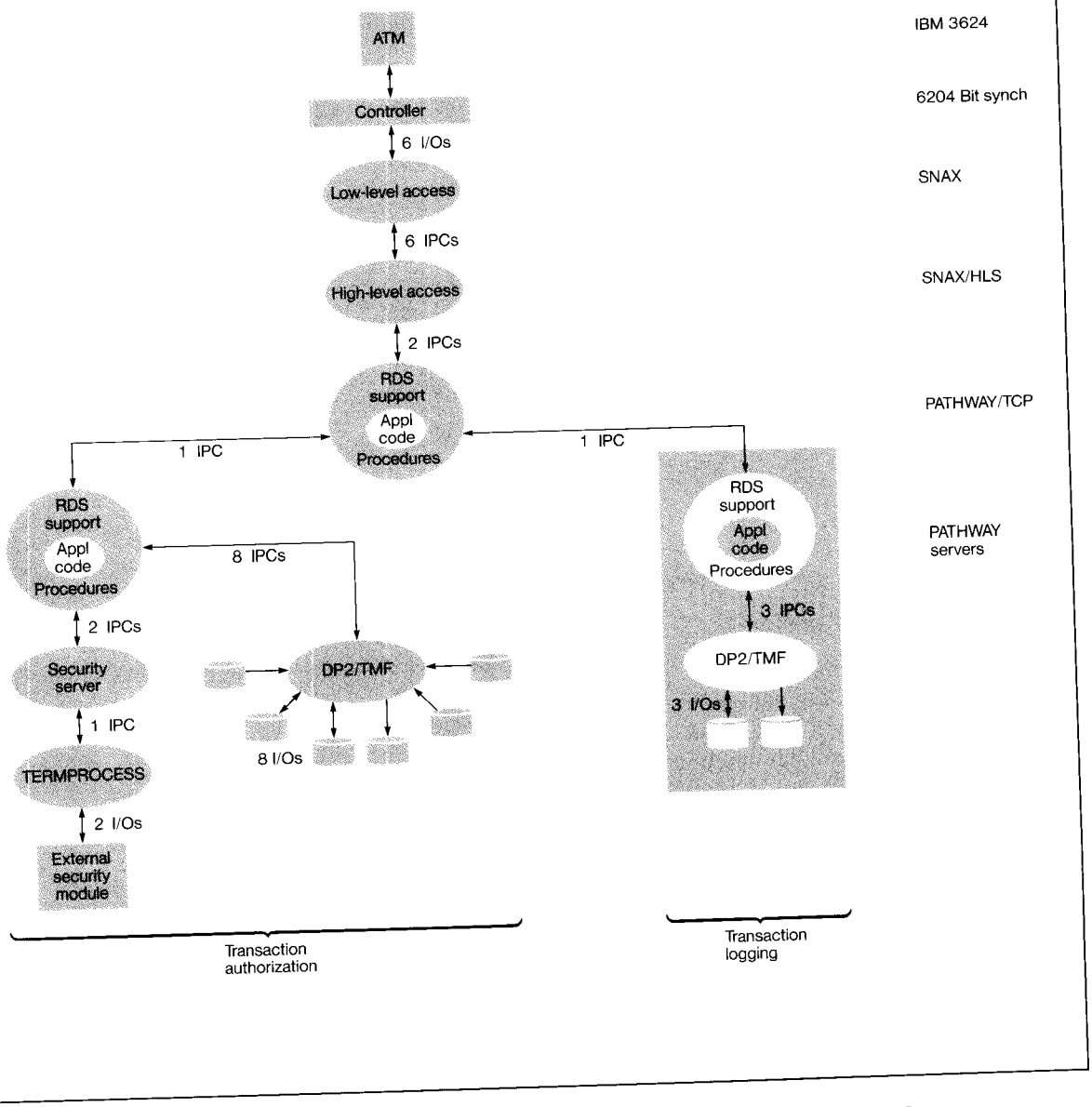


Figure 5.
The RDS ATM withdrawal transaction. Logical control of the transaction is exercised by the application residing within the PATHWAY TCP. Because SNAX, SNAX/HLS, DP2, TMF, and TERMPROCESS are all employed during the life of a transaction, a minimum of 23 interprocess communications (IPCs) are required. However, the steps in the shaded box to the right (transaction logging) do not contribute to the user-perceived response time.

Five PATHWAY environments with five Terminal Control Processes (TCPs) each were configured for the NonStop TXP application benchmark, resulting in 25 TCPs total. Each TCP controlled 20 ATMs. For a given withdrawal transaction, a requester could access two servers (plus one follow-on server) and the servers eight files (see tuning section on page 112).

What Is the ATM Transaction?

The RDS application provides for a variety of transactions to be performed on an IBM 3624 ATM: cash withdrawals from multiple accounts including multiple checking, savings, and MasterCard/VISA accounts; quick withdrawal from checking; deposits to multiple accounts; transfers; loan and credit card payments; and balance inquiries. Figure 5 represents a typical withdrawal transaction.

The transaction request is gathered at the ATM. A sequence number is generated by the ATM, retained in nonvolatile memory, and the request (with debit card information) is passed along to the host.

Figure 6

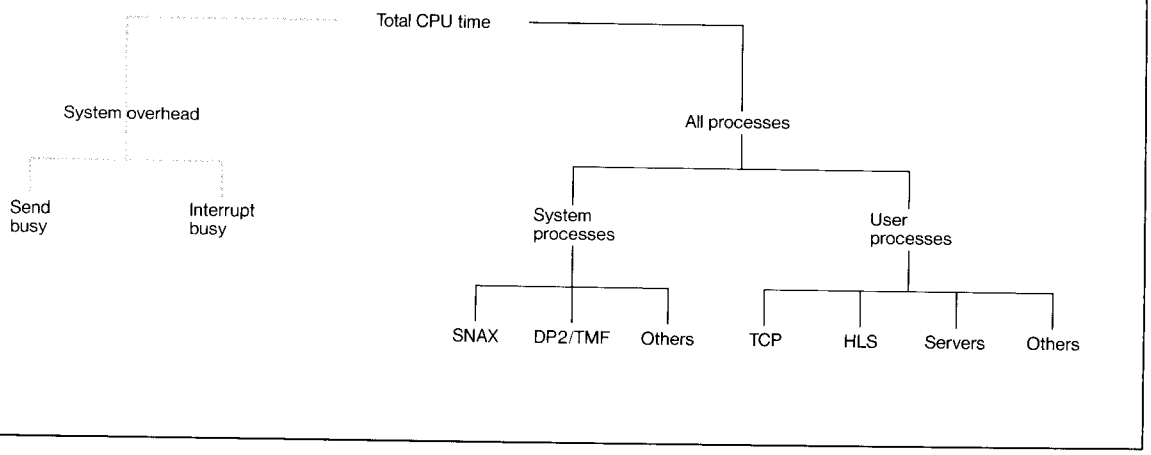


Figure 6.
Measurement entities. The CPU time to execute a transaction can be broken into distinct elements, allowing the analyst to pinpoint different processes or subsystems as they increase or decrease their share of the total CPU cost. In most cases, over 50% of the cost of the RDS transaction was attributable to the TCP and disk processes.

The RDS application receives the request and starts a TMF transaction. The appropriate business application server is called so that account information is read and/or written and the transaction sequence number is saved in the recovery file.

The application delivers the reply to the ATM and waits for the status message. The ATM pays out money, "commits" to the transaction, and replies (sequenced) to the application.

Upon receipt of a "committed" transaction, the log server writes a transaction journal entry and updates the ATM cash settlement file. The application ends the TMF transaction.

Although Figure 5 outlines a withdrawal transaction, measurements given below are for an "average" transaction where the transaction mix is:

- 50% withdrawal transactions with 11 disk I/Os.
- 25% deposit transactions with 9 disk I/Os.
- 5% transfer transactions with 14 disk I/Os.
- 20% inquiry transactions with 7 disk I/Os.

What Was Measured?

Because of the period of time in which the benchmarks were performed, two different measurement tools were used. For the prototype measurements XRAY was used; whereas by the time the application was ready, MEASURE had been released. The elements measured are shown in Figure 6.

In Figure 6, *system overhead* is calculated as the difference between total CPU time and all processes, since interrupt busy and send busy are overlapping counters. Note also the specific processes represented by the following elements:

- *SNAX* represents the SNAX line handler, not including the SNAX controlling process, $SSCP.
- *DP2/TMF* represents the aggregate of all disk processes, both SYSTEM and DATA BASE volumes, and the TMF Monitor Process.
- *Others* (under system processes) represents GUARDIAN 90 system processes such as the MONITOR process, IPB process, FOX line handler, and the MEASURE control processes.
- *TCP* represents PATHWAY TCP processes, thus SCREEN COBOL requester activity.
- *HLS* represents all HLS processes.
- *Servers* represents all user-written processes, both business and user facilities.
- *Others* (under user processes) represents any other process such as COMINT, CMP, and PATHMON.

Figure 7

	Test 1 B10	Test 2 B10	Test 3 B30	Test 4 B30
Total NonStop TXP milliseconds	590	494	435	419
System overhead (includes all other processes and XRAY)	118	101	75	59
SNAX/HLS processes	48	44	45	42
SNAX line handlers	152	55	54	51
PATHWAY TCP	122	135	138	136
TMF/disk processes	112	122	89	97
Application servers	38	37	34	34
Average response time (seconds)	NA	1.0	1.2	2.3
95% transaction response time (seconds)	NA	<3.0	<3.0	<5.5
CPU transactions/second	1.3	1.5	1.8	2.1
Total CPU utilization	77%	74%	78%	88%

Figure 7.
Results of RDS prototype benchmark. During the course of the benchmark, the cost of the prototype transaction was reduced from 590 to 419 CPU ms. The bulk of this 30% reduction came from enhancements to the GUARDIAN 90 Message System (system overhead), SNAX, and the disk processes.

Calculating Performance: Response Time, Throughput, and Transaction Resource

In the RDS benchmark, one purpose of the measurements was to determine the throughput obtainable with proper application design and system tuning, while maintaining response-time goals and without unnecessarily sacrificing application functionality. To understand the results obtained, the terms *response time*, *throughput*, and *transaction resource* require some definition.

In general, *response time* is a measurement of how long a user has to wait between entering a transaction and first seeing a response at the terminal. Traditionally, this amount of time is broken into two parts: communications and host time. Availability of response-time information was a requirement of RDS so that network implementors would know that customer service goals were being met. For the final application, HLS user exits were written to calculate a portion of the host response time. Since the start time was from the point that a transaction request was sent to the PATHWAY SCREEN COBOL requester and the stop time was the point that a SCREEN COBOL reply was received by HLS, this response-time figure is more accurately described as *application response time*.

Host or application response time is only useful in a performance evaluation when viewed in conjunction with *throughput* (i.e., the number of transactions that a CPU can process per second). This is because a change in the transaction rate may affect the response time. For example, if the transaction rate is too high, queuing may cause the response time to degrade. Adding more CPUs may reduce the response time but also make the application unnecessarily costly. Balancing and tuning are necessary to find the proper balance between throughput and response time.

Although throughput and response time are related in this key way, what is considered a *transaction* in response-time measurements may be viewed differently in throughput calculations. The throughput must incorporate the cost of the total transaction, whereas the response time may, by design, refer only to a portion of the total life of the transaction. Such is the case with the RDS response-time measurements. Since the customer could delay an unpredictable amount of time before the status would be sent, the processing of this third step of the transaction is not included in the response time, yet is included in the throughput as part of the cost of a transaction.

For this benchmark, *transaction resource* was calculated by dividing the total cost in CPU milliseconds by the number of transactions completed during the measured period. For example, in a test period of ten seconds (10,000 ms) with two CPUs running at 60%, a total of 12,000 ms of CPU time, or

$$2(0.60 * 10{,}000) = 12{,}000 \text{ ms},$$

would be taken to execute the transactions. If 30 transactions were completed during the test window, one could say that it takes 12,000 ms divided by 30 transactions, or 400 CPU ms, to perform a transaction.

Knowing the resource cost of a transaction allows one to calculate the potential throughput at a given CPU utilization. For example, at 60% utilization, a single CPU has 600 ms available. Since the transaction takes 400 ms to execute, the throughput achieved would be 1.5 transactions per second per CPU. In a real measurement, the number should match the total number of transactions (e.g., 30) divided by the total period of the test (10 seconds) divided by the number of CPUs (2).

Results of the Prototype Application Benchmark

While the software targeted for production was still in development, a prototype application that supported the same RDS transaction was used for the initial benchmark. The benchmark consisted of a series of tests using consecutive B-series GUARDIAN 90 software releases. This was done to isolate which product changes or enhancements were contributing either positively or negatively to the overall performance and cost of a transaction. A comparison of the key test results is shown in Figure 7.

Test 1: B10. This was the initial measurement using the RDS prototype software and Tandem products available as of the B10 release. The cost attributed to SNAX (152 ms) was viewed with some alarm. After investigation it turned out to be the polling overhead when running at low transaction-per-line rates. To confirm this, a special test outside of the benchmark was run using the 6100 controller and the figure was reduced to about 20 ms. This led to an investigation of the SNAX 6204 polling algorithm.

Test 2: B10. In this second test, a significant decrease in the cost per transaction for SNAX line handlers was the result of the inclusion of a preliminary version of a future B-series release for 6204 SNAX, which included an enhanced polling algorithm. This was particularly beneficial in configurations such as Wells Fargo's where the transaction rate for any given device (polled station) was low enough to cause relatively high polling overhead per transaction. The result was a decrease to 55 ms of almost 100 ms, or 16% for the overall transaction.[4]

The full two-node configuration was tested in this run. An increase in CPU ms attributable to TMF, about 7 ms, was the result of network transactions being performed.

The increase of 13 ms for the PATHWAY TCP reflects application changes for added functionality. For future measurements of the prototype, the application was frozen.

Tests 3 and 4: B30. The final benchmarks of the prototype application using B30 software produced a new low of 435 ms per transaction for the NonStop TXP. The dramatic improvement realized in the system overhead figure was the result of B30 Message System streamlining. The interprocess message protocol was enhanced to use a "fast select" method of passing buffers between the Linker and Listener (see Dave Kinkade's article in this issue, "Performance Changes to the GUARDIAN 90 Message System"). The decrease in messages reduced the dispatch rate per transaction, and thus CPU Interrupt and Send Busy time.

Software developers suggest that the significant drop in utilization attributed to TMF and DP2 was partially due to the "boxcar" effect, whereby messages are batched together before being transmitted by TMF. In previous releases, TMF transmitted records one at a time. Additionally, changes were made to the cache management algorithms which may have reduced the cost of flushing cache to disk.

[4]The SNAX cost per transaction varied according to the transaction rate per line, which for Test 2 was an average of 0.18 transactions per second per line.

A significant reduction in the number of I/Os required for a network transaction saved about 8 ms of the TMF cost per transaction.

During Test 4, while trying to reach two transactions per second per CPU, the CPU utilization went well above 80%, demonstrating that far higher CPU utilization could be achieved than had been shown in earlier tests. When the transaction rate was raised to the point at which CPU utilization reached 88%, queuing started to become a negative factor and response time degraded (2.3 seconds average).

The most interesting finding of the prototype tests was the reduction in CPU cost per transaction when running at higher transaction rates. For example, a NonStop TXP system prototype transaction cost 435 ms at 78% utilization, but only 419 ms at 88% utilization. Performance analysts feel this was largely the result of the behavior of two products: DP2 and SNAX. In high transaction environments, DP2 performs fewer physical I/Os per transaction since more writes to cache would be made in the fixed period between physical I/Os to disk. A higher message rate also means that SNAX transmits and receives more messages, thus reducing the amount of time spent in idle polling.

Running the benchmark on NonStop VLX versus NonStop TXP processors, the same application took exactly one-third less CPU time to execute a transaction.

Tuning Improves Throughput

When the B10 benchmark was first run with the full system load of 1000 ATMs (500 per node), the original cost per transaction was significantly higher than is shown in Figure 7. Configuration tuning was found to reduce the cost per transaction by 35 to 40 ms. The major factor turned out to be the location of processes that had a high rate of interprocess communication between each other. SNAX and HLS, for example, send at least six messages to each other for each transaction. Therefore, to keep the message system overhead to a minimum, HLS and SNAX pairs were placed in the same CPU.

Another method found to reduce file system overhead was the clustering of records that were accessed during a single transaction on the same volume. In this way, TMF would send fewer interprocess messages to the disk processes, fewer interprocess messages from those disk processes to the TMF audit disk process, and fewer checkpoints between disk processes at each ENDTRANSACTION. (All records of the transaction would be originating from fewer disk volumes.) Effectively, this gives the user implicit control over the number of TMF IPCs attributed to a transaction.

A third step taken to improve throughput involved the allocation of disk cache. With the release of DP2 and its buffered cache memory, physical I/Os have been reduced since changed data blocks can now remain in memory (cache) for longer periods of time (Schachter, 1985). In fact, performance specialists can actually control the number of physical I/Os, or flushes of changed data blocks, by setting cache large enough to accommodate the number of data blocks that will be changed in the given period between automatic DP2 control points. Since the disk processes write all changed pages to disk at these control points anyway, throughput will be improved if during the intervals the data block is accessible in cache rather than on disk.

For the RDS application, the optimum allocation turned out to be 1.6 Mbytes of cache per disk volume. This was sufficient to contain all of the index blocks for the key-sequenced files on a volume and the maximum number of data blocks that would be updated between the DP2 control points, i.e., when "dirty" pages are flushed to disk.

It was also determined that, for this application, the optimum block size was 2048 bytes. Other block sizes tended to drive the controller and disk busy rates higher.

It should be noted that the above information does not constitute a tuning recommendation. The actions taken by Wells Fargo took into consideration their unique set of requirements and may not be suitable for all applications.

Results of the "Real" Application Benchmark

When the benchmarks were run using the actual ATM application, as planned for use in the production Express Stop network, the CPU cost for a transaction was higher than for the prototype. The 13% increase, nearly 60 ms, resulted from increased complexity and functionality in the application.

When the benchmark was run on NonStop VLX instead of NonStop TXP processors, the same application took exactly one-third less CPU time to execute a transaction (Table 4). This was evidenced by the four-processor NonStop VLX system completing the same number of transactions as a six-processor NonStop TXP system.

RDS and Debit-credit Benchmark Results

The most widely publicized measures of system performance for OLTP applications have been benchmarks using the standard "debit-credit" transaction. In fact, this has been the method used to rate the NonStop TXP system as a 4.5-transaction-per-second CPU. The Wells Fargo benchmark achieved transaction rates far less than this, yet it was a major achievement in that the RDS application performs real business functions for one of the leading retail banks in the country. The debit-credit transaction is, by design, an artificial transaction. Its relative simplicity makes it ideal for comparing the performance of unlike systems, not for modeling business applications.

As Table 5 shows, an RDS transaction is about three times as costly in CPU time as the debit-credit transaction. By analyzing the elements of the two transactions, the reasons for the cost difference are better understood.

Table 4.
Results of the RDS application benchmark.

	NonStop TXP B30 application	Percent transaction	NonStop VLX B30 application	Percent transaction
Total system milliseconds	491		328	
System overhead	74	15%	39	12%
TMF/disk processes	104	20%	66	20%
SNAX line handlers	49	11%	35	11%
PATHWAY TCP	154	32%	115	35%
HLS processes	49	9%	32	10%
Application servers	54	11%	37	11%
Other processes	7	2%	4	1%
Average response time (seconds)	1.3		0.9	
95% transaction response time	<2.8		<2.1	
CPU transactions/second	2.0		3.0	
Total CPU utilization	96%		97%	

The debit-credit transaction is run using the X.25 protocol. X.25 is a protocol most commonly used for Public Data Network access, in which a typical transaction is inquiry-response. Wells Fargo chose to employ the SNA protocol because of the wide use it enjoys in OLTP environments and because of the potential functionality it provides for data integrity and network management.

An example of how this choice adds to the cost of an RDS transaction can be seen in RDS's use of the SNA definite response mode for both the primary and secondary logical units. (The IBM 3624 operates only in definite response mode.) For each of the three steps in the transaction, an SNA positive response (9 bytes) is sent. This doubles the number of messages sent between the ATM and the host.

Table 5.
Cost and performance comparison of the debit-credit vs. RDS application.

	ET1 NonStop TXP	RDS NonStop TXP
CPUs	4	6
CPU milliseconds/transaction	160	491
Transaction rate	22.7	12.0
Transactions/second/CPU	5.7	2.0
CPU utilization	91%	96%
Average response time (seconds)	1.5	1.3
97% transaction response time	<4.1	<3.5

Significantly higher file I/O is performed in the RDS application than in the debit-credit transaction. This is due to the requirement for:

- Geographic independence of the card from the node.
- Multiple accounts linked to a single card.
- Posting of balance changes to the memo account (not on-line posting).
- Maintainance of daily limits on cards.
- Key-sequenced transaction log for random access and manageability of multiple logs.
- Reliable message delivery to ATM (e.g., recovery of three-step transaction and recovery from double failure).

Not only does this high level of functionality significantly increase disk activity, but the front-end application processing (formatting and decision making) is increased, costing more CPU cycles for the TCP.

Finally, the RDS benchmark closely modelled the load variations experienced in a real-world network. Specifically, transaction arrival rates varied from one line to the next and transactions resulted in "uneven" access of disk volumes. The debit-credit benchmark, on the other hand, does not vary the arrival rate from a given terminal, and access of disk volumes is evenly distributed.

Conclusion

The Retail Delivery System comprises a functional application and operations environment for running a retail network in the rapidly changing and highly competitive banking world. The special cooperation between vendor and user in the design and testing of RDS made the RDS benchmark a milestone for both Tandem and Wells Fargo. Because of the number of standard software products used, it demonstrated that Tandem systems provide an ideal environment for developing high volume large network, OLTP applications that perform *real* business functions. By addressing the total cost of a transaction, Wells Fargo ensured that their critical business application will continue to meet the competitive demands of the 1990s without sacrificing their high availability and performance standards. When calculated to include the cost of ongoing maintenance, growth and high volume, and failure then the Wells Fargo RDS application run on Tandem systems may well provide the lowest-cost, high-function transaction in the industry today.

References

Anon, et al. 1985. A Measure of Transaction Processing Powe *Datamation*. Vol. 31, No. 7.

Arthur Andersen & Co. 1983. *New Dimensions in Banking: Managing the Strategic Position*. Bank Administration Institut

Bank Network News. 1985. *1986 EFT Network Data Book*. September 26.

Blake, R. 1979. Tailor: A Simple Model That Works. 1979 Conference on Simulation, Measurement and Modeling of Computer Systems. ACM. Boulder, Colorado.

Dow Jones. 1985–1986. Dow Jones News Retrieval.

EFT Report. 1985. *Newsletter of Electronic Funds Transfer*. Phillips Publishing, Inc. Vol. 8, No. 7.

Schachter, T. 1985. DP2's Efficient Use of Cache. *Tandem Systems Review*. Vol. 1, No. 2.

The 1000 Largest U.S. Banks. 1986. Sheshunoff and Company, Inc.

Acknowledgments

A large part of the technical material in this article is derived from work done by Wells Fargo personnel. Special credit goes to Scott Alexander of Wells Fargo who provided most of the measurements and analysis. The authors also wish to thank Pe Homan and Jim Gray for use of their comments made while performing an RDS data-base review and for their excellent suggestions upon reviewing this article.

Nick Cabell joined Tandem in March 1980 as a data communica tions course developer and instructor. In April 1983, he became manager of the Marketing Technical Support Data Communications Group. Since October 1984, he has been with the Wells Fargo project providing support in the areas of SNA and communications.

Duncan Mackie joined Tandem in San Mateo as a senior systems analyst in April 1980. He became branch systems manage of the Silicon Valley Branch in February 1981 and district systems manager of the Silicon Valley District in September 1981. He transferred to the Wells Fargo Project in October 1984, wher he is currently the project manager.

TANDEM PUBLICATIONS ORDER FORM

The *Tandem Systems Review* and the Tandem Application Monograph Series are combined in one free subscription. Use this form to subscribe, change a subscription, and order back copies.

For requests *within the U.S.*, send this form to:

Tandem Computers Incorporated
Tandem Systems Review
1309 South Mary Avenue, MS 5-04
Sunnyvale, CA 94087

For requests *outside the U.S.*, send this form to your local Tandem sales office.

Check the appropriate box(es):

- ☐ New subscription (# of copies desired _____)
- ☐ Subscription change (# of copies desired _____)
- ☐ Request for back copies. (Shipment subject to availability.)

Print your current address here:

_____
ADDRESS

_____

_____

_____

_____

_____
ATTENTION

_____
PHONE NUMBER (U.S.)

If your address has changed, print the old one here:

_____
ADDRESS

_____

_____

_____

_____

_____
ATTENTION

_____
PHONE NUMBER (U.S.)

To order back copies, write the number of copies next to the title(s) below.

NUMBER OF COPIES

Tandem Journal

_____ Part No. 83930, Vol. 1, No. 1, Fall 1983

_____ Part No. 83931, Vol. 2, No. 1, Winter 1984

_____ Part No. 83932, Vol. 2, No. 2, Spring 1984

_____ Part No. 83933, Vol. 2, No. 3, Summer 1984

Tandem Systems Review

_____ Part No. 83935, Vol. 1, No. 2, June 1985

_____ Part No. 83936, Vol. 2, No. 1, February 1986

_____ Part No. 83937, Vol. 2, No. 2, June 1986

_____ Part No. 83938, Vol. 2, No. 3, December 1986

Tandem Application Monograph Series

_____ Part No. 83900, *Developing TMF-Protected Application Software,* March 1983, AM-005

_____ Part No. 83901, *Designing a Tandem/Word Processor Interface,* March 1983, AM-006

_____ Part No. 83902, *Integrating Corporate Information Systems: The Intelligent-Network Strategy,* March 1983, AM-007

_____ Part No. 83903, *Application Data Base Design in a Tandem Environment,* August 1983

_____ Part No. 83904, *Capacity Planning for Tandem Computer Systems,* October 1984

_____ Part No. 83905, *Sociable Systems: A Look at the Tandem Corporate Network,* May 1985

_____ Part No. 83906, *Transaction Processing on the Tandem NonStop Computer: Requestor/Server Structures,* January 1982, SEDS-001

_____ Part No. 83907, *Designing a Network-Based Transaction-Processing System,* April 1982, SEDS-002

_____ Part No. 83908, *A Close Look at PATHWAY,* June 1982, SEDS-003

_____ Part No. 83909, *A Multi-Function Network for Business Automation,* May 1982, SEDS-004

TANDEM EMPLOYEES: PLEASE ORDER YOUR COPIES THROUGH YOUR MARKETING LITERATURE COORDINATOR.

TANDEMCOMPUTERS

400100 12/86 Printed in USA

Part No. 83938