

T A N D E M
SYSTEMS REVIEW

VOLUME 7, NUMBER 2

OCTOBER 1991



Instrumenting Applications

Rules for Automated Operations

RDF • TCM

Dial-In Security

Index

Volume 7, Number 2, October 1991

Editorial Director

Susan W. Thompson

Editor

Anne Lewis

Associate Editor

Steven Kahn

Technical Advisors

Mark Anderton

Terrye Kocher

Mike Noonan

Assistant Editor

Sarah Rood

Electronic Publishing

Marcy Cross

Art Director

Janet Stevenson

Cover Art

Steve Elwood

Illustrations

Steve Elwood

Cynthia Moore

Circulation

Christina Cary

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

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

Subscription additions and changes:

As of the March 1990 issue, customer subscriptions to the *Tandem Systems Review* must be approved by a Tandem representative. Complete the subscriber portion of the order form at the back of this copy and send the form to your local Tandem sales office. Anyone who does not have a Tandem representative should fill out the subscriber portion and follow the instructions on the form.

Comments: The editors welcome suggestions for content and format. Please send them to the *Tandem Systems Review*, LOC 216-05, 18922

Forge Drive, Cupertino, CA 95014.

Tandem Computers Incorporated makes no representation or warranty that the information contained in this publication is applicable to systems configured differently than those systems on which the information has been developed and tested. It also assumes no responsibility for errors or omissions that may occur in this publication.

Copyright © 1991 Tandem Computers Incorporated. All rights reserved.

No part of this document may be reproduced in any form, including photocopy or translation to another language, without the prior written consent of Tandem Computers Incorporated.

Atalla, Challenge/Response, CLX, Cyclone, Envoy, Expand, Guardian, Measure, NetBatch, NonStop, RDF, Safeguard, SNAX, TACL, Tandem, the Tandem logo, and TMF are trademarks and service marks of Tandem Computers Incorporated, protected through use and/or registration in the United States and many foreign countries.

IBM is a registered trademark of International Business Machines Corporation.

2 Editor's Preface

4 Instrumenting Applications for Effective Event Management

Jean Dagenais

22 Writing Rules for Automated Operations

Jim Collins

34 RDF: An Overview

Jorge Guerrero

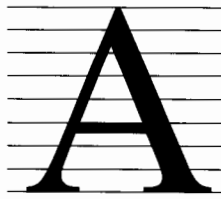
44 Capacity Planning With TCM

Wilbur Highleyman

60 Dial-In Security Considerations

Peter Grainger

71 Index



availability has become a catchword in the online transaction processing (OLTP) industry. An increasing number of computer vendors are claiming that their systems provide high availability and fault tolerance. To achieve high availability, vendors must include fault prevention, detection, and management in the hardware platform and the operating system. They must also provide an environment in which users can build applications that will support effective fault management.

In the April 1991 issue of the *Tandem Systems Review*, the first two articles discussed the level of availability offered by Tandem™ hardware architecture and the Guardian™ 90 operating system. This issue continues the discussion by describing how several Tandem products address availability in the areas of operations management, disaster planning, and system responsiveness.

The fault-tolerant architecture of Tandem systems addresses the system requirements for high availability and reliability. On large systems that include many applications and users, the MIS operations personnel must be able to respond quickly to problems that arise in the user applications as well as the system configuration. The most cost-effective way to meet the service objects of an application, such as continuous availability, is to design operations management instrumentation into the application while it is being developed. The first two articles in this issue discuss how application developers can help the operations management staff with application problem detection, isolation, and recovery.

The article by Dagenais describes a general approach to designing applications to generate effective event messages that will help the operations organization with problem detection, isolation, and recovery. It provides examples that illustrate the instrumentation in the Pathway subsystem. The article emphasizes that good instrumentation benefits human operators and has even greater benefits when it is combined with a complete automated operations solution.

Collins' article continues the discussion by proposing a methodology for designing and building rules for an automated operations system. He advocates breaking complex problems into sets of simpler ones and responding to each individual problem as a unit. The philosophy and techniques proposed in this article apply to any automated operations solution; extended examples are taken from the Programmatic Network Administrator (PNA) system. The article also describes general requirements for writing rules.

Another important element of availability is system protection in the event of a disaster. An effective way to protect critical databases is to duplicate them at a remote site that can be isolated from the disaster. The Remote Duplicate Database Facility (RDF[™]) maintains at a remote site continuously up-to-date databases that can be used for contingency planning.

The article by Guerrero shows how the RDF product can play an important role in a disaster recovery plan. It describes the process by which RDF maintains duplicate databases and discusses how one can use them as the primary database during system outages.

Because system responsiveness is the primary measure of performance for an OLTP system, Tandem provides several tools that monitor and tune system performance based on resource utilization. One of these tools, the Tandem Capacity Model (TCM), allows users to predict changes in system responsiveness as workloads and system configurations change.

Highleyman discusses the functional steps required for capacity planning with TCM and gives examples showing how TCM can apply to typical business problems. The independent audit from which the article is derived concluded that TCM can be a reliable source for the performance predications required by capacity planners.

As host systems become increasingly available through telephone dial-in connections, system managers are examining the important and complex issue of ensuring security while maintaining availability. The final article, by Grainger, explores some of the challenges associated with providing dial-in access and discusses a number of security solutions designed for Tandem systems. The author examines possible problems, such as deliberate attacks on the system and unintentional security breaches, and describes options for recovery and user authentication.

Finally, this issue includes an index of *Tandem System Review* articles. The index is a list of all articles, by subject and product, that have been published in this and each previous issue. If you would like to order back issues, please submit the order form on the last page.

Susan W. Thompson
Editorial Director

Instrumenting Applications for Effective Event Management

In the 1990s, companies can achieve a competitive advantage by being able to respond quickly to rapidly changing business conditions with the appropriate strategic applications. These applications provide new business functions that must meet increasingly stringent service objectives.

MIS organizations face major challenges not only when they design and code these applications, but also when they deliver the new business functions. If organizations do not put effective management instrumentation into their applications, the cost of operation may far exceed the initial cost of equipment and development. The users of an application include not only business customers, but also the operations organizations that must meet service level agreements.

The fault-tolerant architecture of Tandem™ systems addresses the business requirements for high availability and reliability. (Figure 1 shows a formula that defines system availability.) Nevertheless, on large distributed systems that include thousands of components and users, and that change rapidly, problems will continue to arise in the applications, operating systems, hardware configurations, and data communications, and among human operators.

This article describes a general approach to instrumenting applications to help the operations organization with problem detection, isolation, and recovery. It also describes how automated operations software can use this instrumentation to increase system availability. The article assumes that good instrumentation benefits human operators, but provides even greater benefits when it is combined with an automated operations solution.

The article begins by presenting the business case for instrumenting applications. Next, it presents a methodology for defining object state changes, using Tandem's Pathway distributed transaction processing system as an example. Finally, the article discusses important considerations that can help developers design good event messages into their programs.

The first section in the article is intended for all MIS personnel. The remaining sections are intended for readers who will design and implement instrumentation for their applications; readers of these sections should be familiar with the Pathway application environment and with Tandem's Distributed Systems Management (DSM) architecture and set of products.

Instrumenting Applications to Meet Service Objectives

When a company offers a new business function, it must identify and meet the service objectives of that function. Aggressive service objectives allow the company to gain an edge over its competition.

For example, a bank may want to offer financial services such as home banking, ATM services, and loan approvals. The service objectives for one of these functions may be to provide the same quick response to all customers, be available at any hour of the day or night, and lower the cost to customers. To meet these objectives, the application supporting the business function may have to execute hundreds of transactions per second, complete transactions in less than 1.5 seconds 95 percent of the time, and be available continuously. If the application is down frequently or requires a significant amount of operations support, the bank fails to meet its objectives, and the cost of providing the service escalates.

Figure 1

$$\text{Availability} = \text{MTBF} / (\text{MTBF} + \text{MTTR})$$

where:

MTBF = mean time between failure.

MTTR = mean time to repair.

If a system is highly reliable (MTBF is large relative to MTTR), availability is close to 1 (100%). If the MTBF is smaller, availability varies significantly with repair time.

Figure 1.

Defining availability.

Other business functions such as online lottery systems or services for stock market traders involve a high number of financial transactions. The supporting applications must provide high performance and low response times to process these transactions. Any delays in processing the transactions can have a negative impact on the service providers and their customers. For example, delays in processing the sales of state lottery tickets could lose money for the lottery and prevent some customers from buying tickets.

When service objectives are not met, a company can be exposed to financial risks directly associated with its business. Assume, for example, that a financial institution transfers billions of dollars to another company. If a communication line goes down, the money may not be sent on time. The company receiving the money can lose interest until the money is transferred to the proper account. Interest on a few billion dollars accrues rapidly; even a few hours' delay can cause a major financial loss. Restarting a downed communication line is an

obvious task. However, the problem can go undetected because of human oversight or because the application did not issue proper notification.

Choosing to instrument an application is a sound business decision.

Application designers can prevent such a loss. If the application notifies an operator (either human or automated) that the line is down, it can be restarted. However, to have proper notification, developers must design and build instrumentation into the application when it is being developed.

Users often find creative ways to use new business services to enhance their work. In some cases, they can employ the system in ways never thought of by the designers. While some applications may be able to meet a new user requirement, others may have trouble doing so. For example, a new type of transaction (such as a new, exhaustive monthly report) might cause a problem such as degraded system performance. In this kind of situation, it is essential to design instrumentation into the application so that analysts can identify the new types of transactions and properly control system resources.

To meet the service objectives of a business, one would like to develop a way to prevent problems. Since this is not always possible, one must also be able to detect and repair problems in the shortest possible time. Automated operations software, interacting with a properly instrumented application, can help to achieve both goals.

Defining the Application Instrumentation

To find out if a new business function meets its service objectives, analysts need to *instrument* the supporting application. Instrumentation encompasses all the functions required to measure and control the application.

Assume, for example, that users require the application to be running 99.5 percent of the time. This objective permits the application to be unavailable 0.5 percent of the time, or 2,628 minutes per year. If human operators need 20 minutes to resolve the average problem, they can resolve only about 125 problems (that affect application availability) per year. Now assume that an automated operator can resolve two problems per minute. By using the automated operator, the system can sustain more than 5000 problems per year and still meet the stated service objective.

Application instrumentation must provide information that allows automated operations and performance measurement tools (as well as other system tools) to increase availability and measure performance. This information must be provided simply and reliably. In addition, depending on the business functions being supported, the application may need instrumentation to support the measurement or control of security, user profiles, and work profiles.

If the requirements for a new function specify that charge-back services need to be provided, transaction accounting may be essential. Without proper information, it is extremely difficult to keep track of the resources used by individual departments or users and charge them for any work done.

To satisfy these requirements, and to verify that an application meets its service objectives in areas such as transaction counts and response time, one can instrument the application to use the Measure™ system performance measurement product. The Measure product defines a set of standard procedures that help one to instrument applications and provides an extensive set of capacity and performance management functions (Dennison, 1986).

Finally, analysts should define standard ways to support the generation of error and exception messages. Diagnostic facilities should be provided to support the tracing of user requests, transactions, and interprocess messages. Also, if the installation is using automated operations software, the application must define a standard command and control interface such as the Subsystem Programmatic Interface (SPI), a subsystem of Tandem's Guardian™ 90 operating system.

The remainder of this article focuses on using events to help with problem detection, isolation, and recovery. Instrumentation used for other purposes (such as performance measurement) is not discussed.

Deciding Whether to Instrument an Application

Clearly, choosing to instrument an application is a sound business decision. First, however, analysts must decide when and how to make the instrumentation effort. It is significantly easier to incorporate appropriate instrumentation into the application during the design phase than it is later on, when the application is in production.

Because of budget and scheduling constraints, application developers sometimes put off creating instrumentation in their application. They decide to concentrate solely on developing the business functions of the project, which are more visible to their customers. However, without instrumentation, the application cannot be managed effectively. Studies have shown that while applications take one to three years to develop, they can stay in production for five, ten, or even twenty years. Thus, the cost of managing a poorly designed application can be astronomical (Jones, 1991).

Another reason to create a proper design initially is that both design and code are reusable. Studies have shown that if code is properly designed and implemented, programmers can reuse 60 percent of it (Lanergan and Grasso, 1984). Thus, if developers design well-thought-out instrumentation into their code, much of it can be reused in subsequent projects.

Figure 2

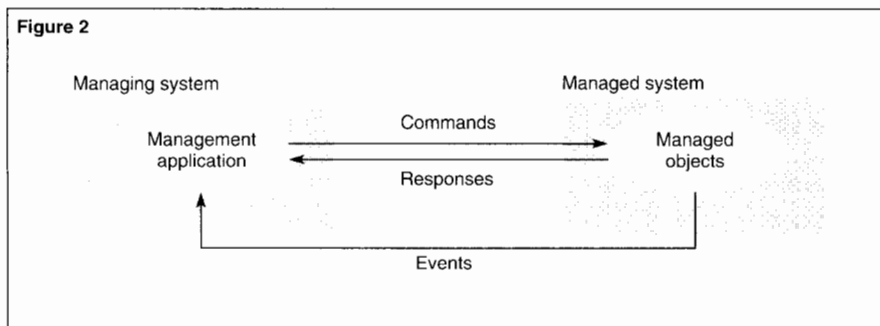


Figure 2.
DSM operations
management model.

Using the DSM Operations Management Model

The Tandem DSM architecture provides a general management model that defines management information structures and standards. (See Figure 2.) Users can realize the greatest benefit from automated operations software if their applications are compatible with the DSM model and properly instrumented.

The DSM framework, standards, and products can increase the overall efficiency of development efforts as well as enable more efficient operation and management of applications. DSM defines a message format and protocol standards (for event messages, commands, and responses) to ensure that managed applications offer consistent interfaces to management applications.

When user applications are compatible with DSM, designers can develop their own management application or use a commercial product such as Tandem's Programmatic Network Administrator (PNA) automated operations software. Management applications can recognize, diagnose, and fix problems before the end users encounter them.

Every Tandem user and third party organization now has a strong incentive to gain compatibility with DSM. Pathway fully complies with the DSM model; thus, it provides developers with a solid foundation on which to build their applications. This article assumes that developers will use Tandem's Event Management Service (EMS), a subsystem of the Guardian 90 operating system, to generate events.

A Methodology for Defining State Change Instrumentation

To instrument an application, one must identify the objects it contains and define their behavior and relationships. The methodology proposed in this article involves developing a dynamic state change model of the application. With the model, users can analyze the operational properties of the application and determine where they may need to add instrumentation to their programs. A comprehensive discussion of dynamic modeling appears in Rumbaugh et al. (1991).

Object States

In this article, an object is a distinct entity that has specifically definable behavior. Examples of objects are a terminal, file, transaction, financial institution, cashier, ATM, and cash card.

An object can have many valid states. For this discussion it is useful to divide object states into four main categories: up, down, unknown, and odd.

An object is *up* when it is started. In this state, the object is defined in the subsystem and fully meets all of its operational objectives. It can be used to provide services. Examples are an active ATM and an executing server process.

An object is *down* when it is stopped. The object is known to the subsystem, but it cannot provide useful services for the application. An example is a stopped ATM.

An object is *unknown* when it is not defined. As far as the subsystem is concerned, the object does not exist. An example is a terminal that hasn't been configured in Pathway.

An object is *odd* when it isn't in any of the other states. An object in the odd state requires some corrective action. For example, an ATM that is low on cash can still provide services to customers, but if a corrective action is not taken before it runs out of money, the ATM will shut down.

Table 1 shows an example of Guardian 90 object states classified according to the categories described above. One can draw several conclusions about these states. First, based on the classification, objects can provide useful business services for applications when they are in the up or odd state.

Whenever an object goes into a down state, one needs sufficient information to recover from this situation. One can think of this kind of recovery as *reactive recovery*. This is similar to what is known as first-level support in many operations groups.

When an object goes into an odd state, one needs sufficient information to bring the object back into an up state. One can think of this kind of recovery as *preventive recovery*. It is preventive because the object is still providing services, but if this situation is not corrected, a more important problem can occur.

Assume, for example, that an application transaction log file becomes over 75 percent full and that this is considered an odd state for this object. The common corrective action is to create a new log file or modify the attributes of the existing file.

However, if this condition is not detected, the log file could become full. If that happens, the application might have to be stopped to correct the problem. This type of recovery (preventive recovery for objects in odd states) may require much more sophisticated instrumentation and recovery procedures than would reactive recovery.

Table 1.
Classifying object states.

Object type	Unknown	Down	Odd	Up
Transaction	Not defined		Aborting	Active
Process	Not started		Suspended Unstoppable Wrong priority Debug, inspect	Running
Disk	Not defined	Down	Revive Wrong path Wrong primary CPU	Up
File	Not defined	Closed Corrupted Broken	Over/Under Threshold	Open

To detect that an object is in an odd state may require supplemental instrumentation for the object. One can think of this kind of instrumentation as *threshold alarm detection*. One can implement it either within or outside the application controlling the object. For example, the Tandem Disk Process 2 (DP2) does not indicate that a file is over 75 percent full. Either the application or, more realistically, a monitoring subsystem must take responsibility for instrumenting for this condition.

This analysis suggests that there are two kinds of events that are of interest for operations management. The first kind tells when an object changes state and requires reactive recovery. The second kind tells when an object goes over a threshold, which may also cause a state change, and requires preventive recovery. A dynamic state change model can help one identify where to design events for both kinds of recovery.

Dynamic State Change Modeling

The dynamic modeling approach is not a mechanical one. Many iterations may be required to create a final model. This approach comprises five steps:

1. Identify the objects that are meaningful to the subsystem and specify their relationships and attributes.
2. Prepare one or more typical dialogs (scripts) between the operations staff and the system to get a feel for expected and unexpected system behavior.
3. Identify the behavior of the different objects.
4. Build a dynamic model of the objects' application.
5. Analyze the model and specify where instrumentation is sufficient and where more instrumentation may be required.

The following discussion of dynamic modeling uses the Pathway subsystem as an example. It focuses on the types of events that are generated when an object changes state. Pathway encompasses a set of products aimed at developing and managing large business applications.

From a developer's point of view, Pathway is a framework for developing applications. From an operator's point of view, Pathway is a subsystem running in a Tandem system. Pathway makes an especially useful example because of this double view. First, Pathway is an example of a well-instrumented subsystem. Second, the extensive instrumentation in the Pathway subsystem makes it easier to manage a user application based on Pathway. The instrumentation in Pathway provides a solid foundation for creating an application framework and using automated operations software, which will increase system availability.

By using the dynamic modeling approach to examine the Pathway subsystem, one can see how Pathway objects change states and which EMS event messages are produced. One can also determine if additional event messages are needed.

Step 1: Identify Objects in the Subsystem

Each application has its own objects. One must understand the problem and select which objects are meaningful in one's situation. For example, one can identify the following basic object classes defined by the Pathway subsystem:

- PATHMON.
- Terminal control process (TCP).
- Terminal.
- Program.
- Server class.

All these objects are meaningful in the context of operating a Pathway environment. From an application programmer's perspective, however, only programs and servers may be relevant; PATHMON, TCPs, and terminals are simply other objects provided by Tandem. Nevertheless, programmers need to understand how these objects interact and what type of instrumentation may need to be added to their programs.

Assume, for example, that an application server abends and is not instrumented to generate the information needed to identify the cause of the problem. In such a case, it may be extremely difficult to detect that a problem exists and, once detected, to analyze and correct it.

After the objects are identified, one must show how they relate to one another. Figure 3 is an object diagram showing the associations (*constraints*) between the Pathway objects. A constraint is a functional relationship between objects: a statement about a condition or relationship that must be maintained as true. For example, objects in a Pathway application must satisfy the following constraints:

- A Pathway system can have only one PATHMON.
- A PATHMON manages one or more TCPs and one or more server classes.
- A TCP manages one or more terminals.
- A terminal can execute one or more programs.

As shown in Figure 3, multiple constraints affect a number of related objects. For example, a Pathway system can *own* only one PATHMON. One terminal can *execute* one or more programs. Developers must understand the constraints in their application in order to design an effective event management strategy.

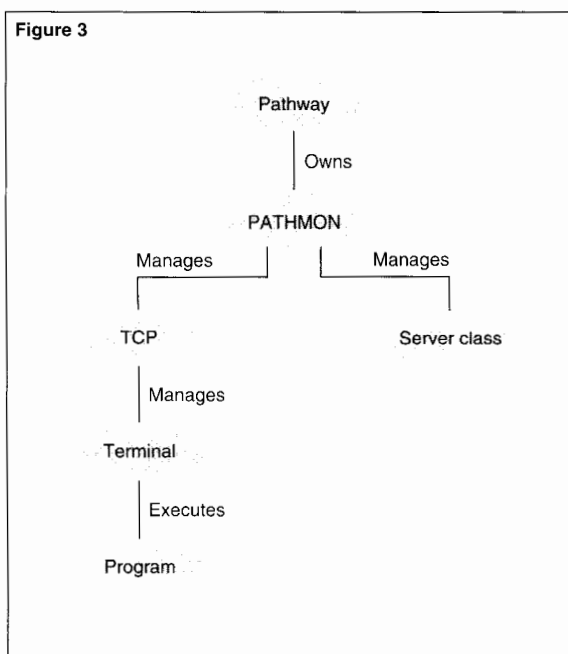


Figure 3.
Pathway objects diagram.

Next, one must identify the *attributes* of each object. An attribute is a named property of an object describing the data value held by that object. Each object can have many attributes. Some attribute values, such as a program name or the object file name of a server, can be static. Other attributes have values that can change frequently; examples include the state of an object (started, stopped, suspended), the CPU and priority of a running process, or the end-of-file (EOF) pointer of a log file. For each object, one must identify at least two key attributes, the object name and state, which will be required for the remaining steps.

Table 2.
Behavior of a Pathway terminal.

Current state	Condition	Action	Next state
Stopped	START command	Generate event 1043	Running
Running	STOP command	Wait for term stopped	Pending
Running	SUSPEND command	Generate event 1049	Suspended
Running	ABORT command	Generate event 1002	Stopped
Running	File system error	Generate event 1048	Pending
Running	STATUS command	—	Running
Running	INFO command	—	Running
Pending	All programs stopped	Generate event 1047	Stopped
Pending	ABORT command	Generate event 1002	Stopped
Suspended	RESUME command	Generate event 1042	Running
Suspended	ABORT command	Generate event 1002	Stopped

Figure 4

State:	Running		
Description:	The terminal is available for user work.		
Condition that produces the state:	START command		
Precondition that characterizes the state:	The TCP that owns the terminal must be running. The PATHMON that owns the TCP must be running.		
Conditions and commands accepted in the state:			
	Condition	Action	Next State
	SUSPEND	Suspend the terminal	Suspended
	FS-ERROR	Suspend the terminal	Pending
	STOP	Stop the terminal	Pending*
	ABORT	Stop the terminal	Stopped
*The terminal remains in a pending state until all programs running on it finish executing.			

State:	Suspended		
Description:	The terminal is not available for user work; it was suspended by a user.		
Condition that produces the state:	A user entered a SUSPEND command from PATHCOM or by using the SPI interface.		
Precondition that characterizes the state:	The TCP that owns the terminal must be running. The PATHMON that owns the TCP must be running.		
Conditions and commands accepted in the state:			
	Condition	Action	Next State
	RESUME	Restart the terminal	Running
	ABORT	Stop the terminal	Stopped

Figure 4.
Running and suspended states of a Pathway terminal.

Step 2: Prepare Operations Scripts

In this step, one prepares *operations scripts*, written dialogs describing the interactions between the operations staff and the system. The scripts may not cover every contingency, but they ensure that common interactions are not overlooked.

For example, consider a problem that can occur frequently in a Pathway environment; a user terminal goes into KBD LOCK because of a transient failure and the user is unable to enter data. The following script describes the interaction that allows an operator to fix the problem.

1. A user terminal encounters an error, which causes the TCP to suspend the terminal.
2. An EMS event message, *terminal-suspended-event*, is generated, which informs the operations staff of the terminal failure.
3. The operator starts PATHCOM and enters the STATUS *terminal-name* command to inquire about the state of the terminal.
4. If the terminal is in the suspended state, the operator issues an ABORT *terminal-name* command to change the state of the terminal to stopped.
5. The operator enters a START *terminal-name* command to restart the suspended terminal.
6. The operator may want to ensure that the terminal was restarted by reissuing the STATUS *terminal-name* command.
7. If the terminal is restarted, the operator can concentrate on other work. Otherwise, he or she may have to open a problem report and call for an escalation procedure to identify the cause of the problem.

Scripts such as this one are extremely useful for identifying the management information that is already available or that may be needed for automatic recovery. In general, operations scripts for the Pathway subsystem have the properties shown below. (One can analyze the operations scripts for other subsystems in a similar way.)

- When an object changes state, the Pathway subsystem manager generates an event message. The structure of these event messages is based on the EMS.
- Commands and responses are used to direct the recovery procedures.
- Two types of commands are used, one (such as STATUS) to inquire about the state of an object and another (such as ABORT or START) to change its state.
- In certain cases, the scripts show how to recover from a failure. In other cases (such as a user terminal power supply failure or data communication failure), no simple recovery is possible; human intervention may be required on site.

Step 3: Identify the Objects' Behavior

For each object identified, one should define its valid states, state transitions, possible conditions that make it change states (such as a user command or an internal error in the subsystem), and the corresponding actions. For example, Table 2 lists the conditions that modify the states of a Pathway terminal; this table represents the behavior of a Pathway terminal.

For each state change caused by an internal or external condition, the object may have a pre-defined set of actions. Often, the action is to generate an EMS event message that informs the system of the object's state change.

For each valid state of the object, one may want to specify more information than is shown in Table 2. For example, Figure 4 presents information that defines the running and suspended states of a Pathway terminal.

Figure 5

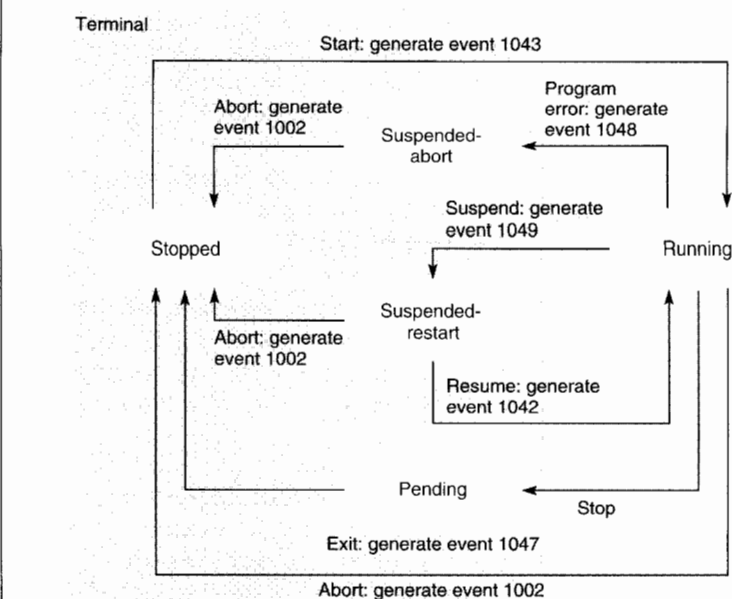


Figure 5.

State transition diagram for a Pathway terminal.

Step 4: Build a Dynamic State Change Model

From the information shown in Table 2, one can produce a state transition diagram that illustrates the dynamic behavior of a terminal. Figure 5 shows the five valid states of a Pathway terminal: stopped, running, suspended-abort, suspended-restart, and pending. A Pathway terminal will change states only when an internal or external condition occurs. (An internal condition can be, for example, a problem with the object; an external condition can be an operator command.)

For example, the Pathway subsystem can only accept a START command if the terminal is in a stopped state. The START command causes a transition from a stopped state to a running state. This causes Pathway to generate an EMS event message (1043), which informs the operator of the state change of the terminal.

If a running terminal encounters a program error, the terminal goes into the suspended-abort state and Pathway generates an EMS event message (1048). In this state, only the ABORT command can be used to modify the state of the terminal. If the operator issues an ABORT command, the terminal goes into a stopped state and PATHMON generates an EMS event message (1002).

A state transition diagram is a powerful tool for representing the dynamic properties of objects. One can also use this dynamic model to represent other aspects of one's application. For example, in an ATM application, one could use a state transition diagram to represent a typical interaction between an ATM user and the banking software. One could include both normal scenarios and exceptions to represent the major interactions between the objects.

In addition, one can use state transition diagrams to represent the states of complex business transactions or a series of batch jobs that have complex interdependencies. With the information provided by these models, it is easy to specify the required instrumentation for the objects. For example, EMS event messages can be generated when a job starts and stops successfully, or stops with errors. With this information, an automated operator can reschedule some of the processing automatically.

Using the Pathway object hierarchy model developed in step 1, one can now represent a whole Pathway system. Figure 6 shows the relationships among the objects in a Pathway dynamic model.

Step 5: Analyze the Model to Discover Missing Instrumentation

Once the dynamic model is constructed, one can analyze it. The Pathway subsystem example shows how one can use the dynamic model to identify where a subsystem requires more instrumentation. The analysis of the Pathway subsystem demonstrates that no messages are generated when a server has a problem during initialization (such as opening files) or when an internal error occurs (such as an arithmetic overflow).

For some of these conditions, the role of the Pathway subsystem may be limited to trying to restart the server. Therefore, application programmers are responsible for designing and implementing supplemental instrumentation in order to identify these problems and recover from them. Messages such as server abending, internal errors, or application errors may be useful for application diagnostics or recovery.

On the basis of the instrumentation analysis in the Pathway subsystem, one can identify the states of a server and create a state transition diagram that will help to identify where supplemental instrumentation may be required. One can use this approach for transaction servers as well as batch servers.

Figure 6

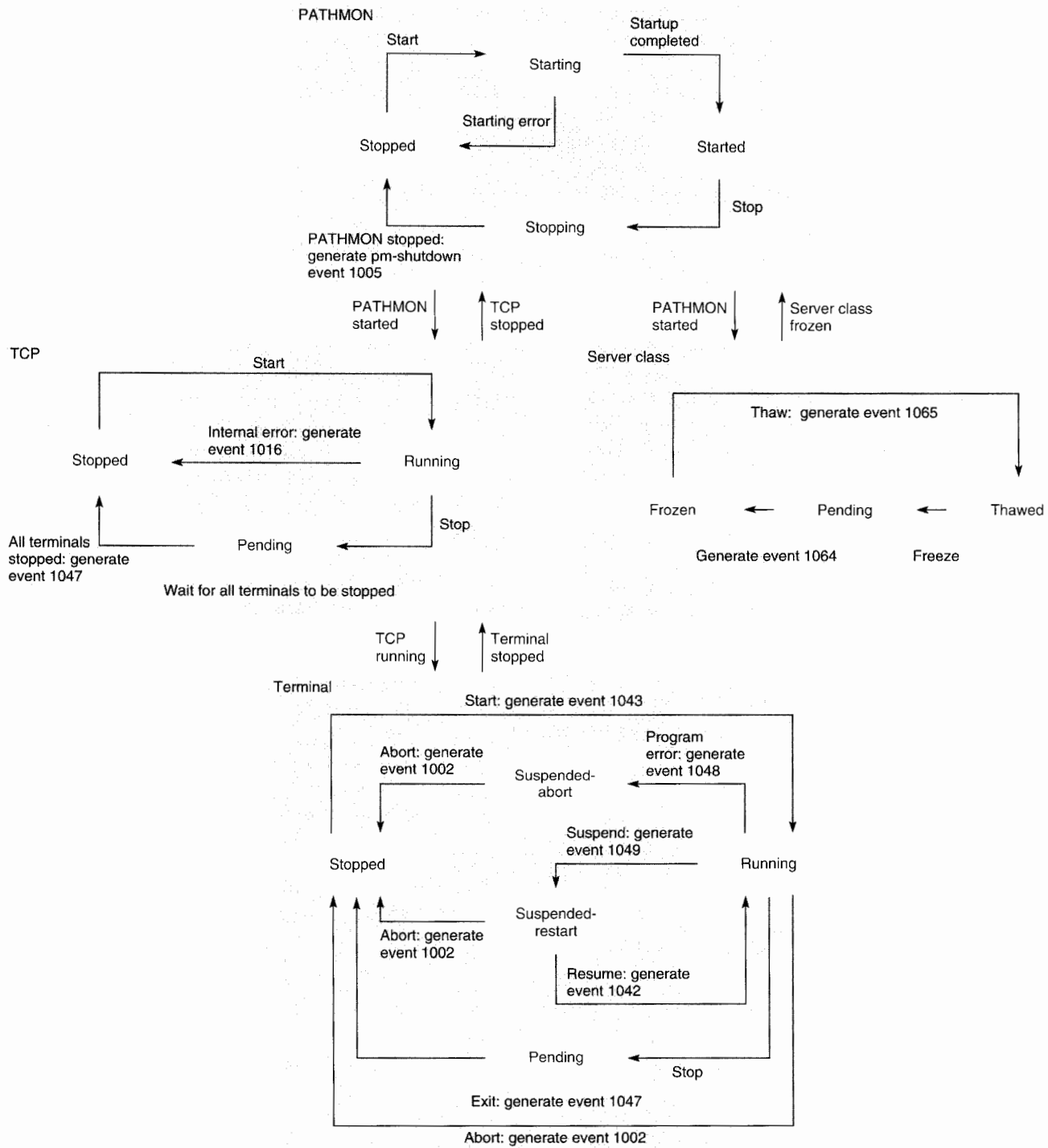


Figure 6.
Pathway subsystem
dynamic model.

Transaction Servers

In a Pathway-based application, it is important to understand the relationship between a *server class* and an individual *server process*. A server class is a logical object defined by Pathway. It can contain one or more server processes, each of which can be in a different state. A server process can be started only when the server class to which it belongs is in the thawed state. (See Figure 6). A transaction server process has the following states:

- *Stopped*. The server is known to the subsystem but not started.
- *Starting*. The server performs two main activities. First, it usually processes initialization parameters and ASSIGN statements that specify its run-time environment and the objects (such as files and spooler locations) with which it will communicate. Next, it tries to create a link with these external objects by opening files or reading initialization tables.
- *Running*. The server is fully functional and can process transactions.
- *Stopping*. The server breaks the links to the external objects by, for example, closing the files and external tables.
- *Abending*. The server terminates processing because of an internal error.
- *Stopped*. The server returns to its initial state.

By default, no messages are generated whenever the server process changes states. (In Pathway, these states are valid only when the server class is in the thawed state.) Thus, without instrumentation, there is no way to find out the logical state of the server. One can solve this problem by instrumenting the server so that it generates event messages during the startup state, while it is started and processing transactions, and when it changes to the stopping state.

If an error occurs during the startup state, exception messages can be generated. For example, one can generate a message if the server receives a wrong parameter or cannot open the database files. Each of these messages should be well documented and contain all the required information to help the human (or automated) operator recover from the situation.

When the server changes to the started state, an event message can be generated that specifies that the object is fully functional. Monitoring subsystems can use messages such as this one to track the health of the application. Without these messages, it becomes much more difficult to determine whether or not an object is operational.

If a business logic error occurs while the server is processing transactions, the server should only reply to the requester with the appropriate information. There is no use in generating an event message stating, for example, that the transaction is not accepted because of a missing account number.

However, any internal program error or severe loss of resources (such as access to the account master file) should be reported promptly. One must understand the application context and envision the recovery scenarios in order to provide all the required information in the event message.

When the server changes to the stopping state, another event message can be generated. This message can also be used by a monitoring sub-system to track the state of the application components. Figure 7 shows a state transition diagram for a transaction server running within the Pathway environment.

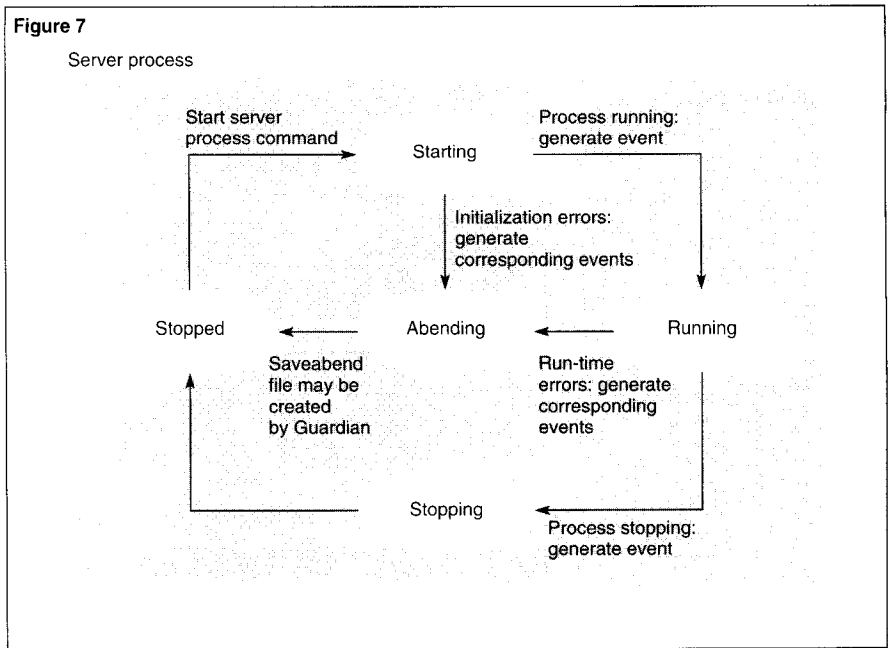
Batch Servers

The instrumentation of batch servers also requires a thoughtful strategy. The states of a batch server are similar to those of a transaction server. However, the starting state may involve reading more parameters and tables, opening many files, and creating temporary files. Also, when reports are involved, special forms may have to be specified.

The major difference between instrumenting a batch server and a transaction server is in the choice of the recovery strategies if a problem occurs. For a batch server, one must carefully analyze problems such as data inconsistency, temporary file full, CPU overutilization, and unexpected termination.

When a prolonged job must be executed, there may be a requirement to provide the right information to restart the job at a specific time or record in order to finish processing within the available time (batch processing window). In this situation, proper instrumentation may be of great help by making it possible to automatically restart the job.

One can use Tandem's NetBatch™-Plus batch scheduling product to control the execution of a series of jobs. Nevertheless, well-instrumented batch programs may greatly improve the control and management of complex jobs.

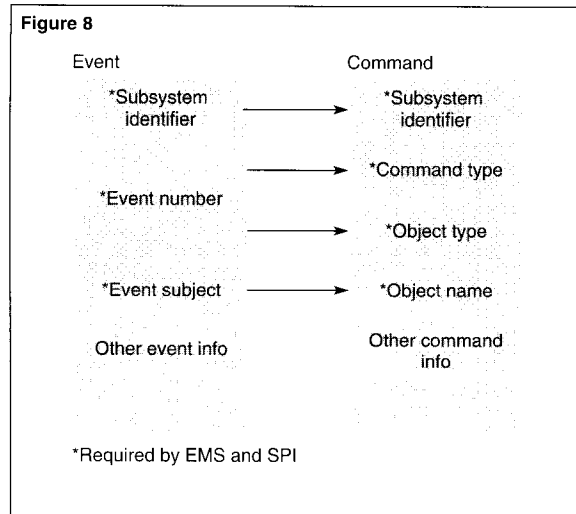


Consider, for example, a scenario in which a batch job is started and a special set of actions is required in order to fully use all the available resources in a system. One could write special rules for an automated operator in order to re-configure the system dynamically for the batch run. When the batch job begins, it sends a message to EMS. The EMS message invokes the rule for the automated operator, which reconfigures the system. When the batch job terminates successfully, it sends another EMS message, which invokes another rule that returns the system to the previous configuration.

Figure 7.
*State transition diagram
for a transaction server.*

Figure 8.

Normalization of events and commands.



Considerations for Instrumenting an Application

There are several considerations that can help one to design effective instrumentation for an application. For example, the structure of the EMS event messages can be optimized for automated operations. One can also define a consistent set of commands that control the application's objects.

Optimizing EMS Event Messages for Automated Operations

Event design is of primary importance for automated operations, because event messages may be the only source of information available when an unexpected condition occurs in an application. For each object, one should define event messages that are consistent with the object's states and command set.

Each EMS event message should contain the required information in the proper format to be used by the appropriate command. For example, a terminal-suspended event should contain all the necessary information in the format required by the ABORT and START commands. Figure 8 shows how pieces of information are mapped from an event to a command.

EMS event messages should contain at least the following fields: a subsystem identifier, event number, and event subject. (By default, each EMS event message also includes other fields such as the time the message was generated, who generated the message, and where the message was generated.)

The subsystem identifier (SSID) field contains the SSID of the application. Each application should have a unique SSID so that the source of the event can be recognized easily by an automated operator.

The event number should be selected carefully. A common mistake in application development is to keep adding new event numbers without verifying that there are no existing events similar to the new ones. Another mistake is to use the same event for multiple purposes, which makes it difficult to isolate problems when they do occur.

An event message can represent an object state transition (such as a stopped terminal). The event number should represent an object state (such as stopped) and an object type (such as a terminal). Thus, from the event number, the monitoring subsystem or application should be able to get directly to the command type and object type that should be issued.

For example, when Pathway suspends a terminal, it generates an EMS event message. The event number is 1048, terminal suspended on program error. From this Pathway EMS event number, one can infer that the next command one must send is ABORT *terminal-name*, which stops the terminal.

In general, the event subject (subject name) corresponds to what this article calls the object name. The subject in the event should be normalized (standardized). Applications should supply a subject in the format expected by the command and control interface, such as the Subsystem Programmatic Interface (SPI).

If the event subject is not normalized, the automated operations software has to determine if the subject is in external or internal format. It is important to keep the subject in a single format, either external or internal. For example, a specific standard could recommend using a 64-byte subject length so that it can accommodate an object name of any size. Also, the event subject should be fully qualified to identify the location of its generation.

If the subsystem has a manager process such as PATHMON, the event message should contain the name of the manager process. This is useful, because an automated operator will normally send the commands to this process.

If possible, an EMS event message should contain other subsystem- or application-related information. For example, some applications use logical names for objects that are different from their physical file names. (In the Tandem SNAX™ communications services environment, the Pathway terminal name MY-APPL-TERM32 can also have the name \$\$NA1.#PU32.AFG0045.) For these applications, one should report both names in the EMS event messages. This may be useful for finding out if the problem is with the logical object or the physical object.

When constructing an EMS event message, one can also add the name of the object file and the name of the procedure that detects the condition that generates the message. With this standard, one can write a specific filter and use an EMS Distributor to generate reports and analyze trends in one's software. From these reports, one can identify areas in the code that may need more testing or review.

Table 3.
Classes of commands.

Class	Commands
State changes	ADD, DELETE, START, STOP, ABORT
State inquiries	STATUS, INFO, AGGREGATE
Attributes modifications	ALTER (static), CONTROL (dynamic)
Human interface commands	HELP, SHOW, ENV, HISTORY

Commands and Responses

To control the objects in an application effectively, each subsystem (or application) should support a set of commands consistent with the management model to be used. Table 3 identifies four classes of commands: state changes, state inquiries, attributes modifications, and human interface commands.

State change commands allow the subsystem to change the state of an object. State inquiry commands allow the subsystem to inquire about the state of an object. The object can return information about the values of its static and dynamic attributes.

Attributes modifications allow the subsystem to modify the values of the different attributes of an object. One uses human interface commands to implement a user interface for the application.

General Application Guidelines

The following guidelines are intended to help developers increase the effectiveness of the instrumentation in their applications. In general, when one designs an event message, one should ask what the purpose of the event is and what a human or automated operator can do in response to it. Event design goes hand in hand with command and response design. They produce related types of messages that need to be considered together as one designs the application.

Missing Objects. No subsystem should report an event (other than an expected object missing event) against an object that is not physically or logically in the system. For example, if a terminal is not yet connected, there is no need to generate an event that says the terminal is down.

Error Events. All error events should be reported immediately when detected. Any delay in generating a critical event message may prevent the operator or automated operations software from recovering from the error quickly. To achieve high availability, one must have a short mean time to repair (MTTR).

Generating a Single Event for a Single Resource. A failure of a single lower-level resource used by multiple entities in a subsystem should be reported in a single event. For example, if a Pathway TCP fails, Pathway generates only one event, even if many terminals are connected to that TCP.

Unspecified Events. No subsystem should generate an event whose meaning is not clear. The reason for this is that the people who program automated management applications are often not the same as those who program the managed applications. Developers' expertise must be captured and made available internally and externally.

Designing a Suitable Module for Event

Generation. A good approach to event design is to develop a common code module for event generation. One can specify parameters for this module that allow one to select dynamically the types of events a particular object will generate. For example, one rule specifies that only exception messages should be generated at system startup time (when one restarts the application). Later, if a problem occurs, it may be useful to allow all event messages (including started terminal messages) to be generated.

It may also be useful, during application certification or testing, to control the kind of information generated about the application's objects. The use of an event generation module can greatly simplify the implementation of these features.

With this facility, one can define many levels of errors in the application, from informative messages to critical messages. By default, EMS defines three types of messages: informative, action, and critical. One can also add other types (levels) of messages specific to one's application. It is extremely useful to design instrumentation services so that they can be configured dynamically.

Old Applications. In many applications designed years ago, most of the event messages do not help one to identify problems and recover from them. Only a few events are useful, and even those have an unstructured message format, which makes it hard to use automated operations software with them.

Converting these applications is not easy, because no management model existed when they were developed. If it is unacceptable to open many mature application modules, one could use an external software product to trap the event messages and convert them to the standard EMS format. However, if an application does not provide a command and control interface, these messages may be of help only to human operators.

Conclusion

The most cost-effective way to meet the service objectives of an application (such as providing continuous availability) is to design instrumentation into the application while it is being developed. To achieve availability, application developers should provide normalized event message generation and command and control interfaces.

To help one understand the properties of an application's objects, this article has examined Pathway, a good example of a well-instrumented subsystem. One can model the dynamic behavior of an application's objects by building a state transition diagram. Using the information provided by the diagram, one can easily identify where instrumentation is sufficient and where more instrumentation may be needed (such as in application server programs).

One can further increase the availability of an application by using automated operations software, which can repair simple problems effectively. Automated operations software can greatly reduce the mean time to repair (MTTR) of problems and also provide a more consistent recovery mechanism. A properly instrumented application can provide the information that allows automated operations software to work most effectively.

References

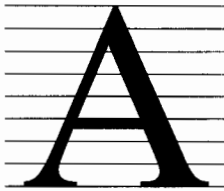
- Dennison, D. 1986. Measure: Tandem's New Performance Measurement Tool. *Tandem Systems Review*. Vol. 2, No. 3. Tandem Computers Incorporated. Part no. 83938.
- Jones, C. 1991. *Applied Software Measurement: Assuring Productivity and Quality*. McGraw-Hill.
- Lanergan, G. and Grasso, A. 1984. Software Engineering with Reusable Design and Code. *IEEE Transaction on Software Engineering*. Vol. SE-10, No. 5.
- Rumbaugh, J. et al. 1991. *Object-Oriented Modeling and Design*. Prentice-Hall.

Acknowledgments

The author wishes to thank all the people who carefully reviewed this article and made constructive comments. Special thanks go to Paul Calhoun for his contribution to the first section of this article. Thanks also to Ted Schachter, Mark Anderton, Terrye Kocher, Mike Choi, Jim Collins, Mark Merala, Jim McHutchion, and Spots Stoddard, as well as all the technical staff of the Centre de Technologie Tandem de Montréal.

Jean Dagenais is the manager of the Centre de Technologie Tandem de Montréal, where he supervises research and development in the area of operations management. He joined Tandem in Montreal as an account analyst in September 1984. In 1989, he transferred to the LSMS tools development group, where he designed and implemented Tandem's EMS FastStart product.

Writing Rules for Automated Operations



s large online transaction processing (OLTP) systems grow, demands on the operations staffs increase at a dramatic rate. Help desk personnel become over-

loaded with mundane tasks while trying to maintain high levels of service to end users. If single component failures are not repaired promptly, they can quickly escalate and cause significant outages.

Automated operations systems can help alleviate these problems by responding immediately and reliably to initial problem reports. By intervening programmatically, automated systems reduce the burden on operations staff and increase the availability of the system. To take full advantage of the benefits of an automated operations solution, users can write rules customized for their particular installations.

This article proposes a methodology for designing and building rules for an automated operations system. The methodology entails decomposing complex problems into corresponding sets of simple problems and dealing with each individual problem as a unit. This approach allows users to realize a rapid return on their investment in an automated operations solution.

One can apply the philosophy and techniques proposed in this article to any automated operations system. The article uses, as an example, the Tandem™ Programmatic Network Administrator (PNA) automated operations system. The article defines the function of a rule in an automated system and describes general requirements for writing such rules. Finally, it discusses specific ways to handle simple and complex problems.

The article assumes that the reader is familiar with the basic architecture of the Tandem Distributed Systems Management (DSM) and the fundamentals of at least one Tandem subsystem such as the Pathway transaction processing system or Expand™ data communications networking software.

Definition of a Rule

A rule is a set of statements that, taken together, comprise an operational procedure. There are many reasons for writing a rule. For example, a rule might schedule an activity to take place at a certain time. A rule might also periodically wake up and poll the environment, looking for abnormalities.

This article focuses on a small subset of reasons for writing a rule. For the purposes of the article, a rule exists to address a specific problem or condition, which is always identified by one or more event messages.

How PNA Uses Rules

The PNA automated operations system analyzes a system or application event message and invokes a rule (procedure) that acts upon the event message under analysis. PNA can analyze several event messages concurrently. Any of these events may have corresponding rules, and PNA can execute these rules concurrently.

A rule analyzes the event message to find appropriate actions to be taken. To carry out those actions, the rule interacts with subsystems or applications through gateway processes. Users have full access to the PNA rules database and can modify, add, or delete rules as system and management requirements change. PNA comes with a set of rules already written. These rules handle a variety of situations that may occur in any installation. The rules also serve as examples for recommended rule-writing techniques.

The Origin of the Rule

A rule has its origin in an installation's *runbook*. The runbook is a collection of procedures, each of which is designed to resolve a specific problem. The runbook may not actually exist outside the minds of the operations staff. If it does exist, the installation has a head start on the composition of rules. If it does not exist, it will be created by the time the installation implements its first operational rule for an automated system. The collection of rules in the rules database will become the runbook.

If there is no runbook, users will have to analyze their installation before they can successfully implement an automated operations system. The analysis should begin with a recent sampling of log files and the help of one or two experienced operators. The log files should indicate the types of problems that occur frequently, and the operators should provide the procedure used to solve each problem.

The log files are an important component of this analysis. The automated operations system can only react to an event that has been produced by some subsystem or application. A human being can react to a telephone call, but the automated operations system must be made aware of a problem by means of an event message.

The log files are also important because they show how frequently any particular problem occurs. Users should address the most frequent problems first. An installation may not realize an early return on its investment in an automated operations solution if that solution concentrates on infrequent problems merely because they were easy to resolve.

Once one determines which set of problems to tackle first, one must determine the exact series of steps an operator would take to resolve each problem. It is important to record each step and any possible results of that step. It is tempting to write a rule that will exactly mimic the operator's actions, but this may not be the proper choice. Instead, one should note how a subsystem or application may react to any one of the steps the operator may take. Determining the operator's steps does not necessarily complete the analysis.

Requirements for Writing Rules

Before writing rules, one should understand the basic requirements for rule writing. There are four such requirements.

1. A single event message may trigger only one rule. (This rule may call other rules.)
2. A rule may be triggered by any one of several event messages. (A single message will cause a rule to be triggered. It is not possible to cause a rule to be triggered by a combination of event messages.)
3. A single event message is independent of all other event messages. (This statement is made from the point of view of the automated operator, not the subsystem producing the event messages.)
4. A rule is independent of all other rules. (A rule may call another rule, but no rule can determine that another rule is executing or that it has executed.)

These requirements may appear to be so restrictive that they could never solve any real problems. This article shows how one can turn these requirements to one's advantage. With a little guidance, the restrictions coerce the rule writer to produce generic rules that are useful not only in any installation but in a variety of circumstances.

Basic Rule Structure

The ideal rule should follow a simple, basic structure. There are four general steps in this structure.

1. Interrogate the event condition. Is the condition that caused the event still in effect? The rule, like the human operator, cannot afford to assume that it is the only agent acting on a problem. Another agent may already have fixed the problem.
2. Attempt to resolve the problem.
3. Interrogate the event condition a second time. The problem may still exist even if the rule did not receive an error when it attempted the resolution. Many subsystems reply to action commands with a status code signifying only that the command was received and was formatted correctly, not necessarily that the command was processed.
4. Take a secondary action if the resolution was not successful. There are several possible secondary actions. For example, the rule could generate a special event message describing the failure. (The event message may end up being displayed on the operator console; it also may be processed by a problem management application.)

Another secondary action might be to exit the rule. The failure of the resolution could cause another event to be produced, thus causing another rule to be executed. Alternatively, the rule could retry the resolution or delay for some time before taking any further action. Often, a rule combines several secondary actions. For example, the rule might issue an event message, delay for a period of time, and then retry.

In PNA, two forms of delay are possible, the `DELAY` statement and the `QUEUE_EVENT` statement. `DELAY` suspends execution of the rule for a specified period of time. `QUEUE_EVENT` removes the rule from current execution and places it on a queue. This frees the thread used by that rule, allowing the thread to be used by another rule. The `QUEUE_EVENT` mechanism allows more events to be in the PNA system than there are threads configured. This is a very powerful feature and is the recommended approach for the delay-retry combination of secondary actions.

The basic rule structure outlined in this article may be more an ideal than a doctrine. Certainly, in some instances, the four basic steps are not used. However, one should be cautious about eliminating any of these steps, particularly the first. It may be superfluous to interrogate the condition if the resolution is nondestructive. For example, assume that an Expand line went down. Attempting to bring up an Expand line that is already up does not result in any significant confusion, so the first step might be omitted.

However, the result can be different if the resolution is destructive. For example, assume that a Pathway terminal went into a suspended state, pending an abort. The proper solution is to abort the terminal. This is a potentially destructive step because the problem may already have been resolved. Failure to test the condition in the first part of the rule may cause a running Pathway terminal to be aborted. This may unnecessarily arouse the ire of the end user.

The Atomic Rule

An *atomic* rule is one that will never cause a subsystem to produce more than one new event message as a result of actions taken by the rule. In normal situations, the atomic rule is the best possible kind, not only because it is inherently simple, but also because nonatomic rules can be so bad. A significant problem with nonatomic rules is the phenomenon of cascading events.

To understand cascading events, consider the problem of a Pathway terminal going into a suspended state, pending an abort. Figure 1 shows a sample script for the solution of this problem.

A rule that exactly mimics the actions of the operator may precipitate cascading events. Figure 2 shows such a rule, as written in the PNA Rules Definition Language (RDL). (Only the most important steps appear in the example.)

The rule shown in Figure 2 causes several new event messages to be generated. An event message is generated at STEP_2, when the terminal is aborted. This necessarily triggers another rule. (After all, aborted terminals have to be restarted.) The rule that is triggered may be another copy of this rule.

An event message is also generated at STEP_3, when the terminal is started. This event is likely to be of little interest to the automated operator because this is the desired outcome.

Figure 1

1. Get PATHMON name and terminal name.
2. Run PATHCOM.
3. STATUS TERM <name>.
4. ABORT TERM <name>.
5. START TERM <name>.
6. STATUS TERM <name>.
7. Repeat, if needed.
8. Exit PATHCOM.

Figure 1.

Solution script for a Pathway terminal going into a suspended state.

Figure 2

Suspended terminal rule

```

LABEL: STEP_1;
  SEND [STATUSTERM] TO PATHWAY;
  ...
  IF RESPONSE = SUSPENDED THEN STEP_2;
  ...
LABEL: STEP_2;
  SEND [ABORTTERM] TO PATHWAY;
  ...
  IF RESPONSE = 0 THEN STEP_3;
  ...
LABEL: STEP_3;
  SEND [STARTTERM] TO PATHWAY;
  ...
  IF RESPONSE = 0 THEN STEP_4;
  ...
LABEL: STEP_4;
  SEND [STATUSTERM] TO PATHWAY;
  ...
  IF RESPONSE = STARTED THEN EXIT_RULE;
  QUEUE_EVENT 1 MINUTE;
  ...
  GOTO STEP_1;

```

Figure 2.

PNA rule mimicking the operator's actions.

What happens if the device is down? Here, further analysis is required. One must know how Pathway reacts in certain circumstances. Even if the device is down, the Pathway terminal will still move to the started state. The only prerequisite for moving to this state was successful communication between the PATHMON process and the terminal control process (TCP) that controls the device.

Figure 3.

Two atomic rules. One handles suspended terminals; the other starts stopped terminals.

Figure 3

Suspended terminal rule

```

LABEL: STEP_1;
  SEND [STATUSTERM] TO PATHWAY;
  ...
  IF RESPONSE = SUSPENDED THEN STEP_2;
  ...
LABEL: STEP_2;
  SEND [ABORTTERM] TO PATHWAY;
  ...
  EXIT;
```

Stopped terminal rule

```

LABEL: STEP_1;
  SEND [STATUSTERM] TO PATHWAY;
  ...
  IF RESPONSE = STOPPED THEN STEP_2;
  ...
LABEL: STEP_2;
  SEND [STARTTERM] TO PATHWAY;
  ...
  EXIT;
```

If the device is down, the TCP encounters an error when it attempts to open the device. This causes the Pathway terminal to move to a suspended state, pending abort. This, in turn, results in the generation of another event message, which triggers another copy of the original rule. If that happens, the QUEUE_EVENT and retry after STEP_4 are useless, because at least two other rules have been triggered. Also, no delays occur anywhere; actions are taken as fast as events can be produced.

Clearly, the problem of cascading events is one to be avoided. In constructing the Pathway solution, one would replace the nonatomic rule with two smaller, atomic rules. (See Figure 3.) The first rule would handle suspended terminals. It would abort the terminal, which would cause another event message to be produced, triggering the second rule. The second rule would start stopped terminals. If the terminal does not start, it will move to a suspended state, thus triggering the first rule. The pair of atomic rules would eliminate cascading events.

Of course, this solution does not allow for any type of delay if the terminal does not start. It would still take actions as fast as events could be produced. Therefore, some delay is required. In this situation, the proper place for a delay is just before the action that causes a rule-triggering event message to be produced. Starting the terminal causes an event message to be produced, but no rule is triggered as a result. (If the terminal were to move to a suspended state, an event message would be produced, but this would not be a direct result of the action that the rule took.) Aborting the terminal always causes an event message to be produced, which always triggers the execution of a rule. Thus, the proper place for a delay is just before aborting the terminal, as shown in Figure 4.

This solution is incomplete because it is skewed toward the inability of the TCP to start the terminal. The solution always waits one minute before attempting anything. A complete solution would respond to a problem that was not so serious, in which one could restart the terminal immediately. A solution later in this article addresses this possibility.

A complete solution may also demand a status check between the delay and the actual abort. An outside agent might already have fixed the problem.

Prerequisites for Atomic Rule Composition

There are two basic prerequisites for atomic rule composition:

- One must know how the subsystem operates. How does the subsystem accomplish its tasks? What states do objects move through, and in what order?
- One must know the events produced at different states.

This is the last phase of the analysis required to convert a runbook procedure into a set of atomic rules. Knowledge of the subsystem is probably the most critical component. A rule is only as smart as its author.

Simple Problems and Complex Problems

The problems an automated operations system attempts to resolve fall into two general categories: *simple problems* and *complex problems*. A simple problem is one in which the solution script does not cross subsystem boundaries. A complex problem is any problem that is not simple.

This may seem like a trivial categorization. In fact, understanding simple problems is key to a successful implementation of an automated operations system. If one is familiar with simple problems, one will understand complex problems and their solutions. Furthermore, understanding these two types of problems should help those responsible for implementing successful event-logging strategies in subsystems and applications.

Simple Problem Message Types

Two types of event messages identify simple problems: *single-purpose event messages* and *multipurpose event messages*. A single-purpose event message has a unique message identifier for each combination of object type and condition. This is the most common type of message. Figure 5 shows examples of such messages.

Figure 4

Suspended terminal rule

```
LABEL: STEP_1;
  SEND [STATUSTERM] TO PATHWAY;
  ...
  IF RESPONSE = SUSPENDED THEN STEP_2;
  ...
LABEL: STEP_2;
  QUEUE_EVENT 1 MINUTE;
  ...
  GOTO STEP_3;
  ...
LABEL: STEP_3;
  SEND [ABORTTERM] TO PATHWAY;
  ...
  EXIT;
```

Stopped terminal rule

```
LABEL: STEP_1;
  SEND [STATUSTERM] TO PATHWAY;
  ...
  IF RESPONSE = STOPPED THEN STEP_2;
  ...
LABEL: STEP_2;
  SEND [STARTTERM] TO PATHWAY;
  ...
  EXIT;
```

Figure 4.

Including a delay in the suspended terminal rule.

Figure 5

```
Expand Event #7
  Meaning: Line is down.
  Subject: ZEMS_TKN_LDEVNAME.

Expand Event #45
  Meaning: Line is not ready.
  Subject: ZEMS_TKN_LDEVNAME.

Pathway Event #1002
  Meaning: Terminal aborted.
  Subject: ZPWY_TKN_TERMNAME.
```

Figure 5.

Single-purpose event messages.

Figure 6.
Multipurpose event messages.

Figure 6

Pathway Event #1038
Meaning: NEWPROCESS error.
Subject: ZPWY_TKN_SCNAME
 or ZPWY_TKN_TCPNAME.

Pathway Event #1047
Meaning: Object stopped.
Subject: Any Pathway object.

Figure 7.
A rule that handles a simple problem.

Figure 7

```
?SUBSYSTEM_EVENT TANDEM, EXPAND, (7, 45)
...
LABEL: STEP_1;
SEND [STATUSL] TO EXPAND;
...
IF RESPONSE = STOPPED THEN STEP_2;
...
LABEL: STEP_2;
SEND [STARTL] TO EXPAND;
...
IF SEND_ERROR = 0 THEN STEP_3;
...
LABEL: STEP_3;
SEND [STATUSL] TO EXPAND;
...
IF RESPONSE <> STOPPED THEN EXIT_RULE;
QUEUE_EVENT 1 MINUTE;
...
GOTO STEP_1;
```

For single-purpose event messages, the rule can assume that the subject of the event message is of a certain type. It can take appropriate actions based on that assumption.

A multipurpose event message has a unique message identifier for each condition, but not each object type. These messages are less common. Figure 6 shows examples of such messages.

For multipurpose event messages, the rule cannot assume anything about the subject. The rule must interrogate ZEMS_TKN_SUBJECT to determine the appropriate actions that must be taken against that subject.

A Very Simple Problem

Figure 7 shows an example of a rule that handles a simple problem. An Expand line changes state to down or not ready. The SUBSYSTEM_EVENT statement identifies which events will trigger this rule.

This is a simple problem because the solution script does not cross subsystem boundaries. The event originated in the Expand subsystem, and that is the only subsystem used in the rule.

The rule is atomic because only one new event is produced. The QUEUE_EVENT statement is there in case the line does not come up. No new event is produced, so some method of retry must be built into the rule.

A Less Simple Problem

Figure 8 also shows an example of a rule that handles a simple problem, but the problem is complicated somewhat by a multipurpose event message. The problem is one in which a Pathway terminal stops or aborts. The SUBSYSTEM_EVENT statement identifies which events will trigger this rule.

The major difference between the example in Figure 8 and the one in Figure 7 is that the rule in the Pathway example (Figure 8) has to determine the type of subject, whereas the rule in the Expand example (Figure 7) can assume that the subject of the message will always be a line handler name.

Suppose the automated operations system is also meant to handle a stopped TCP. A Pathway event number 1047 indicates a stopped TCP. Recall the first requirement for writing rules: An event message may trigger, at most, one rule. Thus, one must handle a stopped TCP in the same rule that handles a stopped terminal. Figure 9 shows a possible example.

As Figure 9 indicates, multipurpose event messages can cause rules to become large. This is not necessarily bad, but if a rule becomes too large to manage, one can use the CALL statement to break it into smaller rules.

Figure 8

```
?SUBSYSTEM_EVENT TANDEM, PATHWAY, (1002, 1047)
...
LABEL: STEP_1;
  IF SUBJECT_EVENT_TOKEN = ZPWY_TKN_TERMNAME THEN HANDLE_TERM;
  EXIT;

LABEL: HANDLE_TERM;
  SEND [STATUSTERM] TO PATHWAY;
  ...
```

Figure 9

```
?SUBSYSTEM_EVENT TANDEM, PATHWAY, (1002, 1047)
...
LABEL: STEP_1;
  IF SUBJECT_EVENT_TOKEN = ZPWY_TKN_TERMNAME THEN HANDLE_TERM;
  IF SUBJECT_EVENT_TOKEN = ZPWY_TKN_TCPNAME THEN HANDLE_TCP;
  EXIT;

LABEL: HANDLE_TERM;
  SEND [STATUSTERM] TO PATHWAY;
  ...
  EXIT;

LABEL: HANDLE_TCP;
  SEND [STATUSTCP] TO PATHWAY;
  ...
  EXIT;
```

Figure 8.

A rule that handles a simple problem associated with a multipurpose event message.

Figure 9.

A rule that handles a stopped terminal or a stopped TCP.

Figure 10.

Using the CALL statement to break a large rule into smaller ones.

Figure 10

Calling rule

```
?SUBSYSTEM_EVENT TANDEM, PATHWAY,  
(1002, 1047)  
?RULE 80  
...  
LABEL: STEP_1;  
  IF SUBJECT_EVENT_TOKEN =  
    ZPWY_TKN_TERMNAME THEN HANDLE_TERM;  
  IF SUBJECT_EVENT_TOKEN =  
    ZPWY_TKN_TCPNAME THEN HANDLE_TCP;  
  EXIT;  
  
LABEL: HANDLE_TERM;  
  CALL 801;  
  ...  
  EXIT;  
  
LABEL: HANDLE_TCP;  
  CALL 802;  
  ...  
  EXIT;
```

Terminal handler rule

```
?SUBSYSTEM_EVENT DUMMY, 1, (1)  
?RULE 801  
...  
LABEL: STEP_1;  
  SEND [STATUSTERM] TO PATHWAY;  
  ...
```

TCP handler rule

```
?SUBSYSTEM_EVENT DUMMY, 1, (2)  
?RULE 802  
...  
LABEL: STEP_1;  
  SEND [STATUSTCP] TO PATHWAY;  
  ...
```

In the example in Figure 10, SUBSYSTEM_EVENT statements in the called rules do not refer to real events. Each rule must have at least one event, real or imaginary, attached to it, even if it is designed to be a called rule and never intended to be triggered directly by an event message. Whether one prefers a larger, single rule over smaller, multiple rules is largely a matter of taste.

Secondary Actions and Atomic Rules

The major question in an atomic rule is where to put the QUEUE_EVENT statement. Consider the case in which a problem is identified by a single event. The rule takes an action, but the action does not succeed. At this point, many subsystems will not produce a new event indicating the failure. If a new event were produced, it could trigger a new copy of the rule. However, if no new event is produced, the rule must assume responsibility for a retry. The rule should not attempt a retry immediately. One should insert a delay before the retry attempt, in order to give another agent a chance to address the real problem. Thus, in this case, the placement of the QUEUE_EVENT statement is obvious, as illustrated by the Expand rule in Figure 7.

Now consider the Pathway terminal example, in which the solution script is broken down into several atomic rules. In this case, the QUEUE_EVENT statement has a slightly different function. In the Expand example, the function of the QUEUE_EVENT statement was to throttle the retries performed by the same rule. In the Pathway example, one cannot delay retrying the same rule because several rules (or multiple occurrences of the same rule) are involved in the solution.

Instead, one must delay the execution of the next rule in the sequence. One can accomplish this by placing the QUEUE_EVENT statement in one of the atomic rules, just before taking the action that would cause the next event to be produced. Thus, in this case, the function of the QUEUE_EVENT statement is to delay issuing the next event, which will give another agent a chance to address the real problem.

The suspended Pathway terminal example would proceed as follows:

1. Get the status of the terminal.
2. If the terminal is suspended, pending an abort, and this is not the first time through, QUEUE_EVENT.
3. Abort the terminal.
4. Exit. The abort will cause a new event to be produced.

Examine step 2. First, one must determine whether this is the first time through. One cannot count the number of times QUEUE_EVENT has been executed, because the rule exits after the queue delay. (PNA provides a QUEUE_COUNT, but, in this case, it is of no use.)

The answer must come from the subsystem. Pathway not only yields the specific error encountered by the TCP, but also tells whether the error occurred in the attempt to access the terminal or to open it. If the error occurred during accessing, the terminal was already open, and one can assume that this is the first time through. If the error occurred during an attempt to open, one can assume that this is a retry, because the original error probably occurred in the attempt to access the terminal. Thus, one should execute the QUEUE_EVENT statement if the error was encountered in the attempt to open the terminal. (Even if the original error occurred during the attempt to open, it is appropriate to delay.)

However, another subsystem may not yield the same type of information as Pathway. In that case, one should execute the QUEUE_EVENT statement each time. It is better to throttle back than to produce events and trigger rules as fast as the system will allow.

The remaining problem with this approach is that one cannot determine how many times the retry has been attempted. QUEUE_COUNT is of no value, because the rule uses QUEUE_EVENT to control when the next event is issued rather than to throttle retries within the same rule. This problem has no ready solution.

The Pathway example used in this article involves a complicated problem. That is why the Pathway rule provided with the PNA package is the most complicated of the standard rules.

Complex Problems

A complex problem is one in which the solution script crosses subsystem boundaries. There are two major types of complex problems:

- *Type 1:* One subsystem is involved in the actual resolution of the problem, but other subsystems must provide information. These problems are usually identified by a single event message.
- *Type 2:* Several subsystems are involved in the resolution of the problem. These problems are usually identified by a combination of event messages.

Type 1 Complex Problems. These definitions use the term *actual resolution* to identify action commands that bring about some change. It is possible that a rule dealing with a type 1 complex problem could execute only the action commands. It is also possible that, if the resolution fails, the rule would not be able to determine that the failure had occurred. One would have to access the other subsystems to determine the probability of the success of the resolution. If the probability of success is low, the rule might queue the event and retry later.

Table 1.
Breaking a complex problem into several simple problems.

Event	Rule
1. ATP6100 line goes down.	Bring up line.
2. ATP6100 subdev goes down.	Bring up subdev.
3. ATP6100 subdev goes down.	Bring up subdev.
4. ATP6100 subdev goes down.	Bring up subdev.
5. ATP6100 subdev goes down.	Bring up subdev.
6. Pathway thread goes down.	Start thread.
7. TACL terminates.	Start TACL.
8. TACL terminates.	Start TACL.
9. TACL terminates.	Start TACL.

For example, assume that a Tandem Advanced Command Language (TACL™) process terminates because of an I/O error on its terminal. TACL produces one, and only one, event message. The message occurs when the TACL terminates because it encounters an I/O error while accessing an already opened terminal. However, if the terminal is inaccessible at TACL startup time, the TACL process abends without producing any event message.

If the rule dealing with this problem only attempts to start the TACL process, it cannot determine that the TACL terminated because it was unable to access the terminal. There is no provision for retry. Therefore, the TACL restart rule must determine whether the terminal is available before attempting to start the TACL process. If the terminal is unavailable, the rule issues a QUEUE_EVENT statement and tries again later.

In general, the ideal solution to type 1 complex problems resides in the subsystem itself. Either the subsystem must produce adequate event messages at appropriate times or it must provide suitable information upon inquiry. For example, TACL might produce a new event

message signifying termination because of an error opening the terminal. This would remove from the TACL restart rule the responsibility for determining whether the terminal is available. Because the TACL restart rule must fulfill this role, a configuration assumption must be built into the rule. This is undesirable because the rule is less portable than it might be.

The concept of type 1 complex problems is important not only for writing rules that resolve the problems. It can also help one to identify these problems, which can usually be avoided by properly instrumenting the subsystem.

Type 2 Complex Problems. Typically, type 2 complex problems are identified by a combination of messages. The messages can be related or unrelated.

Usually, one can break up complex problems identified by multiple, unrelated event messages into several, simple problems. This may be the most important point in this article. Breaking complex problems into simple ones is how one uses the requirements for writing rules to one's advantage. This approach allows one to write generic, portable rules. It also offers a way to understand what future subsystems must do in order to allow automation. Once one understands simple, atomic rules and how to design them, one can make rational decisions about how current and future subsystems should correctly use the DSM architecture.

For example, assume that an ATP6100 line goes down, killing three TACL processes and a Pathway thread. This qualifies as a real and complex problem. It is not particularly difficult to solve, but an operator might take some time to solve it. An automated operations solution could probably solve it faster, and more accurately.

The complex problem is broken up into several simple problems, each identified by an event message. (See Table 1.) Each problem has its own solution script.

If the ATP6100 line failure is a hardware problem, the automated operations solution may be of even greater value. A human operator may take some time to address the hardware problem, but that is the only problem he or she has to address. One can forget about the other problems (one might do so anyway). Moreover, new problems requiring one's attention are likely to arise before one has finished with the old ones. The advantage of automatic restart and recovery is that it focuses the human operator's attention on only those problems that actually require it.

This strategy is based on the rule-writing techniques recommended in this article and on the fact that PNA can execute several rules concurrently. This powerful combination eliminates dependence on the sequence of events received. The techniques illustrated here are the cornerstone of a rules-based automated operations solution.

Type 2 Complex Problems and Multiple, Related Events. Certain complex problems cannot be broken up into several simple problems. For example, a problem might arise when a specific action should be taken only when an event occurs a certain number of times within a certain time limit.

Other types of stubborn, complex problems are variations on this theme. (Assume, for example, that two independent events have happened recently.) PNA does not directly support these problems, but one can address them. One would have to build a special gateway for them. Such a gateway must keep track of all event number-subject-timestamp combinations. Furthermore, it must be able to respond to queries about frequencies of events. Gateways within PNA are much like server classes within Pathway; each is context-free.

Conclusion

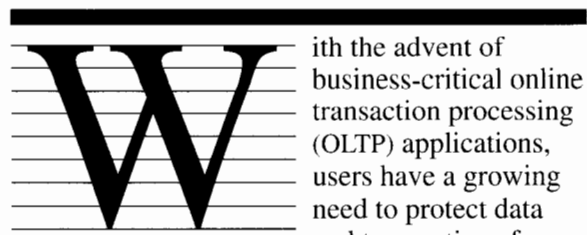
One should take advantage of the fact that PNA is rule-based. In normal applications, a large design effort must precede coding, which must precede testing; one must wait some time before getting a return on one's investment in the application. In a rule-based automated operations system, one can work toward a perfect solution and still receive benefits from the system before the final goal is met.

One should write one rule at a time and put it into production when it is ready. With this approach, the installation receives the immediate benefit of the automated rule, and the rule writer can learn by observing the way the rule operates in the production system. Each completed rule contributes to the rule writer's expertise in creating a comprehensive automated operations solution for the installation.

Acknowledgments

I'd like to thank Larry English for all his contributions to the development of the techniques outlined in this article.

Jim Collins is a senior advisory analyst in Large Systems Marketing Support. He is currently involved in automated operations. He has worked with Tandem systems since 1979.



With the advent of business-critical online transaction processing (OLTP) applications, users have a growing need to protect data and transactions from total system disasters. A common way to protect critical databases is to duplicate them at a remote site that can be isolated from a disaster occurring at the local site.

The Tandem™ Remote Duplicate Database Facility (RDF™) maintains replicated databases at a remote site that can be used for contingency planning. As end users modify the local databases, RDF replicates those changes in the remote databases, keeping them continuously up to date.

This article describes the process by which the RDF product maintains replicated databases. It discusses how one can use the replicated databases for ad hoc query processing, which relieves the primary system from having to perform that task. Next, the article explores how to use RDF as a key element in a data disaster recovery plan. Finally, it discusses how RDF can reduce the duration of planned application outages (when users need to bring the primary system down to perform change management operations).

The Uses of RDF

The RDF product can replicate databases at one site, called the primary system, onto duplicate databases installed at a remote site, called the backup system. RDF performs the replication by detecting all transactions (inserts, updates, and deletes) that change the databases in the primary system and applying those changes to the databases in the backup system.

The main purpose of RDF is to complement a disaster recovery plan. If a mishap at the primary site causes the primary system to be unusable or inaccessible, users have access to an up-to-date copy of the databases maintained on a backup system by RDF. Examples of mishaps from which a primary application would need protection are fire, earthquake, flooding, and sabotage (to mention a few).

RDF replicates the database transactions as soon as the two systems can detect, transmit, and apply them to the backup databases. Because the backup databases are as current as possible, one can use them in an ad hoc fashion for inquiries and reports. By transferring these tasks to the backup system, one can improve the performance of the critical applications supported by the primary system.

A third use for RDF is to reduce the application down time when users have to schedule outages at the primary site to perform tasks such as hardware or software maintenance. With RDF, one can run the primary application on the backup system during the outage, then switch back to the primary system when it becomes available again.

Because only the transactions are replicated, the speed of RDF does not depend on the size of the databases. Instead, it depends on the amount of work being performed to update the primary system's databases.

Topologies

One can set up the RDF product to replicate transactions in different ways, depending on the type of protection one requires. Figure 1 shows the basic configuration, in which one primary system's databases are replicated onto one backup system's databases. This configuration is the most common one. Typically, the primary system supports production and the backup system supports software development and testing.

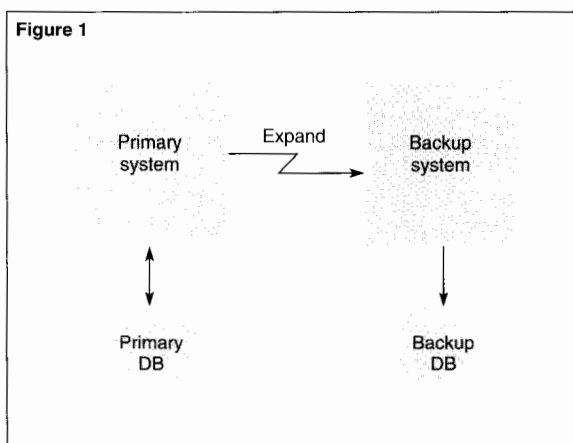


Figure 1.
Basic configuration for use of RDF.

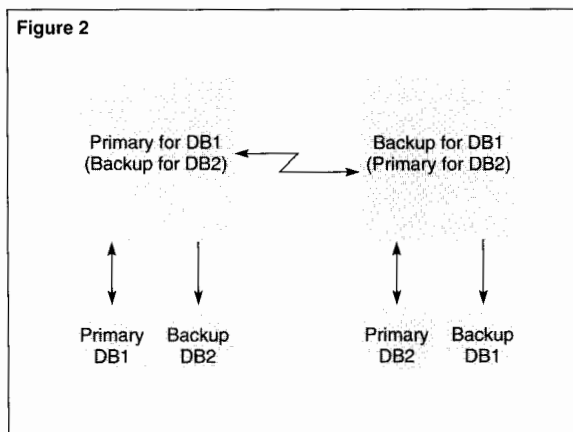


Figure 2.
Reciprocal configuration for use of RDF.

When two systems have their own applications in geographically distinct locations, users can augment each system to serve as a backup for the other. In this configuration, called the reciprocal configuration, each system acts as both a primary and a backup. (See Figure 2.) This is a very cost-effective configuration.

Figure 3

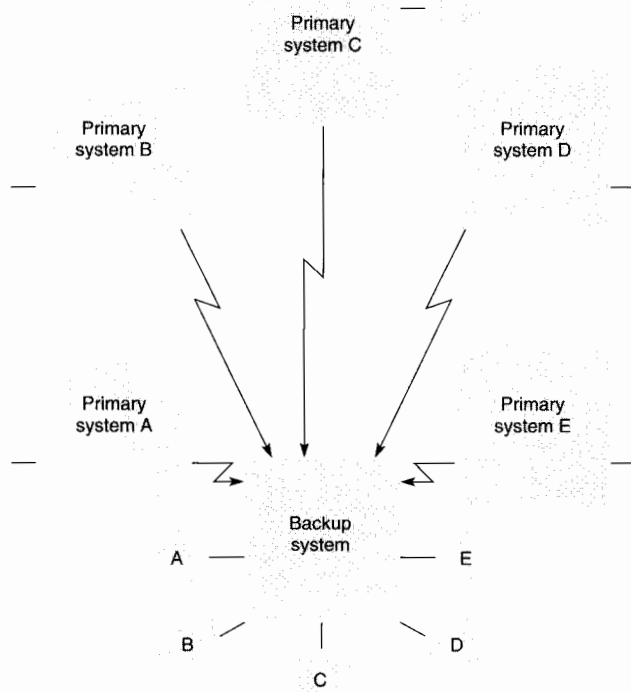


Figure 3.
Multisite configuration
for use of RDF.

When a network contains several remote sites with their own applications and a centralized development or operations system, users can set up one common backup site for multiple primary sites. This is called the multisite configuration. (See Figure 3.) It has the advantage of following the model of centralized operations.

RDF also supports other configurations such as daisy chains and rings. The only restriction in setting up a configuration is that one can set up a particular system only once as the primary system.

How RDF Works

The RDF product maintains an up-to-date backup of the databases by reading a log of the database transactions that have occurred at the primary site and transmitting it to the backup site, where it is applied to the appropriate databases. Tandem's Transaction Monitoring Facility (TMF™) maintains the log of the database transactions. The primary and backup sites must be nodes in a Tandem Expand™ data communications network; the logged data is transmitted through the Expand link between the nodes. Because RDF depends only on TMF, it is totally application independent. User application programs that already use TMF do not have to be modified to be protected by RDF.

The Role of TMF

The TMF product manages the log of database modifications for any audited databases at the primary site. The log consists of audit trails (sets of disk files), one set of which is designated as the master audit trail (MAT).

TMF saves any database modifications in the audit trails in the form of record images. The saved information serves to protect database and transaction integrity. For example, if a transaction is interrupted before it ends (whether by a program abort, an extended power failure, or for another reason), TMF uses the information in the audit trails to restore the databases to the state they were in before the transaction began. TMF thus leaves the databases in a logically and physically consistent state.

RDF also uses the database modification records (images) in the MAT. It uses this information to keep the backup databases up to date. RDF only transmits images for disk volumes that have been configured under RDF protection. Any audited files residing on those volumes become protected by RDF. Unaudited files in those volumes do not generate any audit information and are therefore not in the realm of RDF protection. Applications using some unaudited files, such as error files, need to be examined closely to determine the type of replication required for the unaudited files.

Managing RDF

The user manages the RDF product through a command-level user interface process called RDFCOM. One can use RDFCOM to configure, start, monitor, and stop RDF.

To configure RDF, the user designates the execution priorities, the CPUs to be used, the processes' names, and the disk locations of the swap files for the processes on the primary and backup systems. Proper configuration allows for load balancing, which optimizes the use of system resources by RDF.

Also, the user must designate the volumes that will be protected by RDF. For example, if a system has several applications and only some of them require RDF protection, the user needs to configure only those disk volumes that require replication. After all the configuration parameters are set, the user can verify and correct them with RDFCOM if there are any discrepancies.

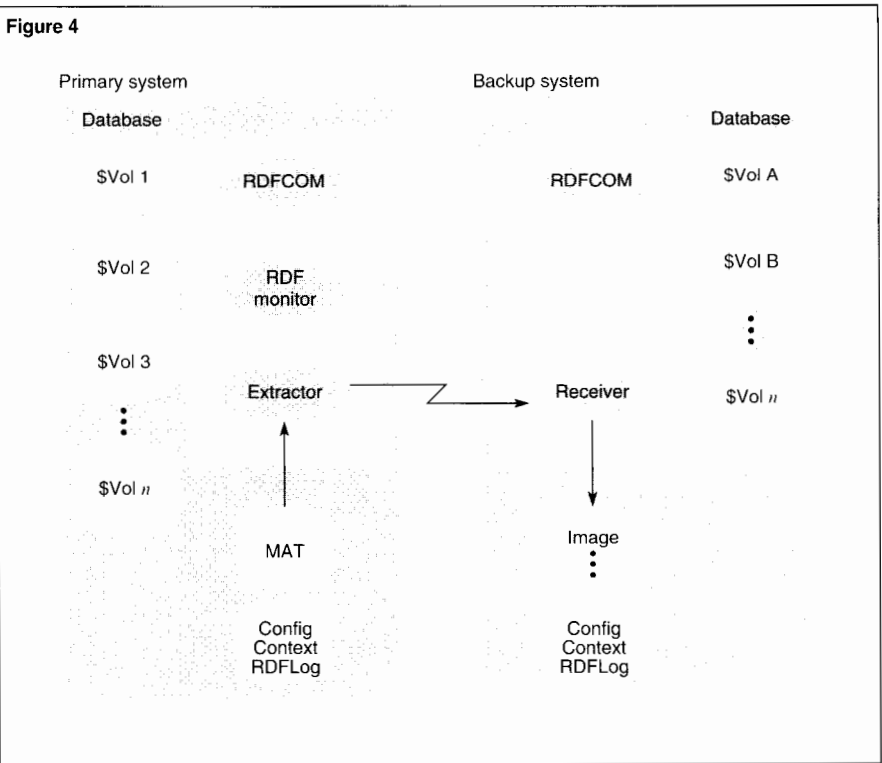
Starting RDF

The user starts RDF through RDFCOM, which starts an RDF monitor process whose main function is to start and control the rest of the RDF processes. These processes send the audit trail data, receive and store the data, and update the backup database files.

Once RDF is running, the user can monitor the activity of all the processes in both systems with RDFCOM (by using a STATUS command). This command displays the current status of each executing RDF process. It also displays the name of the file being processed and an RDF time delay (known as RTD) reflecting the age of the data being handled by the corresponding process. The age of the data is set in relation to the time that it was originally written by TMF into the MAT.

Sending Data

RDF uses a process called the extractor to read each block of data from the MAT on the primary system. Only the records for configured volumes are extracted from the block and transmitted to

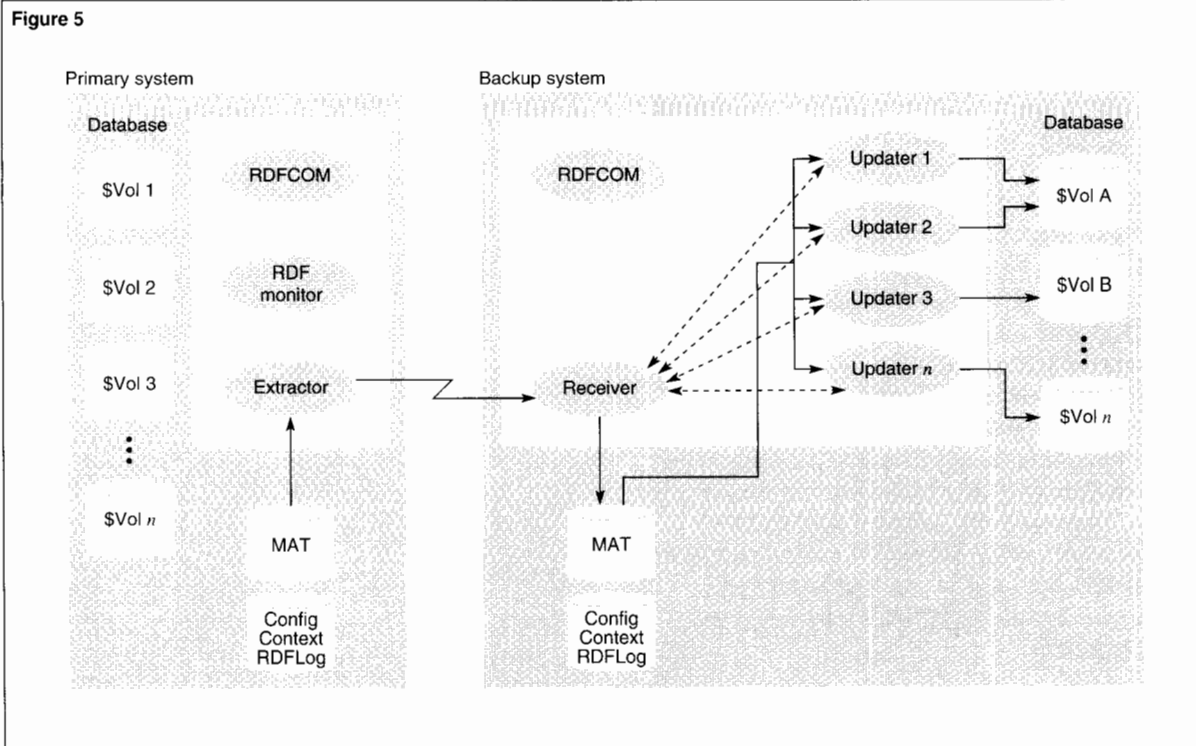


the backup system. (See Figure 4.) As soon as the backup system acknowledges that the block of data has been received, the read-extract-send operation is repeated. While RDF performs this operation, it can encounter an end of file (EOF) if TMF has not yet physically written an audit block. Upon encountering an EOF, RDF waits for a specified time and then performs another read. This wait-read operation is repeated until TMF writes another block of data, which will be detected by RDF on its next read.

Figure 4.
Extractor and receiver processes transfer data from the master audit trail (MAT) in the primary system to the RDF image files in the backup system.

Figure 5.

Update processes update the backup databases.



If the communication line becomes unavailable while RDF is processing, RDF waits for a predetermined, short period of time and retries the send operation. RDF repeats these wait-send operations until the line becomes available again. Operating in this manner, RDF is able to withstand communication line failures.

Receiving Data

As shown in Figure 4, a receiver process receives the blocks of data sent to the backup system by the extractor. This process has two functions. One function is to receive the data and write it into files known as RDF Image files. The other is to communicate with other RDF processes regarding the status of the transactions being received.

The RDF image files are sequentially numbered files. As one file becomes full, RDF creates a new file with the next higher number and uses it immediately.

Updating the Backup Databases

The updater processes update the backup databases. (See Figure 5.) There is one updater process for each primary system volume protected by RDF. For load balancing, the user can configure more than one updater for a backup system volume.

Through RDFCOM, the user can start or stop the updating function separately from the other RDF functions discussed so far. When the updater processes are stopped, the image files will continue to grow in size. Thus, there is a potential for filling up the disk volume on which the image files reside unless the user manually stores, archives, and purges them. Therefore, it is recommended to allocate one disk volume for image file usage.

To accomplish its task, each updater reads the image files in bulk transfer mode, looking for data modification records corresponding to the disk volume it is protecting. When an updater process encounters such an image record, it extracts the transaction ID and solicits the status of that transaction from the receiver.

The receiver keeps track of the completion status (committed or aborted) of the transactions being received. The receiver keeps this information in a status table cache that resides in an extended-memory segment. It keeps this table only when the updater processes are enabled. When the receiver gets the transaction status request, it looks in its status table cache and responds with a buffer containing the status of the requested transaction. It also sends the status of other transactions not yet encountered by the updater, in anticipation of future requests. This minimizes the message traffic between receiver and updater.

When it receives the transaction status, the updater determines which action to take. For transactions that have been committed, the updater applies the modification records to the database files in its corresponding volume on the backup system. For transactions that have been aborted, the updater bypasses the modification records. After the current block of image data is dealt with, the updater reads another block from the image file and repeats the updating process.

When the updater encounters the current end of the image trail, it waits for a predetermined period of time (to minimize CPU resources). Then it reads once again, repeating the update cycle.

Stopping RDF

The user can stop the RDF processes in two ways. First, one can issue a TMF stop command at the primary system. This causes TMF to write a TMF stop record in the MAT after completing all transactions currently in progress. Eventually, the extractor reads the TMF stop record, which will signal all RDF processes to stop after they process all the image records.

Second, one can issue an RDF stop command through the RDFCOM process. This notifies all RDF processes to stop immediately.

Regardless of the way in which RDF is stopped, the processes write the position of the file they are currently processing into a file called the context file. Each system has one context file to keep track of the corresponding processes' locations. On subsequent startups, the RDF processes read the context file, which allows them to continue processing where they stopped.

Database Synchronization

Before starting RDF for the first time, one must synchronize the primary and backup databases. In this context, database synchronization means that the primary and backup databases must be logically equivalent at a time when there is no subsequent audit.

One can synchronize the databases by performing a static backup of the primary databases to be protected. As soon as the backup is finished, one can start RDF with updater processes disabled. This approach will allow the primary application as well as RDF to be started without having to wait for the backup databases to be restored on the backup system. While the databases are in transit, RDF will accumulate audit in the image files until the updater processes are enabled. As soon as the restore is finished, one can enable the updater processes through an RDFCOM command, which causes the updater processes to start functioning. Subsequent RDF startups do not require synchronizations.

The databases will have to be resynchronized if a simultaneous double-CPU failure occurs in CPUs in which the extractor or receiver processes are running. These are the only circumstances in which resynchronization is required.

Ad Hoc Query Processing

Users can perform ad hoc query processing on the backup databases while RDF is running. They can do this because the RDF updater processes open the files in write-protected mode. During query processing, the backup databases are changing dynamically in a manner similar to the primary databases, but no records are locked. (Records in the backup databases do not have to be locked because there is only one updater process modifying each file or partition. In the backup system, two updaters will never attempt to change the same record at the same time.)

Batch reporting or query functions that do not require database activity to be quiescent can also be performed on the backup databases. By performing ad hoc queries and generating batch reports on the backup databases, users can reduce the workload on the primary databases.

Takeover Processing

Whenever the primary system is no longer available (because of a catastrophe or for any other reason), the user can tell RDF to finish processing all the audit the backup system has received and then stop. One does this by indicating takeover processing through RDFCOM on the backup system. RDFCOM directs RDF to process all the image files to completion. Any transactions found to be incomplete at the end of the image file (because abort or commit records were lost during the disaster) are not applied to the backup databases. At the end of takeover processing, the backup databases will contain only completed transactions. Therefore, they will be in a logically consistent state.

After RDF has finished processing the image files, one can begin enabling the backup system to assume all the functions of the primary system. At a minimum, this task consists of enabling all the backup databases for auditing, starting the necessary applications, and switching the communication lines from the primary system to the backup system. One can perform some of these tasks in parallel, depending on the way they are originally set up. Completing these tasks quickly depends on the degree of preparation and testing done beforehand. The more one tests the systems before a disaster occurs, the easier it will be to perform a switchover.

Because RDF has no control over the application or the communication lines, it is only considered a complement to a disaster recovery plan. Users should not think of it as the complete solution for disaster recovery.

Minimizing Application Outages

The RDF product performs another important service; it can help to minimize the duration of planned application outages required to perform change management operations such as:

- System software changes.
- Operating system releases.
- System hardware changes.
- Migrating to a new hardware platform.
- Moving hardware to a new site.

For all of these changes, the system manager has to stop the primary application for the duration of the operation. To minimize the impact the outage will have on the application, one can use RDF to perform a planned switchover to the backup site. One can restart the application there, allowing access to the backup databases. Before one can start the application again on the original primary system, any updates performed on the backup system must be replicated on the primary system. Again, one can use RDF to replicate the data and switch back to the original primary system.

RDF is meant to minimize the time during which the application is unavailable. To further reduce the outage, one should perform certain operations concurrently on both systems. Using RDF in a carefully planned switchover can greatly reduce the outage.

For example, installing a new version of the application software may require an outage lasting a few hours. If one uses RDF and applies some concurrent manual intervention, the outage may last only a few minutes.

To perform a planned switchover, one must analyze the required steps. One needs to stop the application running on the primary system, switch the communication lines, and prepare the backup system to start up the application as soon as possible.

Stopping the Application on the Primary System

When the primary application is stopped, RDF must apply to the backup databases all the audit generated up to that point. A recommended way to perform this step is to stop TMF before stopping the application. This step disallows any new transactions from taking place while allowing any transactions in progress to be completed. One can oversee this process through TMFCOM, the TMF command interface. Recommended application design methodology allows applications to be quiesced when new transaction generation has been disabled.

After all the transactions are completed, one can stop the primary application. (See Figure 6.) Now, because the primary databases are no longer being accessed, one can disable them for audit. This prepares the primary databases for their backup function when RDF runs in reverse mode. Stopping TMF causes RDF to shut down automatically. One can oversee the RDF shut-down through RDFCOM.

Once the primary application has stopped, one can switch over the communication lines from the primary system to the backup system. This part of the switchover tends to vary greatly, as there are many different ways to set up the communication lines.

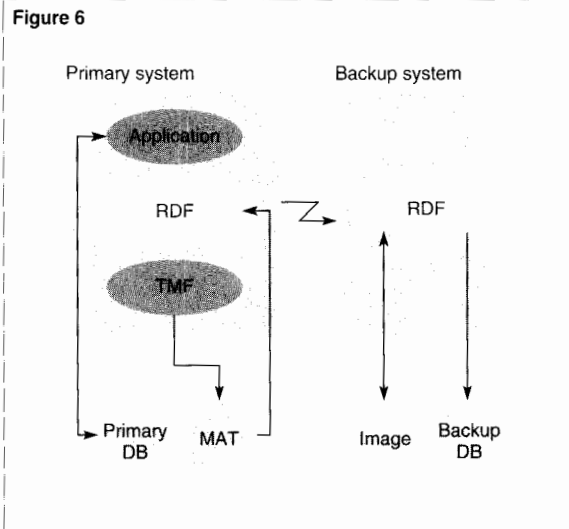


Figure 6.

A recommended approach to managing a planned outage involves first stopping TMF and then stopping the application on the primary system.

Preparing the Backup System for the Switchover

One can prepare the backup system for the switchover ahead of time or at the same time as one stops the application on the primary system. Preparing and testing it ahead of time will ensure smoother operations during the actual switchover. As part of the preparation, one must configure RDF at the backup site to indicate that the application will be running on the backup system. Because the copy of RDF that will run on the backup system is independent of the copy running on the primary system, one can take this step before one stops the application on the primary system.

Figure 7.

Stopping TMF to prepare the backup system for takeover. This involves having TMF write shutdown records into the MAT and the EMS log.

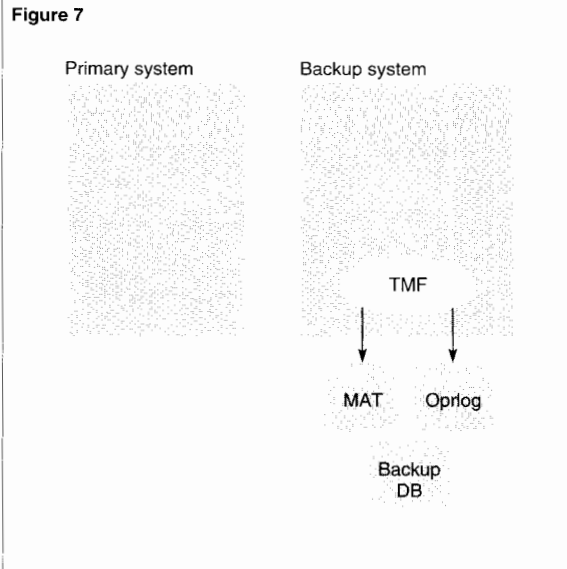
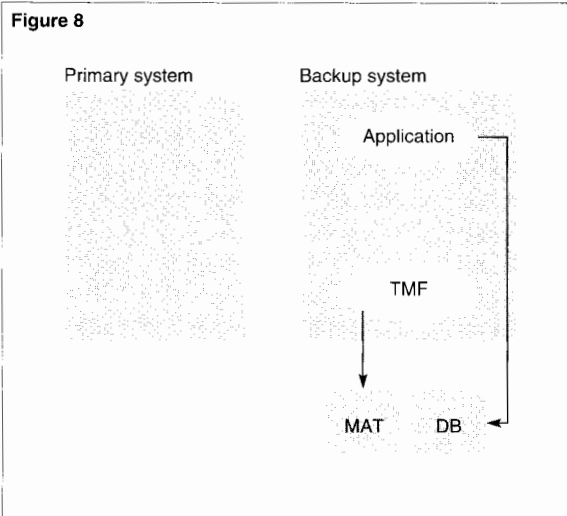


Figure 8.

Starting TMF and the application on the backup system to prepare for transmission of image data to the primary system.



How one performs this initialization step depends on whether or not TMF is already running at the backup site. Although TMF does not have to be running before the takeover, it may already be running to support other applications or SQL catalogs. Assuming that TMF is already running, one will have to stop it momentarily and restart it with transactions disabled. (See Figure 7.)

The object of this step is to set up a starting point at which RDF will begin sending replicated data from the backup system to the primary system. (After the outage, RDF will run in reverse mode to bring the primary databases up to date.) Just before TMF stops, it writes a date-time stamped shutdown record into the TMF audit trail. TMF also writes a log record with the date and time of the shutdown into the Tandem Event Management Service (EMS) log file. One can view the EMS log record and use its date-time information to initialize RDF. This operation ensures that RDF will start at a point at which no transactions are in progress.

Next, one must configure RDF in reverse mode (since it has just been initialized.) Although RDF is configured, it does not run now because the primary system is not available to receive replicated data. One can verify the completed configuration through RDFCOM.

Starting the Application on the Backup System

As soon as RDF is configured in reverse mode, one can start TMF transactions and enable the backup databases for auditing to allow TMF and RDF protection. Now one can start the application on the backup system, as shown in Figure 8.

In some cases, one can start the application before or during any of the previously described steps. This further reduces the switchover time. One constraint is that the database files should not be opened until all of the preparation steps have been completed. However, this is the typical situation with most database applications that use Tandem's NonStop SQL relational database management system.

When users finish the maintenance operations at the primary site and the Expand line is re-established between the systems, one can start RDF at the backup site. RDF begins transmitting the image data created since the beginning of the outage. The image data is transmitted from the backup system to the primary system, as shown in Figure 9.

Next, the updater processes bring the primary site's databases up to date. One can supervise the activity of the updater processes through RDFCOM, which shows the RDF time delay (RTD).

Switching Back to the Primary System

When the updater processes finish applying modifications, one can switch back to the primary system, restoring the original RDF configuration. Switching back to the primary system involves the same steps as switching over to the backup system (described above). One must identify the correct system names, if they were used. Otherwise, switching back involves no special considerations.

As noted, one can perform certain steps concurrently to minimize the time it takes to switch over. Other steps can be performed by an automated operations tool to minimize the need for human intervention. For information about designing applications for automated operations, see the article by Dagenais in this issue of the *Tandem Systems Review*.

Conclusion

The RDF product provides a mechanism for maintaining up-to-date databases at a remote site. Primarily, one can use this facility to complement a comprehensive disaster recovery plan. RDF can also relieve the primary databases of tasks such as ad hoc query processing and certain batch reporting jobs. In addition, RDF provides a way to minimize the duration of planned application outages at the primary databases' site, where high availability is of utmost importance.

Figure 9

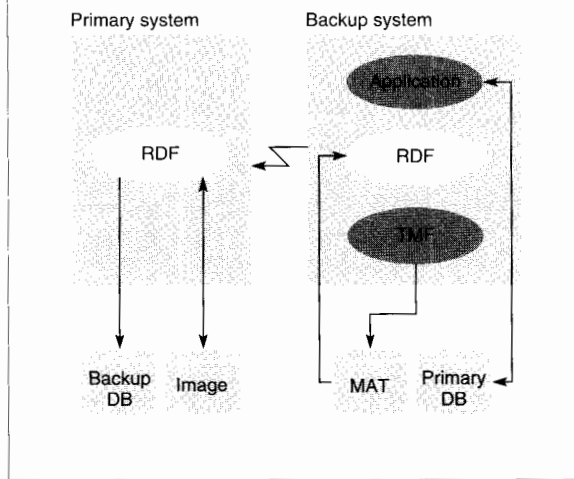


Figure 9.

Transmitting image data from the backup system to the primary system.

References

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

Jorge Guerrero is an advisory analyst in Large Systems Marketing Support. He has worked with RDF since joining Tandem in 1987. Previously, he worked for ten years supporting data management products at another major computer vendor. Jorge has a B.A. in Mathematics and Computer Science from the University of Maine, at Orono, where he was a LASPAU scholar.

Capacity Planning With TCM

Responsiveness is the primary measure of the performance of an online transaction processing (OLTP) system. Tandem™ provides several tools that monitor and tune system performance based on resource utilization. Now Tandem also offers a capacity planning product, the Tandem Capacity Model (TCM). With TCM, users can predict changes in system responsiveness as workloads change and grow, the system configuration is modified, or applications are added or changed.

TCM is a powerful workstation-based modeling product that uses performance data gathered by Measure™, Tandem's system performance measurement product. TCM can calculate changes in the average transaction response time based on planned or potential changes such as:

- Adding Tandem NonStop™ processors or disks to the system.
- Changing the transaction mix.
- Increasing the transaction throughput or number of end users.
- Adding a new function to an application.

Tandem recently asked the author of this article to perform an independent audit of TCM. The audit results indicated that TCM can be a reliable source of the performance predictions required by capacity planners, system managers and analysts, and MIS managers.

This article begins by discussing the business uses of TCM. It then describes the functional steps required for capacity planning with TCM: gathering data, creating a capacity planning model, validating the model, making predictions based on a variety of hypothetical changes, and reporting the predictions. Examples show how TCM can apply to real-life problems.

The article also discusses how to use TCM successfully in environments it does not directly support, such as environments that have significant batch components or are heavily prioritized. A supplemental note that follows this discussion briefly explains the performance modeling principles on which TCM is based. This article is based on the C20 version of TCM (release 1.0).

Business Uses of TCM

TCM has a variety of business purposes. It can help users plan for an expanding system, determine the maximum capacity of the current system, determine the effects of expanded functionality, perform cost-benefit analyses, and provide support for third-party software vendors. In minutes, TCM can calculate answers that would take days or weeks to calculate manually.

The most common use of TCM is for capacity planning, determining the expansion of a system as transaction volumes grow. With TCM, one can increase the volumes of different transactions at different rates. For each change in transaction volume, one can try various configurations of CPUs and disks to determine an appropriate growth plan for the system for the foreseeable future. Alternatively, one can determine the least expensive system that will meet response-time constraints under the modified transaction volumes.

A related purpose is determining the capacity of the current system. Given the response-time constraints of the application, TCM can calculate the largest transaction volume the current system will handle.

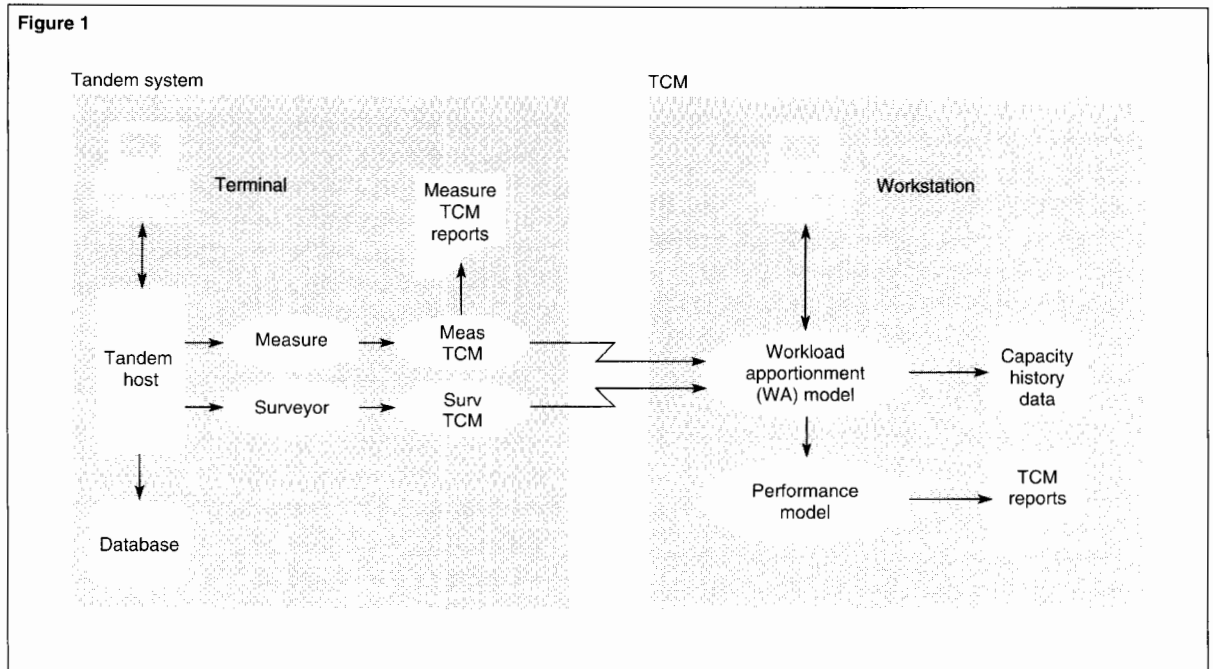
TCM can also help users predict the impact of expanded functionality on the system, that is, the effect of adding new functions to the application. One can include new, unimplemented transaction types by entering programmer-supplied best estimates of the CPU and disk resources used by the new transactions. (One can also change the profile of an existing transaction.) With this information, TCM can determine the system configuration required to support given transaction rates (for the old and new transactions) and meet specified response-time criteria.

One can use TCM for cost-benefit analysis, estimating the performance impact of proposed modifications or enhancements to a system. If users want to change the system to improve its performance, TCM can accurately determine the performance benefits of a given configuration, which can be compared to its dollar costs.

TCM provides invaluable support for third-party software products. For users purchasing third-party software packages, TCM can predict the system configuration required to meet a specified combination and volume of transactions. By using a validated TCM model supplied by the third-party vendor, users can specify the required workload and response time, and TCM will determine the least-cost system configuration to meet those requirements.

A common question concerns the relationship of TCM and Surveyor, Tandem's performance database management product. TCM is a capacity planning product that can answer what-if questions regarding the impact of future scenarios. Surveyor can automate performance data collection and exception reporting, report system usage on an ongoing basis, and can be used to help in capacity planning efforts by forecasting future system usage based on historical trends. The two products complement each other, especially because the performance data stored in a Surveyor database can be used to build TCM models. By incorporating the usage trends provided by Surveyor, capacity planners can refine the analyses they perform with TCM.

Figure 1.
TCM architecture.



TCM Components

To create and use a successful capacity planning model with TCM, one must do five things: gather data from the Tandem system, create a transaction performance model that accurately reflects the real system, validate the model, make predictions based on hypothetical scenarios (furnished by the user), and report the predictions.

TCM uses Measure to gather critical CPU and disk utilization data during representative periods of system use. (Measure gathers information from an operating Tandem system.) MeasTCM, a TCM component running on the Tandem host, categorizes and summarizes Measure data and transfers the summarized results to the TCM workstation. SurvTCM, a similar TCM component, can also perform this function.

The remaining components of TCM run on a PC or Macintosh workstation as a Microsoft Excel application. Figure 1 shows how TCM extracts data from the Tandem host, downloads the data to the TCM workstation, and, guided by the user, calculates capacity planning scenarios.

With the workload apportionment (WA) component of TCM, users define the transactions in their system. Next, they allocate to those transactions the proportion of the CPU and disk resource utilizations reported by MeasTCM or SurvTCM. The WA component can then calculate the CPU and disk resources used by each individual transaction.

Once the workload apportionment model has been created, users validate the model by testing it for consistency against a variety of Measure data samples. One can perform a second type of model validation by testing whether the TCM model accurately predicts response times for the system as it is currently configured.

With the performance model (PM) component, users can change transaction rates, modify the system CPU and disk configuration, and add projected new transaction types to determine the system's reaction. The PM component calculates resource utilizations and response times according to these what-if scenarios.

In addition, users can print a variety of presentation-quality charts and reports to illustrate selected scenarios. For complete information about how to operate TCM, refer to the *TCM User's Guide and Reference Manual* (1990).

Measurement

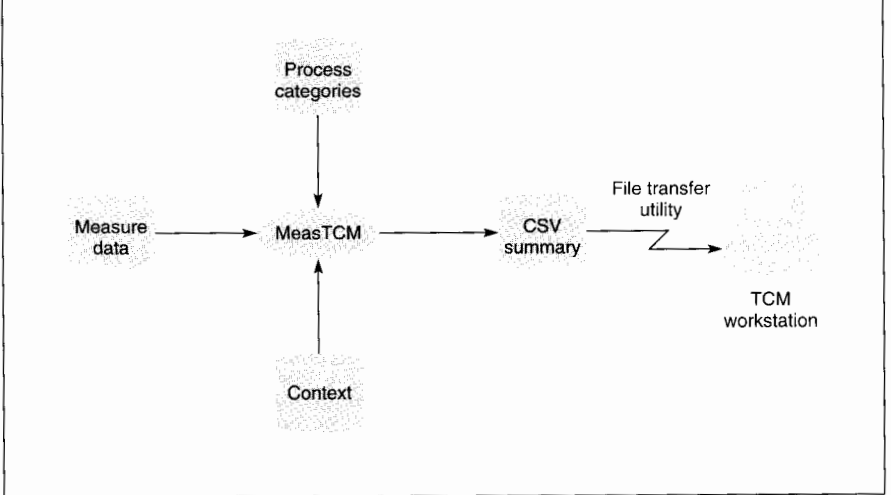
The TCM process begins with Measure, which accumulates performance data on the Tandem system. Two similar tools are available to categorize and summarize Measure data: MeasTCM and SurvTCM. SurvTCM interfaces with Surveyor, Tandem's performance database manager. Surveyor uses Measure data to establish a performance database created by Tandem's NonStop SQL relational database management system; SurvTCM processes the Surveyor data for TCM. MeasTCM interfaces directly with Measure. This article discusses MeasTCM only.

MeasTCM can merge Measure data taken from multiple Tandem systems to create one set of performance data that represents the activity of multiple nodes in a Tandem network. By using this merged data, TCM can perform its capacity planning predictions for the Tandem network.

For MeasTCM, one must collect data on all CPU, process, discopen, and disc entities. These entities provide a variety of useful information to TCM:

- The CPU entity provides the CPU type and count as well as the interrupt busy time.
- The process entity provides CPU times and interprocess message counts for each process.
- The discopen entity provides cache activity and physical disk access counts for each process.
- The disc entity provides the number of disk volumes and total seek-read-write disk times.

Figure 2



The user invokes MeasTCM with an interactive or batch Tandem Advanced Communications Language (TACL) routine. MeasTCM summarizes the large amount of Measure data (many megabytes) into a compact file (a few kilobytes) in comma-separated value (CSV) format. (See Figure 2.) MeasTCM operates in part under the control of a context file, which specifies the input and output files to use, the CPUs to use, the Measure time window of interest, and the CPU type for normalization if the system comprises multiple CPU types.

Figure 2.

TCM host component.

Table 1.
MeasTCM process category types.

S	=	Server processes
R	=	Requester processes
L	=	Line handler processes
D	=	Disk processes
T	=	TMF processes
O	=	Other processes
F	=	Fixed processes (not related to transactions)
Z	=	Ignore these processes

Fundamental to the MeasTCM function is the process category control file. In this file, the user specifies a series of process categories and defines the criteria for assigning each process in the system to a particular category. Each process category is also assigned a process category type. TCM recognizes eight process category types, as shown in Table 1.

MeasTCM allocates CPU and disk activity to processes and thus to process categories. It accomplishes this by linking process activity (such as CPU seconds) with discopen activity (such as cache hits).

CPU times are allocated directly to processes by Measure. CPU interrupt seconds are allocated to process categories in proportion to the send and receive message counts for each process category.

Disk read, write, and seek seconds are allocated to process categories in proportion to their read, write, and seek counts. The seek seconds are further divided between reads and writes within a process category according to the proportion of read and write counts in that category.

Disk process CPU time is allocated among process categories in proportion to their cache activity. Cache misses are weighted twice as heavily as cache hits because a cache miss implies two cache searches, one before the I/O operation and one after.

MeasTCM writes the generated process category summary information to a CSV file. The user then downloads the CSV file to the TCM workstation, using any standard file transfer utility. TCM provides a screen report of this data. (See Figure 3.)

Figure 3

TCMC10		CATEGORY DEMAND REPORT						
		Application:			Example			
		Category File:			EXAMPLE1.CSY			
		Sample Window:			16Jun89	12:24 PM	12:50 PM	
		Window Size:			1560	Sec		
		Interrupts:			608.995	Sec		
Cat Name	Type	Revs	Sends	CPU Sec	C Hit	C Miss	Rd Sec	Wr Sec
Summary	S	22244	118258	474.965	160664	85362	655.040	2564.549
	R	40781	85325	1062.313	0	0	0.000	0.000
	D	239194	118076	1307.431	0	0	0.000	0.000
	T	0	0	0.000	0	0	0.000	0.000
	L	22239	486	183.779	0	486	0.000	20.773
	O	1444	57	2.130	0	7	10.924	6.964
	F	5060	2707	22.424	539	1661	3.114	29.777
ST1	S	12908	90309	292.342	127375	66363	369.605	2181.132
BIG	S	1099	6689	33.915	8521	5124	79.857	98.986
INQ	S	1852	1853	10.978	4118	1702	53.417	0
UPDT	S	6385	19407	137.73	20650	12173	152.161	284.431
NUCLEUS	O	798	0	0.5	0	1	3.84	2.383
\$VIRTUAL	O	0	1	0.001	0	0	0	0
\$TMF	O	636	36	0.849	0	6	1.073	0.962
DISC	D	239194	118076	1307.431	0	0	0	0

Figure 3.

TCM summary of Measure data.

Workload Apportionment

The TCM model is driven by transaction primitives that specify the CPU, disk read, and disk write activity associated with each transaction. Because Measure does not provide transaction-oriented data, the process category data accumulated by MeasTCM in the CSV file must be apportioned to the users' transactions. This procedure is called workload apportionment (WA).

Using the WA module of TCM, users specify the relative amount of each process category used by each transaction. They also specify how transactions will be counted by associating each occurrence of a particular transaction with a message sent or received by a key process or processes. TCM notes the number of sent or received messages and determines the number of each transaction type represented in the Measure data.

Knowing the actual CPU and disk seconds to be allocated to each transaction type, and knowing the number of transactions of each type seen in the Measure run, TCM can calculate the CPU and disk resources used by each individual transaction.

Figure 4

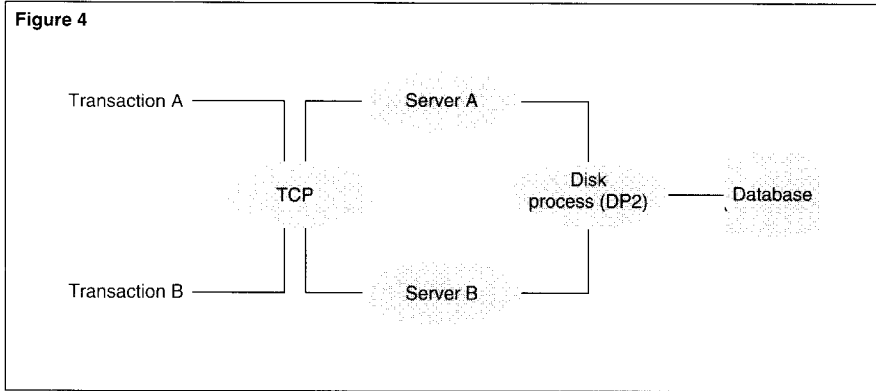


Figure 4.
A simple transaction
model.

Table 2.
Workload apportionment weights.

Process category name	Type	Transaction	
		A	B
TCP	R	2	1
Server-A	S	1	—
Server-B	S	—	1
Disk process	D	1	1
LH1	L	1	1
LH2	L	1	1
TMF	T	1	1
PATHMON	O	1	1
Measure	F	1	1

Workload Apportionment Weights. TCM provides default values for the relative use of process categories. When users indicate that a particular process category is used entirely by one transaction, TCM allocates that process to the corresponding transaction.

For example, consider the transaction flow shown in Figure 4. Transactions A and B are handled by the same terminal control process (TCP) but routed to different servers, Server-A and Server-B, respectively. The example assumes that the requester for Transaction A requires twice as much processing time as the requester for Transaction B. The transactions use the same amount of server and disk time.

Table 2 shows the process categories in this simple model. They include the processes shown in Figure 4 (TCP, Server-A, Server-B, and Disk) and a few other representative processes such as line handlers (LH1 and LH2) and PATHMON. Server-A and Server-B are the server process categories (type S).

In the Server-A category, Transaction A has a weight of one. (Transaction B will have a weight of zero.) Whenever possible, TCM allocates the entire server category to the corresponding transaction. TCM assumes that the allocation of the other process categories is uniform across all transaction types. Thus, by default, TCM allocates the other process categories to all transactions. For example, both transactions, A and B, have a weight of one in the PATHMON category.

Users can modify the default values based on their knowledge of the system. For each process category requiring refinement, users enter the relative weights used by each transaction. For example, in the TCP category in Table 2, users give Transaction A a weight of two and Transaction B a weight of one. (The requester for Transaction A is twice as large as the one for Transaction B.)

The workload apportionment weights illustrated in Table 2 show the relative process category usage. TCM calculates the actual proportionate usage based on these relative usage values.

Transaction Counts. Once transaction weights are allocated, users determine how to count the transactions that were included in the Measure sample. Table 3 shows a transaction count table. With this table, users specify how TCM should calculate the number of transactions of each type. Measure reports the number of interprocess messages sent and received by each process. TCM assumes that a message received by a server process represents one transaction passing through the system. Thus, in Table 3, each message received by Server-A represents one execution of Transaction A. Each message received by Server-B represents one execution of Transaction B.

To count a more complicated transaction, users can modify the default values by choosing a combination of messages sent or received by multiple processes. They can add or subtract messages to arrive at an accurate formula.

Table 3.
Transaction count table.

Process category name	Type	Transaction A		Transaction B	
		Receive	Send	Receive	Send
TCP	R	—	—	—	—
Server-A	S	1	—	—	—
Server-B	S	—	—	1	—
Disk process	D	—	—	—	—
LH1	L	—	—	—	—
LH2	L	—	—	—	—
TMF	T	—	—	—	—
PATHMON	O	—	—	—	—
Measure	F	—	—	—	—

If transaction counts have been determined by another method, users do not have to use the transaction count table. Instead, they can supply TCM with a transaction count file containing each transaction name together with the number of those transactions that occurred during the Measure sample.

Figure 5.

Transaction primitives
created by the workload
apportionment process.

Figure 5

TCMC10		CAPACITY BASELINE				
		Application:	Example			
		Sample File:	EXAMPLE1.SWA			
		Sample Date:	16Jun89			
		Planning Unit:	Hour			
		Apportionment:	EXAMPLE			
		CPU Config:	4	TXP		
		Disk Config:	14	Unaud		
	Txn Name	TPH	CPU Sec	Disk Sec		
				Read	Write	Total
	Average Txn	51330.	0.162	0.030	0.117	0.146
	Fixed	per sec	0.031	0.002	0.019	0.021
	ST1	29787.	0.192	0.029	0.170	0.199
	BIG	2536.	0.182	0.073	0.092	0.165
	INQ	4273.	0.089	0.029	0.002	0.031
	UPDT	14734.	0.121	0.024	0.046	0.071

Combining the user-supplied control information illustrated in Tables 2 and 3 (or supplied in a transaction count file) with the actual measured data, TCM can calculate the transaction primitives (CPU seconds, disk read and write seconds). Figure 5 shows a TCM summary of these transaction primitives.

Model Validation

TCM bases its predictions on the transaction primitives created by the workload apportionment (WA) model. One caveat applicable to any performance modeling effort is that the model must be valid. A useful model must be accurate (consistent), and it must correspond reasonably well with reality. Before using the TCM results, one should make sure that the model satisfies these two important criteria.

Testing the Model for Accuracy. To test the accuracy of the WA model, users should apply the same WA model to several Measure data samples with different transaction mixes and workloads. Users can archive each set of transaction primitives in a TCM history file and then compare the primitives created for each Measure sample. If the results of various samples are consistent, the WA model is correct. If they are inconsistent, users must manipulate the WA model until they achieve consistent results across multiple Measure samples.

Figure 6

TCMC10	CONSUMPTION MODEL								
	Sample Date: 16Jun89								
				Actual	Constraint	Plan			
	CPUs:	Number		4		5			
		Type		TXP	CLX7	CLX7			
		Avg Utilization		58.68%	63%	62.24%			
	Disks:	Number		14		4			
		Type		Unaud		Unaud			
		Avg Utilization		8.79%	35%	32.46%			
	Avg Response Time:			0.505	0.700	0.696			
Txn Name	Txn Per Hour			CPU Seconds		Disk Seconds		Avg Resp Time	
	Actual	Plan	MAX	Actual	Plan	Actual	Plan	Actual	Plan
Average Txn	51330	53866	58122	0.162	0.205	0.146	0.147	0.505	0.606
ST1	29787	29787	32140	0.192	0.241	0.199	0.199	0.610	0.835
BIG	2536	5072	5472	0.182	0.229	0.165	0.165	0.584	0.709
INQ	4273	4283	4610	0.089	0.112	0.031	0.031	0.252	0.346
UPDT	14734	14784	15898	0.121	0.152	0.071	0.071	0.353	0.484

Figure 6.

A TCM Consumption model.

Validating the Model's Predictive Ability. The next test is to validate the model's ability to predict the performance of a real system. To do this, one uses the performance modeling component of TCM (based on the WA model) to predict response times for the system as it is currently configured under known throughput conditions. If the model results accurately reflect reality, the model is valid.

If the model fails the test, the problem must be found and corrected. The problem could be in the model, which could require a finer categorization of processes, different process categorization, a different workload apportionment, or a different transaction definition. Or, the problem could be in the real system, which may be poorly balanced or have a bottleneck caused by a design issue. TCM assumes a well-balanced system with no bottleneck processes and computes response times assuming that all processors and disks are equally utilized. If the system is not balanced, one should tune it to improve the performance of the current system as well as to validate the TCM model. Once the problem is found and corrected, one should revalidate the model before using it.

Performance Modeling

The forecasting function of TCM begins with performance modeling. TCM provides three performance models for evaluating what-if scenarios: the Consumption Model, the Sensitivity Analysis Model, and the Planning Timeline Model.

The Consumption Model allows the user to determine the performance implications of various system configurations and transaction loads. (See Figure 6.) The user can change the number and type of processors and disks, specify the maximum utilization of processors and disks, establish the required average response times, and specify transaction volumes. TCM will determine the least-cost system (if it is given flexibility with CPU and disk counts) and calculate utilizations and response times consistent with the constraints imposed by the user.

Figure 7.
Sensitivity analysis model.

Figure 7

TCMC10		SENSITIVITY ANALYSIS						
		Sample Date:		16Jun89				
		Number of CPUs:		5 CLX7				
		Number of Disks:		4 Unaud				
Txn Name	Increment							
	10%	CPU Util	40%	50%	60%	70%	80%	90%
	5%	Disk Util	15%	20%	25%	30%	35%	40%
Average Txn		Avg R/T	0.463	0.539	0.65	0.831	1185	2.225
		TPH	24615.0	32991.0	41368.0	49745.0	58122.0	58499.0
	Mix %							
ST1	55.30%	Avg R/T	0.558	0.648	0.78	0.994	1.41	2.534
		TPH	13611.0	18243.0	22875.0	27508.0	32140.0	36772.0
BIG	9.42%	Avg R/T	0.535	0.621	0.746	0.949	1.344	2.504
		TPH	2317.0	3106.0	3895.0	4683.0	5472.0	6261.0
INQ	7.93%	Avg R/T	0.226	0.266	0.325	0.421	0.611	1173
		TPH	1952.0	2617.0	3281.0	3946.0	4610.0	5275.0
UPDT	27.35%	Avg R/T	0.318	0.373	0.454	0.585	0.845	1611
		TPH	6732.0	9024.0	11315.0	13606.0	15898.0	18189.0

One can use the Consumption Model to determine the least-cost system configuration. Often, a given response-time requirement can be met by more than one configuration of CPUs and disks. TCM can, in effect, search through the possibilities to find the least-cost configuration that meets the response-time requirement. This least-cost recommendation is not based on empirical cost data, but rather on a proportional costing of CPU to disk.

The Consumption Model is very effective in helping companies establish new data centers that will run new or existing applications. For example, one retail company used the Consumption Model to help size two new regional data centers created to service their two new distribution centers.

The company's order and stocking applications resided on their existing centralized system. Analysts developed a TCM workload apportionment model for these transactions. To simplify the model, they decided to focus on their ten most critical transaction types. They found that these ten transactions accounted for most of their workload and that the profiles of the remaining transactions mapped closely to one of the ten modeled transactions. This enabled them to represent their entire anticipated workload with a simplified model. By specifying their desired average response time and the anticipated transactions per second (tps) for each transaction type, they were able to use the Consumption Model to arrive at the least-cost number of CPUs and disks required to meet their objectives in each of the new data centers.

The Sensitivity Analysis Model lets the user determine the effect of a growing transaction volume on the existing system. (See Figure 7.) It assumes the configuration proposed by the Consumption Model as its baseline. From that level, the user can specify incremental growth of both CPU and disk utilization to get the resulting knee of the performance curve, where increasing utilization levels cause the response time to rise exponentially.

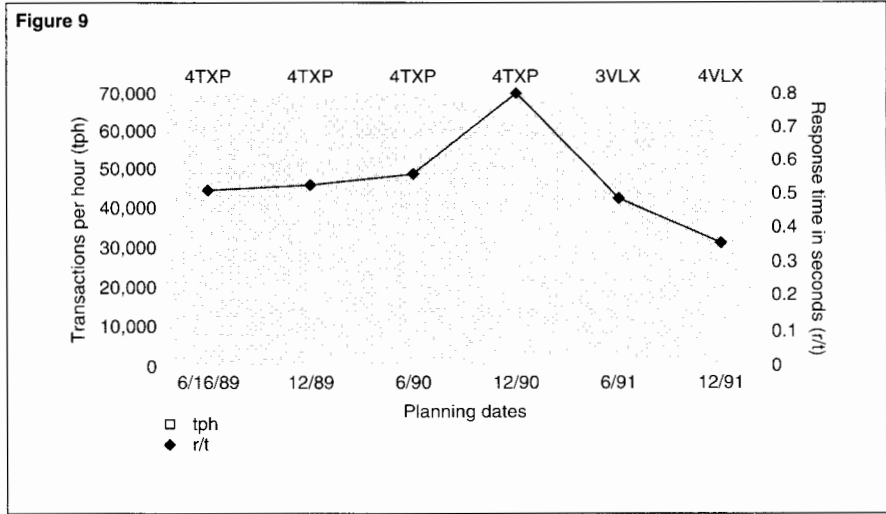
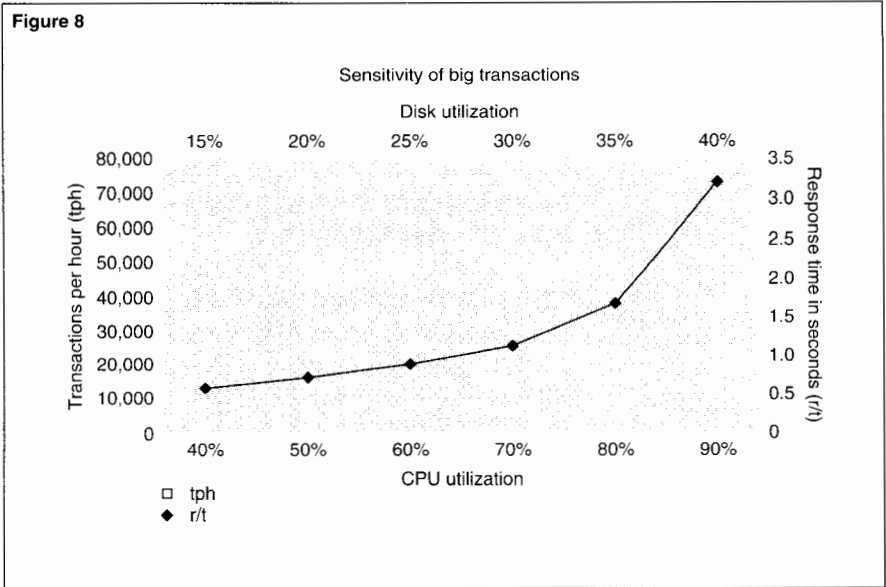
Charting this model into the Sensitivity Chart gives the classic picture of the knee of the curve. The Sensitivity Chart is one of the two TCM presentation-quality graphical reports for management purposes. It shows how response times and CPU and disk utilizations increase as throughput increases. (See Figure 8.)

Consider the example of the expanding retail chain. After analysts used the Consumption Model to determine the least-cost system for their new data centers, they used the Sensitivity Analysis Model to determine the approximate amount of growth the proposed systems could sustain. Next, they charted this model into the Sensitivity Chart, which made it apparent that the proposed system would not be adequate, given the current growth rate and management's desire to install a system that would be adequate for two years.

As a result of this finding, the analysts increased the tps input to the Consumption Model to represent the anticipated two-year growth. Again, TCM forecast the least-cost solution. Using this forecast as the baseline for the Sensitivity Analysis Model, the analysts produced a Sensitivity Chart that met the requirements for the new data centers.

The Planning Timeline Model shows the user how system expansion will meet throughput projections for the coming months or years. The user specifies the length of time being analyzed, the expected increases in transaction volume, and the corresponding proposed changes to the system configuration. TCM will calculate the resulting response times for various transactions at each of the future dates. (The user picks an interval such as 1 month, 8 months, or 24 months, and TCM uses that interval to extrapolate to six dates in the future.)

The Planning Timeline Model benefits high-growth industries and service bureaus that maintain service-level agreements with their users. For example, analysts at one fast-growing telephone company used the Planning Timeline Model to help them plan for system upgrades to ensure that the promised response times were obtainable. By being able to extrapolate increased transaction rates into hardware demands the company gained a much needed perspective



on their resource requirements. They were thus better able to respond to the user community's needs and budget for hardware expenditures.

TCM also provides a Planning Timeline Chart, which indicates how planned system expansions will meet throughput projections by showing the responsiveness of the system at each projection point. (See Figure 9.)

Figure 8.
A TCM sensitivity chart.

Figure 9.
A TCM planning timeline chart.

Other Considerations

TCM is designed for systems running OLTP applications in which a work unit is an individual transaction requiring a rapid response. However, TCM can also work successfully in environments that include elements such as batch jobs and SQL queries.

Using TCM to Model Batch Workloads

TCM can handle small batch jobs that consume negligible processor and disk resources. It does not address large batch runs that consume the surplus processor or disk resources not used by the concurrent OLTP functions.

However, by using the results of TCM, the user can calculate manually the resources and time needed to execute a batch job. For a given OLTP load, TCM calculates the processor and disk utilization, from which the user can determine the amount of processor and disk resources left over for batch processing. Assuming that the batch workload can be characterized in terms of total processor and disk time, the user can determine how long the batch job will run.

For example, assume that a batch workload comprises multiple threads (so that it can use all processors) and requires 60 minutes of processor time and 60 minutes of disk time. It runs in a three-processor system that is 60 percent loaded with OLTP functions. (Disks are 20 percent loaded.) In this example, the processor time is the limiting resource, not the disk time. Three processors that are 40 percent available produce the equivalent of 1.2 processors available for the batch run. The job will be completed in a period of 60 minutes divided by 1.2, or 50 minutes.

NonStop SQL Applications

NonStop SQL attempts to optimize its access strategies based on the available access paths and the sizes of the various tables. Therefore, as an application grows, NonStop SQL may from time to time change its strategies. When one applies TCM to NonStop SQL applications, one must remember that TCM is based on Measure data valid for the strategy in effect when the measurement was taken. TCM cannot predict changes in NonStop SQL strategy.

However, NonStop SQL changes its strategy only if it is more efficient to do so. Therefore, for a NonStop SQL application, the TCM results are conservative. The actual system should perform as well as or better than the TCM prediction.

One can obtain a more accurate picture from TCM by running the application with test tables that represent future growth. This gives the NonStop SQL optimizer the opportunity to devise new strategies consistent with the size of the expanded database. One could then use Measure samples taken during the test run as the basis for more accurate TCM predictions.

TCM Limitations

TCM imposes certain modeling limitations. It is important to understand these limitations when using TCM and interpreting its results.

Process Priorities. TCM assumes that all processes run at the same priority. However, the responsiveness of some OLTP systems depends on priority assignments. TCM calculates the average response time correctly for the whole system. However, response times for transactions using predominantly higher priority processes are reported conservatively, and response times for transactions using lower priority processes are reported optimistically.

Communication Network. TCM specifically does not address delays caused by communication systems. It deals only with the data processing environment. (Data communications is a highly complex subject requiring a different approach to performance modeling than the one TCM uses.)

The Audit Results

The TCM audit analyzed the adequacy of the modeling techniques, examining their accuracy as well as their applicability to a real Tandem system. It concluded that TCM is based on accurate modeling equations and is a powerful and flexible tool for making capacity planning predictions.

There were a few recommendations for enhancements, both to improve the modeling accuracy and to add functionality. For example, the audit suggested that TCM add functions to address priority-assigned processes and batch workloads.

In general, the enhancements recommended to improve the modeling accuracy would have small effects. In almost all cases in which the model could be improved, the model erred on the conservative side. Therefore, TCM response-time predictions should be slightly higher than the actual response times found on the system, leading one to safe conclusions.

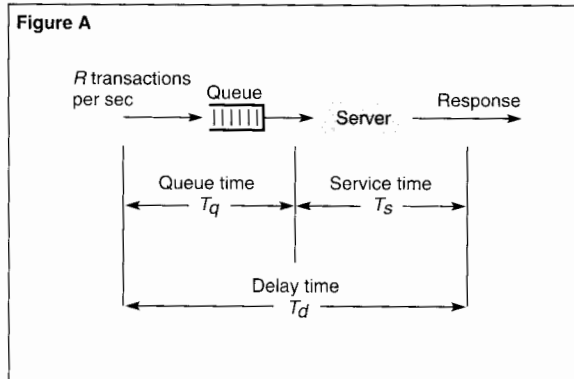
A small part of the audit dealt with the TCM user interface. The author learned to use TCM without benefit of a TCM course. With the help of the *TCM User's Guide and Reference Manual* (1990) and the intuitively obvious windows environment, the author learned TCM reasonably well in a single day.

Although Tandem recommends using a 386-based PC for TCM because of the product's substantial processing requirements, the author used a 386SX-based PC for the audit and found it to be quite adequate. Its multisecond processing time for complex calculations compared favorably to the many hours the computations would take if done manually. However, a faster 386 PC could further cut the calculation time for very large models from minutes to seconds.

Conclusion

The Tandem Capacity Model plays an important role among Tandem's performance monitoring products. It provides reliable performance predictions to support capacity planning and system management decisions. Aimed at the system-knowledgeable person versed in his or her application, it requires no specialized knowledge of performance issues.

Figure A.
A simple queueing model.



The M/M/1 Model

TCM is based on the M/M/1 model for performance evaluation. Figure A shows a simple illustration of this model.

One can think of a transaction processing system as a collection of *servers*. A server is a component that provides a service to a transaction as it proceeds through the system. Typical servers include processors, disks, and processes. Communication lines or networks and I/O subsystems are also servers.

A transaction competes with other transactions for the services of a server. Because most servers can only service one transaction at a time, a transaction may have to wait for a server to become free before it can be serviced. It may have to wait in line behind other transactions that have arrived earlier or have a higher priority. This line is called a *queue*.

The server has an average service time of T_s . (Typical processor service times are 1 to 20 milliseconds; typical disk service times are 20 to 50 milliseconds.) If transactions arrive randomly at a rate of R transactions per second, the server is busy RT_s of the time. This is called the utilization of the server, or the load L on the server:

$$L = RT_s \quad (1)$$

For example, if a transaction requires 50 milliseconds of CPU time (T_s), and if the CPU is processing two transactions per second (R), the load L imposed on the CPU by this transaction is $2 \times 0.05 = 0.1$. That is, the transaction utilizes 10 percent of the CPU's capacity.

When a transaction arrives at the queue of a busy server, it will find that the transaction currently being processed has, on the average, a remaining service time of kT_s . The average time a transaction has to wait in line in the queue before being serviced is called the queue time T_q (Highleyman, 1989). T_q can be shown to be:

$$T_q = \frac{kL}{1-L} T_s \quad (2)$$

The total time the transaction is delayed as it travels through the server system is called its delay time T_d . The delay time is the sum of the queue time and the service time:

$$T_d = T_q + T_s = \frac{kL}{1-L} T_s + T_s \quad (3)$$

Equation (3) is the Khintchine-Pollaczek equation. The term k is a function of the distribution of the service time. (Remember that k is the average portion of service time remaining for the current transaction when a new transaction arrives at the queue.) For randomly distributed service times, $k = 1$. It doesn't matter when one looks at a transaction being serviced. Whenever one views it, it will always have an average service time of T_s remaining. This is called the memoryless feature of random distributions. Typically, CPU processing times are randomly distributed.

Disk service times tend to be more uniformly distributed. (They have an equal probability of service between zero and the maximum access time.) For uniform distributions, $k = 2/3$.

A communication line with fixed-length messages has a constant service time. On the average, a message will be half-transferred, and $k = 1/2$.

If $k = 1$, then, from equation (3),

$$T_d = \frac{L}{1-L} T_s + T_s = \frac{T_s}{1-L} \quad (4)$$

This is called the M/M/1 model. (M stands for memoryless random distributions. The first M specifies random inputs, the second M random service times, with one server.) The M/M/1 model results in the familiar performance curve shown in Figure B.

Assume, for example, that the average transaction processing time in a CPU is 50 milliseconds (T_s), and that the CPU is 75 percent loaded ($L = 0.75$). The average response time for a transaction is then $50 / (1 - 0.75) = 200$ milliseconds. A transaction must wait in line for 150 milliseconds before it is given the CPU, and then it requires an additional 50 milliseconds to be serviced.

How TCM Uses the M/M/1 Model

Currently TCM uses the M/M/1 model for processors and disk servers. This model is appropriate for processors and conservative for disks (for which k is closer to 2/3).

TCM calculates transaction response times as follows. The first task is to calculate CPU and disk utilizations. TCM determines CPU utilization by summing the products of the transaction rate and calculated CPU seconds for each transaction. Next, it adds fixed CPU seconds and scales the result if the calculation is being made for a CPU type different from the one that was measured. Then it divides this total by the number of CPUs.

TCM calculates disk utilization in an analogous manner. One complication is that measured disk activity represents the activity of all physical disks, but Measure reports the number of logical disks. TCM deduces the number of physical disks from the logical disk types (audited, mirrored with serial or parallel writes, or unmirrored).

The response time of each transaction is calculated as the sum of the CPU and disk response time components, using the M/M/1 model. From equation (4),

$$\text{Response Time} = \frac{\text{Service Time}}{1 - \text{Load}}$$

For each response time component, the service time is the total CPU or disk time demand allocated to that transaction. The load is the average CPU or disk utilization, as previously calculated. TCM adjusts disk service time to account for mirrored disks and TMF audit file activity. TCM accounts for the fact that audit activity is capped by the *boxcar* effect; in a busy system, the audit function becomes more efficient as multiple transaction audit records are buffered into a single audit write.

Finally, TCM calculates the least-cost system (in terms of numbers of processors and disk spindles) that satisfies the constraints specified by the user. Typically, many solutions satisfy the constraints. For example, a 1-second response time may be achieved with 2 processors and 100 disks, or with 5 processors and 12 disks. TCM must choose the most reasonable solution.

TCM does this by relating the cost of a disk to the cost of a processor and then solving for the least-cost combination of processors and disks that satisfies the user's response time requirement. It substitutes the disk-to-processor cost relationship into the response time equation and develops an equation for the cost of solutions that satisfy the response time requirement. Next, it differentiates this equation to find its minimum solution. This usually results in a non-integer number of processors and disks. For example, the least-cost solution may be 4.3 processors and 8.7 disks.

Figure B

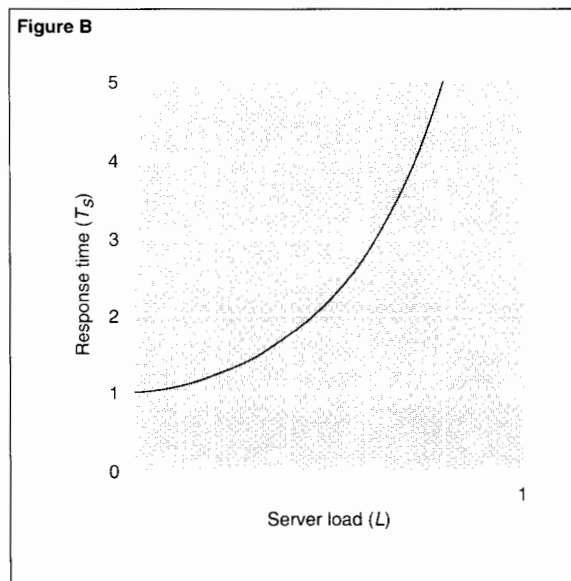


Figure B.
The M/M/1 performance curve.

TCM evaluates the relative cost of whole numbers of processors and disks around this solution point and proposes the minimum cost solution. In the preceding example, it would evaluate the relative costs for 4 processors plus 9 disks and 5 processors plus 8 disks, verify that each combination satisfies the user's constraints, and propose the combination with the lower cost.


References

- Highleyman, W. 1989. *Performance Analysis of Transaction Processing Systems*. Prentice-Hall, Inc.
- Tandem Capacity Model (TCM) User's Guide and Reference Manual*. 1990. Tandem Computers Incorporated. Part no. 50220.

Acknowledgments

I would like to thank the Tandem developers and customer support staff who reviewed this article. Special thanks are due to Jane Jensen, who provided the examples of how Tandem customers have used TCM. (Jane works in the Performance Center in Large Systems Marketing Support.)

Wilbur H. Highleyman is chairman of The Sombers Group, Inc., a Tandem Alliance member specializing in heterogeneous applications. He holds a doctorate in Electrical Engineering from the Polytechnic Institute of Brooklyn. He is past president of the International Tandem Users' Group and is the author of *Performance Analysis of Transaction Processing Systems* (1989).



s data processing becomes a larger part of business life, the need to have more data available and accessible becomes increasingly important. Dial-in access to mainframe systems is one method that makes data stored on mainframes available to individuals. Additionally, a growing need exists for providing distributed data and allowing users to have both local and network access. Access to data is often required by different groups of people, for example, by data processing staff and ATM users of the same bank, at the same time, albeit for different purposes.

The security issues inherent to dial-in access are generic problems facing the computer industry. This article, focusing primarily on the Tandem™ environment, explores some of the challenges associated with providing dial-in access to computer systems and discusses a number of security solutions designed for

Tandem systems. Various forms of deliberate attacks on the system, as well as unintentional security breaches, can exist. This article examines these possible problems, outlines some deterrent or preventive actions, and describes some options for recovery action to ensure the application is confident of the identity and status of the dialed-in user. Several examples of potential security breaches show the sort of risks that must be considered, and suggestions address ways to circumvent these threats.

Ensuring Security With Increasing System Access

The explosion in the number of personal computers and the amount of people using them to work from home has greatly increased the need for dial-in access to both development and live systems. While it is desirable to make access from home as fast and easy as possible, it is essential to maintain system security. Access to the system must be limited to authorized users. The number of unauthorized callers, those people who may want to explore the system for amusement or with criminal or malicious intent, is increasing.

These factors, combined with increasing government legislation on data confidentiality, have led to the realization that information, being much more freely available, must be more carefully protected than in the past. The level of security that controls dial-in access to the computer must at least match the level previously used to limit physical access to the system. Physical security controls, such as identity cards or keys to the building, are effective when users tend to work inside the user premises or from physically controlled locations.

Various communications protocols can be used for dial-in system access. Asynchronous, bisynchronous, X.25, and Systems Network Architecture (SNA) protocols or a combination of protocols, such as dialing into a public packet assembler-disassembler (PAD) asynchronously and making an X.25 call into the target system, are all commonly used access methods. While the security implications are similar for all these methods, this article concentrates mainly on asynchronous dial-in access, which tends to be the most common method of attack, as it has fewer controls against errors or abuse than the other protocols mentioned.

Deciding where and how to implement and control the system's security means considering a number of choices. Security can be executed by the application, by the system, or by some form of front-end device that filters access to the application. One implementation may not work perfectly in all cases, and a combination may be the best solution. The application may not have access at a level low enough to intercept certain attacks, and complex customization may make it unwieldy. Yet security implemented at the system level may not fully understand the meaning of the application objects, such as a transaction type or a data element, that are being protected.

Additionally, the security implementation must allow for the features of the Tandem hardware architecture. The fault-tolerant Tandem system may, in the event of an I/O failure, allow applications to continue while retrying the I/O. Other security systems without such fault-tolerant features may terminate the session, or even the system, in the event of such an error.

There is, therefore, a balance to be struck between allowing the application to be available for as long as possible and ensuring that a security breach cannot take advantage of a real or simulated I/O problem. System intruders may try subverting or bypassing security measures in operation at the initial connection by attempting an intrusion at retry time. With Tandem, the security implementation can be designed to tolerate communication errors and invisibly reconnect a caller, or it can force the caller to reauthenticate when the connection has lost the assurance of the caller's identity.

Figure 1

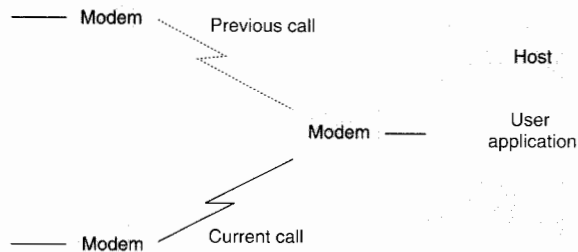


Figure 1.
Multiple user dial-in.

Problems Associated With Dial-in

When dial-in communications allow multiple users to use the same line, the security considerations can be divided into two main areas. One consideration is the need to identify the user to an appropriate level of assurance. The other relates to the problems associated with unexpected events or malfunctions. Figure 1 illustrates a dial-in configuration that accommodates multiple users.

Reliable Authentication: Security Considerations

Reliable authentication of a dial-in user must establish that the user is indeed who he or she claims to be. The level of assurance needed should be determined by the facilities that become accessible to a user after authentication. Although a user password may usually be sufficient for most general applications, access to a funds transfer application, for example, often requires a much higher level of assurance.

Allowing system access to an unauthorized user could have many adverse effects, depending upon the facilities an authenticated intruder can initially access. Sensitive data could be disclosed, altered, or deleted, or the availability of the application could be impaired. The whole network could be probed or attacked if the intruder is able to break out of the application or environment that is initially accessed.

There are, in addition, other considerations, such as legal and contractual obligations as well as the possible loss of business or business confidence following any security breach. A risk analysis should highlight these and other areas.

Various methods, such as biometrics or challenge-response devices, are available for user authentication. Biometrics involves recognition of specific physical attributes of the user, while challenge-response devices generally require the use of some form of hardware device. Most commercial systems, however, base user authentication on passwords. Safeguard™, the Tandem system protection product, has extensive password management facilities. It enforces such features as frequent password change and minimum password length and prohibits reuse or rollaround of passwords.

User authentication is based on three fundamental criteria:

- What you know (such as a password).
- What you have (a personal token, such as a smart card or authentication device).
- Who you are (as established, for example, by a biometric method of fingerprint recognition, retinal eye scan, or voiceprint).

More secure authentication can combine these criteria. The thoroughness of the method or combination chosen should be defined by the security policy implementation. This should usually depend on how seriously the implementer views the perceived cost of any possible breach and on the acceptability to the end user of the authentication method.

Quite often in a support or development environment, it is necessary to allow a dialed-in user to access data on another node, for reference or for problem-solving assistance. For the sake of convenience, the passwords for these user IDs are not always changed as frequently as they should be. Such network-wide access could give the caller opportunity to probe the system for inadequately secured files. Therefore, authentication for any such users must be strictly controlled.

Effects of Disorderly or Enforced Disconnection

In many cases, dial-in lines are notoriously unreliable. Line noise with a subsequent disconnection is the most common error requiring some form of recovery action. If the line noise or parity error rate is excessive, the modems typically break the connection. Sometimes the line noise becomes so unacceptable that the caller just hangs up, instead of shutting down the session in an organized way. When a modem disconnects or a user hangs up, the application is left in a disorderly or confused state. The application must then take some form of cleanup action, such as requiring a new logon, as soon as it detects that the initial user is off the line.

Cleanup is easier for protocols such as X.25 and SNA, which require a form of handshake (a call packet or bind). Both ends establish a session and agree to a starting point where any disorderly connection can be resynchronized. Most asynchronous connections are much more prone to confusion, as there is usually no handshake. This type of protocol may merely acknowledge that there is a call on the line, indicated by the modem signals.

Any confusion due to a disorderly disconnection could cause a new caller to inherit the session of a previous user. The new caller would have access to the privileges and opportunities granted the initial user. Obvious security problems, such as those discussed above, can result from an unauthorized user inheriting a previous session.

Additional problems occur if multiple processes were communicating with the same initial user. The process detecting the broken connection would have to find some means of informing all other processes that the initial caller was lost. The following examples describe the confusion that could arise when the connection with the caller is lost in an unexpected way.

Inheriting a Previous Session. Suppose a local bank has an application based on the Pathway transaction processing system that allows dial-in access by its customers. If a requested transaction is slow, the dial-in user may become impatient and hang up, disconnecting the line before receiving the reply from the Pathway server. If another dial-in occurs before that transaction is completed (before the display is sent to the caller), the new customer may see possibly confidential financial information when it, is displayed from the previous customer's dial-in session.

This problem exists for Tandem's older 6303 controllers and for some half-duplex modem connections. Upgrading to the newer 6100-style communication subsystem allows a disconnection always to be noticed for full-duplex operation. With older controllers and half-duplex connections, the Pathway terminal control process (TCP) has no way to detect that while it was processing a WRITE to the terminal, the modem signals had changed and a new caller has been connected. Some modems are switchable between full-duplex and half-duplex operation, which helps avert this problem.

Asynchronous Configurations

Communications between the application and the dialed-in user are made via the pins between the Tandem controller and the modem. Figure A illustrates a typical connection. The signals that have significance for dial-in connections are:

- Data terminal ready (DTR)
- Data set ready (DSR)
- Data carrier detect (DCD)
- Request to send (RTS)
- Clear to send (CTS)
- Transmit data (TD)
- Receive data (RD)

DTR is controlled by the application, which indicates that a session is available at the host. DSR and DCD are both controlled by the modem. DSR indicates that the modem is available, and DCD indicates the presence of a connected remote modem. RTS is raised by the host to request permission to transmit data, and CTS is raised by the modem to indicate the availability of the modem to receive data. TD carries data transmitted to the modem, and RD carries data sent to the host.

When the modem port is opened by a process and a CONTROL 11 (wait for modem connect) is issued, the DTR pin is raised. This allows an incoming call to be accepted (DCD and DSR raised) if the modem is configured for automatic answer. Many modems allow for these actions to be individually altered.

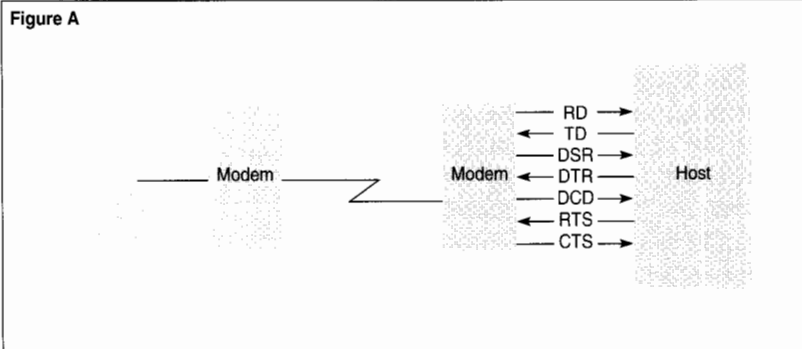


Figure A.
Modem connections.

continued on next page

If, instead, the initial caller disconnects and no new call is received while a transaction is in progress, some modem signals are low instead of high, and the TCP detects the loss of the caller and returns an error. This allows the screen COBOL (SCOBOL) requestor to handle the disconnection and reconnection programmatically, inhibiting the display from the previous transaction. But whenever modem signals are the same as when the transaction was issued, the TCP does not know that a new user is connected and the process therefore proceeds with the next screen display.

Inheriting a Delayed Process. This second scenario examines a development environment wherein a security process screens and authenticates users before allowing them to start their own processes. Suppose a user is successfully authenticated and starts a Tandem Advanced Command Language (TACL™) process, while in the meantime the security process waits for the TACL to terminate before resuming control of the line. The user starts a file utility program (FUP) command to duplicate a file, which may take a minute to complete. In that time, perhaps the noise level on the line increases or the user hits the break key and either logs off or hangs up.

The security process detects the loss of the caller, issues a communication error, breaks the connection, and waits for the next call. When it arrives, the call is authenticated. Meanwhile the FUP completes and, based on the ID of the previous user, still attempts to talk to the previous user. The next caller not only has a new TACL process started but also gets the FUP prompt from the previous user. That user's privileges are now vulnerable to use or abuse.

Methods of Control

A variety of methods are available to counter some of these potential security threats. Using Tandem security products such as Safeguard correctly is one option. Customizing or introducing application code to protect port access or implementing a solution such as a challenge-response device, a modem dial-back facility, or a smart card are other possibilities. Depending on the particular threat or set of threats being faced by a system, security concerns can be addressed by choosing a single method or by using a combination of security controls.

Safeguard

With the introduction of Safeguard into the Tandem Guardian™ 90 operating system environment, control over the use of dial-in lines has greatly improved. An access control list can be set up against a dial-in line to ensure that only specified individuals or group users have access to open the device.

Because the current version of Safeguard offers greatly enhanced password management, its abilities to ensure frequent changes of passwords and to deny previously used passwords help ensure that "well-known" logons do not remain as a target. Safeguard can be configured to disable both user IDs and ports after a threshold number of failed logon attempts has been reached.

Another recently released Safeguard feature allows terminals to be centrally controlled until after a Safeguard logon has been accepted, whereupon a user-specified process is started against that terminal. A consistent system-wide ring of security can be imposed before any caller can obtain access to an application process.

If a full-duplex modem is being used and autodisconnect is specified, DTR is dropped if DCD is lost. Any attempt to read when DCD (or DSR) is low, or if CTS does not rise and fall in response to the raising or lowering of RTS, causes an error 140 (communication error). If a CONTROL 12 (disconnect modem) is issued from an application, DTR is lowered until another CONTROL 11 is issued. After that, another call can be accepted. DSR is also monitored once per second, and an error 140 is issued if DSR is found to be low.

If, however, a half-duplex modem is used, DCD cannot be monitored as it rises and falls with data transfers unless the modem has been adapted to keep DCD high. Caller breaks and reconnects can be invisible, although modern modems are now much more flexible and can sometimes be configured to detect these problems.

Tandem's older 6303 asynchronous controller works in a slightly different manner than the newer 6106 and 3606 controllers. The 6303, which uses Tandem Terminalprocess I/O process, does not notice an I/O error (such as DSR dropping) unless there is an I/O outstanding on that port when the error occurs. It is possible that a call can be broken and another call established without the application being alerted that it is now connected to a new user.

Upgrading to the new controllers or using the READ CONTINUOUS feature to ensure that a read is always posted solves this potential problem. This situation should not exist for the NonStop™ CLX™ and NonStop Cyclone™ system emulation of the 6303 controller, the 3603. This controller is for Envoy™ use, and the READ CONTINUOUS feature can be used.

Similar problems exist with cluster controllers. The asynchronous caller may be lost while the SNA or X.25 session possibly remains. Tandem's 6600 intelligent cluster controller is sensitive to this potential threat and returns an error 140 to the SNA session when a logoff or disconnect occurs. The recent 3270 Release 4 Port Expansion Feature by IBM now addresses this problem. This feature also does not allow a dial-in asynchronous session to inherit an SNA session left incomplete by a previous user.

Pathway Error Handling

Pathway offers two options for managing errors. Pathway can handle errors itself, or the SCOBOL programmer can handle errors using the USE FOR TERMINAL-ERRORS feature. If Pathway handles errors internally, an error 140 causes that terminal thread of the TCP to become suspended.

The CALL ON ERROR command used from the logon module level can trap all errors. Any module doing local error correction can use the USE FOR TERMINAL-ERRORS declarative. If the application is handling them, the SCOBOL RECONNECTMODEM statement is used to do a control 12 and 11 sequence. Good programming here would include a logical logoff sequence before accepting the next call when dial-in access is possible.

Figure 2

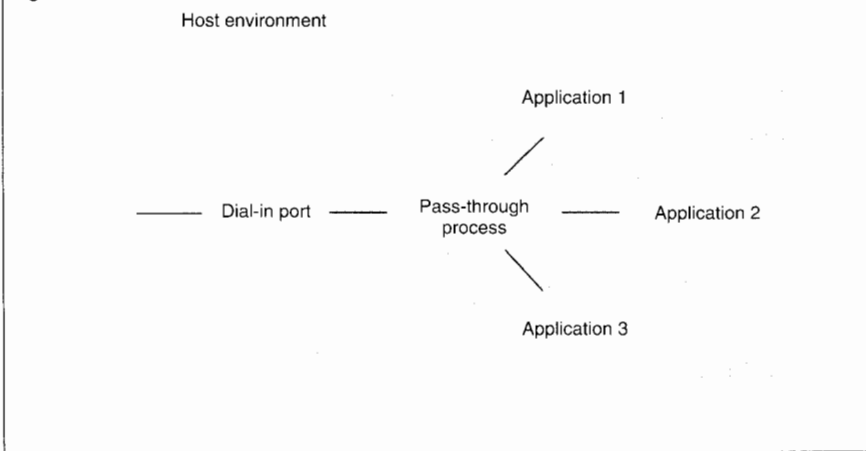


Figure 2.
The pass-through process.

Application Control

One can see that to control a port securely and maintain full knowledge about the connection between a user and any application, one solution is to use a pass-through process that opens the port exclusively. Figure 2 illustrates how a pass-through process can be implemented.

In this configuration, all applications open this process as if it were a port. Any error and recovery handling is done by the pass-through process, which sees all errors and retries on the line and can enforce a new logon by the caller to the application.

The pass-through process opens the modem device and its own \$RECEIVE file for communication to the application (both nowaited). Requests from the application are forwarded from the \$RECEIVE file to the modem port, returning the data received from the modem in the REPLY call back to the application. The status from the modem is also passed back to the application in this REPLY call (for all kinds of operations).

If the pass-through process detects a serious error (such as a communication error or a line, device, or path failure) when the line communication has failed, it marks all current application requester processes as invalid and constantly returns an error 140 (communication error). This continues until the process either closes the line or issues CONTROL 12 and 11 messages to disconnect and wait for modem connect, respectively.

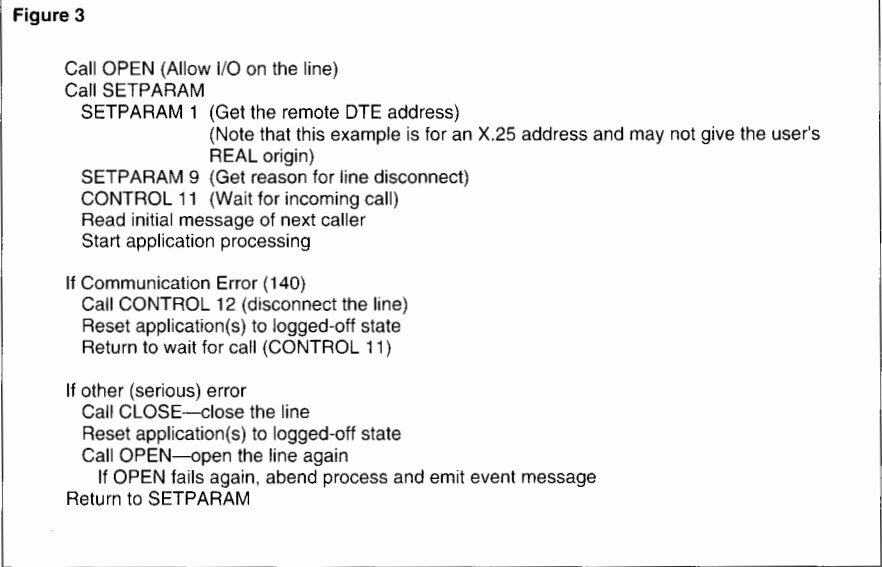
Figure 3 provides an example of recovery logic for errors of this type. For a SCOBOL program, the terminal type must be explicitly specified. Special consideration must also be given if the process is to simulate a block mode terminal by running it as a subtype 30 process. This allows the process to receive DEVICEINFO messages. The process appears as the device controller to the application, which then sends device-specific requests to the process.

One advantage of a pass-through process, in addition to new user detection, is that various degrees of customization can be included. For example, it is possible to specify a time of day or a particular day when certain functions may be performed. The pass-through process could also monitor all processes communicating to that port, all communications of a certain user ID, or all the processes started by itself. If necessary, the pass-through process can send shutdown requests and warning messages when the integrity of the caller or the application becomes dubious.

There are several software packages that implement these features, one being the ION family of products developed by TransComm, a division of Tandem. Using such packages would allow a large degree of customization, the possible inclusion of a challenge-response device, and error translation on pass-through as well as some options for reauthentication after line failures.

Challenge-Response Devices

One authentication procedure that is more secure than the use of static passwords involves using a device that gives some form of individual response which varies with each logon attempt. In this scheme, the user must have knowledge of a password plus possession of a physical device. Usually the device is activated by using an individual piece of knowledge such as a personal identification number (PIN) as the password for the device.

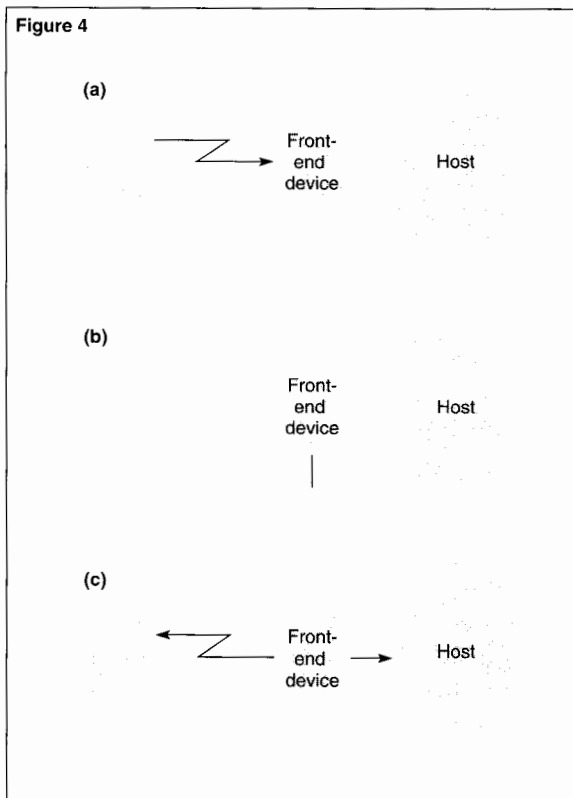


This type of double check is similar to a user having a cash card for an automated teller machine as well as a corresponding PIN in order to withdraw funds. This type of security blends the what-you-have and what-you-know authentication methodologies.

Figure 3.
An example of recovery logic.

Figure 4.

The modem dial-back facility. (a) Front-end device accepts a call and authenticates the user. (b) The front-end device breaks the call and validates the dial-back number. (c) Finally, the front-end device dials back and connects the user to the host system.



The Atalla™ Challenge/Response™ (ACR) product employs this dual authentication method. The ACR provides secure customer or employee identification that is both geographically and terminal independent. It is a small hand-held device, similar in size to a pocket calculator, containing a unique data encryption algorithm (DEA) key. This key is used to ensure that the requester has both knowledge of a PIN and physical access to the device.

When a user begins a logon request, the host machine generates a random numeric challenge and sends it to the user's terminal. The user enters that value into the ACR, after activating it with the user's unique PIN. The ACR calculates and displays a response to the challenge, which can then be entered into the terminal. This response is checked at the host, using one of Atalla's host security modules for response verification with the database of user devices, thus ensuring a unique challenge-response for each logon attempt. When the user receives verification, the logon request is completed and the user is granted system access. The ACR meets two of the criteria for secure authentication, and does so at a low cost.

Atalla also provides other security products. One is a secure identification card that does not require a challenge; it requires merely the input of the currently displayed number for authentication. Another is a software interface layer (the Cryptographic Security Manager) that allows many functions to be easily added into the application at a high level to ensure ease of implementation and maintenance.

Front-End Devices and Modem Dial-Back Facilities

Several suppliers provide equipment that intercepts calls to the host and performs some type of user authentication before the call is transferred to the host system. Several methods are used to authenticate the caller, such as the challenge-response technique or receipt of a specific number. The database of authorized users and telephone numbers is maintained by the front-end device.

Another authentication method often used with front-end devices is the modem dial-back facility. The front-end device receives a call and performs a handshake whereby the caller's phone number or password is accepted. (See Figure 4a.) The call is then terminated while the device scans its internal lists to validate the data it received. (See Figure 4b.) The authenticated caller is then dialed by the device once it has determined that the call originated from an authorized address. (See Figure 4c.)

Some users prefer a modem dial-back facility because the host pays the cost of the line connection and keeps a central record of the costs and the locations it redials. Additional features can also include audit trail facilities and both fixed number dial-back and variable number dial-back. For variable dial-back, users enter the number on dial-in if they are authorized to do so. Both fixed and variable dial-back can have time constraints to allow more control of system security.

Automatic Logoff

Configuring TACL for automatic logoff can greatly reduce the chances for an intruder to access the system on an idle terminal while a TACL prompt is on the screen. TACL can be directed to log off the user automatically if no activity is seen for a fixed, preferably short amount of time since the last TACL command. Applications can also be written this way with the use of nowaited I/O. While this does not eliminate the risk of an unauthorized user accessing a logged-on terminal, it helps to ensure that a terminal is not vulnerable for long periods, such as lunch times or meetings, or when the authorized user has forgotten to log off.

Smart Cards for Personal Computers

Personal computers offer various methods for authentication. Security boards or the addition of smart card readers are two of the most common options. These allow local processing of any security operations. A smart card is the more sophisticated method, in that both authentication and security functions are possible right on the card. This ensures that smart card users have the knowledge to identify themselves to the card and also have physical possession of their cards.

General Security Guidelines

A tradeoff always exists between security and ease of operation. Each installation must choose the appropriate dial-in security for avoiding any damage that could result from inadequate control of dial-in users.

The level of security should be scaled to match the threat of intentional or accidental abuse, and it should be justifiable on those grounds. At the same time, the security should be as unobtrusive as possible and should neither encourage anyone to attempt to break into the system nor discourage anyone from using it legitimately.

The methods used should be decided not at random but with reference to the company's security policy. That policy should state the terms of reference for the implementation of the chosen security. In this way all data can be protected in a consistent way, avoiding a situation of having too much protection in some areas while leaving an alternative path with much slacker security in others. Security is only as good as its weakest link.

The following points suggest some general security guidelines.

- If the user can in some way break out of an application, such as starting a new process from TACL, then it is appropriate to have a much higher level of authentication and auditing.
- The dial-in security method should make it impossible for a break in a call to result in a session inheritance for the next caller.
- Half-duplex asynchronous operation should be avoided if possible. This reduces the possibility that the security method will not detect a session break and a new dial-in user. Check the modem specification for clarification.
- Any session controller should ensure that a disconnect and reconnect (RECONNECTMODEM in SCOBOL) are issued in any situation wherein the identity of the user has become uncertain. One such situation is an unexpected error or logoff.
- Only one process should be actively using the terminal at any point in the session. Truly asynchronous processing is usually illogical in a dial-in environment.
- Any process started from another and detecting an error on its connection should report the condition to the initial session logon process. The session controller should abort the session as well as any existing slave processes that may have the terminal open. This minimizes the risk of session inheritance.

Conclusion

Dial-in access is a necessary and desirable feature in both development and live transaction processing environments. If the power and flexibility of this means of communication can be harnessed without compromising the security of the system, all of the services offered can be made more available and the system resources can be utilized more fully and efficiently.

Peter Grainger works in the European Consultancy Group and is based at High Wycombe in the United Kingdom. He has been with Tandem for ten years and specializes in security systems.

TANDEM SYSTEMS REVIEW INDEX

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

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

As of this issue, 16 issues of the *Tandem Systems Review* have been published:

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

The articles published in all 20 issues are arranged by subject below. (*Tandem Journal* is abbreviated as TJ and *Tandem Systems Review* as TSR.) A second index, arranged by product, is also provided.

Index by Subject

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

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

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

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

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

Index by Product

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

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

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

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

Article title	Author(s)	Publication	Volume, Issue	Season or month and year	Part number
Miscellaneous¹					
A Performance Retrospective	P. Oleinick	TSR	2,3	Dec. 1986	83938
Batch Processing in Online Enterprise Computing	T. Keefauver	TSR	6,2	Oct. 1990	46987
Buffering for Better Application Performance	R. Mattran	TSR	2,1	Feb. 1986	83936
Capacity Planning Concepts	R. Evans	TSR	2,3	Dec. 1986	83938
Configuring Tandem Disk Subsystems	S. Sitter	TSR	2,3	Dec. 1986	83938
Credit-authorization Benchmark for High Performance and Linear Growth	T. Chmiel, T. Houy	TSR	2,1	Feb. 1986	83936
Data-window Phase-margin Analysis	A. Painter, H. Pham, H. Thomas	TSR	2,2	June 1986	83937
Estimating Host Response Time in a Tandem System	H. Horwitz	TSR	4,3	Oct. 1988	15748
Getting Optimum Performance from Tandem Tape Systems	A. Khatri	TSR	2,3	Dec. 1986	83938
Network Design Considerations	J. Evjen	TSR	5,2	Sept. 1989	28152
New Software Courses	M. Janow	TSR	1,2	June 1985	83935
New Software Courses	J. Limper	TSR	4,1	Feb. 1988	11078
Optimizing Batch Performance	T. Keefauver	TSR	5,2	Sept. 1989	28152
Optimizing Sequential Processing on the Tandem System	R. Welsh	TJ	2,3	Summer 1984	83933
Performance Considerations for Application Processes	R. Glasstone	TSR	2,3	Dec. 1986	83938
Performance Measurements of an ATM Network Application	N. Cabell, D. Mackie	TSR	2,3	Dec. 1986	83938
Peripheral Device Interfaces	J. Blakkan	TSR	3,2	Aug. 1987	83940
Predicting Response Time in On-line Transaction Processing Systems	A. Khatri	TSR	2,2	June 1986	83937
Remote Support Strategy	J. Eddy	TSR	3,1	March 1987	83939
Streaming Tape Drives	J. Blakkan	TSR	3,2	Aug. 1987	83940
Subscription Policy for Software Manuals	T. McSweeney	TSR	2,1	Feb. 1986	83936
Tandem's New Products	C. Robinson	TSR	2,1	Feb. 1986	83936
Tandem's New Products	C. Robinson	TSR	2,2	June 1986	83937
Tandem's Software Support Plan	R. Baker, D. McEvoy	TSR	3,1	March 1987	83939
Terminal Connection Alternatives for Tandem Systems	J. Simonds	TSR	5,1	April 1989	18662
The Performance Characteristics of Tandem NonStop Systems	J. Day	TJ	1,1	Fall 1983	83930
The Role of Optical Storage in Information Processing	L. Sabaroff	TSR	3,2	Aug. 1987	83940

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

TANDEM SYSTEMS REVIEW ORDER FORM

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

- ☐ If you are a Tandem customer, complete Part A of this form and send it to your Tandem representative. Your request is subject to approval.
- ☐ For other subscribers, complete Part A of the form and send it to the address below. Enclose a check or money order, payable to Tandem Computers Incorporated, for the subscription and back copies that you order. The cost is \$40 for a one-year subscription and \$15 for each back issue.

Part A. To be completed by the subscriber.

Subscription Information

- ☐ New subscription
- ☐ Subscription renewal
- ☐ Update to subscription information
- Subscription number: _____

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

COMPANY

NAME

JOB TITLE

DIVISION

ADDRESS

COUNTRY

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

Title or position:

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

Your association with Tandem:

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

Back Order Requests

Number
of copies

Tandem Systems Review

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

Tandem Journal

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

Tandem Application Monographs

- _____ *Developing TMF-Protected Application Software*
March 1983
- _____ *Designing a Tandem Word Processor Interface*
March 1983
- _____ *Application Database Design in a Tandem Environment*, Aug. 1983
- _____ *Capacity Planning for Tandem Computer Systems*
Oct. 1984
- _____ *Sociable Systems: A Look at the Tandem Corporate Network*, May 1985

Tandem customers should send this form to their Tandem representative.

Other subscribers send this form to:

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

Part B. To be completed by the Tandem representative.

Subscription Processing

Please complete this portion of the form to approve your customer's subscription. Your department will be charged \$40 per year per subscription. Incomplete requests will be returned for resubmittal.

Back Order Processing

If your customer requests back issues, you must order them through Courier. The menu sequence is:

Marketing Information → Literature Orders → Tandem Systems Review → Back Orders

Your department will be charged \$15 for each back issue.

NAME

TITLE

DEPARTMENT NUMBER

LOC

TELEPHONE NUMBER

CUSTOMER NUMBER

SYSTEM NUMBER

SIGNATURE

Send completed approvals to:

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

Process back order requests through Courier.

For our tracking purposes, please indicate the
date you submitted the back order request:

Comments:

TANDEM SYSTEMS REVIEW CUSTOMER SURVEY

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

1. How useful is each article in this issue?

Instrumenting Applications for Effective Event Management

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

Writing Rules for Automated Operations

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

RDF: An Overview

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

Capacity Planning With TCM

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

Dial-In Security Considerations

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

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

- 21 ☐ Overview discussions of new products and enhancements. 22 ☐ Performance and tuning information.
23 ☐ High-level overviews on Tandem's approach to solutions. 24 ☐ Application design and customer profiles.
25 ☐ Technical discussions of product internals.
26 ☐ Other _____

3. Your title or position:

- 27 ☐ President, VP, Director 28 ☐ Systems analyst 29 ☐ System operator
30 ☐ MIS manager 31 ☐ Software developer 32 ☐ End user
33 ☐ Other _____

4. Your association with Tandem:

- 34 ☐ Tandem customer 35 ☐ Tandem employee 36 ☐ Third-party vendor 37 ☐ Consultant
38 ☐ Other _____

5. Comments

NAME _____

COMPANY NAME _____

ADDRESS _____

► FOLD



► FOLD

BUSINESS REPLY MAIL

FIRST CLASS

PERMIT NO. 482

CUPERTINO, CA. U.S.A.

POSTAGE WILL BE PAID BY ADDRESSEE

TANDEM SYSTEMS REVIEW

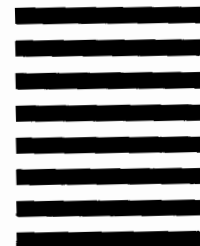
LOC 216-05

TANDEM COMPUTERS INCORPORATED

19333 VALLCO PARKWAY

CUPERTINO, CA 95014-9862

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES



► FOLD

► FOLD

