TANDEM JOURNAL

The Model 6VI Voice Input Option: Its Design and Implementation

The NonStop TXP Processor: A Powerful Design for On-line Transaction Processing

Optimizing Sequential Processing on the Tandem System

T A N D E M J O U R N A L

Volume 2, Number 3, Summer 1984

Editor
Carolyn Turnbull White

Associate Editor
Susan Thompson

Production Editor
Anita Van Auken

Technical Advisor
Bjoern Lindberg

Design
Craig Frazier Design

Cover Art
Craig Frazier

The Tandem Journal is published by Tandem Computers Incorporated.

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

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

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

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

The Model 6VI Voice Input Option: Its Design and Implementation

The marriage of speech recognition with the transaction-oriented terminal is an advance in terminal interaction that is both user friendly and cost effective. It is particularly useful for those who must use their hands and eyes to perform other tasks while interacting with the terminal. Speech recognition allows users with "busy hands and busy eyes" to enter transactions easily and quickly.

This article gives an overview of voice recognition technology and describes the Model 6VI voice input option (for the 6530 Terminal), explaining its design rationale and applicability. It discusses the problems of recognizing speech and the means of solving the problems, particularly as they are implemented in the Model 6VI. Finally, it describes the performance features of the Model 6VI.

Voice Recognition for the 6530 Terminal

Voice recognition is part of a trend to add alternate input devices to terminals. In 1983, Tandem introduced the Model 6LA/AI alternate input option for the 6530 terminal line. This option allows input from devices such as bar-code readers, optical-character readers, magnetic-card readers, and scales. The Model 6LA/AI enters information from these devices more quickly and accurately than the user can manually.

Recent advances in digital-signal microprocessing and automatic speech recognition (ASR) have made it practical to offer speech as another means of input. In April 1984, Tandem made the Model 6VI voice input option available. This addition to the 6530 terminal allows the user to enter data into a NonStop system by speaking into a microphone. The Model 6VI enhances productivity when the user is occupied with tasks that prevent efficient manual entry.

Applications

Speech is our fastest means of discourse. While a skilled typist can type about 1 word per second, speech can occur at rates from about 2.5 words per second (spontaneous speech) to 4 words per second (reading aloud). It is also nearly effortless; if you use speech to interact with a computer system, there is no need for visual or physical contact, and there are no restrictions on the use of the hands or the mobility of the body.

In a receiving-inspection station of a manufacturing facility, parts undergo an inspection before entering the manufacturing process. The inspectors handle the material (perhaps wearing gloves) and record the inspection. Using ASR, they can enter the data without removing their eyes from the material. An ASR terminal can also be equipped with a wireless transmitter to give the operators additional freedom of movement.

In pharmaceutical research, pathologists evaluating tissue slides under microscopes make constant use of their hands and eyes while inserting slides, adjusting the focus, and moving the slides under the lens to evaluate the tissue. ASR allows efficient use of the microscope while the data is recorded.

Microelectronics inspection stations require the inspectors of printed circuit boards to hold and manipulate the board under an illuminated magnifier. ASR allows them to record each observed defect without putting down the board and refocusing the eyes.

The Technology

ASR systems have a wide variety of capabilities (and concomitant costs). Figure 1 illustrates the broad classifications of speech recognition technology.

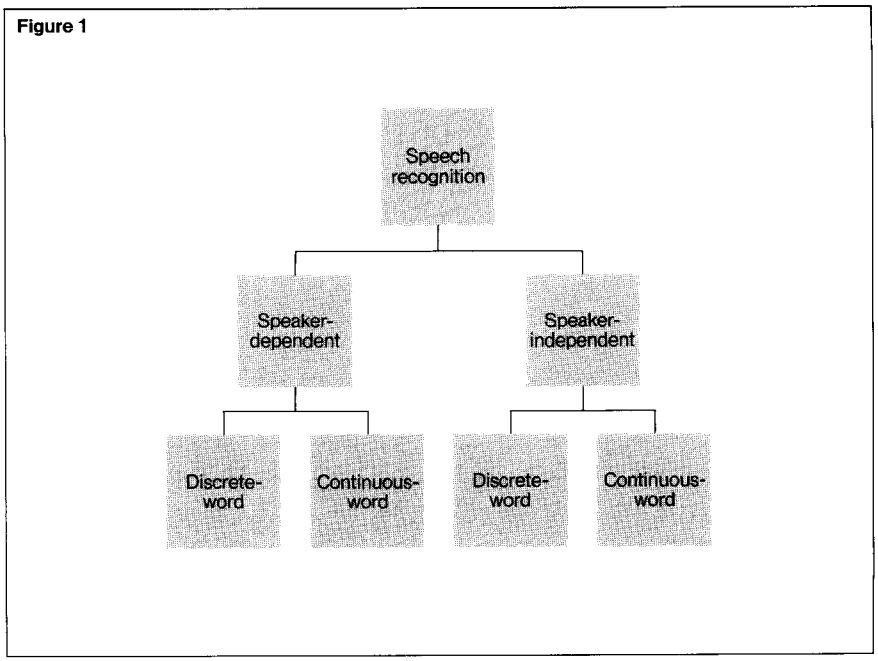
A *speaker-dependent* speech recognizer is "trained" to recognize a word or phrase as spoken by a particular person. During training, the user speaks each word in the vocabulary into the ASR microphone. This creates a digitized template (reference pattern) for each word and stores it in the recognizer's memory.

A *speaker-independent* recognizer allows designated words or phrases to be spoken without requiring the users to train the system for their individual voices. It does this by using a set of representative templates created from a statistical sampling of a large population. This population takes into account differences in accent, tonality, and speed.

Speech recognition systems are also classified as *discrete* (isolated-word) or *continuous* (connected-word). The discrete system recognizes words or phrases under two seconds in length and requires a brief pause between words to determine their boundaries. The continuous system accepts strings of words without pauses between words. However, even the few commercially available continuous-word recognition systems perform better on isolated speech than on connected speech.

Continuous-word recognition requires more processing power (with accompanying cost) than discrete-word recognition, and speaker-independent recognition requires more processing power than speaker-dependent recognition.

Figure 1



A speaker-independent, continuous-word ASR usually has a vocabulary of about 50 words. In these systems, the vendor preselects the words in the vocabulary. This has the added disadvantage of restricting the system's use to the language in which it was developed. A speaker-dependent, discrete-word system can have a vocabulary of about 200 words. In this case, the user selects the vocabulary and trains the system.

The Model 6VI is a speaker-dependent, discrete-word recognizer. It has a vocabulary of up to 200 words and/or phrases. This technology was selected because it is cost effective and has a vocabulary size that is favorable for transaction processing.

Figure 1

Automatic speech recognizers can be classified as being speaker-dependent or speaker-independent, continuous- or discrete-word recognizers.

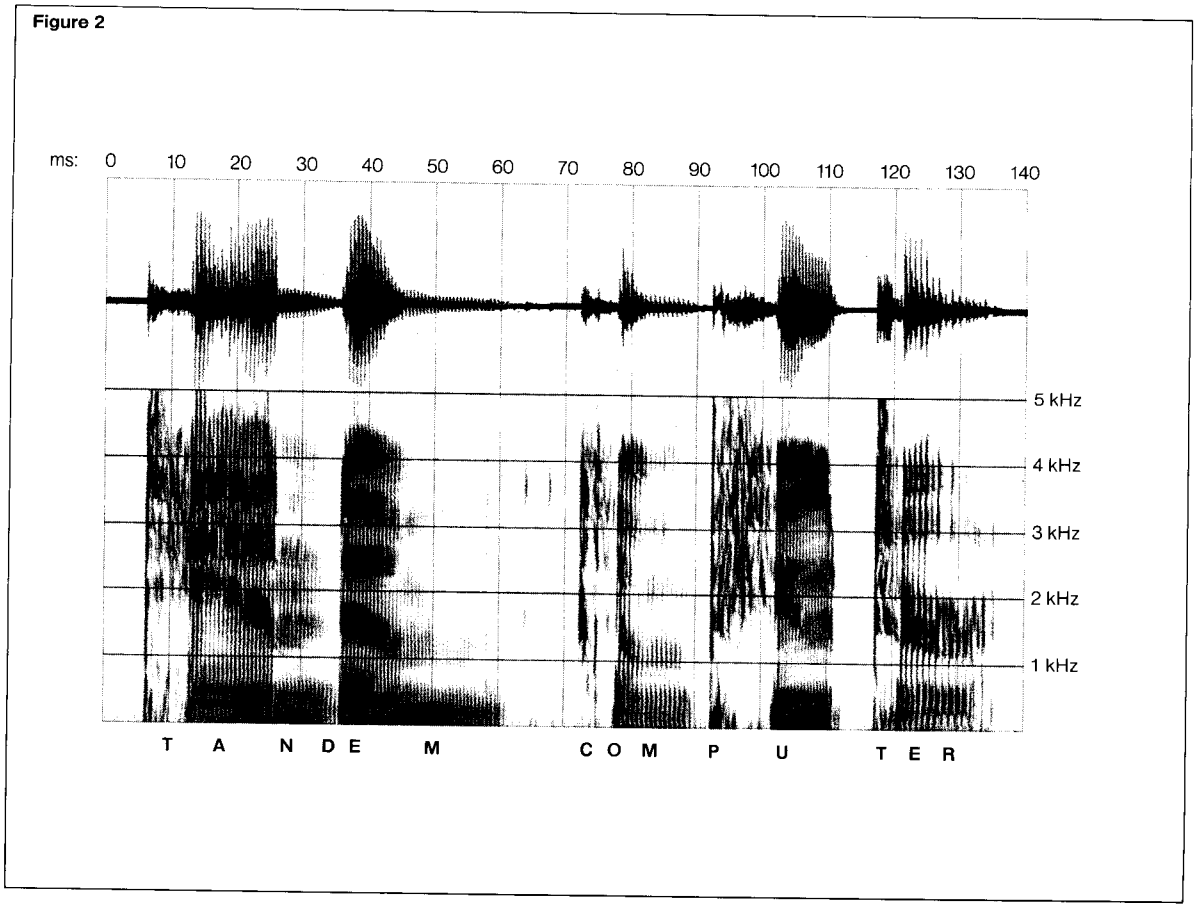
The Problem

A speech recognizer accepts human speech as input, and when it recognizes the word, performs an action such as closing a relay or routing a message to another device or host.

When a word is spoken, the movement of the vocal tract creates disturbances in the air. A microphone converts the sound waves into an analog electrical signal. The resulting speech wave consists of a complicated assembly of overlapping frequencies with amplitude (loudness) and frequency varying throughout the utterance.

Figure 2

Speech waveforms are a complicated assembly of overlapping frequencies. Here, a spectrogram produced by a spectrum analyzer displays the waveforms of the words "Tandem Computer". The top waveform is in the time domain (amplitude versus time), and the bottom waveform is in the frequency domain (frequency versus time versus amplitude). The bottom waveform shows amplitude (signal strength) as lighter or darker shades in the frequency ranges.



Reprinted with permission of Kay Elemetrics Corp., Pine Brook, N.J.

The problem lies in the processing of the speech signal. In areas such as seismic exploration, sonar detection, and engine vibration analysis, major advances have been made in hardware and processing. Similar progress has not yet been made in speech recognition, however, because speech waveforms are more complex than these waveforms and require more sophisticated hardware and processing techniques.

Transforming the Waveform into a Digital Signal

Speech produces a waveform with three dimensions: frequency, amplitude, and time. The ASR must use a mathematical technique to transform all three of these dimensions into a digital signal. Some of the mathematical techniques available for this purpose are linear predictive coding, variations of the Fast Fourier Transform (FFT), dynamic programming, and proprietary spectral-transform algorithms such as "spectral slicing".

The Model 6VI uses the technique referred to as spectral slicing. This technique splits the input speech signal into sixteen frequency bands (channels) and samples each channel periodically (every 5 ms) to measure the amplitude within each frequency range. This forms a "profile" of the significant features of the spoken word.

Accounting for Frequency Variations among Speakers

For any utterance, the two-dimensional waveform of amplitude versus time (as shown in the top half of Figure 2) varies little from speaker to speaker. The three-dimensional waveform of amplitude versus frequency versus time (as shown in the bottom half of Figure 2), however, changes significantly if the words are from a speaker of a different gender or age.

Voiced sounds are roughly periodic with fundamental frequencies of 210 Hz for men, 220 Hz for women, and 300 Hz for children. This is one factor that makes speaker-independent recognition difficult.

Determining the Boundaries of a Word

In isolated-word recognition the ASR determines the boundaries for the beginning and end of a word. After determining the word's pattern, the ASR matches it to a *template* that was made during training.

Continuous-speech recognition may also use template matching as part of the recognition process. However, its templates correspond to phonemes (the smallest units of speech) instead of words. (There are approximately 40 phonemes in the English language.)

The continuous-speech recognizer has the additional task of linking phonemes to construct valid words. The adjacent words "six seven", where the phoneme /s/ in the word "seven" combines with the end of "six", demonstrates the difficulty in distinguishing and linking phonemes.

The Solution—The Anatomy of the Model 6VI Voice Input Option

The functional components of the Model 6VI controller are shown in Figure 3, and the hardware components are shown in Figure 4. This section describes how the hardware components perform each major function.

Spectral Feature Extraction

The analog signal generated by the microphone feeds into a preamplifier. The preamplifier output then goes into a speech spectrum equalizer. The equalizer compensates for the "roll-off" (or decrease) in strength of speech frequencies above 750 Hz, adjusting the amplitude of the signal upward with increasing frequency. This removes the distortion and preserves the significant features of the signal.

The equalized signal goes either to the Automatic Level Control (ALC) or to the linear path circuitry. (The user selects one of these paths with a switch on the controller.)

The ALC circuit controls signal variations caused by variations in the microphone location and speaking effort. When selected, ALC compensates for a variation of approximately 12 dB. This is useful, for example, when the user must move around while the microphone remains in a fixed place.

When the microphone is mounted on a head-set, ALC is unnecessary. Also, in a noisy environment, ALC might cause system

Figure 3

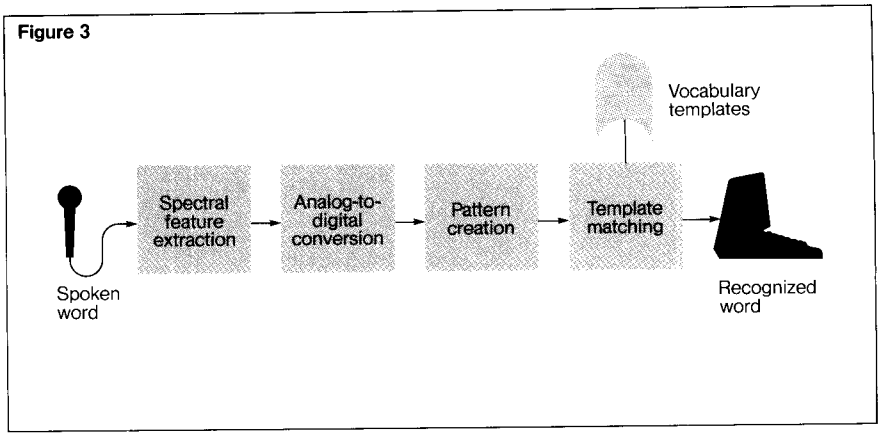


Figure 3.
A typical template-matching speech recognizer performs four basic functions to accomplish recognition: spectral feature extraction, analog-to-digital conversion, pattern creation, and matching.

Figure 4

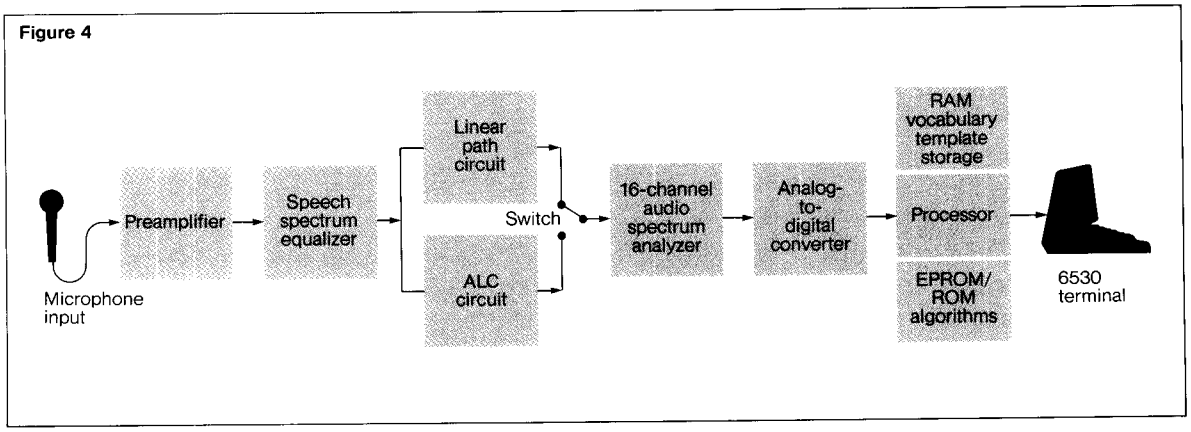


Figure 4.
Fundamental components of the voice recognition module used on the Model 6VI voice input option controller.

performance to degrade as the system tries to "hear" the background noise. Linear path circuitry considers the ambient noise level and is the best choice for a noisy environment. (The user can also adjust for background noise by setting the "gain" or the sensitivity of the microphone. This is described later under "Performance Considerations").

The signal from either the ALC path or the linear path is input to the 16 bandpass filters that make up a 16-channel spectrum analyzer. The spectrum analyzer separates the speech spectrum (200 Hz to 7 kHz) into 16 adjacent frequency bands. A 16-channel multiplexer within the spectrum analyzer chip provides spectral data for each of the 16 bands.

Analog-to-digital Conversion

The output from the spectrum analyzer is a signal of 0 to 5 V for each channel. It feeds into an 8-bit analog-to-digital (A/D) converter, which scans each of the 16-channel outputs and digitizes the signal to 8 bits (0 to 255). This sampling and conversion occurs on all 16 channels every 5 ms.

Pattern Creation

Once the onset of a word is detected, the controller uses a feature-extraction algorithm to extract the recognition information from the digitized data. When the end of the word is found, the feature data is time-normalized to compensate for differences in speech rate. The result is a 67-byte digital pattern.

Because of the time and frequency characteristics of the human voice, several methods of feature extraction are possible. They include linear predictive coding, format filtering, and digitization. The Model 6VI uses the digitization method. It encodes the extracted feature information so that approximately half of the information in the 67 bytes represents spectral slope coding (frequency information) and half represents binary spectrogram data (energy information).

During *training mode,* feature extraction creates a template (reference pattern) for each word in the vocabulary. The controller stores the templates in 20K of RAM. The user can permanently store these templates by uploading them into a file on the host computer.

Once the user creates a table of reference patterns, the Model 6VI is ready for *recognition mode.* In this mode, the system uses feature extraction to process an unknown utterance into a "token" template. The system then attempts to find a match between the token template and one of the templates made during training mode.

Matching

Each time the same word is spoken by the same person, the waveform of that word is at least slightly different. In the same way, the waveform of a word spoken during training mode is not identical to the waveform of the same word spoken in recognition mode. A matching strategy is, therefore, necessary.

Distance-measurement algorithms compute the distance between the incoming utterance and the vocabulary's reference patterns. This algorithm is important to the recognition process. The more accurate the distance measurement, the better the recognition will be. The following are some distance measurement methods:

- *Chebychev method* sums the differences between the reference patterns and the utterance.
- *Euclidean method* squares the differences between the reference patterns and the utterance.
- *Correlation measure* uses the well-known statistical technique, coefficient of correlation.
- *Polynomial measure* refines the correlation measure by rewarding small differences that are exaggerated by the correlation measure.
- *Hamming-distance method* determines the difference between the corresponding digits of two binary words. It is widely used in digital-signal processing because of its computational simplicity and high accuracy.

The Model 6VI uses the Hamming-distance method. The algorithm from this method produces a "score" indicating the similarity between the token template and a reference pattern. (A score of zero indicates no similarity, and 128 is a perfect match.) The algorithm produces one score for each reference pattern in the vocabulary.

The reference pattern having the highest score over the *recognition threshold* (RTHL) and differing from the closest runner-up by at least the *delta* is declared a match. The user-defined RTHL sets the minimum score for recognition, while the user-defined delta sets the minimum difference between the score of the recognized word and the next-highest score.

When a match is made, one of the following (depending on the host program's directive) is routed to the host:

- A user-defined ASCII string of up to 198 bytes.
- The vocabulary-word number of the recognized word, its score and delta, the user-defined ASCII string associated with the recognized word, the runner-up word number, and the ASCII string associated with the runner-up word.

If no match occurs in the first case, the terminal sounds a beep. If no match occurs in the second case, the terminal displays the ASCII string "FFF" for the word number.

As the above discussion indicates, the ASR can perform one of the following:

1. Recognize a word correctly.
2. Confuse the input word with another word in the vocabulary. (This is known as a substitution error.)
3. Reject the input utterance.
   a. Identify a valid word, but threshold parameters are not met.
   b. Fail to identify a valid word with no attempt at recognition.
4. Recognize a non-word input as a word. (This is known as a spurious error.)

Performance Considerations

The user can adjust several recognition performance factors on the Model 6VI. These are described below.

Word Rejections and Substitution Errors

The user can adjust the RTHL and delta at any time to accommodate changes in the speaker's voice (such as those due to a cold, stress, or hoarseness).

A change in the speaker's voice increases the number of word rejections and substitution errors. (In general, a value of 100 for RTHL and 5 for the delta results in good recognition.) Lowering the RTHL decreases the number of rejections but increases the number of substitution errors. Raising the delta decreases the number of substitution errors but increases the number of rejections. Depending on the changes in the speaker's voice and the vocabulary, these two parameters can be manipulated to improve recognition.

The user can adjust voice-recognition parameters on the Model 6VI to accommodate temporary changes in the speaker's voice.

Background Noise Level

As discussed earlier, the controller constantly samples the incoming audio signal, looking for the beginning of a word. The controller detects a beginning-of-word boundary when the spectral energy of the filters exceeds the ambient-silence background by a certain value. The user can adjust for background noise by setting the "gain" or the sensitivity of the microphone.

When the gain is high, the user can speak softly and still be "heard". (A high gain setting, however, is sensitive to background noise.) When the gain is low, the user must speak louder. A low setting accommodates a noisy background such as a factory shop floor.

Length of a Word
Once an incoming word is detected by the ASR, it must last for a specified period of time before it is processed. This prevents accidental noise (such as a tool being dropped) from being mistaken for a word (resulting in a reject). This parameter defaults to 80 ms, but the user can adjust it to accommodate a vocabulary containing shorter words.

Minimum Word Interval
A discrete-word recognizer requires a pause between words to perform recognition. The default for the minimum time between words is 160 ms, the norm for general vocabularies. At this setting, words with intra-utterance gaps (e.g., "Los Angeles") can be spoken at approximately 42 wpm and still be successfully recognized.

Good speech recognition and small vocabularies go hand in hand.

Reducing the interval between words allows a higher rate of speech. However, this increases the risk of an intra-utterance gap being mistaken for an end-of-word boundary. Of course, if the vocabulary has no words such as these, no such risk exists.

Word Selection
Word selection is important to recognition performance. Users should select the vocabulary so that the reference pattern formed by each word is unlike other reference patterns. They should avoid homonyms (e.g., "one" and "won", "two" and "to", "hear" and "here"), as well as those words that differ by a single phoneme (e.g., "five" and "file", "time" and "tire", "logon" and "logoff").

Some words that seem to sound significantly different may form reference patterns similar enough to cause substitution errors. A Model 6VI program detects similar reference patterns. It examines a vocabulary after training and reports all words whose reference patterns are within a specified tolerance. The user can then add more training to create a more effective set of reference patterns or replace the words with synonyms.

Partitioning the Vocabulary
In speech recognition, good recognition performance and small vocabularies go hand in hand. Since it is not always desirable to limit an application's vocabulary to achieve higher performance, the Model 6VI uses a vocabulary organization scheme that can partition a vocabulary into *syntax nodes,* giving the effect of smaller vocabularies.

At any time in the recognition process, only one syntax node or subset of the vocabulary is active (available for recognition search). Thus, if the reference patterns for the words "file" and "five" are in different syntax nodes, they cannot be mistaken for each other.

Microphone Selection
Microphone selection is an important consideration in ASR systems. A microphone should be chosen with two factors in mind: the background noise of the environment and the system application.

In general, microphones with low signal-to-noise (S/N) ratios increase the probability of speech-recognition errors. For high performance, S/N ratios should be higher than 24 dB.

A popular microphone with good noise-cancelling ability is the Shure SM-10. Because this is a headband microphone, it is suitable for "busy-hands, busy-eyes" environments. For quiet environments, the lapel microphone, which clips onto clothing, is a good choice. For the user who is close to the terminal and whose mouth is normally a fixed distance from the terminal, a directional microphone mounted on a flexible gooseneck is the best choice.

The Future of Voice Recognition Technology

Voice recognition is not a typical engineering problem; it is a multi-disciplinary field that uses signal processing, LSI and VLSI microprocessor design, linguistics, artificial intelligence, and the mathematics of stochastic processes. Because of this, voice recognition technology has not had the dramatic progress that many had predicted. This is especially true, for example, in the area of speech understanding, which draws heavily on the use of artificial intelligence.

Research continues in three major areas: vocabulary size, continuous-word recognition, and speaker independence. Speaker-independent systems that recognize continuous speech currently exist only on large, fast mainframe computers. Significant breakthroughs in hardware and/or algorithms are necessary for this type of voice recognition to become practical for large vocabularies.

Meanwhile, as the refinement of isolated-word, speaker-independent systems continues, their performance increases and their cost decreases. Current advances in VLSI also contribute to this trend. The most immediate application of this progress will most likely be telephone-driven applications, such as remote transaction processing.

In the mid-1980's, significant advances in voice recognition technology will continue. The commercial availability of automatic dictation equipment (voice-actuated typewriters), however, will probably not occur before 1990. Eventually, the ability to use contextual, syntactic, and linguistic rules in a speech recognition device will make "natural language" speech a reality.

References

Crochiere, R.E., and Flanagan, J.L. 1983. Current perspectives in digital speech. *IEEE Communications Magazine,* vol. 21, no. 1.

Geyarter, W.B. 1983. *An overview of artificial intelligence and robotics.* National Aeronautics and Space Administration. Report no. NASA TM-85838.

Interstate Electronics Corporation. 1961. *Voice recognition chip set, Model VRC100 Series.* Document no. TMP00701949.

Nance, J. 1983. *Implementation strategies for voice-processing terminals.* Votan Corp.

Pierce, J.R. 1961. *Symbols, signals, and noise.* New York: Harper & Row.

_____. 1969. Whither speech recognition? *Journal of the Acoustical Society of America,* vol. 46.

Bill Huggett joined Tandem in January of 1982 as a software designer at the Austin, Texas facility. Since then, he has been involved with the 6530 terminal family primarily as the developer of the Model 6LA/AI alternate input option and the Model 6VI voice input option. Since graduating from Tulane University in 1963 with a B.S. in Mathematics, he has held a variety of positions involving the design, programming, and management of systems and application software.

The NonStop TXP Processor: A Powerful Design for On-line Transaction Processing[1]

The Tandem NonStop TXP™ processor was introduced to *Tandem Journal* readers in the Winter 1984 issue ("The High-Performance NonStop TXP Processor"). In this article, its design and performance are discussed in more detail. First the design goals of high performance and compatibility with the NonStop II™ processor are defined, and the innovations in hardware and software through which these goals were met are described. The major portion of the article is then devoted to a discussion of the resulting performance of the NonStop TXP processor. In this discussion, accepted performance metrics for computer systems are identified and explained, the problem of defining an accurate performance metric for on-line transaction processing systems is discussed, and the results of various performance comparisons between the NonStop II and NonStop TXP processors are presented. The performance criteria used in this discussion equate processing power to the following:

1. Block-move time.
2. Interprocess-communication time.
3. Process-dispatch time.
4. Data-base-access time.
5. Response time.

[1]Portions of the hardware discussion, Table 1, and Figures 1 and 2 originally appeared (in a slightly different form) in an article entitled, "New System Manages Hundreds of Transactions per Second", by Robert Horst and Sandra Metz in *Electronics*, volume 57, number 8 (April 19, 1984).

Design Goals for the NonStop TXP Processor

Higher Performance

As the transaction processing market has matured, users have demanded ever more powerful transaction processing systems capable of handling ever larger transaction workloads. Early transaction throughput requirements of dozens of transactions per second have grown to hundreds per second and show no sign of slackening in the future. One primary design goal for the NonStop TXP processor was to support these higher performance systems by providing at least twice the processor performance of the NonStop II processor.

The performance improvements were achieved through a combination of advances in architecture and technology. The NonStop TXP processor employs dual 16-bit data paths instead of the more traditional single 32-bit data path. Both 16-bit paths are used simultaneously more than 80% of the time, far more frequently than a 32-bit path would be fully utilized.

The parallelism results from the ability of the main arithmetic and logic unit (ALU) to perform an operation in parallel with a different operation executed by one of several special ALU modules. For example, the inner loop of a compare-byte instruction that would take three clock cycles in the traditional architecture (i.e., the extraction of byte 1, followed by the extraction of byte 2, followed by the comparison) takes only two clock cycles on the NonStop TXP processor (i.e., the parallel extractions by the main ALU and the special ALU of byte 1 and byte 2 in the first clock cycle, followed by the comparison). This is shown in Table 1.

Table 1.
Compare-byte instructions (inner loop).

Clock cycle	NonStop TXP processor		Traditional architecture
	Main ALU	Special ALU	
1	Extract byte 1	Extract byte 2	Extract byte 1
2	Compare bytes	—	Extract byte 2
3	(Repeat)	(Repeat)	Compare bytes
4	—	—	(Repeat)

Although the frequency of 32-bit arithmetic operations is insignificant relative to data-movement and byte-manipulation instructions in typical transaction processing applications, the performance of these instructions is also enhanced by the dual 16-bit paths. The two clock cycles required to perform an add operation is partially offset by the availability of the other 16-bit path to perform another 32-bit operation or two 16-bit operations in parallel with the main ALU.

The NonStop TXP processor supports 32-bit addressing much more efficiently than the NonStop II processor. The 32-bit virtual addressing built into the hardware is capable of addressing a gigabyte of virtual memory.

The 32-bit virtual address is used to access a large, 64K-byte memory cache. In comparison with the IBM 4361, which can access 32 bits of cached data every 100 ns out of its 16K-byte cache, the NonStop TXP processor can access 16 bits of cached data every 83 ns out of its 64K-byte cache. In practice, the effective cache bandwidth of the NonStop TXP processor is comparable to the 4361 because the considerably higher cache hit-rate offsets the larger 32-bit words retrieved by the 4361. However, the cost of a NonStop TXP processor with 4M bytes of main memory is approximately one-half that of an IBM 4361 Model 5, similarly equipped (Carr and Campbell, 1984; IBM, 1984; Tandem, 1984.)

The technology of the NonStop TXP processor includes 25-ns programmable array logic (PAL), 45-ns static RAM chips, and Fairchild Advanced Schottky Technology (FAST) logic. This newer technology and a reduction in the number of logic levels in each path has resulted in a reduction of the basic microinstruction cycle time from 100 ns in the NonStop II processor to 83.3 ns in the NonStop TXP processor.

Instruction pipelining has been increased from two levels on the NonStop II processor to three levels on the NonStop TXP processor. Figure 1 illustrates the operation of the

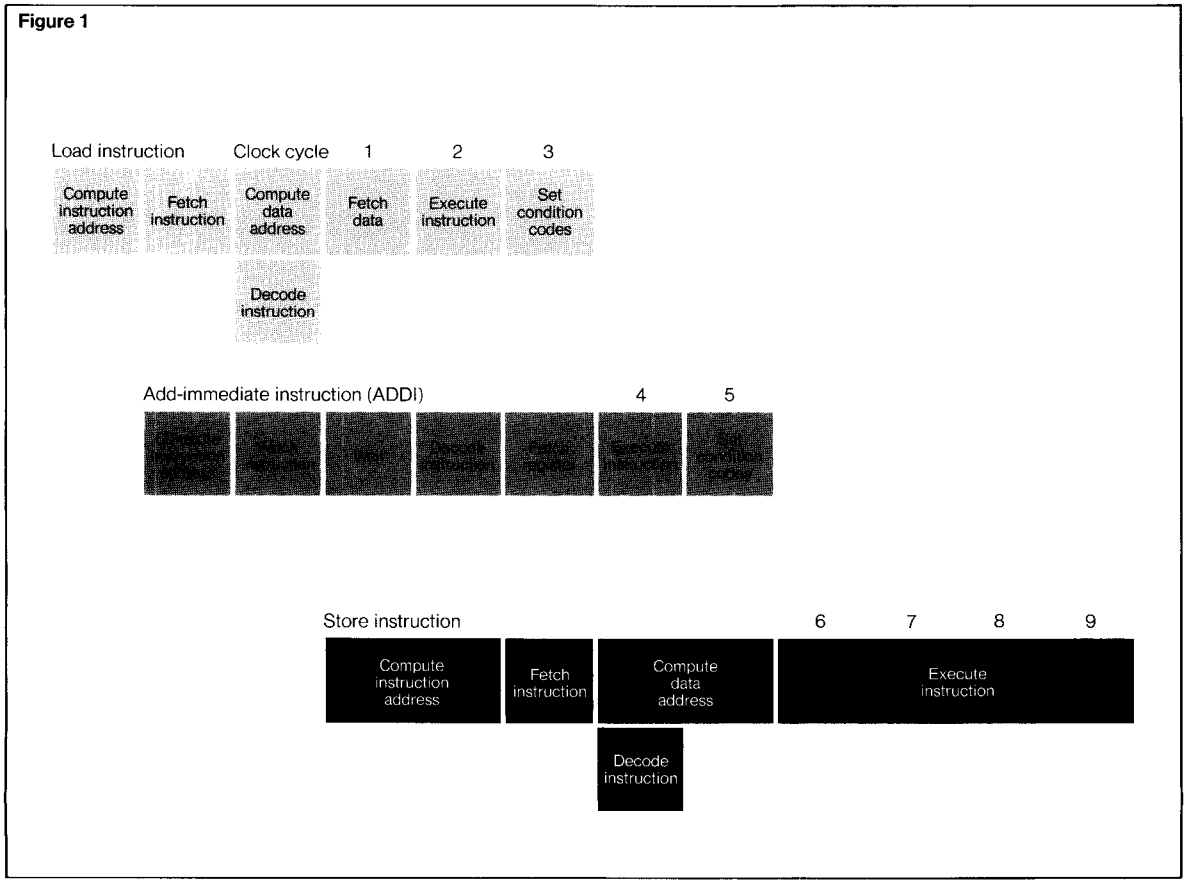
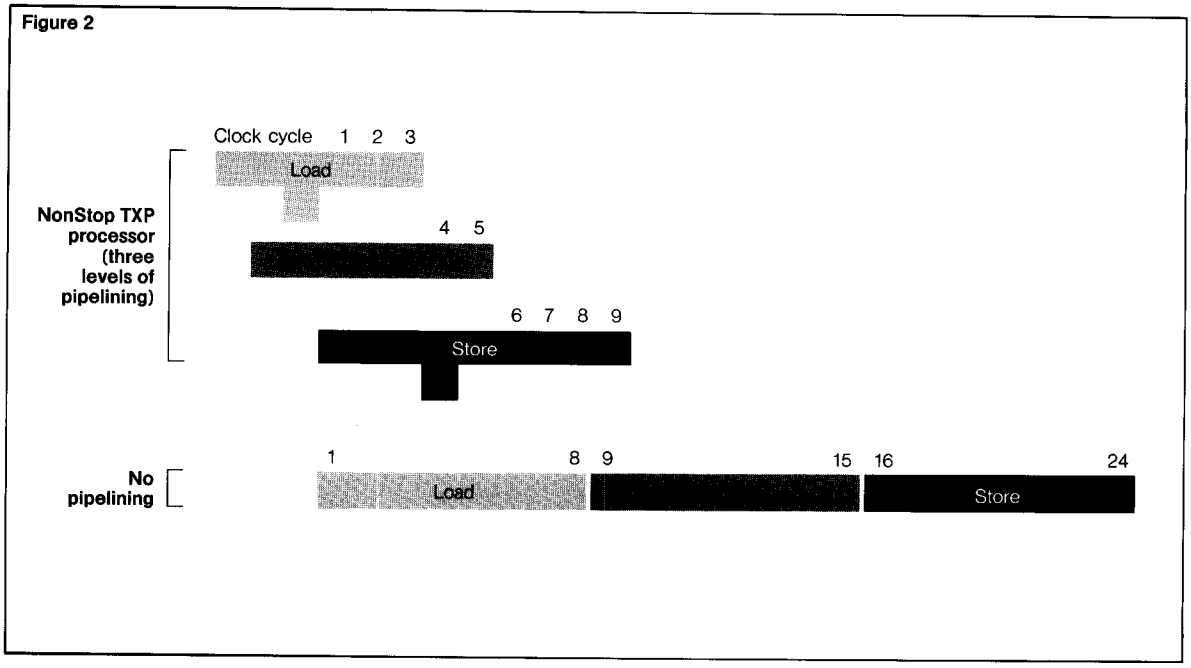


Figure 1
The NonStop TXP processor employs three levels of pipelining.

Figure 2

With three levels of pipelining, the NonStop TXP processor can execute a combination of three typical instructions in only nine clock cycles. Without pipelining, 24 clock cycles would be required.



macroinstruction pipeline for a sequence of instructions. As shown in Figure 2, a combination of three instructions (a load, followed by an add immediate, followed by a store) that would require 24 clock cycles with no pipelining requires only 9 clock cycles with pipelining because the prefetch and part of the execution of each instruction can be overlapped with previous instructions. Pipelining permits the NonStop TXP processor to execute many of its most frequent instructions in just two or three clock cycles.

Compatibility

The design goal of compatibility between the NonStop II and NonStop TXP processors was established to provide an upgrade path for systems based on NonStop II processors. It was decided that users must be permitted to mix NonStop II and NonStop TXP processors within the same system and within the same physical cabinet, as well as to mix any combination of Tandem processors in the same network. These constraints translated into specific mechanical, hardware, and software goals.

Software Compatibility. The main goal was to mask from the user as many of the differences between the two processors as was possible. Ideally, the user was to see nothing different in the processors but their speeds. To meet this goal, the following were implemented:

1. A single operating system, GUARDIAN™, supports both the NonStop II and NonStop TXP processors.
2. The firmware for the NonStop TXP processor implements the existing instruction set.
3. All nonprivileged software runs without change on both machines.
4. To facilitate processor swapping without system reconfiguration, the type of each processor does not have to be specified at system-generation time. The operating system automatically loads the appropriate instruction-set microcode at load time.
5. The procedures for controlling the system and the system console are the same for both machines.

Hardware Compatibility. Because the NonStop TXP processor was intended to replace only the central processing unit (CPU)

in Tandem systems, it was important that it not require new buses, channels, or controllers. Thus, it had to support all the controllers and devices currently in use in systems based on the NonStop II processor. Fortunately, the existing 5M-byte I/O bus and the dual 13M-byte DYNABUS™ have more than enough bandwidth to support the more powerful NonStop TXP processor. The main problem facing the hardware design team, then, was to develop a new microarchitecture that would efficiently support a 32-bit processor at much higher speeds, using only 33% more printed-circuit-board real estate and the same backplane.

Mechanical Compatibility. The NonStop TXP processor is physically very similar to the NonStop II processor. The CPU and memory (MEM) boards are the same size and draw comparable amounts of power. The NonStop TXP processor uses 4 CPU boards; the NonStop II processor uses 3. All cable connections to the NonStop II processor are in locations similar to those on the NonStop TXP processor. The CPU backplanes are identical.

The NonStop TXP processor uses the same power system as the NonStop II processor, and AC power requirements have not changed. The CPU power requirements have increased from 60A to 80A while the I/O power requirements are unchanged. The battery backup characteristics of the NonStop TXP processor are similar to those of the NonStop II processor.

In summary, both of the somewhat-conflicting goals of higher performance yet strict compatibility were met by the development groups. The resulting performance of the NonStop TXP processor is discussed in the following section.

Comparative Performance Data

Although determining the relative performance of computer systems seems like a straightforward task, it is almost always more difficult than it first appears. Comparisons can be made in so many different ways that the confusion begins with the selection of the performance criteria upon which to base the performance comparison.

The earliest performance indices were those used to compare the CPU-primary-memory complex of various computer systems, as this complex was then the most expensive part of the system. Rudimentary indices, such as the internal clock rate of the CPU, the execution time of certain arithmetic instructions, or the cycle time of the primary memory, formed the basis of most early performance comparisons.

Although determining the relative performance of computer systems seems straightforward, it is more difficult than it appears.

These early measures evolved into slightly more sophisticated definitions of computer processing power. Schneidewind (1966) introduced the definition of power

$$P = \frac{M}{t_{cycle}},$$

where M is the primary-memory size in bytes, and t_{cycle} is the primary-memory cycle time. Notice that this definition lacks any workload dependence and is a purely hardware-based definition.

Gruenberger (1966) amplified Schneidewind's definition to include some elementary calculations

$$P = M\left(\frac{1}{t_{add}} + \frac{1}{t_{mpy}}\right),$$

where M (the primary-memory size) is in words and t_{add} and t_{mpy} represent the time it takes the computer to execute addition and multiplication operations. The main problem with indices of this type is their workload independence.

A further refinement in this form of performance index attempts to characterize the power of a computer at processing its typical workload by using the instruction-mix frequencies and instruction-execution timings

$$P = \sum_i f_i t_i \; ; \; \sum_i f_i = 1,$$

where f_i is the relative frequency of instruction i, and t_i is the execution time of instruction i.

Since these early attempts, many subsequent definitions of computer power have been proposed, and some are still in use today. Some contemporary yardsticks used to compare computer systems are: millions of instructions per second (MIPS), floating-point operations per second (FLOPS), whetstones per second (WHETS), batch job turn-around time, transactions per second, and transaction response time.

Because performance analysts have so many metrics to choose from, and because the relative performance of computer systems vary according to the performance metric used, it is possible to draw misleading performance comparisons. Of course, there is nothing wrong in using any well-defined performance metric to compare computer systems. However, it would be incorrect to compare them with a performance metric designed to compare scientific processing power and then postulate that this relative performance ranking would also hold for transaction processing workloads, for example.

Block-move time equates processing power to the processor's ability to move blocks of data in primary memory.

Because no single universal performance metric has been adopted to compare the relative performance of computer systems, the best one can do is to examine relative performance across a spectrum of performance metrics and provide this information in standardized formats. It is then up to the performance analyst to select the appropriate measure(s) of performance.

The remainder of this article summarizes the performance of the NonStop TXP processor using several performance metrics. It begins with low-level performance data on CPU-intensive operations that are designed to measure the performance of the CPU hardware and the code compilers. It then concludes with high-level performance data on system-wide performance.

Block-move Time (*PI*)

Block-move time (*PI*) equates processing power to the processor's ability to move blocks of data in primary memory. For example, if process A wants to send a message to process B, this operation typically moves the actual message from one location in memory to another. Block-move operations are typical in message-based computer systems.

Even for this simple yardstick many environmental sources of variation exist. The performance of a block move can be affected by the location of both the source and destination in memory, by whether or not the memory is organized hierarchically, and by the mode of addressing employed to access the data.

For the purposes of this article, this first performance index (*PI*) is defined as

$$PI = \frac{1000}{t_{move}},$$

where t_{move} is the time in microseconds to move a 1000-word block of memory from the source location to the destination. Experiments performed by the NonStop TXP System Support Team (1983) yielded the results in Table 2.

In Table 2, the first row represents the moving of a block of 1000 words from some location on the data stack of the processor to another location on the stack using the 16-bit addressing mode. In the second row, the block is moved from some location in an extended data segment to a different location in the segment, using absolute extended addresses to access the block. The final column shows the ratio of *PI* for the NonStop II processor, *PI*(II), to that of the NonStop TXP processor, *PI*(TXP).

While many other combinations of memory locations and addressing modes can be used, these two have been chosen as representative for performance index *PI*. The results for *PI* indicate that the NonStop TXP processor is between 2.5 and 3.1 times as powerful as the NonStop II processor.

Table 2.
Relative performance of the NonStop II and NonStop TXP processors for block-move time (*P1*).

Source	Destination	S-Address	D-Address	Ratio *P1*(II):*P1*(TXP)
Stack	Stack	16-bit	16-bit	1:2.5
E-segment	E-segment	A-extended	A-extended	1:3.1

Interprocess Communication (*P2*)

Like *P1*, interprocess communication, or *P2*, also attempts to measure processing power by a measure related to sending messages. In this case, it measures the ability of the processor and operating system (and message system) to support interprocessor communication. In a system in which processes cooperate to accomplish a task, messages are used to communicate requests for processing, processing results, and status information. Although often equated with overhead, the ability of a computer system to facilitate messages between cooperating processes is an important measure of how effective a computer system is in supporting a distributed processing environment.

For the purposes of this article, two variations of *P2* are defined. One corresponds to interprocessor communication ($P2_{inter}$), i.e., the two communicating processes are in different processors, and the other corresponds to intraprocessor communication ($P2_{intra}$), i.e., the two processes are in the same processor.

In an attempt to minimize environmental variations, this investigation was restricted to "waited" communication only. In this mode of communication, the sending process waits for the reply to come back from the process to which it sent the message before it resumes execution. (An example of "nowaited" communication is the sending process asynchronously executing other work before receiving the reply from the process to which it sent the message.)

As one might expect, the elapsed time for interprocess communication can be approximated as a linear function of the message length and takes the form of $y = mx + b$. In this equation, y, the elapsed time, is equal to a fixed cost, b, plus a variable cost per byte of data transmitted, m. Because values of P should increase with increasing processing power, *P2* is defined as follows:

$$P2_{inter} = \frac{1}{t_{var1}x + t_{fixed1}} ; \text{CPU(A)} \neq \text{CPU(B)}$$

$$P2_{intra} = \frac{1}{t_{var2}x + t_{fixed2}} ; \text{CPU(A)} = \text{CPU(B)}$$

Both *P2*s are equal to the inverse of the elapsed round-trip time of a message and thus specify the number of messages that can be sent and subsequently received by that process per second. The variables t_{var1} and t_{var2} are the cost per byte of inter- and intraprocessor communication and are multiplied by x, the total number of bytes transmitted by both the sending and the receiving process. Included in this portion of the processing time are the costs of data transmission and message packetization, for example. The t_{fixed1} and t_{fixed2} variables are the fixed overheads associated with process communication performed by the operating system (and message system) on behalf of the processes, e.g., process dispatching, message initialization, and interrupt handling.

Notice that it is the round-trip elapsed time for process A to send a message to process B and then receive a reply that is important here. Clearly, this measure includes events other than those that are purely message-transfering operations; for example, process dispatching must be occurring. This is obviously the case for intraprocessor communication; the two processes take turns receiving service at the CPU.

Although a simpler definition that counted only message transmission would more accurately measure how fast a message can be transmitted from process A to process B, most computer workloads involve the repetition of the following scenario: process A formulates a request for data that is sent to process B, process B decodes what to do, locates the data process A wants, and responds with the requested information in a reply to process A. Thus, the frequency of round-trip interprocess communication that a processor can support is an appropriate measure of a computer's power.

Table 3.
Relative performance of the NonStop II and NonStop TXP processors for interprocess communication (*P2*).

	$x = 100$	250	500	1000
Ratio $P2_{intra}$(II):$P2_{intra}$(TXP)	1:3.0	1:3.0	1:2.9	1:2.8
Ratio $P2_{inter}$(II):$P2_{inter}$(TXP)	1:2.8	1:2.8	1:2.8	1:2.7

Again, the data in Table 3 is derived from a study by the NonStop TXP System Support Team (1983). As the data indicates, the relative power of the NonStop TXP processor is between 2.7 and 3.0 times that of the NonStop II processor for typical message lengths. This observation is in agreement with the findings for *P1*.

Several interesting relationships exist behind the data of Table 4. First, intraprocessor communication is faster than interprocessor communication for both the NonStop II and NonStop TXP processors. For the NonStop TXP processor, intraprocessor communication is about 18% faster than interprocessor communication. For the NonStop II processor, this difference is somewhat less, approximately 11%.

It is also of interest to observe that as the length of the message increases, the relative difference in power declines. This occurs because the data transmission component of interprocess communication is the same for both processors. Hence, the larger the message, the more this dominates the elapsed time.

Process Dispatch (*P3*)

The process-dispatch performance metric (*P3*) equates processing power to the processor's ability to dispatch processes. A process dispatch occurs for a variety of reasons, but the result is usually the same: the currently executing process can no longer proceed, its context must be saved so that it can be resumed at some later time, the context of a different process is loaded by the processor, and this process resumes or begins execution.

In transaction processing systems, dispatching occurs quite often, therefore making the speed with which a process switch can be made an important indicator of processing power. A processor that spends 20% of its time dispatching processes is only providing 80% of its power for useful work.

For this performance index only one major environmental influence exists: the number of memory pages that must be made present with the process. In typical systems, although the pages are not actually brought in from secondary memory, the system registers containing the address of the physical-memory pages are usually pre-loaded before the process can execute. Thus, the dispatch time is approximated as a linear function of the number of process pages plus a fixed overhead, i.e.,

$$t_{disp} = (p_1 + p_2)x + f,$$

where p_1 is the number of pages that must be unmapped for process p_1, and p_2 is the number of pages that must be mapped for process p_2. This sum is multiplied by the cost of mapping or unmapping a page, x. Finally, f is the fixed overhead associated with performing a dispatch.

The NonStop II processor employs this type of algorithm, but the NonStop TXP processor uses a "demand-based" loading scheme and a cache of map entries to reduce the execution time of the dispatch. For the NonStop TXP processor, p_1 and p_2 both equal zero, and the dispatch time is not a function of the number of pages, it is a constant.

For the purposes of this article, *P3* is defined as

$$P3 = \frac{1}{t_{disp}},$$

where t_{disp} is the time to dispatch a process. Thus, *P3* is, in fact, the maximum possible number of dispatches per second.

To study the impact on *P3* performance, two identical processes were made to execute in the same CPU at the same priority and to do nothing but cause themselves to be dispatched. Thus, the process activity for this processor consisted entirely of two processes that were alternately dispatched. The measurements from which this data was extracted were conducted by Singh (1983) during the pre-release testing of the NonStop TXP processor.

As expected, varying the number of memory pages owned by these processes produced a variation in the value of *P3* for the NonStop II processor, but not for the NonStop TXP processor. As the data in Table 4 indicates, the ratio of *P3*(TXP) over *P3*(II) increases with the number of memory pages.

The ratio of *P3* for the NonStop II, *P3*(II), and *P3* for the NonStop TXP processor, *P3*(TXP), indicates that the NonStop TXP is between three and four times as powerful as the NonStop II. This performance difference is greater than that exhibited by the previous indicators *P1* and *P2*. The explanation for the larger difference is that the improvement in performance is due not only to the faster processing speed of the CPU, but also to the way dispatching is performed on the new hardware. While in the NonStop II processor, map entries for memory pages accessed by a process are loaded by the process before execution begins, in the NonStop TXP processor, map entries are brought in as needed. Thus, the dispatching of a process is faster on the NonStop TXP processor because there is also less work to be done.

Table 4.
Relative performance of the NonStop TXP and NonStop II processors for process dispatching (*P3*).

Number of process pages	Ratio *P3*(II):*P3*(TXP)
Small	1:3.17
	1:3.35
	1:3.52
	1:3.66
	1:3.78
	1:3.88
	1:4.00
Large	1:4.11

Data-base Access Time (*P4*)

The data-base access time performance metric (*P4*) equates processing power to the computer system's ability to manipulate data stored in data bases. This metric is the first one to include the processor's ability to manage an I/O device as a measure of the computer system's processing power. This measure of performance is of interest because it compares the ability of computer systems to perform operations involving more than CPU processing.

The time to access data from a disc can be approximated by

$$t_{access} = t_{cpu} + t_{disc} .$$

Assuming no overlap of CPU processing with disc-device processing, the access time, t_{access}, is equal to the sum of the CPU processing time, t_{cpu}, and the disc processing time, t_{disc}. Note that there is some overlap in Tandem's disc subsystem and in that of most computer systems. This approximation is to simplify the discussion that follows.

An interesting performance phenomenon is explained by this rather simple equation. A CPU that is twice as fast as another does not halve the data access time (unless t_{disc} is zero). For example, if $t_{cpu} = 20$ ms and $t_{disc} = 35$ ms, then $t_{access} = 55$ ms. If the speed of the processor is doubled, t_{cpu} is reduced to only 10 ms, but t_{access} still equals 45 ms, an improvement of only (55-45)/55 = 18%. In fact, an infinitely fast CPU ($t_{cpu} = 0$) would only serve to reduce t_{access} by 20 ms, for a performance increase of (55-35)/55 = 36%. This relationship should be kept in mind when the *P4* performance indices are discussed.

In most computer systems, several predefined file types are provided by the system to hold data. Each of these has a different structure that provides access to the data in a particularly useful manner. For example, Tandem supports four file types: key-sequenced, relative, entry-sequenced, and unstructured. Key-sequenced files contain records that are stored in ascending sequence, ordered by a field within each record called the primary key. Relative files contain records that are stored in a position relative to the beginning of the file, according to a record number supplied by the application program. Entry-sequenced files store records by appending new records to the end of the file in the order they are received. Unstructured files have no system-defined structure; data stored in them can be considered a large byte array.

Each type of file is useful to hold data that is accessed in certain familiar patterns. For example, entry-sequenced files support log files very nicely. An example of an application for a key-sequenced file is an inventory file in which each record describes a part. The primary-key field for the file would probably be the part number, and other fields could contain information such as the vendor name and quantity on hand.

Table 5.
Relative performance of the NonStop TXP and NonStop II processors for data-base I/O (*P4*).

Index	Ratio *P4*(II):*P4*(TXP)
$P4_a$	1:1.27
$P4_b$	1:1.39
$P4_c$	1:2.33
$P4_d$	1:1.69
$P4_e$	1:1.39
$P4_f$	1:1.45
$P4_g$	1:2.46
$P4_h$	1:1.93

Data-base I/O operations are the basic units from which transactions are built. In fact, because transaction processing involves almost no "number crunching", transaction workloads are typically approximated by the sequence of data-base operations they entail. For example, a typical banking transaction might include an update of the customer account file, then an update to the teller file, and finally, a write to the transaction log file.

Thus, before a computer system's performance at processing transaction workloads is measured, it is worth examining its performance in terms of simpler I/O operations. For this purpose, let P4 be defined as the following sequence of I/O operations:

$P4_a$ = Random read from a key-sequenced file.
$P4_b$ = Random write to a key-sequenced file.
$P4_c$ = Sequential read from an entry-sequenced file.
$P4_d$ = Sequential write to an entry-sequenced file.
$P4_e$ = Random delete of a relative file.
$P4_f$ = Random update of a relative file.
$P4_g$ = Sequential read of an unstructured file.
$P4_h$ = Sequential update of an unstructured file.

For this comparison, identical disc devices were chosen to eliminate any environmental variations stemming from the devices. For each of the *P4* indices, it is the inverse of the operation elapsed time, or the theoretical maximum number of operations per second, that is of interest.

The data in Table 5 compares the relative performance of the NonStop II processor to the NonStop TXP processor as measured by *P4*. $P4_a$ through $P4_h$ show that the NonStop TXP processor improves data-base access time by an amount that varies considerably with the type of data-base operation performed. The explanation for the large variance in the improvement is somewhat complicated.

The most significant factor contributing to the spread in relative performance is whether the operation is sequential or random in nature. The four indices $P4_a$, $P4_b$, $P4_e$, and $P4_f$ involve I/O operations that are randomly distributed throughout the file. These four indices also exhibit the smallest improvements in access time. Conversely, the four sequential I/O indices ($P4_c$, $P4_d$, $P4_g$, and $P4_h$) show the most significant improvements in access time.

The reason for this behavior becomes apparent if the proportions of the access time, t_{access}, stemming from t_{disc} and t_{cpu}, are compared. For the random I/O indices, t_{disc} accounts for more than 60% of the access time on a NonStop II processor, a high percentage when compared to less than 50% of the time (one as low as 3%) for the sequential I/O operations. As mentioned before, t_{disc} is essentially unaffected by the introduction of a faster processor; therefore, sequential I/O processing exhibits more significant performance improvements than random I/O processing because it is more CPU intensive.

When the geometric mean is used to obtain average values for *P4*, three averages can be defined for the ratios of $P4_a$ through $P4_h$ from Table 5:

$$P4_r = (P4_a \cdot P4_b \cdot P4_e \cdot P4_f)^{1/4} \quad \text{(random I/O)}$$

$$P4_s = (P4_c \cdot P4_d \cdot P4_g \cdot P4_h)^{1/4} \quad \text{(sequential I/O)}$$

$$P4 = (P4_a \cdot P4_b \cdot P4_c \cdot P4_d \cdot P4_e \cdot P4_f \cdot P4_g \cdot P4_h)^{1/8}$$

For this comparison, the eight data-base I/O operations are weighted equally in importance. Using the arithmetic mean of the execution times does not provide equal weighting. For example, the NonStop TXP can perform 30 times as many $P4_c$ I/O operations per second as it can $P4_f$ operations. If the numbers were merely added together, much more weight would be given to the operations with larger *P4* values than those with smaller *P4* values.

The ratios of *P4* values for the NonStop TXP and NonStop II processors indicate that when compared against a NonStop II processor performing a mix of random and sequential data-base I/O operations, a NonStop TXP processor (on average) performs those operations 68% faster, i.e., *P4*(II):*P4*(TXP) = 1.68. If the operations are strictly random in nature, only 37% improvement can be expected, i.e., $P4_r$(II):$P4_r$(TXP)=1.37. However, if the operations are primarily sequential in nature, an improvement of 107% can be realized, i.e., $P4_s$(II):$P4_s$(TXP)=2.07.

The range of *P4* ratios is below the ranges for *P1* through *P3*, as can be expected. The more powerful processor speeds up the t_{cpu} portion of t_{access} by a factor of two to three, but t_{cpu} is less than half of t_{access}. Thus, the total improvement in performance of t_{access} from the introduction of the faster NonStop TXP processor should fall somewhere in the range of about 50% to 150%, as is verified by the measurements.

On-line Transaction Processing (OLTP) Benchmark (*P5*)

So far, the comparisons of performance for the NonStop TXP and NonStop II processors has been based on artificial workloads. But the ultimate performance question for a user of the NonStop II system is, of course, "How much better will my application perform on a system of NonStop TXP processors?"

Performance indices *P1* through *P3* are CPU-intensive measures of performance, and *P4* is a fairly disc-intensive measure. Typical OLTP applications fall somewhere between these two extremes. Consider that between performing data-base I/O operations, the typical application performs some data processing on the retrieved data. Another, less apparent type of processing that occurs in OLTP systems is the background processing performed by the system to provide continuous operation, fault tolerance, and transaction integrity. This "system" processing is not considered overhead, rather, it is a part of the OLTP application that the customer would have to develop if it were not already provided by the system.

For OLTP applications, a meaningful measure of performance is the responsiveness of the system as perceived by the user, or how fast the computer responds to requests. The performance index that corresponds to the system's responsiveness is called *response time.*

In typical OLTP systems the response time does not appear to be constant. As the workload on the machine increases, the response time lengthens. This occurs because, as more jobs demand resources from the system, queues begin to form at those resources that are most in demand. The time that a request spends waiting in these queues is the main source of longer response times. A simple equation that expresses this relationship is

$$t_{response} = t_{service} + t_{queueing} .$$

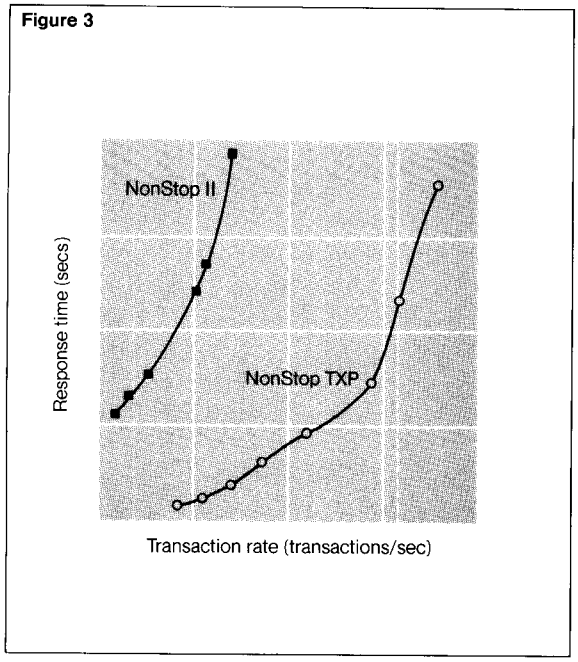
The system's transaction response time, $t_{response}$, is equal to the sum of $t_{service}$, the delay the request would have if no other request were in the system, and $t_{queueing}$, the time the request spends waiting for resources.

For this simple investigation, assume that $t_{service}$ is a constant, not affected by the workload in the system. The other component of response time, $t_{queueing}$, is definitely a function of the workload. In lightly-loaded systems, little or no queueing takes place and the system's response time is very nearly $t_{service}$. In heavily-loaded systems, the response time is eventually dominated by $t_{queueing}$ and can even approach infinity.

For the DP manager who has a problem with response time, the performance question might be expressed as, "What will the transaction response time of my system be at current transaction workloads if I replace my NonStop II processors with NonStop TXP processors?"

Another related problem is the capacity of the system. If response time goes to infinity, no work is done. The point at which response time becomes unacceptable (whether it be three seconds, three minutes, or three years) can be considered the maximum transaction workload (capacity) of the system. In this case, the question being asked is, "What will the maximum capacity of my system be if I replace my NonStop II processors with NonStop TXP processors?" Another way to ask this is, "What is the new peak load my system can handle?"

Figure 3
Benchmark response times for the NonStop II and NonStop TXP processors at various transaction rates.



One variation on the throughput question is, "What new transaction throughput at the same transaction response time(s) can my system handle if I replace my NonStop II processors with NonStop TXP processors?" In this case, the system manager would like to increase the number of transactions per second that the computer can process at some predefined response time(s). This question is similar to the previous one but usually involves increasing the number of users on the system, not determining the peak loading point. This situation arises when new terminals (and work) are added to the system while the responsiveness of the machine must be maintained at present levels.

These questions and others can be answered by determining the transaction response time as a function of the transaction rate. Thus, in this article, the following definition of *P5* is the performance index for OLTP workloads:

$P5_{rt}$ = Response time at a transaction rate.
$P5_{thru}$ = Transaction rate at a response time.
$P5_{cap}$ = Maximum transaction-rate capacity.

In early comparisons of the NonStop TXP and NonStop II processors, response-time curves like those in Figure 3 were obtained (NonStop TXP System Support Team, 1983).

To obtain these results, the same benchmark was measured on two configurations in which the only difference was that, in one instance, NonStop II processors were used, and in the other, NonStop TXP processors were used.

Building, configuring, loading, and running benchmarks is a time-intensive and resource-intensive exercise. Although benchmarks are the best models of the real workload, and therefore yield the most accurate information about the system's performance, many difficult operational problems limit their use. Benchmarks are usually difficult to modify. Often, privacy and security requirements limit the use of application files in the benchmark. Finally, tuning and balancing benchmarks for optimum performance is a non-trivial task. Thus, developing performance information through benchmarking usually involves many days of intensive work on the real system hardware.

For all of these reasons, a method that consumes fewer resources is called for to predict the performance impact caused by changes to the system (e.g., replacing NonStop II processors with NonStop TXP processors). Based on the data from many of these benchmarks and similar measurements, an on-line transaction processing model has been developed to predict the performance of OLTP applications on both the NonStop II and NonStop TXP processors. The model is composed of two major submodels: a resource-demand model for sizing a system and a response-time model for predicting the system's performance. ENVISION, the OLTP performance modeling tool, is used by Tandem system analysts to accurately size, tune, manage the growth of, and predict the performance of OLTP systems (Chou, Oleinick, and Singh, 1984).

The model predicts transaction response times, given a hardware and software configuration and workload description. To demonstrate the effectiveness of this tool, an OLTP application was modeled. The resulting transaction response-time curves appear in Figure 4. Three curves appear in the figure: those representing measured data for a real system of NonStop II processors, predicted data for the system of NonStop II processors, and predicted data for the system of NonStop TXP processors.

Although not exact, the modeled performance of the system of NonStop II processors compares well with the measured data. At low-to-medium transaction rates, the model and the measured data agree to within 5%. At medium to higher transaction-throughput rates, the model predicts response times that are from 7% to 13% too low. Still, this degree of accuracy is sufficient for the analysis that follows.

There are many ways to interpret these curves. If reducing response times is of interest, a vertical line can be drawn at the appropriate transaction rate, and the response-time values of the two systems can be compared. The curves indicate that in this benchmark, at five transactions per second, for example, response time can be reduced from 1.46 seconds to 0.7 seconds, a reduction of more than 50%. To continue the ratio method of comparison of the NonStop II and NonStop TXP processors, the transaction response times of both machines should be compared.

Table 6 summarizes this type of analysis. By this measure of performance, the NonStop TXP processor is dramatically more powerful than the NonStop II processor. The first row of the table compares the service times of the transactions, i.e., the no-load response time. The ratio for $P5_{rt}$ is 1:1.61 in this case. As the transaction rate increases, this improvement increases, and eventually no comparison is possible; the NonStop II processor response time becomes unbounded, and $P5_{rt}$ approaches infinity.

The data in Table 6 indicates that the improvement in the responsiveness of the system varies considerably with the quantity of work. If the DP manager were interested in improving system responsiveness within the typical transaction-throughput range of a system (e.g., 4 to 8 transactions per second), this data indicates that the amount of improvement would be between 100% ($P5_{rt}$ ratio = 2.00) and 153% ($P5_{rt}$ ratio = 2.53).

By drawing horizontal lines across the curves, the DP manager can answer the question about increasing throughput rates while the same response time is maintained. For example, the transaction throughput capacity at a one-second response time is approximately

Figure 4

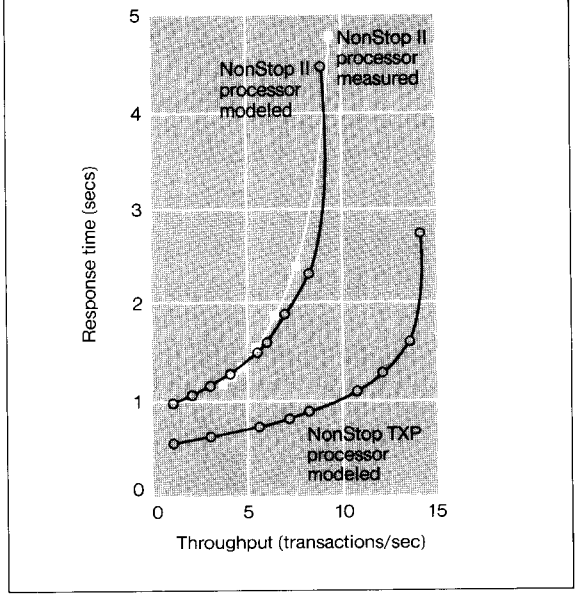


Figure 4
Modeled and measured response-time curves (P5).

Table 6.
Relative response times for the NonStop II and NonStop TXP processors ($P5_{rt}$).

Transaction rate	Ratio $P5_{rt}$(II):$P5_{rt}$(TXP)
0.0 (no load)	1:1.61
2.0	1:1.90
4.0	1:2.00
6.0	1:2.10
8.0	1:2.53
9.0	1:4.60
10.0	1:infinity

Table 7.
Relative throughput rates for the NonStop II and NonStop TXP processors ($P5_{thru}$).

Response time	Ratio $P5_{thru}$(II):$P5_{thru}$(TXP)
2.50	1:1.70
2.00	1:1.91
1.50	1:2.48
1.00	1:9.30
0.75	1:infinity

one for the NonStop II processor, but more than nine for the NonStop TXP processor. The DP manager can increase the system's transaction processing capacity by a factor of 800% by replacing the NonStop II processors with NonStop TXP processors, given this response-time requirement for this system's workload and configuration. This type of analysis is summarized in Table 7.

By this measure of performance, the NonStop TXP processor is again dramatically more powerful than the NonStop II. As the response time requirement becomes more and more stringent, the difference in transaction processing capacities widens. Eventually, as before, the NonStop II processor cannot deliver any transactions at the indicated response time, and the ratio of $P5_{thru}$(TXP) over $P5_{thru}$(II) becomes unbounded.

The third *P5* metric, $P5_{cap}$, is easily obtained from the curves. It is approximately 9 for the NonStop II processor and 14 for the NonStop TXP processor. Thus, for *this* system configuration, running *this* workload, switching from NonStop II to NonStop TXP processors would permit the system to handle a peak load 1.55 times the present maximum.

A balanced system, in which the disc-processing and CPU capacities match the demands of the workload, will always be the most efficient system to configure.

What is the reason for the rather small difference in peak load capacity? The answer can be found by examining the rate of utilization of the processors at the peak load points. The NonStop II processors were running at an average of 87% busy. The NonStop TXP processors were much less busy, only 49%. Clearly the bottleneck in the NonStop TXP processor system is not the processors.

It should come as no surprise that the disc subsystem is the bottleneck in the NonStop TXP processor system at peak load. The utilization data confirms this hypothesis. For the NonStop II processor system, the disc drives were only 27% busy on average (54% for each mirrored pair) at peak load. When the NonStop TXP processors were introduced, the disc devices could be used more. The average utilization of the disc drives for the system of Nonstop TXP processors increased to 38% (76% for each mirrored pair) at the higher peak load. Thus, the NonStop TXP processor better utilizes the disc devices.

Note that of all the performance indices examined so far, *P5*, and especially $P5_{cap}$, should be viewed with the most caution. Because the values of *P5* were derived from the performance of a single OLTP application on a single configuration of processing elements, the performance results cannot be used to size or configure systems for other than OLTP applications or workloads. The *P5* data merely demonstrates the performance impact of the CPU elements when NonStop II processors are replaced by TXP processors in one situation.

The results of *P5* could have been adjusted to any value desired to represent a different disc and processor configuration or different application-processing demands. The values presented for *P5* were chosen to provide insight into typical system-configuration problems.

Conclusions

It is difficult to specify unequivocably the relative performance of two computer systems. It is clear, however, that when the performance of computer systems is compared, the workload to be used for the comparison must be specified as accurately as possible. This is important because the slightest variation in the workload can produce a profound change in observed performance.

In this article, several essential computational elements in an OLTP environment have been discussed, and the corresponding performance values for very similar processors, the NonStop II and the NonStop TXP, have been compared. The data in Table 8 summarizes the performance comparisons.

While, as discussed earlier, the processing performance for OLTP workloads varies, depending on the nature of the transactions and the configuration of the systems, the throughput of the NonStop TXP processor has been observed to be two to three times greater than that of the NonStop II processor for a variety of OLTP applications and workloads for numerous Tandem users.

The information in this article represents the performance criteria and resulting measurements available at the time of publication. The available data was gathered and presented in order of increasing complexity, beginning with the measurement of certain

basic system functions and ending with the measurement of a mix of more complex OLTP functions. This particular set of criteria is not intended to become the standard for evaluating the performance of OLTP systems. A more rigorous standard workload/benchmark, designed by "those who develop standards", is needed. Until this standard OLTP benchmark is available, ad hoc performance comparisons will continue to be the only method available for evaluating relative computer power in the OLTP arena.

A final note: Instead of replacing all the NonStop II processors with NonStop TXPs, the DP manager might consider adding a few NonStop TXP processors (and disc drives) to augment the eight NonStop II processors. As the performance data indicated, replacing all eight processors with the faster NonStop TXP processors substantially increased the amount of CPU power in the configuration but did not much alleviate the system's disc-drive bottleneck. Much of the added CPU power is, then, essentially wasted.

The example from which performance metric *P5* was derived was chosen to compare the NonStop II processor to the NonStop TXP processor. It was purposely restricted to illustrate the performance improvement that would result from merely replacing the CPU components. Deriving the ideal system (built with NonStop TXP processors) for this OLTP workload was not the purpose of this investigation. Nevertheless, an important lesson in system configuration was demonstrated; i.e., a balanced system, in which the disc-processing capacity matches the disc-processing demands of the workload and the CPU capacity matches the CPU-processing demands of the workload, is the most efficient system to configure.

In conclusion, the results of this study indicate that the NonStop TXP processor is a worthy successor to the NonStop II processor. Its designers have achieved compatibility with the NonStop II processor while providing much-improved performance. The NonStop TXP processor is clearly a high-performance processor for on-line transaction processing systems.

Table 8.
Summary of performance index comparisons for the NonStop II and NonStop TXP processors.

Performance index	Ratio P_x(II):P_x(TXP)
P1	1: (2.5-3.1)
P2	1: (2.7-3.0)
P3	1: (3.2-4.1)
$P4_r$	1: 1.37
$P4_s$	1: 2.07
P4	1: 1.68
$P5_{cap}$	1: 1.55
$P5_{rt}$	1: (2.5-infinity)
$P5_{thru}$	1: (2.7-infinity)

References

Carr, R., and Campbell, G. 1984. The NonStop TXP Processor word length. Tandem Computers Incorporated internal report.

Chou, T.C.K., Oleinick, P., and Singh, A. 1984. Language-directed modeling. 17th Hawaii International Conference on System Sciences, Honolulu, Hawaii, January 4-6.

Curnow, H.J., and Wichmann, B.A. 1974. A synthetic benchmark. *The Computer Journal,* vol. 19, no. 1, 43-48.

ENVISION Reference Manual. 1984. Tandem Computers Incorporated internal manual. Part no. 82351 B00.

Gruenberger, F. 1966. Are small, free-standing computers here to stay? *Datamation,* vol. 12, no. 4 (April), 67-68.

Horst, R., and Metz, S. 1984. New system manages hundreds of transactions per second. *Electronics,* April 19, 147-151.

IBM Corporation. 1984. *Service for Consultants.* Hardware prices, vol. 1, 48.

NonStop TXP System Support Team. 1983. Atoms of NonStop TXP processor performance. *Focus* (Tandem Computers Incorporated internal publication), vol. 3, no. 3, 10-18.

Schneidewind, N.F. 1966. *Analytic model for the design and selection of electronic digital computers.* Ph.D. thesis, University of Southern California, Los Angeles.

Singh, A. 1983. NonStop TXP Processor atoms. Tandem Computers Incorporated internal report.

Tandem Computers Incorporated. Tandem product and price guide (internal). 1984.

Peter Oleinick is the manager of the Performance Group in Software Development. He joined Tandem in August 1978 to work on performance evaluation. As Software Development grew, a group evolved to concentrate on performance modeling, measurement, and analysis under Peter's direction. Before coming to Tandem, Peter obtained a B.S. in Electrical Engineering from the University of Michigan, and a Masters and Ph.D. in EE/CS from Carnegie-Mellon University.

Optimizing Sequential Processing on the Tandem System

For programmer-analysts used to traditional sequential processing, designing on-line transaction processing applications for a multiprocessor architecture often requires an adjustment in thinking. Their traditional approach to application design must change so that they can optimize the processing of the on-line transactions and fully utilize the features of the multiprocessor system. Often, the methods they use to optimize the sequentially-oriented portions of the application must be reconsidered also. Requirements such as fault tolerance, data integrity, and tunability are different for sequential and on-line environments. Table 1 identifies some of these fundamental differences.

In this article, techniques for optimizing sequential processing on the Tandem system are explored. System configuration, file blocking, and parallel processing techniques are discussed in the order in which they should be implemented (i.e., parallel processing is not as effective if system-configuration and file-blocking possibilities have been overlooked).

Table 1.
Fundamental differences between sequential and on-line processing.

	Sequential processing	On-line processing
Typical file access	Sequential	Random
Recoverability	File level	Transaction level
Data integrity/ Data-base consistency	File level	Record level
Tunability	Larger data blocks, fewer physical I/Os	Modular growth
Ultimate goal	Shortest wall-clock time	High availability and fast response time

Finally, a general approach to optimizing sequential processing is presented, illustrated with examples of typical applications.

A primary consideration in the presentation of these techniques is their ease of installation into an existing program. The changes required are generally minimal and do not affect the application portion of the code. In some instances, the programs need not be recompiled. In others, small, simple programs or pieces of code must be added.

Example programs are available either in current Tandem manuals or through Tandem support analysts. The list of references identifies specific manual sections and other supporting written information.

Benchmarks

Simple COBOL benchmark programs were used to test the effects of each technique. The results of these benchmarks are presented as each technique is discussed. The values of appropriate parameters were varied to explore their effects on wall-clock time and system resource usage. The benchmarks were run on the A06 version of the GUARDIAN operating system, on both the NonStop II and NonStop TXP systems.

These benchmarks are intended as guidelines for comparing various techniques. The results should not be used as absolutes in measuring system performance. Many different factors contribute to performance measurement, and while controlling all of them is very difficult in a lab, it is nearly impossible in a production system.

The XRAY performance analysis tool was used to monitor and verify the benchmark results and show the impact of each test on system resources. XRAY should be used in conjunction with any performance optimization. Taking XRAY measurements before any modification is made is the surest way to determine where optimization is actually needed.

Those analysts familiar with XRAY statistics will observe that sequential processing has significantly different optimal measurement ceilings than those of on-line transaction processing. The value for the XRAY metric CPU BUSY can be much higher, disc utilization can be driven to the limit for sequential access, cache hits can be very high (this is not necessarily optimal, as is later shown), and queue lengths for resources can be longer. Finally, CPU workload over an entire system can be extremely unbalanced.

System Configuration Considerations

Several options in the system generation (SYSGEN), hardware configuration, and general procedures for running applications can be useful in improving sequential processing.

Disc Configuration

At SYSGEN, the following disc parameters can be configured according to how the disc is to be used.

On the Tandem NonStop system, mirrored disc volumes can be configured for either PARALLELWRITE or SERIALWRITE. PARALLELWRITE is much faster for individual writes because of the overlap gained with parallel-disc seeking and writing. SERIALWRITE can provide another access path to disc since two controllers are not tied up for a single operation. On the NonStop II system these parameters are not available; the system automatically selects the appropriate option, based on the dynamic configuration. If both halves of a mirrored volume are on the same controller, seeking occurs concurrently on both, but all other operations are done one at a time. If each device in a mirrored pair is on a separate controller, seeking or writing occurs concurrently on both.

For most applications, reads on a disc volume outnumber writes by a wide margin. The SPLITSEEK option provides a faster seek time when records are read randomly, by positioning the heads for the two mirrors on the inside and outside of the spindles. If writes are proportionally higher, the SLAVESEEK option, which keeps both sets of heads positioned at the same place, should be considered. For example, if a file is written and read sequentially, there is no advantage to SPLITSEEK; SLAVESEEK saves more time.

The "phantom-disc" technique is useful for systems with removable disc volumes. This simply involves adding extra disc definitions on mirrored-disc strings in the SYSGEN so that more logical discs are configured than physically exist. This gives the system manager the capability to bring down half a mirror, remove the pack, install another pack, and bring it on line as a disc volume that can be addressed separately. This process can be reversed at any time to bring the "downed" half of the mirror back on line. A REVIVE can then be used to synchronize it with the primary.

A primary consideration in the presentation of these techniques is their ease of installation into an existing program.

If temporary work files are extensively used, consideration should be given to unmirroring the disc volumes on which they reside. In most instances, data in work files is not critical, since it can be rebuilt. If the normal recovery for the job is to rerun it, regenerating the output data when a disc fails is not a serious problem.

Unmirroring disc drives has three advantages in sequential processing:

- Physical disc capacity is doubled.
- About 20 ms per access is saved over PARALLELWRITE-configured mirrored discs, and more than 50 ms is saved per access over SERIALWRITE-configured mirrored discs. Of course, this is dependent on the size of the block being written and the mode of access.
- Depending on the configuration of the disc controllers, unmirroring may provide another physical access path.

Critical or permanent data should not be kept or modified on unmirrored packs since this would introduce a single point of failure. Files audited by the Transaction Monitoring Facility (TMF) and all audit trails should always be mirrored.

File System

The following considerations about the file system are also important. First of all, making sure the REFRESH flag in the file header block is off prevents the file header block from being rewritten to disc whenever a file's EOF changes. While useful for on-line processing, rewriting the file block header to disc is inefficient for sequential writes since the entire job is rerun in the event of a failure. REFRESH can be called at any time by the program and is invoked when the file is closed.

If two files on the same volume are heavily accessed simultaneously (e.g., one is read from, then another is written to), moving one of the files to another pack can provide an immediate reduction in seek time and up to a 50% improvement of the job's run time in some instances.

When jobs access a number of files simultaneously, or when several processes access files heavily, additional disc controllers are an asset. The more paths there are to the data, the more options there are for performance improvement, both for sequential and on-line environments.

Larger block sizes for structured files ensure better access times, especially for sequential reads. A larger block size means a higher blocking factor, which reduces the number of physical I/O operations.

If a system has several disc volumes and some fairly large files or files that are used heavily, partitioning should be considered, whether the processing is on-line or sequential. Even if a file is not in danger of overflowing its current disc volume, partitioning offers a number of advantages:

- *More access paths to the file.* This can improve the throughput for both sequential processing (see "Parallel Processing") and on-line processing (by reducing the bottleneck potential and better balancing the system).
- *Increased fault tolerance.* If one partition of the file is down because of a disc, CPU, or node failure, the other partitions are still available as long as the root partition is available when the file is opened. Also, when a RESTORE or TMF ROLLFORWARD is required, only the partition that is down need be recovered, and this can be done while the other partitions are being used by the application.
- *Larger file sizes.* Instead of being limited to the size of a single disc volume, up to 16 discs can be used for a single file. These discs need not be the same type or capacity, and each partition can be a different size.
- *More options for data distribution.* A single file can span discs and nodes, offering a very flexible environment that can be seen logically as a single entity.

The RUN Statement

While sequential jobs are often executed with a simple RUN statement and no Command Interpreter options, the following are some options for improving their performance:

- *Specify a CPU.* This avoids having $CMON or the Command Interpreter select one. (The latter might create contention in the processor if the CPU selected were busy when the job was run.) Each job should be reviewed, and the best processor for it should be determined by the system load and whether or not it will contend with other processes using a large amount of the total system resource.
- *Set the job's priority.* This can improve its performance if it is the most important of several processes running at the same time.
- *Specify the NOWAIT option.* This frees the terminal for other uses. While this does not improve the job's performance, it may help the operator. It should not be specified if COBOL DISPLAY or ACCEPT to the home terminal are used within the program or if other job steps that are dependent on the first job's completion follow in the same obey file.
- *Specify a file name for a temporary unstructured file.* This avoids having the COBOL run-time library create the file on the default volume, which would cause contention if other needed files were already on the volume.

Also, the default size may not be large enough. The job might run at first, but if the number of records increased, the job might fail because of inadequate disc space. This is especially important for a sort-work file; the default is to allocate enough room for approximately 10,000 records.

- *ASSIGN all application files in an obey file.* This makes it easier to keep track of a program's input and output and is a convenient way to allow a program access to different files without recompiling.

- *Name the process.* This makes it easier to monitor its operation and performance. If parallel processing is implemented and the processes involved send data to each other, a named process is much easier to work with.

Below is a sample of the parameters that should be set, not defaulted, for the RUN statement:

```
CREATE $VOL.SUBVOL.TEMP, EXTSIZE
    :
ASSIGN COBOL-FILE, $VOL.SUBVOL.FILE
    :
RUN X /CPU X, PRI n, NOWAIT, NAME $NAME/
```

The SORT Utility

Whenever possible, the SORT utility should be used instead of sorts called by COBOL. This is because, in the SORT utility RUN statement, MEM 64 can be specified to make more memory available for the sort process. This improves sort times significantly. This option cannot be passed to SORT from COBOL.

Also, for optimal sort time, SORT should read the file directly, regardless of whether the SORT utility is used or the file name is specified in the COBOL SORT statement. If an input procedure is used, sorting is much slower, as COBOL must pass each record separately to SORT with an interprocess message. Sequential block-buffering, discussed later, is used by SORT for all structured file reads.

The Spooler

If the spooler collector process is overloaded with jobs opening and writing to the spooler, adding another collector process can help to spread the workload. Each collector process has its own data file, which can help to balance the spooler requests across separate disc volumes. This can be run out of a single spooler supervisor and is transparent to the SPOOLCOM or PERUSE user. Many systems are already configured with two collector processes, with one used for short jobs and the other for longer report output. (See the subsection on Level 3 spooling in "File Blocking" for more ways to streamline spooler processing.)

Even if a file is not in danger of overflowing its current disc volume, partitioning offers a number of advantages.

TMF

The following are some guidelines for using TMF in sequential processing:

- Avoid using TMF to audit intermediate or work files.

- Use the LOCKFILE procedure to eliminate record-level locking on audited files. Note that only one process may lock a file at any given time.

- Block TMF transactions for audited files between BEGIN-TRANSACTION and END-TRANSACTION statements. With the A06 version of TMF, a maximum of 1808 records and 922 key locks can be held simultaneously.

- Leave audit flags on. Turning them off to permit faster updating is dangerous, since it creates a grave risk that the flags might not be turned back on again. This would make ROLLFORWARD impossible for those files until another on-line dump were taken.

Current Releases

The best system-configuration tip of all is to use the current releases of Tandem software. Tandem continuously upgrades its products in response to user needs. Significant improvements can often be realized with the installation of the latest system software release.

File Blocking

At the application level, blocking data into and out of a program is probably the performance factor that is easiest to control and it is one that has a major impact. The goal is to reduce the number of physical I/O operations and the number of system-level calls to the Disc process or cache management function. The three methods discussed below are simple to implement, can show impressive improvements in run time, and can significantly reduce the load on system resources.

Sequential Block-buffering

One of the easiest ways to boost serial-read performance for structured files is to use sequential block-buffering. This technique (described in the *ENSCRIBE Programming Manual* and further explained in an unpublished paper, "Sequential Block-buffering for COBOL Programmers," by Chris Ohland of Tandem) can be used on any structured file. Essentially, in the NonStop processor, sequential block-buffering moves a structured file block directly into the user data area, and in the NonStop II and NonStop TXP processors, it moves it into the Process File Segment (PFS). Then it is deblocked without having to go through cache management or an interprocess communication. The savings in read time are remarkable; smaller records are read three times faster.

Table 2 compares the results of a benchmark testing standard and sequential block-buffering. The program read 10,000 records, first with standard I/O operations and then with sequential block-buffering added. Block size was 4096 bytes. The record sizes were varied.

The XRAY statistics show that for smaller records, the number of read requests to the Disc process dropped 95%. The cache hit rate (the number of times the appropriate record was found in memory) fell from 90 per second to 21 per second, reflecting the fact that the index blocks were still kept in cache. The rate of disc physical reads tripled, showing much higher throughput, but the Disc process CPU BUSY percentage dropped from 43% to 6%. The test program CPU BUSY percentage increased 33%, showing that deblocking was still being done, but outside of the Disc process and cache manager.

Not only can a much higher throughput be obtained with sequential block-buffering, but the load on the disc resource is significantly reduced. The disc could be accessed for another file without serious contention problems. Without sequential block-buffering, cache hits on the sequential reads consume the Disc process resource while the physical disc is almost idle.

Sequential block-buffering is intended for sequential read-only requests on ENSCRIBE structured files. The file can be STARTed at different points and then read sequentially. Random reads or writes work, but do not show a performance improvement since the working storage buffer is completely replaced with each I/O operation. The file can be shared.

Table 2.
Time required to perform standard versus sequential block-buffering. (Sequential reads of structured files, 10,000 records.)

NonStop II system

Record size (bytes)	Time (secs), keyed files (a) Standard buffering	(b) Sequential block-buffering	Time (b) as a percentage of time (a)
100	111	38	34%
200	127	55	43%
500	162	98	60%
1000	200	145	72%
1500	278	276	99%

NonStop TXP system

Record size (bytes)	Time (secs), keyed files (a) Standard buffering	(b) Sequential block-buffering	Time (b) as a percentage of time (a)
100	44	19	43%
200	52	28	54%
500	68	48	71%
1000	96	94	98%
1500	190	190	100%

Finally, updating by other processes is possible, but once a block is read into the user data area, any subsequent updates to that block by other processes do not update the user data area copy of the block.

Unstructured Files

Four file types are available in the ENSCRIBE file system: keyed, relative, entry-sequenced, and unstructured. If a file has special attributes such as keys, alternate keys, variable-length records, ordinal record positioning, update capability, or ENABLE or ENFORM requirements, the appropriate type of structured file should be used.

If the file holds only sequential, fixed-length records it should be unstructured for faster processing. An unstructured file has no pointers, control blocks, or slack space, and records are physically adjacent. There is no record size on an unstructured file; whatever record size is specified in the COBOL file description is read into the record area. Maximum record size is 3584 bytes.

A prime application for unstructured files is for intermediate data that is generated sequentially to be read sequentially.

Reading Unstructured Files

Using a BLOCK CONTAINS clause in the file-definition statement improves read access to unstructured files. It causes the entire block to be read into the user program data area. Reads of the file go outside of the user program environment only when a new block is needed.

In Table 3, the results of a benchmark using unstructured and structured files with a BLOCK CONTAINS clause are shown. The benchmark compared entry-sequenced and unstructured files by reading 10,000 records from each type. Record sizes varied from 100 to 3000 bytes. The entry-sequenced file was blocked at 4096 bytes.

The XRAY analysis showed a dramatic drop in the Discprocess and cache overhead and an increased number of physical disc requests for the unstructured files, demonstrating higher throughput. (This was especially noticeable for smaller record sizes.) The difference is due to the reduced number of interprocess communications needed to obtain record data and the subsequent reduction in load on the Discprocess and cache management for unstructured files.

Table 3.
Time required to perform reads on structured versus unstructured files. (10,000 records.)

NonStop II system

Record size (bytes)	Time (secs) (a) Structured files (entry-sequenced)*	(b) Unstructured files†	Time (b) as a percentage of time (a)
100	117	13 (35)	11%
200	136	24 (17)	18%
500	143	58 (7)	41%
1000	183	128 (3)	70%
1500	257	188 (2)	73%
2000	280	361 (1)‡	129%
3000	393	378 (1)	96%

NonStop TXP system

Record size (bytes)	Time (secs) (a) Structured files (entry-sequenced)*	(b) Unstructured files†	Time (b) as a percentage of time (a)
100	44	6 (35)	14%
200	53	12 (17)	23%
500	68	17 (7)	25%
1000	96	33 (3)	34%
1500	190	100 (2)	53%
2000	185	190 (1)	100%
3000	214	210 (1)	100%

*Entry-sequenced files blocked at 4096.

†Unstructured blocking factor shown in parentheses.

‡With a record size of 2000 bytes, no blocking can be done on an unstructured file, but the entry-sequenced file can still fit 2 records in a block.

Writing Unstructured Files

A good technique for generating records for an unstructured file is to block the data in the user program before issuing a write. This can be done with a simple blocking paragraph and is a very efficient way to minimize the number of physical writes for every logical write.

Table 4.
Time required to perform writes on structured versus unstructured files. (10,000 records.)

NonStop II system

Record size (bytes)	Time (secs) (a) Structured files (entry-sequenced)*	(b) Unstructured files†	Time (b) as a percentage of time (a)
100	533	33 (35)	3%
200	512	60 (17)	12%
500	527	143 (7)	27%
1000	524	303 (3)	58%
1500	561	447 (2)	80%
2000	557	710 (1)‡	127%
3000	594	887 (1)	149%

NonStop TXP system

Record size (bytes)	Time (secs) (a) Structured files (entry-sequenced)*	(b) Unstructured files†	Time (b) as a percentage of time (a)
100	335	16 (35)	5%
200	339	32 (17)	9%
500	340	78 (7)	23%
1000	348	182 (3)	52%
1500	360	270 (2)	75%
2000	355	510 (1)‡	144%
3000	384	542 (1)	141%

*Structured files blocked at 4096 bytes.

†Internal blocking factor is shown in parentheses for unstructured files.

‡At 2000 bytes, only one unstructured record can fit. The file system must move data in adjacent records on each write.

Table 4 shows the results of a benchmark that added a simple blocking paragraph to a program that wrote sequential records to an unstructured file. The benchmark consisted of writing 10,000 records to both structured and unstructured files. Record sizes were varied to show their impact at different points.

The results of the XRAY analysis were very similar to the unstructured read results: the Disc process and cache overhead dropped, and physical disc activity increased.

No other programmatic changes are needed to make a file unstructured. Incorporating these suggestions would help to move the data through the system faster and with lower system overhead.

Level 3 Spooling

The Tandem spooler function receives print data from processes, saves it in a spooler data file, and manages the printers. Standard COBOL WRITE requests to the spooler require an interprocess message for each write. The spooler collector blank-compresses this data into a block, and once the block is full, writes the block to the spooler data file. This is known as Level 1 spooling. In Level 3 spooling, the compression procedures are in the system library, and can be called directly by the application code without an environment switch. An interprocess message to the spooler collector is needed only when a block is full.

The performance improvement that can be obtained by using Level 3 spooling is impressive: writing to the spooler can be 3 to 15 times faster, depending on the number of blanks in the print data. Level 3 spooling is also useful for specifying report names, print priorities, the number of copies, and special flags from the program.

A complete description of Level 3 spooling, including a sample program, is in the *Spooler/PERUSE User's Guide.* The change from standard COBOL I/O is fairly simple; the programmer need only make separate calls for pagination and line feeds. (The sample program, on which this benchmark was based, demonstrates this.)

The benchmark to test both standard writes and Level 3 I/O wrote 10,000 lines of 132 characters each, with a range of nonblank characters. The results are shown in Table 5.

Because far fewer interprocess messages are needed with Level 3 spooling, a more predictable range of timings can be found, especially when there are other active processes on the system. The XRAY analysis bears this out. Using standard writes consumes much more of the total system resource in the form of message-system overhead.

Parallel Processing

While parallelism is an established concept for on-line processing on any multiprocessor system, it is not often considered for use in optimizing the sequentially-oriented portions of an application. Perhaps it is felt that the effort to design and implement a parallel architecture would not be worth the possible improvements in run time. There may be a suspicion that the overhead needed to run processes in parallel would overshadow any gains made through the concurrency obtained.

In reality, parallel processing can greatly improve run times for sequential processing. Its design and implementation for applications is straightforward and simple, especially when the Tandem architecture is utilized. Since parallel processing enables the application to use more system resources simultaneously, the overhead to run functions in parallel is not a significant factor. In many cases, total system resource usage is less in a parallel environment because some steps can be eliminated and information can be shared, as is discussed later in this section.

The Tandem system is message-based; i.e., all information is passed between processes via messages. A part of the operating system controls the message traffic by routing and queuing to the appropriate process. Another integral part of the operating system is the file system, which treats all I/O operations as messages. Whether a block of information is to be read or written to disc, or to another process, is essentially transparent to the sending or receiving process. This simplicity in passing data between processes allows the workload to be distributed, files to be shared, and some intermediate steps in the sequential processing environment to be eliminated.

Three basic parallel processing concepts are important in sequential processing. Each of these is explained below.

Table 5.
Time required to perform standard versus Level 3 spooling. (10,000 lines; 200 pages, 50 lines/page.)

NonStop II system

Number of blank characters	Time (secs) (a) Standard spooling	(b) Level 3 spooling	Time (b) as a percentage of time (a)
0	210	62	30%
33	203	36	18%
66	195	30	15%
132	189	8	4%

NonStop TXP system

Number of blank characters	Time (secs) (a) Standard spooling	(b) Level 3 spooling	Time (b) as a percentage of time (a)
0	97	41	42%
33	86	40	47%
66	82	11	13%
132	78	3	4%

Figure 1

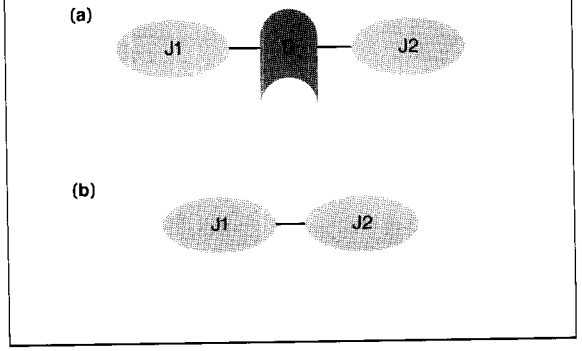


Figure 1
Eliminating intermediate files. (a) Before the intermediate file is eliminated, one job (J1) writes the intermediate data to disc (D) and another job (J2) accesses the data from the disc at a later time. (b) After the intermediate file is eliminated, the first job writes the intermediate data directly to the second via an interprocess message.

Eliminating Intermediate Files

In the traditional single-processor environment, intermediate files are often used for saving information for another job to access later. After this job is complete, the file is purged. This is easy to program. Each unit of work can be considered as separate instead of all units being viewed as a single complex multi-pass program. This modularity is not only simpler to work with in the case of the single processor, it provides the basis for running some of these units of work concurrently in the multiprocessor environment.

In the Tandem message-based environment, it is as easy to write to another process as to a disc file. This allows information to be passed from one program to another very quickly, eliminating the disc time required to write and read the file, and allowing the jobs to run concurrently, further reducing the run time. Disc space is also saved. A further benefit is the ability to increase the amount of data sent per message by internal blocking. Finally, the limitation on disc block sizes is no longer a constraint; up to 32K-byte messages can be sent with system procedure calls. Standard COBOL I/O allows 4K-byte blocks.

Figures 1a and 1b show the flow of data with and without the intermediate file. Note that intermediate disc access is no longer needed. An interprocess message for each record/block is also eliminated.

Figure 2

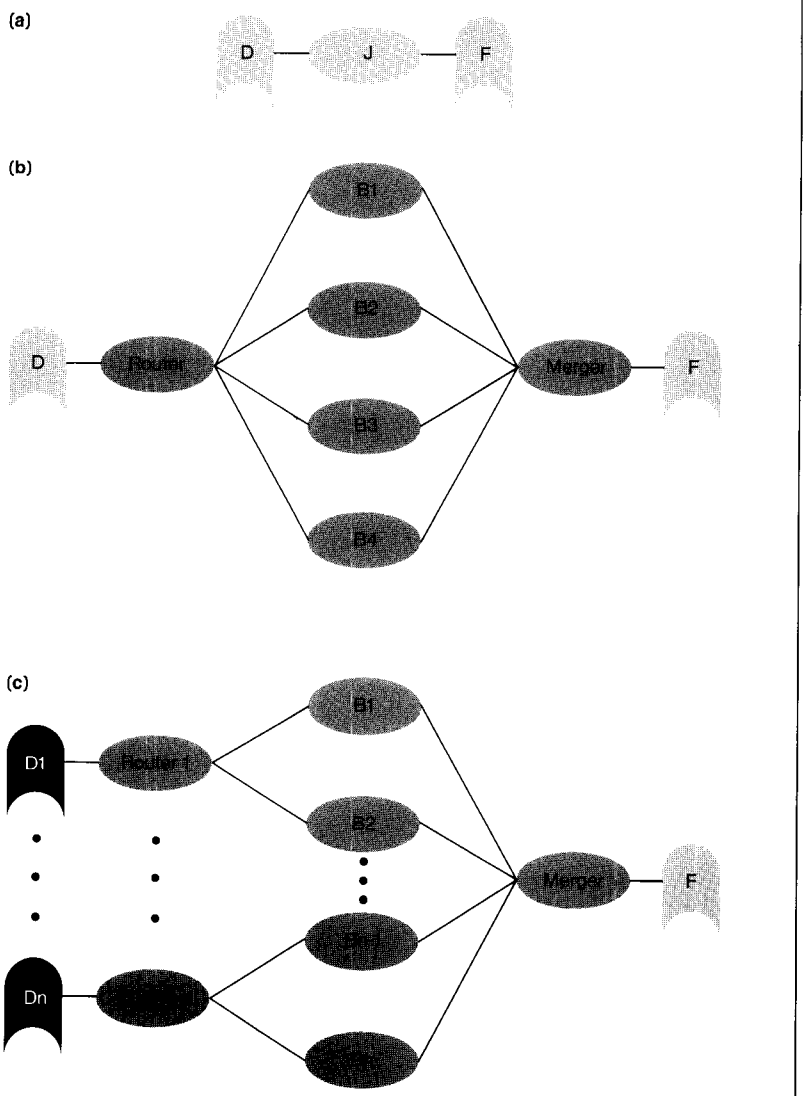


Figure 2

Routing and merging. (a) Before routing and merging is used, the job (J) reads the data file (D), transforms the data, and then writes the transformed data to a file (F). (b) When routing and merging is used, the job reads the data file and routes segments of the data to the bottleneck processes (B1-4). They transform the data and send it back to the main job to be merged and written. (c) When routing and merging is used with a partitioned input file, copies of the router program read each partition of the input file and route segments of the data to the bottleneck processes. These processes then transform the data and send it to a single merger program that combines the data and writes it out as a single stream.

Routing and Merging

Sometimes a single job takes a long time to run. This job may take in one or more large files, transform them, and generate large files or reports. A classic example is the large sort job, in which a file is read, sorted, and written out in a new order. The speed of the sort is usually the constraining factor, especially if the file-blocking techniques discussed earlier have already been applied.

With routing and merging, this bottleneck can be eliminated. Multiple copies of the bottlenecked portion are started in separate processors, a segment of the work is passed to each, and then the segments are merged to produce the final output. Figures 2a and 2b show the flow of data before and after routing and merging is done.

Only two programs are needed for this technique. The first has three components: the application code (without the bottlenecked portion), a round-robin write procedure to route segments of the input file to the bottleneck processes, and a merge function to receive the transformed data from these processes and combine the segments. The second program, the bottleneck process, is a simple routine that reads segments of the data file from the router, performs the necessary function, and then writes the transformed data to be merged. The bottleneck process can be replicated as many times as needed. For optimal performance, the route-merge program should run in a processor of its own, and each bottleneck process should have its own processor.

An extension of this approach is to multiprocess a partitioned file on input. Partitioning allows the data to be further processed in parallel. Multiple copies of the router portion can read the individual partitions and then either transform the data or pass the data to multiple subprocesses as before, for further gains in throughput. For this, three programs would be needed: a router, a bottleneck process, and a merger. (See Figure 2c.)

All message I/O operations are done with COBOL READ and WRITE statements. The route and merge functions are straightforward.

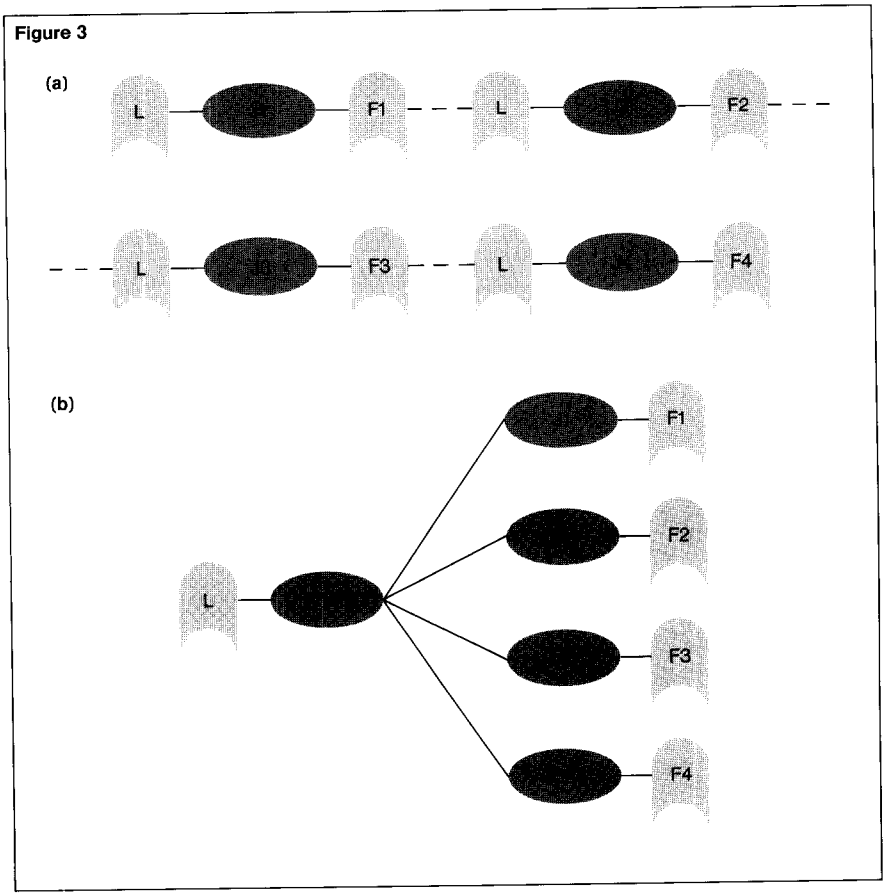
Distributing

Often a single file is used as sequential input by several programs. Each program transforms the data differently, generating separate reports, files, or tapes, and the entire file is read each time. Running several of these jobs concurrently, even with a blocking technique, can cause severe contention problems on the disc and Discprocess, and no significant improvement in overall run time.

A remedy for this is to place a distributor program in front of these programs to read the file once and pass the data to each of the programs. Then, not only do the jobs run in parallel, but the disc file is read only once. Once again, standard COBOL I/O can be used. Blocking data from the distributor process to the application programs further enhances performance.

Figures 3a and 3b show the data flow before and after this type of parallel processing is implemented.

Figure 3



A General Approach to Optimizing Sequential Processing

The following procedure for optimizing sequential processing is recommended:

1. Create a system-configuration chart that shows all devices and their locations.
2. Diagram the sequential processing portions of the application, noting all files, major program steps, interrelationships, and the CPU in which they run.
3. Monitor the system with XRAY during the entire sequential processing window.
4. Check for the opportunity to use any system-configuration parameters that streamline sequential processing.
5. For files with sequential I/O, check for opportunities to implement file blocking.
6. Implement the changes indicated in steps 4 and 5.
7. Monitor the system again.
8. If time constraints are still not met, evaluate and implement appropriate parallel-processing techniques.
9. Monitor the system again. Continue to evaluate opportunities for improvement until the necessary time constraints are met or no further optimization can reasonably be performed.
10. Continue to monitor the system periodically for additional opportunities, especially those due to an increase in application volume.

To illustrate this general approach, three examples based on actual application situations are presented below. Benchmarks for each example were run to compare the performance of the applications before and after optimization techniques were applied.

Figure 3

Distributing. (a) Before distributing, each job (J1-4) reads the log file (L), transforms it in a different way, and writes out the new information to files (F1-4). These jobs run one at a time. (b) With distributing, a distributor program reads the L file once and passes copies of the data to each of the concurrently running jobs.

Figure 4

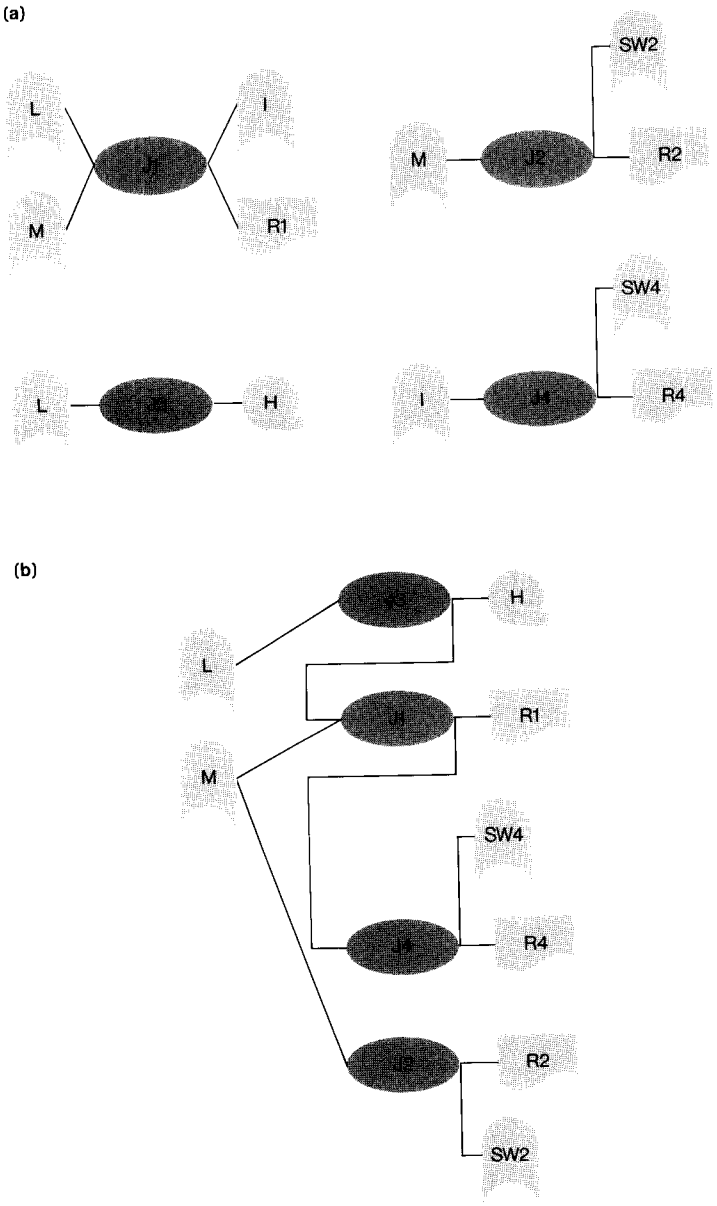


Figure 4

Example application 1, interdependent processing. (a) In the base application, each of the four jobs accesses a different combination of files. Job 1 reads log file L and master file M, and writes intermediate file I and report R1. Job 2 reads M, sorts using a sort-work file (SW2), and writes R2. Job 3 reads L and writes to history tape H. Job 4 reads file I (generated by Job 1), sorts using SW4, and writes R4. (b) When parallel processing is used, all four jobs run concurrently. Job 1 reads M, reads the data from L from Job 3, writes the former I data directly to Job 4, and writes R1. Job 2 reads M, sorts using SW2, and writes R2. Job 3 reads L, writes a copy to Job 1, and writes to tape. Job 4 reads the former I data directly from Job 1, sorts using SW4, and writes R4.

Example 1—Interdependent Processes

In the first application example, four jobs ran consecutively, performing the following functions (see Figure 4a):

- Job 1 read the log file sequentially, did a keyed read from the master file for each log file record, wrote to an intermediate file, and generated a report.
- Job 2 read the master file sequentially, sorted it, and generated a report.
- Job 3 read the log file sequentially, generating a history tape.
- Job 4 read the intermediate file sequentially, sorted it, and generated a report.

The files and reports were characterized as follows:

- The log file was entry-sequenced and had 10,000 records of 256 bytes each.
- The master file was keyed and had 10,000 records of 512 bytes each.
- The intermediate file was entry-sequenced and had 10,000 records of 768 bytes each.
- The reports were 10,000 lines long, 132 characters per line.

At first glance, the jobs seem interdependent. The log file was needed by two programs, as was the master file. The first job generated a file needed by the fourth job. Three reports were written.

To analyze each job's system-resource usage accurately, however, the XRAY performance analysis tool was used. System configuration problems were checked first (as they should be). In a benchmark running these jobs, the following was observed with XRAY.

- The log, master, sort-work, and spooler-data files all resided on the same disc volume. The intermediate file was on a different volume attached to the same controller string. This created severe contention on the disc for Jobs 1, 2, and 4. This could be alleviated by moving the master and log files to discs on other controllers on other processors.

- The sort-work files were defaulted, ending up on the same disc as the other files. Explicitly creating these files on separate discs would reduce contention in Jobs 2 and 4. Specifying larger extent sizes would allow room for growth.

- The processor in which the jobs were to run was not specified. This resulted in a variation in run time, depending on whether or not the jobs' disc files were in the same CPU and whether or not other processing was taking place. Identifying the best CPU for a job would help.

Next, the jobs were examined for file-blocking possibilities. The following items were noted:

- The log file could be read with sequential block-buffering in Jobs 1 and 3.

- The master file could be read with sequential block-buffering in Job 2.

- Since the intermediate file was only read sequentially and the record size was fixed, the file could be made unstructured and read with the BLOCK CONTAINS clause in Job 4, and written with an internal blocking routine in Job 1.

All of the changes suggested above were made. The resulting benchmark and accompanying XRAY analysis showed much better utilization of disc and Discprocess resources. The run times for all jobs were significantly reduced.

Next, possibilities for parallel processing were checked. The configuration and file-blocking changes had already reduced the run times considerably and had prepared the jobs for making optimal use of concurrency by reducing their system resource needs. Because of this, contention for system resources was not a risk, as can be seen below:

- In Job 2, the master file could be read sequentially, with sequential block-buffering, at the same time it was being read randomly by Job 1. The file access with sequential block buffering would permit both accesses with no major contention. This would allow the two jobs to run concurrently. (Note that the file was not being updated. Sequential block buffering should not be done while the file is being modified.)

- The log file was read sequentially twice. Even with sequential block-buffering, this is not as efficient as it could be. If the distributor technique were applied, Job 3 could read the log file from disc and then pass a copy to Job 1 via an interprocess message. This means that the sequential block-buffering code in Job 1 would not be needed since it would get the data directly from Job 3 via interprocess writes. This would save a complete read of the log file and would allow Jobs 1 and 3 to run concurrently without contention. Job 3 is a better choice to read the file from disc because the run time for Job 3 is much less. This also gives Job 1 more parallelism by overlapping log-file reads with the other processing.

The total wall-clock time to run all four jobs became the run time of the longest job.

- Instead of being written to disc in an intermediate file in Job 1, the data could be passed directly to Job 4 as an interprocess message. Elimination of the intermediate file would save two logical disc accesses per record (a write, then a read), not to mention the disc space no longer needed. Jobs 1 and 4 could then run concurrently.

The data flow for the final version is shown in Figure 4b. The benchmark run times for all passes are shown in Table 6 (page 36). Note that the total wall-clock time to run all for jobs became the run time of the longest job. XRAY analysis showed a fairly well-balanced system. Disc accesses were not being excessively queued, and the system was no longer bottlenecked at any point. Doubling or tripling the number of records would not affect this balance, and the overall run time would only increase at the rate of the longest job.

Example 2—The Big Job (Overcoming the Bottleneck)

Optimizing bottleneck processing concentrates on the elements of the sequential cycle that take the most time or affect the most jobs. Improving the run time for this type of processing by a factor of two or three significantly shortens the entire job schedule. Also, if record volume increases, the bottleneck

is most affected. Thus, designing expandability into this type of job is necessary for keeping the sequential functions within their windows.

With this kind of optimization, XRAY is invaluable for identifying the true time consumers within the job. The analyst should not attempt to tune any program without an understanding of how the program uses system and application resources. First, the components of the program should be isolated, then the component that is using the most time or system resources must be identified, and finally, if the processing can be optimized, the job can be tuned and retested.

A typical example of bottleneck processing (and the backbone of much sequential processing) is the reading, sorting, and reporting of a very large data file. This type of job has three components:

- Reading the file.
- Sorting the file.
- Writing the sorted file.

The benchmark run to test the optimization of bottleneck processing used a 10,000-record keyed file composed of 100-byte records and a 10-character sort key. The output file, a report, also used 10,000 records to print that number of lines. Figure 5a shows the data flow for this job.

The XRAY analysis revealed that the input file and sort-work file were on separate discs and the program ran in the same CPU as the disc with the input file. Reading the records took approximately one half the run time; writing the report took the other half. The sort time was difficult to detect because sorting was done as the records were being read from the input file.

To shorten processing time, sequential block-buffering could be used with the read, and Level 3 spooling could be used with the write. When these enhancements were made, the run time was reduced by half.

Results of benchmarks previously discussed in this article show that reading 100-byte structured records can be much faster than this, however. Level 3 spooling should have written data to the spooler at a much higher rate. It was the interaction with the sort process that caused the processes to continue to run more slowly than necessary.

While there was no way to make the sort process run faster, using routing and merging could reduce the bottleneck. Several sort processes could be created and each could be passed a portion of the input file in "round-robin" fashion so they could sort in parallel. Once all the records were read and passed to the sort processes, a simple merge of the output from each would return the sorted file for reporting. This approach would divide the sorting across several processors and split the work-file access between multiple discs. Passing data to and from the multiple sort processes, called subsorts, would be handled by interprocess messages, using COBOL READ and WRITE statements. Figure 5b illustrates the new data flow when subsorts are used.

The changes required for the application program would be straightforward. The call

Table 6.
Processing times for example application 1, interdependent processes.

NonStop II system

Version of application	Run time (secs)					Percentage of base
	Job 1	Job 2	Job 3	Job 4	Total	
Base	2012	930	231	872	4046	100%
System-configuration modifications added	1658	851	229	865	3603	89%
File-blocking modifications added	1110	810	123	580	2623	65%
Parallel processing (4 cpus) added	875	1062	874	1620	1620	40%

NonStop TXP system

Version of application	Run time (secs)					Percentage of base
	Job 1	Job 2	Job 3*	Job 4	Total	
Base	1222	348	81	354	2005	100%
System-configuration modifications added	941	339	81	350	1711	85%
File-blocking modifications added	615	263	62	248	1188	59%
Parallel processing (4 cpus) added	477	391	476	747	747	37%

*Trident tape drive used.

to COBOL SORT would be moved to a separate program, which would be replicated as needed. (Even though multiple subsorts were to be used, only one program would be needed.) The application program would require a routing section to pass the data to the subsorts in round-robin fashion and a merge section to combine the sorted portions. The application-related code itself would require no changes.

When these changes were made, the XRAY measurements showed a much better distribution of the workload. The more copies of the bottleneck process and disc paths available, the faster the job ran. For this application, the best policy was to have each process run in its own processor, using disc space local to that processor for work files.

At this time, the components of the job were analyzed again to determine if its performance could be further optimized. Since gains in processing time from adding more subsorts become minimal after a certain point, simply creating more subsorts would not significantly increase run time.

An opportunity for further optimization through file partitioning was found, however. While the report had to be generated as a unit, completely sequentially, the order of the input file was not crucial. It was currently being read in order, but if it were partitioned, a separate router process could read each partition and either do the sort locally or pass off segments to the subsort processes. Figure 5c shows the data flow for this approach.

Figure 5

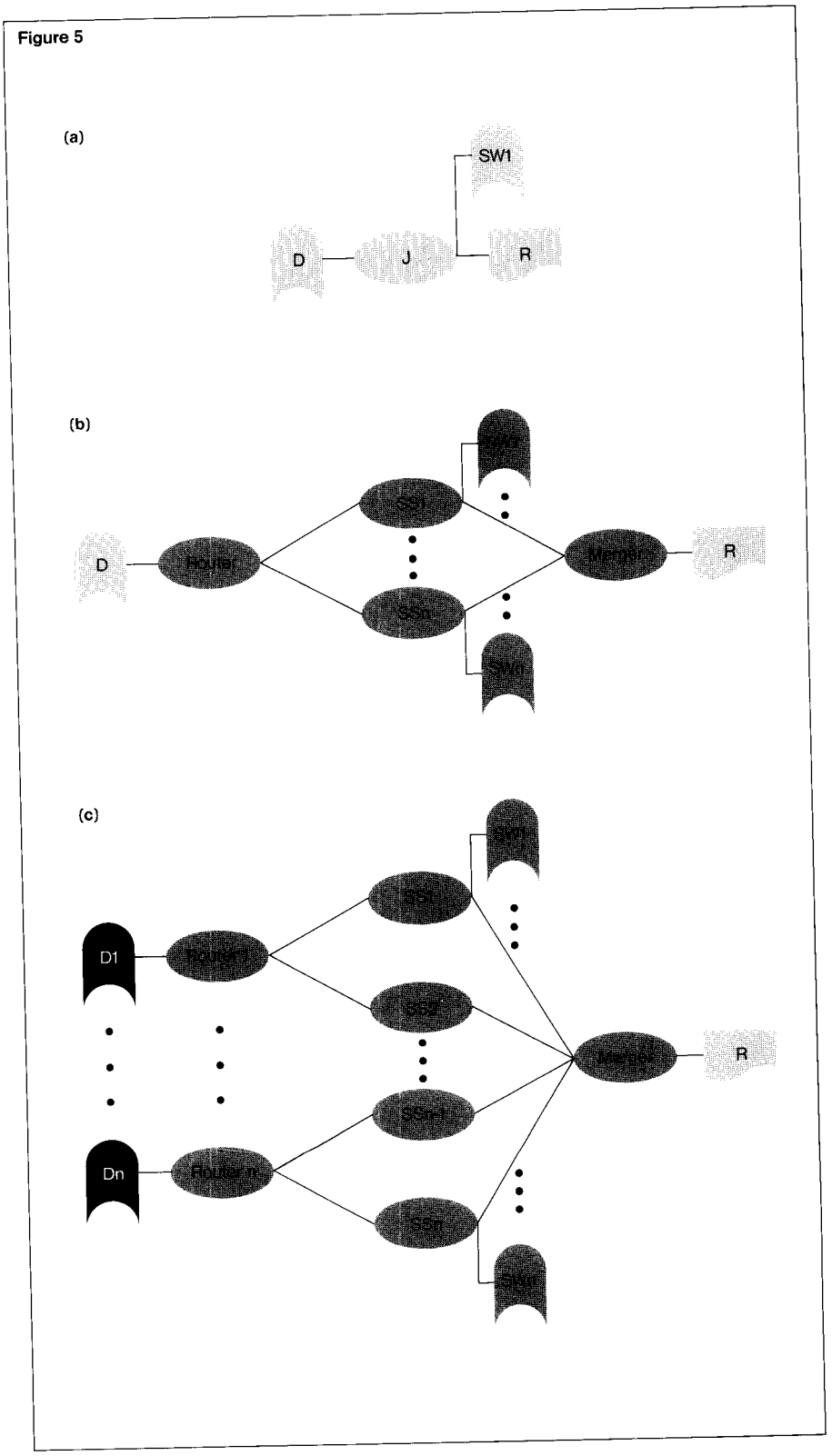
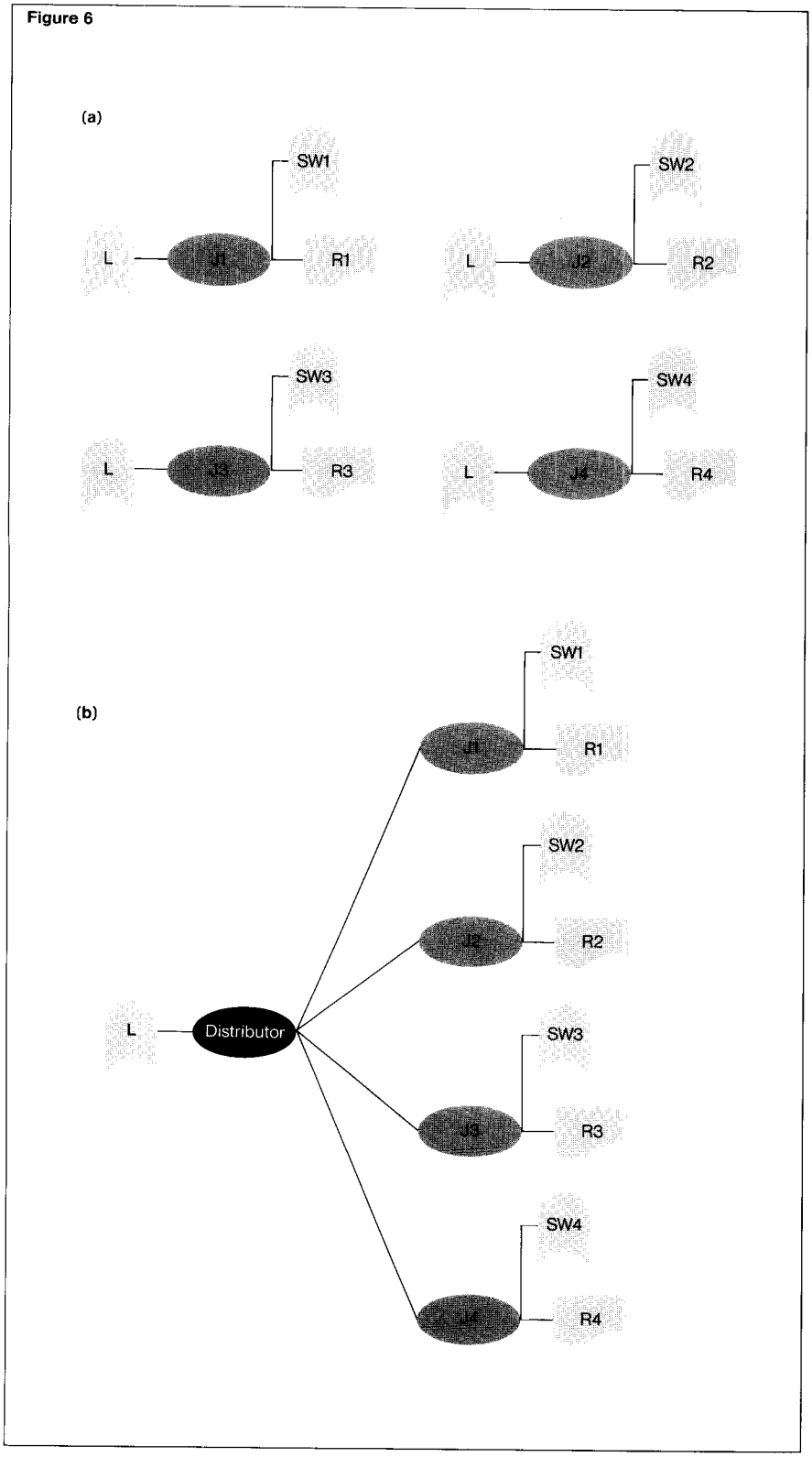


Figure 5

Example application 2, the big job (overcoming the bottleneck). (a) In the base application, the job reads data file D, sorts it using sort-work file SW1, and prints report R. (b) When routing and merging is used, the following occurs: The router portion of the main job reads the D file, passes segments of the file to the subsort processes (SS1-n), which individually make COBOL SORT calls. The returned data is sent to the merger portion of the main job to be combined and printed in the report. Each subsort process has its own sort-work file. (c) When routing and merging is used with a partitioned input file, the following occurs: Copies of the router program read each partition of the D file and route segments of the data to the subsort processes. Each subsort process makes a COBOL SORT call to transform the data and then sends it to a single merger that combines it and prints it. Each subsort process has its own sort-work file.

Figure 6



When the input file was partitioned, the job run time was again reduced. Table 7 shows the benchmark results for all the techniques used on this job.

A more detailed examination of this approach to sorting, often called Supersort, can be found in an article entitled, "Large Scale Sorting Using Multiple Processors," by E. L. Ashbaugh, published in *Focus* (an internal Tandem publication), volume 2, number 2.

Example 3 — The Popular File

In this example, a log file was generated by the on-line portion of the application. 10,000 records, 200 bytes each, were written each day into an entry-sequenced file. Four sequential programs accessed the log file. Each of them read the entire file, sorted it on a different key, and generated a full report. The flow of this job is represented in Figure 6a.

The XRAY analysis revealed that each of the jobs saturated the Discprocess when reading the data. Sorting 10,000 records also took time, although this was masked by the reading step. The sort-work files defaulted to a volume different from that used by the log file.

The first improvements indicated by this analysis included:

- Specifying a CPU for the jobs.
- Fully qualifying the names of the sort-work files.

Figure 6

Example application 3, the popular file. (a) Before parallel processing is used, each job reads the log file (L), sorts it in a different way, and prints a report. The jobs run one at a time. (b) When parallel processing is used, the following occurs: A distributor program reads the L file once and passes copies of the data to each of the concurrently running jobs. Each job reads the data, sorts it, and prints a report. Each job has its own sort-work file.

- Implementing sequential block-buffering for the entry-sequenced reads of the log file.
- Implementing Level 3 spooling for report generation.

When used, these techniques reduced the run times of the individual jobs considerably. The XRAY analysis showed that the reads and writes were much more efficient, the Disc-process was no longer overworked, and the physical disc was moderately active.

This job also provided a good opportunity for parallel processing. Since all four of the programs read the same file, running them concurrently would create a severe bottleneck on the log file, even with sequential block-buffering. A better approach would be to add a distributor program that would read the file and pass copies of the data to each of the application programs. Blocking the records between the distributor and the application programs (with standard COBOL I/O) would provide a further improvement. Sequential block-buffering could be used by the distributor program to read the log file. The data flow for this approach is shown in Figure 6b.

When these further improvements were implemented, the resulting run times were impressive; the total run time for all jobs was now the length of the longest job. Table 8 lists the run times for the job after the various techniques were applied. Note also that because of the overlap now obtained in reading and in releasing records to the sort process, the overall run time for the last run was less than the previous individual runs.

Table 7.
Processing times for example application 2, routing and merging.

NonStop II system

Version of application	Run time (secs)	Percentage of base
Base	616	100%
File-blocking modifications added	318	52%
Parallel processing added		
2 subsorts, 3 cpus	141	23%
3 subsorts, 4 cpus	101	16%
Parallel processing added, with partitioned input file (2 partitions, 4 subsorts, 5 cpus)	77	13%

NonStop TXP system

Version of application	Run time (secs)	Percentage of base
Base	202	100%
File-blocking modifications added	115	57%
Parallel processing added		
2 subsorts, 3 cpus	49	24%
3 subsorts, 4 cpus	37	18%
Parallel processing added, with partitioned input file (2 partitions, 4 subsorts, 5 cpus)	27	13%

Table 8.
Processing times for example application 3, distributing the popular file.

NonStop II system

Version of application	Run time (secs)		Percentage of base
	Single job	All 4 jobs	
Base	608	2432	100%
System-configuration and file-blocking modifications added	349	1396	57%
Parallel processing (5 cpus) added	—	330	14%

NonStop TXP system

Version of application	Run time (secs)		Percentage of base
	Single job	All 4 jobs	
Base	247	988	100%
System-configuration and file-blocking modifications added	143	572	58%
Parallel processing (5 cpus) added	—	130	13%

Conclusion

Tandem NonStop computers are designed for fault-tolerant, on-line transaction processing in a multiprocessor environment. For the optimization of sequential jobs that often accompany on-line transaction processing, techniques that take advantage of the Tandem architecture should be employed. The techniques and general approach discussed in this article can provide significant improvements in the run times of sequential jobs.

Acknowledgments

The author would like to thank the following people who contributed their time and ideas to this article. Without them it could not have been completed.

Wendy Bartlett
Jim Gray
Sue Kleiman
John Nauman
Gil Siegel
Skip Straus
Anne Wu

References[1]

Ashbaugh, E.L. 1984. Large-scale sorting using multiple processors. *Focus* (Tandem Computers Incorporated internal publication), vol. 4, no. 2.

ENSCRIBE Programming Manual. Section 2, File structures, and Section 4, File access. Part No. 82083 B00.

GUARDIAN Operating System Command Language and Utilities Manual. File utility program (FUP). Part No. 82073 F00.

Ohland, Chris. 1981. Sequential block-buffering for COBOL programmers. Tandem Computers Incorporated internal paper.

Spooler/PERUSE Users Guide. Section 4, Spooler interface procedures. Part No. 82093 C00.

Spooler System Management Guide. Collectors and data files. Part No. 82094 C00.

System Management Manual. Section 6, Configuration file. Part No. 82069 G00.

System Operations Manual. Section 5, Peripheral utility program (PUP). Part No. 82075 F00.

System Description Manual: NonStop System. Part No. 82000.

System Description Manual: NonStop II System. The Tandem architecture. Part No. 82077 D00.

Transaction Monitoring Facility (TMF) Reference Manual. Part No. 82341 B00.

Transaction Monitoring Facility (TMF) System Management and Operations Guide. TMF guidelines. Part No. 82343 B00.

[1]All manuals and user's guides listed are published by Tandem Computers Incorporated and are the A06 version.

Rob Welsh is a lead analyst in Tandem's San Francisco sales office. Before joining Tandem in April 1983, Rob led a project which converted a large sequentially-oriented warehouse distribution and accounting system to the Tandem system. He also has several years of experience in operating systems and applications design. Rob holds a degree in computer science from the University of Calgary, Canada.

TANDEM

NonStop™ Computer Systems

400094 08/84 Printed in USA