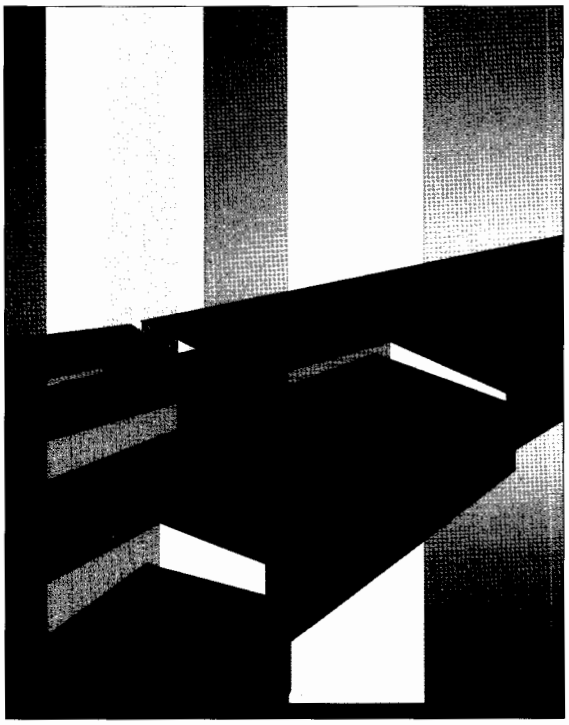
T A N D E M

# SYSTEMS REVIEW

VOLUME 10, NUMBER 3 JULY 1994



*NonStop SQL/MP Overview*

*Hash Joins • NonStop TM/MP*

*Database Configuration Operations*

*NonStop ODBC Server*

*RDF Enhancements*

*Product Update*

T A N D E M

# SYSTEMS REVIEW

VOLUME 10, NUMBER 3

JULY 1994

*The* Tandem Systems Review *publishes technical information about Tandem software releases and products. Its purpose is to help programmers, analysts, and other IS professionals to plan for, install, use, and tune Tandem systems.*

TANDEM

## Editor's Note

The six articles in this Tandem Systems Review focus on products that support major Tandem initiatives: NonStop Availability (NSA), Decision Support Systems, National Language Support, and Open. The first three articles describe the new NonStop SQL/Massively Parallel (SQL/MP) relational database management system. "An Overview of NonStop SQL/MP" (Ho, Jain, Troisi) describes the many performance, availability, and manageability enhancements and new internationalization features. "NonStop Availability and Database Configuration Operations" (Troisi) explains how the enhanced database configuration features in NonStop SQL/MP improve application availability. "A New Hash-Based Join Algorithm in NonStop SQL/MP" (Zeller) compares the hash join algorithm with other types of join strategies and discusses the performance benefits of hash joins.

"NonStop ODBC Server" (Mahbod, Slutz) describes the NonStop ODBC Server, which provides open, transparent access to NonStop SQL/MP databases for applications written to the Microsoft ODBC or Microsoft/Sybase DBLIB API.

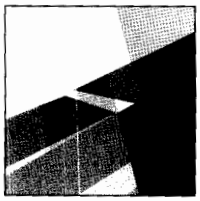
The final articles highlight NSA enhancements in two Tandem products. "Enhancing Availability, Manageability, and Performance With NonStop TM/MP" (Eicher) describes the new features and advantages the NonStop TM/MP product offers over its predecessor, TMF. "RDF Enhancements for High Availability and Performance" (Mosher) describes recent and forthcoming enhancements in the Remote Duplicate Database Facility (RDF) product and provides an updated procedure for a planned switchover.

*–AL*



*Tandem Systems Review* is published quarterly by Tandem Computers Incorporated. All correspondence and subscriptions should be addressed to *Tandem Systems Review*, 10400 Ridgeview Court, Loc 208-65, Cupertino, CA 95014.

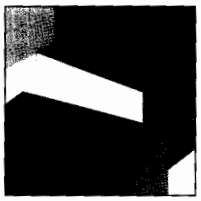
Subscriptions: $75.00 per year; single copies are $20.00. Detailed subscription information is provided on the subscription order form at the end of this book.



NONSTOP SQL/MP



OPEN ACCESS

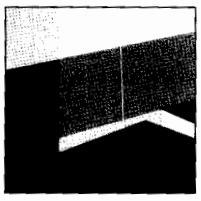




AVAILABILITY



DEPARTMENTS



# Product Update

## Systems Products

### Tandem Himalaya Intelligent Network Server Family

*March 1994*

The Himalaya Intelligent Network Server (INS) family of products assists telecommunications providers in offering intelligent network (IN) services. For global interoperability, it supports the Bellcore AIN, ETSI INAP, and CCITT CS-x call models. A complete Himalaya INS system forms an end point in an intelligent network, providing enhanced telephony and switching services such as calling-card validation, customized number translation service, and custom local area signaling services (CLASS).

The Himalaya INS family includes the following products:

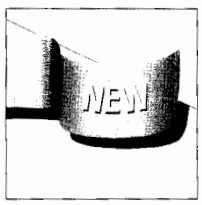
- INS Core software. This system-level software is distributed across Tandem NonStop Himalaya servers, an INS Communications Server, and a Tandem Craft workstation. It provides the software foundation for an online, real-time, fault-tolerant call processor that can meet IN needs ranging from laboratory trials to global telecommunications networks.
- INS/CS Communications Server. This SS7 communications server provides a high-speed interface to call-processing applications residing on a NonStop Himalaya server.
- Flexible Service Logic software. This product creates a data-driven service-logic execution environment based on the Flexible Service Logic standard.1
- TSCE-2000 software. This product provides an open, standards-based development environment for the rapid deployment of new IN services.
- Tandem Craft workstation. This workstation provides a graphical interface for viewing the entire Himalaya INS system and the user's SS7 network.

### TorusNet Interprocessor Connections

*March 1994*

TorusNet connections provide high-speed communications between processors through efficient communications units, the use of high-speed fiber-optic media, and low message latency. TorusNet high-speed fiber-optic connections make it possible to efficiently interconnect from 2 to more than 4,000 processors in a Tandem Himalaya K10000 server. Processors interconnected with TorusNet form a single, parallel system capable of running the most powerful online transaction processing (OLTP), decision support, and messaging applications at peak performance.

Designed to provide flexible configuration options, TorusNet connections allow processors and resources to be located for optimal application performance. They also connect to Tandem's existing FOXII fiber networks. Like the Tandem servers, TorusNet connections offer extreme reliability; each connection contains multiple fibers each of which can take over for another fiber if one should fail.




*The Product Update department provides brief descriptions of new products announced by Tandem. For more information on any of these products, please consult your local Tandem representative.*

## NonStop Kernel Based Software

### NonStop SQL/MP

*March 1994*

NonStop SQL/MP is the next generation of Tandem's NonStop SQL relational database management system. Using the same robust architecture as NonStop SQL and adding parallel database technology, NonStop SQL/MP increases scalability and performance for decision support applications as well as for online transaction processing and messaging. NonStop SQL/MP incorporates many new features to facilitate decision support applications, higher availability, and use of national languages.

The new decision support features include new table join methodologies, accelerated groupings, aggregates in the data access manager, parallel read-ahead, early evaluation of index predicates, improved expression evaluation, forced plan options, improved sequential insert performance, improved predicate selectivity computation, CAST function, faster sorts, and improved accuracy of table statistics.

New availability features include online physical database reconfiguration, lost-partition recovery, late bind, high-availability compilations, and object versioning. New national language support features include single-byte and double-byte character sets, collations support, and text translation facility.

### NonStop Transaction Manager/MP

*March 1994*

NonStop Transaction Manager/MP (TM/MP) provides transaction protection and database consistency in demanding online transaction processing (OLTP) and decision support environments. NonStop TM/MP replaces its predecessor, Tandem's Transaction Monitoring Facility (TMF). Like TMF, NonStop TM/MP provides full protection for transactions accessing distributed Tandem SQL and Enscribe databases, as well as recovery capabilities for transactions, online disk volumes, and entire databases. In addition, NonStop TM/MP adds new architectural and performance features that give it outstanding scalability, availability, and manageability, as well as the capability to support open transactions, as defined by X/Open.

## Client/Server Computing Products

### NonStop ODBC Server

*March 1994*

NonStop ODBC Server software allows users of desktop computers to access Tandem's high-performance, massively parallel NonStop SQL/MP relational database management system. The efficient client/server architecture of NonStop ODBC Server enables users to take full advantage of workstations and Tandem NonStop servers for decision support and online transaction processing (OLTP) applications.

NonStop ODBC Server allows client applications that use either the Microsoft Open Database Connectivity (ODBC) interface or the Microsoft/Sybase SQL Server interface to access databases controlled by NonStop SQL/MP. In addition, NonStop ODBC Server enables Tandem systems to act as transaction servers as well as database servers. Using stored procedures, application tasks can be split between the workstation and the NonStop server. This gives applications greater speed and efficiency in communications between workstations and NonStop SQL/MP databases.

## NonStop-UX Based Software

### Integrity Systems Manager for SGI

*March 1994*

The Integrity Systems Manager (ISM) for SGI is a new addition to Tandem's Integrity System Management Suite (ISMS). The new ISM for SGI software allows the use of Indy™ and Indigo$^{2}$™ workstations for system management of a network of Integrity FT systems. The full-featured Integrity Systems Manager for SGI combines with the price-performance leadership and advanced graphics of the Indy and Indigo$^{2}$ workstations to offer Tandem users exceptional performance and flexibility in managing networks of Integrity FT systems.

The ISM for SGI includes these important features: Administrator's Workbench; log file browser; document reader; physical, schematic, and power views; shell tool; and System Activity Monitor. The System Activity Monitor (SAM), a vital feature of the ISM software, provides a powerful graphical user interface to display Integrity FT system performance metrics. The SAM allows the system administrator to monitor system activities, set thresholds for activity values, and define actions triggered when a value exceeds a threshold.

## Communications and Networking Products

### CA-SESMAN (Secured Session Manager) from Tandem

*April 1994*

CA-SESMAN provides a low-cost software architecture offering reliability, scalable growth, comprehensive network access security, powerful network session management services, and seamless network access to legacy applications (6530 and 3270) over a variety of protocols. CA-SESMAN offers many features for Tandem and IBM host application access, including:

- Front-end security gateway support for Himalaya and IBM host-based applications.
- Single sign-on technology, single password and ID for multiple application access (Pathway, FUP, TACL, Transfer, CICS, TSO, etc.).
- Single system image, seamless network access using Tandem standard communications protocols (TCP/IP, SNA, X.25)
- Hot-key switching and automatic application scripting/network navigation.
- Complement to Tandem Safeguard and Atalla security products.
- Dynamic menu of all authorized applications.

## Application Development Software

### Tandem Service Creation Environment (TSCE-2000)

*March 1994*

Tandem TSCE-2000 software lets users develop applications for use on Himalaya Intelligent Network Servers (INS) or open intelligent network (IN) platforms from other vendors. Residing on UNIX workstations, the product provides a completely self-contained service creation environment. Optional modules let users simulate the performance of their applications on the target execution platform. No interaction with the execution platform is required until the finished application is compiled.

TSCE-2000 software is particularly well suited for developing applications for Himalaya INS systems, because it can directly call APIs on those systems. This capability gives it dynamic and efficient access to the SS7 protocol, TCAP, and Tandem's upcoming Flexible Service Logic (FSL) software for Himalaya INS systems.1. The product can be used with both the Guardian and UNIX personalities on Himalaya INS systems.2. By bridging the NonStop Kernel, different server personalities, and the UNIX development environment, it integrates heterogeneous elements of the user's intelligent network.

## Workstation and Terminal Products

### AST PowerExec 4/33SL Notebook Computers

*March 1994*

The AST PowerExec 4/33SL models are the latest additions to the Tandem notebook product line. The two new models feature a 33MHz, Intel 486SL processor, 200-megabyte or 340-megabyte removable hard drive, 4 megabytes of standard RAM on the 200-megabyte model, 8 megabytes of standard RAM on the 340-megabyte model (both upgradable to 32 megabytes), 3.5-inch diskette drive, 2 PCMCIA expansion slots, and a vivid active-matrix color display. Other features include a battery life of up to 6.5 hours, two levels of password protection, and an individual SmartKey intelligent ROM key that lets the user in if the user forgets the password.

### High-Capacity SCSI Disk Drive

*March 1994*

Tandem now offers a new high-capacity SCSI disk drive that can greatly increase the internal data storage capacity of PSX and NDX workstations. The new drive has a capacity of 2 gigabytes and is available for use with all PSX and NDX systems.

### PSX LP 4/50s Desktop Computer

*June 1994*

The PSX LP 4/50s desktop computer is a new addition to the PSX LP product line. This model is housed in a low-profile chassis and is based on the new Intel 486SX2-50 processor. On average, this new chip outperforms the Intel DX-33 by 22 percent and it costs less.

Like the other PSX LP models, the LP 4/50s features an energy-efficient design that exceeds the guidelines established for the Energy Star power management program. Additional features of the LP 4/50s include 32-bit local bus graphics, 512 kilobytes of standard graphics memory (support for up to 2 megabytes), CPU upgradability to Pentium Overdrive processors, support for up to 256 kilobytes cache, two 16-bit ISA expansion slots, and extensive security features to protect system components and data.

### Pentium Upgrade Modules

*June 1994*

Users can now step up to Pentium operation with the new Pentium P/60 upgrade modules. These modules can be used to update any NDX ST tower server (CUPID or Intel architecture), or any PSX SP desktop system.

### Megahertz PCMCIA Data/Fax Modems

*June 1994*

Tandem now offers a new line of data/fax modems from Megahertz. The new modems combine PCMCIA standards with Megahertz's patented XJACK™ connectors for credit-card-sized modems that can be hooked directly to a telephone jack. The simple and practical XJACK connectors feature a retractable RJ11 tray that pops out for use and easily pops back in when not in use, saving valuable space and eliminating the need to purchase or carry external connectors.

All new Megahertz modems use the Hayes AT command set, as well as Bell and CCITT standards and protocols to provide worldwide compatibility. In addition, the new modems incorporate the V.42bis protocol, which produces a data-compression ratio of up to 4:1 to enhance the data-transmission speed and thus reduce the amount of time required on telephone lines.

### New IDE Hard Drives

*June 1994*

Two new IDE hard drives, with a storage capacity of 270 megabytes and 540 megabytes, are now available for all PSX and NDX platforms. The new drives replace the previously offered 200-megabyte and 520-megabyte IDE drives, respectively.

### 32-Megabyte Memory Upgrade Kit

*June 1994*

A new 32-megabyte memory option is now available for the PSX SP P/60 computers. With the addition of this upgrade kit, the maximum memory capacity of the SP P/60 increases from 32 megabytes to 64 megabytes.

## Printer Products

### 5577-3 Laser Printer

*March 1994*

The 5577-3 and 5577-3PC laser printers, which supersede the 5577 and 5577-2 laser printers, are 17-ppm desktop printers designed for distributed/networked printing applications for print volumes of up to 50,000 pages per month. Both printers feature 600-dpi resolution, which combined with Resolution Enhancement technology and micro-fine toner, yields truly outstanding print quality. The 5577-3 printer family supports industry-standard PCL 5E and prints complex documents faster with the use of an Intel 80960CF 25MHz RISC processor. It also supports Adobe PostScript Level 2, and automatically switches between the PCL and PostScript languages.

The new printers are compatible with virtually all popular UNIX and PC-based application packages. The printers come with 2 megabytes of standard memory, expandable to 32 megabytes. They have a total of 45 internal scalable typefaces in both Intellifont and TrueType formats. A variety of font cartridges is also available.

The 5577-3 includes Tandem's unique T-TAP (Transparent Tandem Asynchronous Protocol) MIO interface to ensure that printing is completed correctly, even if there is a power outage or the printer runs out of paper. This printer is compatible with all NonStop servers and can also be connected to Integrity systems, PCs, and LANs. The 5577-3PC is identical to the 5577-3 except that it does not include the T-TAP MIO card and is thus for use in applications not including a NonStop server. Each printer has two standard 500-sheet cassettes. Duplex (both sides) printing is an option.

## Availability Enhancement Products

### Enhanced Availability Kit

*March 1994*

The Enhanced Availability Kit is a hardware upgrade option that provides the capability to have a fully redundant cabinet power system. It is designed for installations where single power sources have proved inadequate in supplying continuous and steady power levels or where operating conditions could allow a power source to become disconnected. It is also designed for users whose availability requirements demand that hardware upgrades be done online. This availability feature, which has been included in Tandem's Himalaya K10000 packages, is now available for other Tandem systems with similar architecture.

The upgrade allows a Tandem system to connect to redundant ac power sources through the use of dual power distribution units and dual power cords. This hardware upgrade replaces a single power distribution unit (PDU) in each cabinet with two PDUs per cabinet. Each PDU powers half of the system components in the cabinet and contains one of the two main breaker switches used for system power-up and power-down operations. Dual power cords allow the system to be connected to redundant power sources. Losing one ac power source or disconnecting one of the power cords from any outlet removes power to only one of the processors, its memory, and half the disks, while the other processor, disks, and the I/O remain operational, thus preserving fault tolerance.

# An Overview of NonStop SQL/MP

The Tandem™ NonStop™ SQL/Massively Parallel (SQL/MP) relational database management system introduces many new features and enhancements that support three major Tandem initiatives: Decision Support Systems (DSS), NonStop Availability (NSA), and National Language Support (NLS).

In a typical DSS application, ad hoc queries execute against a database much larger than most operational or online transaction processing (OLTP) databases. NonStop SQL/MP provides several features that significantly improve the performance of query processing for such large databases. This makes it both feasible and cost-effective to implement DSS applications on Tandem systems.

While offering high performance for query processing, NonStop SQL/MP also adds features for managing and increasing the availability of large database applications. These NSA features facilitate the development and management of applications by reducing downtime caused by database reconfigurations and by minimizing SQL compilations.

As part of Tandem's company-wide NLS initiative, NonStop SQL/MP supports multibyte character data types, alternate collating sequences, and the display of output messages in languages other than English.

Part 1 of this article discusses the new NonStop SQL/MP features that support the DSS initiative. Part 2 describes the NSA features and Part 3 introduces the NLS features. The article points the reader to other articles in this issue of the *Tandem Systems Review* that give detailed information on many of these features.

## Part 1: Support for DSS

In 1993 Tandem forged an initiative to build massively parallel systems, geared toward decision support applications. The hardware foundation for the initiative is the Tandem NonStop Himalaya™ K10000 server. In a recent Transaction Processing Performance Council TPC-C benchmark, a 64-processor Himalaya K10000 server executed 12,021 transactions per minute (tpmC). Tandem's TorusNet™ technology facilitates massive scalability for Himalaya servers, connecting up to 224 processors through fiber optics and up to 4,080 processors through LAN, WAN, and other connection options.

In addition, Tandem's NonStop ODBC Server product gives client tools access to the large databases required for decision support applications. With the NonStop ODBC Server, client tools using the ODBC or the Microsoft/Sybase SQL Server interface can access NonStop SQL databases. Moreover, enhanced capabilities such as query caching allow one to use the NonStop ODBC Server for low-volume OLTP as well as query processing.

The database is at the heart of a decision support or online query processing (OLQP) system. NonStop SQL/MP contains all of the features and benefits provided by previous releases of NonStop SQL: parallelism in queries and utilities, high availability, near-linear speed-up and scale-up, high performance, data integrity, and transparent data distribution. In addition, NonStop SQL/MP significantly improves the performance of query processing against massive decision support databases.

## Performance Considerations for Decision Support

Query processing for decision support differs from OLTP both in the size and definition of the database and in the methods of accessing the database. The decision support database usually dwarfs the operational or OLTP databases. Because of the size of the DSS database, mirrored disk volumes may not be used even though they provide higher availability. In addition, the database is historical in nature; the stored data has a time dimension.

One can have more indexes for a DSS database than for an OLTP database because their detrimental impact on updates is not felt. Often, updates are either performed in batch processing on a periodic basis or trickle-fed from the operational databases. Response time on updates is not as essential for DSS as for OLTP.

If most of the queries are ad hoc in nature, the database is implemented fairly close to its logical third through fifth normal forms. That is, little denormalization may be possible. If queries are predictable, one can perform denormalization in the DSS environment with less penalty than in an OLTP environment because of the lower priority of updates.

Database access also differs in the DSS and OLTP environments. Usually, in the case of the DSS database, a large number of rows must be scanned to deliver the results of a query. Because of the number of rows scanned, and since linear speed-up and scale-up of queries is an inherent objective, queries almost always need to be processed in parallel.

Most queries have joins (often between a larger table and smaller reference tables), require aggregation of data by groups, and require the results in a specific ordering sequence. Because of the lack of adequate indexes on large tables, the need for grouping and ordering typically means that a sort may be involved.

## Performance Features for Decision Support

To address some of the performance considerations described above, NonStop SQL/MP introduces the following new features and enhancements. Some of these improvements can benefit OLTP as well as DSS applications:

- Hash joins.
- Hashed groupings.
- Aggregation at the disk-process level.
- Early evaluation of index predicates.
- User process sort.
- Cross product.
- Forcing an access plan.
- Update statistics enhancements.
- Sequential block split control option.
- FASTSORT performance improvements.
- Performance improvements for table and index creation.

## Hash Joins

Often, in DSS, the historical data resides in a single large table. For most queries, the large table must be joined with smaller reference (*lookup*) tables. Because of the size of the table, denormalization is prohibitive. In many cases indexes are not affordable on large DSS tables or may not be used by the optimizer. The join strategies in previous releases of NonStop SQL are nested joins and sort-merge joins. Now a hash join strategy is available as well.

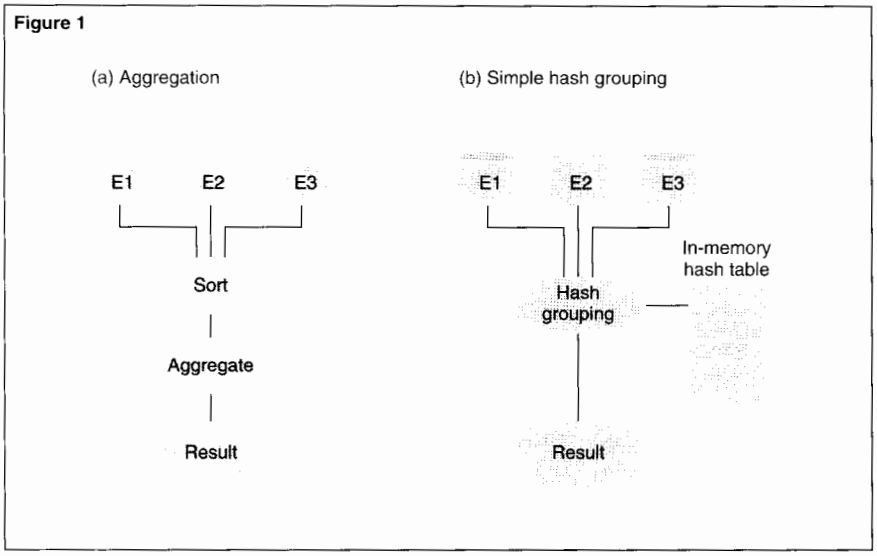



**Figure 1.**
*An example of a simple hash grouping. (a) Aggregation performed in previous releases of NonStop SQL. (b) Hash grouping performed by NonStop SQL/MP.*

***Simple Hash Join.*** In previous releases of NonStop SQL, the nested join was typically the most efficient of the join strategies. If there was an index on the joining columns for the inner table of the join, a nested join was performed.

The hash join improves on the nested join by reading the required columns for the qualifying rows of the inner table and building a hash table in memory (by hashing on the join columns). The hashing of these columns may sometimes result in the same hash key. In these situations the rows are chained into a linked list. When an inner-table row needs to be accessed for the qualifying outer-table row, the hash table is probed to perform the join. This quick-memory access improves performance considerably compared to sending messages to the disk process for each qualifying row of the outer table.

The performance savings provided by the hash join are even greater for sort-merge joins. Sort-merge joins are needed when there are no indexes on the joining columns on either table. The two tables are sorted on the joining columns and then a merge join is performed. Hash joins eliminate the need for sorting by loading the inner table, as mentioned above, into an in-memory hash table. Then, essentially, a nested join is performed. Especially when combinations of larger and smaller tables are involved, this technique can save substantially in the space and time required for sorting.

The NonStop SQL/MP optimizer chooses a *simple hash join* as a hash join strategy when it predicts that the inner (smaller) table can be accommodated in memory.

***Hybrid Hash Join.*** When the NonStop SQL/MP optimizer predicts that the inner table may not fit in memory, it chooses a *hybrid hash join* designed to handle overflow. The inner table is first divided into buckets (or sections). For each bucket, an in-memory hash table is built. For the buckets that fit comfortably into memory, the rows in the outer table are accessed once and a simple hash join is performed, as before.

However, if at run time there is not enough memory to perform the join, and the SQL executor determines that many page faults are occurring, some buckets would have to be written to disk. In this situation, as the outer table is read, some rows would not find a match in the memory-resident buckets. (Their matches may reside on buckets written out to disk.)

To handle this situation, the needed columns of the qualifying outer table rows are read and written out to bucket files on disk as well. The SQL executor writes out the outer table buckets using the same bucket-partitioning scheme it uses for the inner table buckets. After a single scan of the outer table has been completed, the buckets on disk are read and the join peformed. The resulting rows may not be in any particular order. If a specific ordering sequence is desired, the executor has to sort the resulting set. Although the hybrid hash join mechanism is complex, it is far more efficient than the traditional strategy, because of the size of the tables involved and because it avoids having to presort the qualifying rows from both tables in the case of a merge join.

NonStop SQL/MP also considers parallel and repartitioned parallel hash join strategies. Zeller (1994) discusses hash joins in more detail elsewhere in this issue of the *Tandem Systems Review*.

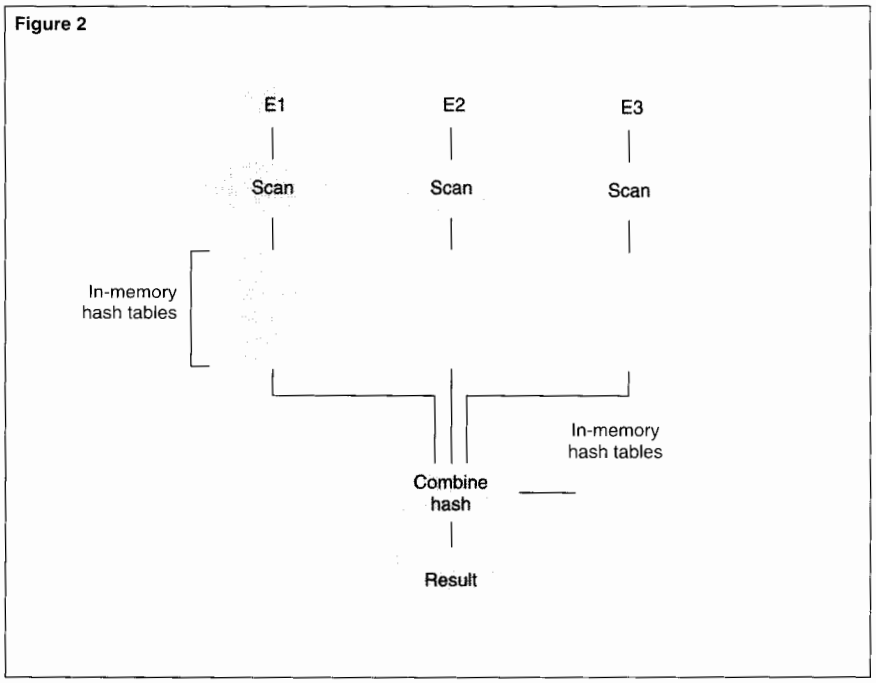
## Hash Groupings

Often, when group aggregations such as AVG, COUNT, MAX, MIN, and SUM are requested, the table does not have an index on the columns by which the aggregation is being grouped. To get the data into the right order so that the group aggregates can be calculated, previous releases of NonStop SQL must perform an expensive sort operation. When the rows to be sorted are in the millions, the cost of the sort is accentuated. To alleviate the sort, NonStop SQL/MP performs hash groupings.

***Simple Hash Groupings.*** Essentially, NonStop SQL/MP builds a hash table in memory by hashing on the GROUP BY columns. The hashing of these columns may sometimes result in the same hash key. In these situations, the rows are chained into a linked list. As the rows are read, the SQL executor aggregates the columns into the appropriate buckets for the resulting hashed key entry. When it is done scanning all the rows, if an ORDER BY is specified, it performs a sort on the hash table to deliver the results in the desired sequence. This operation is called a *simple hash grouping*. Figure 1 shows an example of a traditional aggregation using a sort operation and a simple hash grouping as executed by NonStop SQL/MP.

***Hybrid Hash Groupings.*** When the entire hash table cannot fit in memory (indicated at run time by the occurrence of too many page faults), NonStop SQL/MP chooses a *hybrid hash grouping* strategy to handle the overflow. Here, when the in-memory hash table is full and a row is read that yields a hash key not in the hash table, the raw column data required for the aggregation is written out to disk. At the end of the scan, the rows written out to disk are read and aggregated.

Figure 2




***Parallel Hash Groupings.*** The parallel hash-grouping scenario is similar to the sequential one discussed above. Each executor server process (ESP) does its own hash grouping. Each ESP then passes its results back to the master executor, which combines the results from the various ESPs. Figure 2 shows an example of parallel hash grouping as executed by NonStop SQL/MP.

For parallel plans, hash grouping replaces the repartitioned GROUP BY method. No repartitioning is necessary. Hash groupings work with hash joins as well.

**Figure 2.**
*An example of a parallel hash grouping in NonStop SQL/MP.*

## Aggregation at the Disk-Process Level

If there is no index on the GROUP BY columns to support aggregations, NonStop SQL/MP considers a hash grouping strategy. If, however, there is an index on the GROUP BY columns, the disk process (DP2) performs the aggregations and sends back only the resulting rows. In earlier implementations, NonStop SQL would send the rows back to the executor to perform the aggregations. To accomplish this task, DP2 and the executor would exchange data and messages through the file system, which incurred additional performance overhead. Aggregation at the disk-process level, provided by NonStop SQL/MP, essentially eliminates this overhead.

However, GROUP BY aggregations are not the only aggregations performed at the disk-process level. An even greater performance enhancement can be obtained for single-table aggregations without a GROUP BY clause such as COUNT (*). Also, for joins without a GROUP BY clause, if all aggregations are performed on the innermost table of the join, they are pushed down to the disk process as well.

## Early Evaluation of Index Predicates

In previous releases of NonStop SQL, predicates on nonpositioning columns of an alternate index were evaluated against base-table columns if a base-table predicate existed as well. In NonStop SQL/MP, such predicates are evaluated against the index before base-table predicates are evaluated. This technique can considerably reduce the number of random accesses made to the base table to eliminate undesired rows. The performance improvement increases as the selectivity of the index predicates and the size of the table increases.

Assume, for example, that a table T contains columns PK1, PK2, C1, C2, C3, and C4 with an alternate index on C1, C2, and C3. Then the query

```
SELECT columns
  FROM T
  WHERE C1 = ?c1
    AND C3 = ?c3
    AND C4 = ?c4;
```

would position on column C1 of the index. However, instead of evaluating C3 = ?c3 against the index column, earlier implementations of NonStop SQL would first access the corresponding row from the base table and evaluate both C3 = ?c3 and C4 = ?c4 against it. This resulted in unnecessary accesses to the base table for index rows not meeting the C3 = ?c3 criteria. In NonStop SQL/MP, such predicates are evaluated on the index.

## User Process Sort

Previous releases of NonStop SQL provided two sort options. When many rows were to be sorted, the rows were passed to SORTPROG, an external sort process. Parallel sorting techniques could be used even if the query itself was not being executed in parallel. However, when the number of rows to be sorted was small, a key-sequenced sort was performed. (A sort process was not used.) The SQL executor created a temporary key-sequenced table (buffered and nonaudited) with a key matching the sort sequence. It then inserted the rows to be sorted into this table and read them back out. For the most part, NonStop SQL/MP replaces this technique with a user process sort (UPS), an in-memory sort for fewer than 32,768 rows.

If the optimizer plans to use the UPS process, but the number of rows at run time exceeds 32,767 or the sort workspace is exhausted, UPS gracefully migrates to the SORTPROG process. It passes the data in the workspace to SORTPROG in 8-kilobyte segments, while accepting new rows from the executor. It passes the new rows to SORTPROG in 8-kilobyte segments as well.

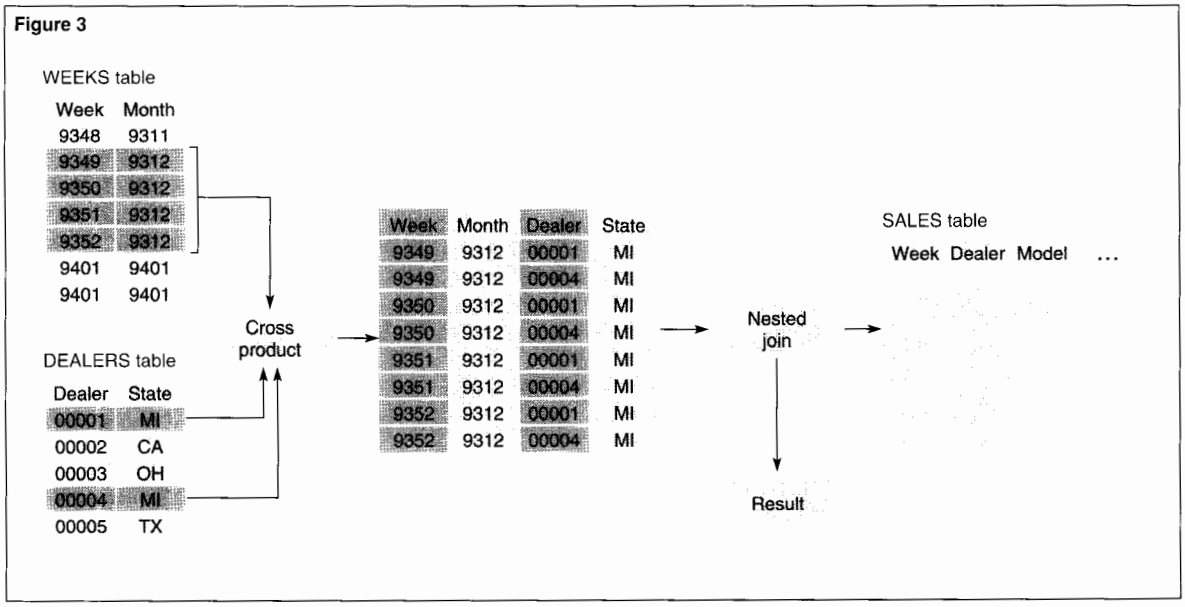



**Figure 3.**
*An example of the NonStop SQL/MP cross product feature.*

## Cross Product

A cross product (or Cartesian product) is the product of two tables, or a join of two tables without a join column being specified, so that each row of one table is joined with each row in the other.

An example would better explain how this concept is used to enhance performance of queries. Assume there is a large historical automobile SALES table. Assume further that the WEEK (year and week number), DEALER, and MODEL columns make up its primary index. Each row has automobile sales data for a specific week for a car model sold in a specific dealership. Suppose a query is submitted that requires all sales in dealerships in Michigan for the month of December 1993. Because the query specifies a range of weeks and WEEK is the leftmost column in the index, all dealers and car models for that range of weeks will be scanned. The ideal way to avoid this excessive scanning would be to perform multiple parallel scans against the SALES table for each of the weeks in December 1993 for only Michigan dealerships.

Now assume that there are two more tables. One table, WEEKS, represents all the weeks for which there is sales information. This table would have a YEAR_MONTH column to facilitate access by month. A second table, DEALERS, represents all the dealerships in the country. The cross product feature in NonStop SQL/MP would facilitate a cross product of the qualifying rows from the WEEKS and DEALERS tables. (These tables do not have a common join column.) The composite table created from this join would then be joined against the SALES table to reduce the excessive scanning. Figure 3 shows an example of the cross product feature.

Cross product is especially beneficial when a query is processed in parallel. In the example, the WEEKS and SALES tables could be partitioned by WEEK. Parallel processes would then perform the joins for each qualifying week. The NonStop SQL/MP optimizer automatically determines whether to perform the cross product operation between these two tables.

## Forcing an Access Plan

New options in the CONTROL TABLE command allow a user to force NonStop SQL/MP to choose a particular access path, join sequence, or join method for a query. The access path indicates to the optimizer that either the primary index or a specific alternate index is to be used. The join method indicates that a nested, merge, or hash join be used. The join sequence indicates the sequence in which the tables should be joined.

In rare circumstances, usually because of nonuniform distribution of data, the optimizer may not choose the best plan for a query. Some users believe this happens fairly often. Experience in tuning many user applications has shown that the problem usually lies in the database design, the construction of the query, or the statistics in the catalog being used for the compilations.

The plan-forcing feature should be used with extreme caution. When users find that the optimizer is being stubborn about a specific query, they should consult with a Tandem NonStop SQL expert before forcing the plan, or report the issue for consideration (if it is not related to skewed data distribution).

Users can prove that the forced plan is better than the one chosen by the optimizer by running the query with both plan options against a production-size database. Measurements taken in both situations can substantiate the reasons for the decision to force a plan. These reasons must be weighed against the fact that the database and the DBMS are constantly changing. A statement using this feature can be embedded in a program permanently and may cause some of the following problems later on:

- The program may not be able to use a more efficient index that may be created in the future.
- If the index being used is dropped, the program will need to be changed and recompiled.
- The program may not be able to benefit from future NonStop SQL/MP enhancements.

## Update Statistics Enhancements

Several enhancements in NonStop SQL/MP have greatly improved the accuracy of statistics.

The EXACT and SAMPLE *n* BLOCKS options were added to the UPDATE STATISTICS command so that users can balance the accuracy of the statistics against the time required to update statistics on large tables. If sampling results in statistics that will generate the same query plans as would be generated if one gathered exact statistics, one can save substantial time by updating statistics with the sampling option.

To determine an appropriate sample size, one can use a trial and error method with different sample sizes. One can compare statistics from each sample with those generated by using the EXACT option. A sample size resulting in statistics with acceptable variances can be used for the table from then on. Obviously, it would not be acceptable to use a sample size that results in plans different from those generated when the EXACT option is used.

If neither the EXACT nor the SAMPLE *n* BLOCKS option is used, the default is that all rows are read to gather statistics for partitions containing 1,000 blocks or less. (The default block size is 4 kilobytes.) When the size of a partition exceeds 1,000 blocks, an average of 500 blocks is read per partition to gather the statistics. If the table is not partitioned, all rows up to 1,000 blocks are read.

Other statistics-related enhancements in NonStop SQL/MP include the following:

- Column statistics (such as UNIQUEENTRYCOUNT, SECONDHIGHVALUE, SECONDLOWVALUE) are collected on the whole table and not its individual partitions.
- The null value is ignored when determining the SECONDHIGHVALUE, but is still counted as a unique value.
- Special adjustments are made for the presence of null values when the UNIQUEENTRYCOUNT value is very small.

## Sequential Block Split Control Option

In many cases, a sequential block split algorithm for inserts may be more efficient than the fifty-fifty block split algorithm. Row insertions using the sequential block split algorithm result in less slack space per block and a fewer number of blocks in the table. More important, they cause less fragmentation in the file. Logically contiguous blocks tend to be physically contiguous as well. This is good for sequential scans. Noonan and Gordon (1993) explain sequential block splits and fifty-fifty block splits in *A Performance Guide to NonStop SQL.*

A new CONTROL TABLE option allows users to direct DP2 to use the sequential block split algorithm when a block split is needed. DP2 does not select this algorithm if it does not detect sequential inserts, as in the case in which there are interleaved inserts from multiple programs simultaneously.

## FASTSORT Performance Improvements

The FASTSORT component of NonStop SQL/MP has been enhanced to take advantage of the larger main memory available in new Tandem processors. This very large memory (VLM) feature removes a previous limit of 32-kilobyte records that can be sorted at one time in main memory without the use of a scratch file. The new theoretical limit is 2 gigabytes. More important, it makes the processor memory and the maximum size for an extended segment, rather than an internal software limitation, the limiting factors in the number of records to be sorted in main memory.

Moreover, for sorts that require more than a single run, the additional memory can be used to store some of the intermediate runs. This reduces the number of read and write operations to the scratch file, thus resulting in significant performance improvements.

## Performance Improvements for Table and Index Creation

NonStop SQL/MP significantly reduces the time it takes to create a table or index with a large number of partitions. NonStop SQL builds large tables by creating a table with a single partition and then iteratively adding partitions, one at a time, until the specified number of partitions is created. When previous implementations of NonStop SQL added a new partition, the file labels of all previously created partitions were altered, resulting in order(*n*-squared) file-label operations (where *n* was the number of partitions). These label operations were quite costly both in CPU and in disk-I/O wait time.

NonStop SQL/MP instead builds a partition template once that contains all the partition information provided in the Data Definition Language (DDL) statement. It then uses the template to build each partition. This technique results in only as many file-label operations as there are partitions.

The enhancement can improve performance by more than an order of magnitude. An added benefit is that the enhancement significantly reduces the audit generated during the DDL operation.

## Part 2: Improving Availability

A survey[1] of computer users found that the application downtime they experienced had a surprisingly high business cost. Although the numbers varied across companies, the average was $1,300 per minute of outage. In the past, companies had large batch-processing windows (time periods) during which it was understood that their applications would be inaccessible. During these periods, companies ran their large batch reports. They also had time to perform any database operations required to maintain their systems.

[1]See *The Impact of On-line Computer Systems Downtime on American Businesses: A Survey of Senior MIS Executives* (1992).

Pressures to provide a competitive differential, together with the move to OLTP, meant that companies had to cut back, and in certain cases eliminate, their maintenance windows. Many companies now need to provide permanent, continuous availability of applications. For this reason, they are scrutinizing database maintenance and program management operations to see if the associated downtime can be shortened or eliminated. Some typical long-running operations include the following:

- Physically reorganizing the data.
- Increasing the size of a table.
- Moving a portion of a table's data, perhaps to a larger or faster disk, to improve performance.
- Creating indexes on a table.
- Balancing the amount of data between partitions.
- Checking that access paths are unchanged after programs are recompiled.

To see how often these operations are required in a typical user application using NonStop SQL, an informal survey of 15 current Tandem users was conducted. Although one can perform many operations online with the previous releases of NonStop SQL, some operations require portions of the database to be unavailable. The survey requested information about each user's downtime and asked specifically about the database maintenance operations listed above. From this survey, outage numbers were projected for an average user.

The data collected had a large standard deviation.[2] The larger the user's database, the larger the outage the user experienced. In all cases, however, one or more of these operations made up a large part of the user's database downtime. Also, in all cases, competitive analysis has shown that these numbers are far smaller than those for competing database products. For example, physical reorganization of the database on Tandem systems caused zero minutes of outage because a Tandem utility can perform this operation online.

[2]The exact outage a user experiences yearly depends greatly on the user's environment. For example, Wood (1994) noted that database reconfiguration operations accounted for 360,000 individual end-user outage minutes for a Tandem server. This is about a third of the total user outage minutes in Wood's model.

To address the other outages listed above, three groups of features were added to NonStop SQL/MP: late binding, high availability compilation, and physical database configuration. In addition to improving availability, these features make it easier to manage both the database and the development, compilation, and installation of application programs. Two other features, not directly affecting the operations examined in the survey, are also included in NonStop SQL/MP: PURGEDATA PARTONLY and Recover Lost Partition. These features are described briefly in the following sections.

Given the outages discussed above, a user requiring permanent, continuous availability can achieve a potential savings of over $3.5 million per year[3] by reducing downtime with the new NonStop SQL/MP features. The NonStop SQL physical database configuration and late binding features provide part of these savings. Full read and write access to all partitions of a table is available during lengthy data-movement operations. Recompilations at the end of DDL operations are no longer necessary. These capabilities limit the outages associated with the Split Partition, Move Partition, Move Partition Boundary, and Create Index operations to about a minute per operation. In addition, the late binding and high availability compilation features reduce the need for recompilation and the outages associated with them. Altogether, the new features reduce the database outages associated with these operations by two orders of magnitude.

## Late Binding and High Availability Compilation

Late binding resolves a number of availability and manageability issues. First, for programs using static SQL statements, name resolution can occur at execution time. In previous releases of NonStop SQL, users who wanted a transaction to execute against one of several different tables had to use dynamic SQL. Dynamic SQL compilations, however, could affect performance. In NonStop SQL/MP, users can get run-time name resolution while using static SQL.

Second, in previous releases of NonStop SQL, DDL operations such as Split Partition invalidated programs that accessed the affected table. Users would then have to recompile the affected programs. In NonStop SQL/MP, users can specify compile options that allow them to avoid recompilation if the DDL operation does not affect the query plan.

Third, users can now install programs in a new system without having to recompile them. For example, users can move programs from a development system to a production system without recompilation; this feature also preserves the existing query plans.

The high availability compilation feature complements late binding by allowing users to recompile individual SQL statements within a program. In previous releases of NonStop SQL, users had to recompile the entire program if a query plan for one SQL statement in the program was invalid. In NonStop SQL/MP, users can selectively recompile plans.

## Physical Database Configuration Features

In previous releases of NonStop SQL, many DDL partitioning operations allowed concurrent read and write access to all partitions of the table except the affected partitions, which allowed write access only. This limitation led to outages, since most applications require full read and write access to all of the data in a table.

[3] A cost of $1,300 per outage minute multiplied by 2,640 application outage minutes per year (for an average user in the informal survey) equals $3,511,200.

As part of Tandem's NSA Initiative, the Split Partition, Move Partition, and Create Index operations have been changed.[4] In addition, a new operation, Move Partition Boundary,[5] is being added in a later release. These new operations allow full concurrent read and write access to all data in the affected table during the DDL operation. Troisi (1994) describes these techniques elsewhere in this issue of the *Tandem System Review*. Late binding features, used together with these new operations, eliminate the need for recompilations after DDL operations.

### PURGEDATA PARTONLY

Often, users keep historical information in tables. As new information is added at the end of the table, old data is deleted from the beginning of the table. Typically, one partition's worth of data is added and one partition's worth removed. To remove this data, one can use the SQL DELETE query, but this operation causes significant amounts of audit to be generated, and its performance may not be acceptable in certain circumstances. One can use other techniques such as the LOAD command with the PARTONLY and EMPTYOK options, but they require that the audit attribute of the table be turned off. User applications cannot operate against the table during such an operation.

The PURGEDATA PARTONLY feature in NonStop SQL/MP solves this problem by allowing users to purge the data in a single partition of an audited table. In this way, user applications are unaffected, yet data can be removed from a table quickly and without large amounts of audit being generated.

### Recover Lost Partition

Users often employ the BACKUP and RESTORE utilities as a way to recover from failures. One popular technique is to use the PARTONLY option of BACKUP to create a backup tape for each partition of a table. Then, if a table's partition is lost, one can recover that partition using the RESTORE utility. Previous releases of NonStop SQL required that the partition of the table that was destroyed still had to exist for the recovery to be successful. In the case of media failure, however, the partition label was unavailable and this recovery strategy did not work.

To solve this problem, the RESTORE utility has been changed to restore data even when the partition label is not present. This is the new default behavior in NonStop SQL/MP.

## Part 3: Features for Internationalization

The following internationalization features are planned for the General Availability release of NonStop SQL/MP.

### Multibyte Character Set Support

Multibyte Character Set Support (MCSS) extends the use of the existing CHARACTER data type in NonStop SQL and implements the NATIONAL CHARACTER (NCHAR) data type. These extensions enable Tandem to be compatible with the ISO/ANSI SQL92 standard, which generalizes the CHARACTER data type to support multiple character sets on the same Tandem system. These enhancements allow NonStop SQL/MP to support multibyte character sets such as KANJI (Japanese), BIG5 (Chinese), KSC5601 (Korean), and ISO88591 (European).

[4]The Split Partition operation moves a key range of the data from one partition into another new partition. The Move Partition operation moves a partition that resides on one disk to another disk. Create Index creates an index that can be used to improve performance for queries that access specific columns in the table.

[5]The Move Partition Boundary operation moves rows between adjacent partitions, typically shifting rows from nearly full partitions into less full partitions. One can use this operation to delete a partition by moving all of its rows into an existing adjacent partition.

## Collations

The collations features in NonStop SQL/MP allow character strings to be ordered in a different collating sequence than the ASCII character-code collating sequence. These features are particularly important for languages other than English, because the ASCII collating sequence does not provide the correct ordering sequence for these languages. NonStop SQL/MP provides the following additional capabilities:

- Characters can sort to the same weight (for example, sort "A" and "a" as equal).
- Characters can have a null collation (for example, D'Anza = DAnza).
- Certain characters can collate as two characters (for example, "O umlaut" = OE).
- Certain double characters can collate as one character (for example, "ch","ll","rr" in Spanish).

## Text Translation

The text translation feature in NonStop SQL/MP allows users to display all error messages, help text, and other informational messages in national languages other than English. One can set up this mechanism before the start of SQL processes or switch it during an SQLCI session.

# Conclusion

Tandem's NonStop SQL/MP provides significant improvements in parallel performance against large databases. New database configuration operations and late binding features reduce downtime for applications requiring high availability. In addition, internationalization features enhance support for languages other than English. NonStop SQL/MP, Tandem's massively parallel database management system, offers users a single-vendor solution for OLTP and decision support applications.


### References

Noonan, M. and Gordon, D. 1993. A Performance Guide to NonStop SQL. *Technical Information Series.* Vol. 2, No. 2. Tandem Computers Incorporated. Part no. 400075.

*The Impact of On-line Computer Systems Downtime on American Businesses: A Survey of Senior MIS Executives.* 1992. FIND/SVP Strategic Research Division.

Troisi, J. 1994. NonStop Availability and Database Configuration Operations. *Tandem Systems Review.* Vol. 10, No. 3. Tandem Computers Incorporated. Part no. 104400.

Wood, A. 1994. Client/Server Availability. *Tandem Systems Review.* Vol. 10, No. 2. Tandem Computers Incorporated. Part no. 104398.

Zeller, H. 1994. A New Hash-Based Join Algorithm in NonStop SQL/MP. *Tandem Systems Review.* Vol. 10, No. 3. Tandem Computers Incorporated. Part no. 104400.



### Acknowledgments

The authors would like to thank the members of the NonStop SQL Development, QA, and Publications teams, who made the NonStop SQL/MP software possible. Thanks also to Dave Liles for providing the original graphics.



**Fred Ho** is the development manager for the NonStop SQL Optimizer, Executor, and Preprocessor. He also managed the Decision Support Systems initiative for NonStop SQL/MP. Fred joined Tandem in 1985 to work on NonStop SQL Release 1. After that he worked in TMF QA and Software Engineering before rejoining the NonStop SQL group as QA manager in 1989. He has been a development manager since 1992.

**Rohit Jain** has worked for the past three years in the Professional Services group delivering application and database design and tuning services to Tandem users. He was also involved with the design and implementation of a large decision support system. Rohit joined Tandem in 1987 as a district analyst. He recently joined the NonStop SQL development group.

**Jim Troisi** is the development manager responsible for the availability features in the NonStop SQL/MP product. Since joining Tandem in 1982, Jim worked as the principal architect of the Tandem Maintenance and Diagnostic System. Since joining the NonStop SQL group in 1989, he has managed the NonStop SQL Utility area and worked to release several NonStop SQL availability and manageability features. In addition, he has been project manager for several NonStop SQL releases, including D20.

# NonStop Availability and Database Configuration Operations

Database configuration and reconfiguration operations can have a significant effect on the availability of user applications. Although most users perform these operations infrequently, their duration can account for thousands of minutes of application outages per year. The enhanced database configuration features in the Tandem™ NonStop™ SQL/Massively Parallel (SQL/MP) relational database management system eliminate most of this downtime.

The cost of application outages has been estimated to be over a thousand U.S. dollars per minute. Thus, a user with an application requiring permanent, continuous availability could save 3.5 million dollars per year by using the new NonStop SQL/MP features. A discussion of the cost of application outages appears in the article "An Overview of NonStop SQL/MP" (Ho et al., 1994) in this issue of the *Tandem Systems Review*.

This article describes the changed database-configuration features in NonStop SQL/MP. It focuses on one operation, Split Partition, in particular. The article explains each phase of the operation and shows how it maintains application availability. (The few required outages are noted, together with estimates of how long they last.) The other database configuration operations function in a similar way and have a similar effect on availability. Thus, the description of Split Partition also applies, for the most part, to the other operations.

## Physical Database Configuration Features

As part of Tandem's NonStop Availability (NSA) Initiative, the SQL/MP developers have added new options to the Split Partition, Move Partition, and Create Index commands and are adding a new command, Move Partition Boundary. One of these new options, WITH SHARED ACCESS, allows concurrent read and write transactions to execute during the operation.

The Split Partition command splits one partition into two. Move Partition moves a partition that resides on one disk to another disk. Create Index orders data according to the value specified in the key columns. As a result, queries that use these key columns run quickly.

The Move Partition Boundary command moves rows between adjacent partitions, typically shifting rows from nearly full partitions into less full partitions. One can use this command to delete a partition by moving all of its rows into an existing adjacent partition.

The new implementations of these commands improve database availability. They minimize, but do not eliminate, associated outages. Even with the improvements in NonStop SQL/MP, user database activity continues to be restricted for about one minute or less per operation. The outage time varies depending on the number of user transactions running against the table, the size of the transactions, and the number of partitions in the affected table. For example, with long-running user transactions, these commands need more downtime because they require locks that cannot be granted until all outstanding transactions against the source table have been completed. For this reason, one should run these commands at times when the user workload against the database is at a low point. Consult the documentation for NonStop SQL/MP for additional considerations.

These NonStop SQL/MP operations use a technique that requires the use of the Tandem Transaction Manager/Massively Parallel (NonStop TM/MP) audit trail.[1] When the WITH SHARED ACCESS option is specified, the operations work only on audited tables; they cannot work when the source table is unaudited. When the WITH SHARED ACCESS option is not specified, the Split Partition, Move Partition, and Create Index operations work for both audited and unaudited tables.

[1] The NonStop TM/MP product is the new version of TMF™ (Transaction Monitoring Facility) software. Chandra and Eicher (1994) describe the new features in NonStop TM/MP in this issue of the *Tandem Systems Review*.

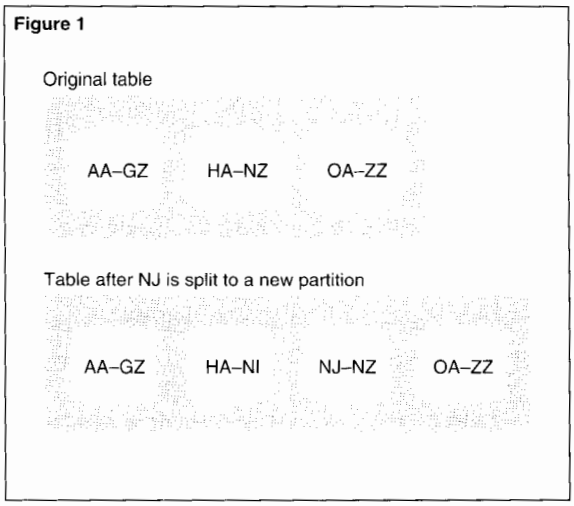
## The NSA Split Partition Operation

In NonStop SQL, partitions are used to create a large logical table from multiple smaller physical files and to enhance performance by balancing the I/O workload among multiple disk devices. Currently, a partition can grow to 2 gigabytes and a table can consist of up to 286 partitions, the exact number of partitions depending on key size.[2] One can enlarge a table by splitting a partition and moving data from the original partition into another, newly created partition, or by adding a new partition where no data movement is required. The Add Partition command, available in previous releases of NonStop SQL, accomplishes the second of these operations; it requires minimal outage times.

When splitting a partition, one decreases the size of the source partition; some percentage of the source partition's data goes to the new partition. To accomplish this, previous releases of NonStop SQL provided the Split Partition command, which allowed concurrent read access to the partition being split and read and write access to all other partitions. In contrast, the NSA Split Partition operation, available in NonStop SQL/MP, allows complete read and write access to all partitions during the split operation.

[2] The limit of 286 partitions will be raised in a future release.

**Figure 1.**
*Table partitioning before and after the Split Partition operation.*




The NSA Split Partition operation works in four phases. The phases guarantee that modifications (inserts, updates, and deletes) made to the data in the original table by user applications are present in the table after the split finishes executing.

Figure 1 shows an example of a table partition being split to accommodate growth in user data. The table's primary key is the user's U.S. state. The table has three partitions, one for states whose two-letter postal code precedes HA, one for states whose postal code is HA through NZ, and one for the states after that. Because sales in New Jersey (NJ) have far exceeded expectations, the second partition of the table is becoming full. Therefore, the user wants to split this partition, moving the entries from NJ through NZ into a new partition. Figure 1 shows the partitions in the original table and the partitions planned for the new table.

## Phase 1: Sequential Read

In the first phase, Sequential Read, of an NSA Split Partition operation, the SQL catalog manager (SQLCAT) process creates the new target partition. Since a new partition is not yet logically a part of the table's definition, user applications cannot use it.

The SQLCAT process then sequentially reads the existing (original) partition and writes the relevant portions of its data to the new target partition. (The reads begin at the location at which the data will be split.) The SQLCAT process uses browse access to perform the reads. Therefore, the data is moved without any user applications experiencing additional lock contention.

This technique, however, allows inconsistent data to be copied to the target partition. (Another name for this technique, *dirty reads*, comes from this inconsistency.) Assume, for example, that a row is copied that is part of a user transaction and the transaction is later aborted. Moreover, since other applications can continue to insert, update, and delete rows already read by the sequential-reading SQLCAT process, the data may be incomplete or even incorrect. In the second phase of this operation, the audit for the table is read to correct this situation. To prepare for the second phase, the Split Partition operation notes the current end-of-file (EOF) of the audit trail for the original partition before the sequential reads begin.

Figure 2 shows the sample table in phase 1, during which the NJ through NZ data is split off from the original partition to a new disk. To understand how availability is maintained during the operation, assume that while the sequential reader is running, two rows (containing the keys MI and NJ) are inserted into the original partition. Assume also that the new NJ row is added after the reader has already read all the NJ rows and written them to the new partition. Under these circumstances, the new NJ row is not copied to the target partition during phase 1.

During phase 1, user applications have complete read and write access to the data in all partitions of the table. Selects, updates, inserts, and deletes are executed against the original table.

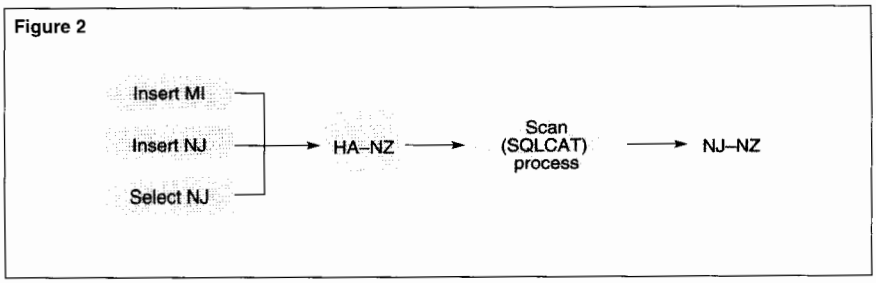
## Phase 2: Audit Fixup 1

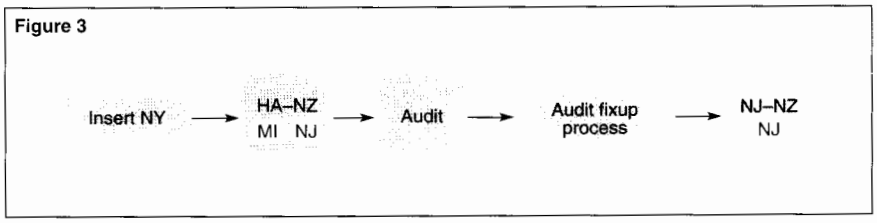
In the second phase, Audit Fixup 1, the dirty copy of the data is brought up to date with the original partition's data. This is accomplished by examining the audit trail for the original partition to find records that have changed since the dirty copy was made. In this phase, the audit-fixup process reads the audit trail, starting with the first record inserted into the audit trail after the dirty read began. From these audit records, the audit-fixup process identifies any changes (inserts, updates, or deletes) made to the data that also resides in the new partition. It then applies those changes to the data in the new partition.

This phase ends when the audit-fixup process reads the last record in the audit trail. At this point, the data in the target partition looks exactly like the data in the original partition except for uncommitted transactions. To find these last changes, the audit-fixup process continues to read and apply audit, as needed, to the new partition.

Figure 3 shows the sample table in phase 2. The audit-fixup process finds the new NJ and MI records in the audit trail. Since the MI record will not reside in the new partition, the process discards it. The NJ record, however, should reside in the new partition, so the audit-fixup process applies this change to the new partition.

However, the target partition still may not be an exact copy of the original one. Assume, for example, that as soon as the audit-fixup process reaches the end of the audit trail, a new record, NY, is added to the original partition. If this record is inserted after the audit-fixup process reaches the audit EOF, the record is not copied to the target partition during phase 2.




During phase 2, user applications have complete read and write access to the data in all partitions of the table. Selects, updates, inserts, and deletes continue to execute against the original table.

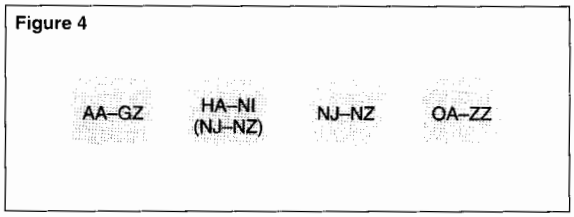
## Phase 3: Audit Fixup 2

In the third phase, Audit Fixup 2, locks are requested to make sure all outstanding user transactions against the table are completed and applied to the new partition before it is made logically part of the table's definition. First, the Split Partition operation requests a file lock against the original partition being split. The lock is made as part of a transaction; it does not allow any other locks to exist concurrently against the specified partition. Thus, once the lock is in effect, no changes outside the transaction can be made to the partition until the transaction is completed.

**Figure 2.**
*The user application inserts records while the table is sequentially read.*

**Figure 3.**
*The user application adds records as the audit-fixup process reads audit.*

**Figure 4.**
*The table's partitioning after phase 3.*




When the lock is granted, the audit-fixup process is called again to finish applying relevant audit. It stops examining audit when it reaches the point where the file lock was granted. In the example, the process reads the NY record at this time.

After this audit is applied, a file lock is requested for all other partitions of the table. When the file lock is granted, the labels of each partition are updated to include the necessary information about the new partition. When the file lock is released, the table physically and logically has four partitions. The redefinition timestamp of the table is updated to indicate that a change has been made. Figure 4 shows the sample table after phase 3 is completed.

When an executing user application makes its next request to the table, the NonStop SQL/MP file-system and executor components notice that the table has undergone a change since their last request. Recompilation occurs if the user has allowed runtime compilations for the application. Since the table now logically has four partitions, the NonStop SQL compiler produces appropriate query execution plans so that Data Definition Language (DDL) and Data Manipulation Language (DML) operations occur against the proper partitions. Recompilation in itself can be a source of outage. A separate NonStop SQL/MP feature, late binding, is available to handle this problem.

At the beginning of phase 3, users have complete read and write access to all unaffected partitions, but by its end, the table is inaccessible to all applications. The time it takes to complete phase 3 depends on the number of partitions of the table and the transaction activity on the table. In practice, this phase should usually last less than a minute.

## Phase 4: Cleanup

In phase 4, Cleanup, the Split Partition operation cleans up the table, removing the physically present, but logically absent, rows from the (original) split partition. In the example, the rows that have been moved from the second partition to the third partition need to be deleted from the second partition. The cleanup operation occurs in the background and does not affect user applications. Phase 4 ends when all the unnecessary records have been deleted. During this phase, the user has complete read and write access to the table.

## Database Recoverability

A major design goal for the NSA database configuration operations required that users who had media protection (used TMF rollforward or NonStop TM/MP file-recovery protection) would be able to recover their database tables if a media failure occurred at any point in the operation. To this end, the NSA operations allow users to make online dumps of the target partition before the end of phase 3. Because of this capability, and because audit is generated for the target partition in phases 2 and 3, one can recover from a media failure at each point in the operation. If a failure occurs before phase 3 ends, the new partition is not visible to the user application, and one can use the original table's online dump. If a failure occurs after phase 3, one can use the online dump made in phases 2 and 3 to recreate the new partition.

## Conclusion

The NSA physical database configuration features introduced in NonStop SQL/MP significantly reduce database outages. By allowing full read and write access to all partitions of a table during the lengthy data-movement phase, these features limit the outages associated with the Split Partition, Move Partition, Move Partition Boundary, and Create Index operations to about one minute per operation.

**References**

Chandra, M. and Eicher, D. 1994. Enhancing Availability, Manageability, and Performance With NonStop TM/MP. *Tandem Systems Review*. Tandem Computers Incorporated. Part no. 104400.

Ho, F., Jain, R., and Troisi, J. 1994. An Overview of NonStop SQL/MP. *Tandem Systems Review*. Tandem Computers Incorporated. Part no. 104400.

**Jim Troisi** is the development manager responsible for the availability features in the NonStop SQL/MP product. Since joining Tandem in 1982, Jim worked as a principal architect of Tandem's Maintenance and Diagnostic System. Since joining the NonStop SQL group in 1989, he has managed the NonStop SQL Utility area and worked to release several NonStop SQL availability and manageability features. In addition, he has been project manager for several NonStop SQL releases, including D20.

# A New Hash-Based Join Algorithm in NonStop SQL/MP

A join is a database operation that combines rows from two tables into a single table. In many contexts, such as decision support and data warehousing, it is necessary to execute ad hoc queries that generate large joins. Based on extensive research in the academic world, a hash join is the algorithm of choice for many join queries, particularly when the tables do not have an index defined on the join columns. The Tandem™ NonStop™ SQL/MP relational database management system implements a hash join algorithm in execution plans for both sequential and parallel processing. This is the first implementation of a hash join algorithm in a major commercial database management system.

This article describes several types of join algorithms, discusses sequential and parallel execution plans for hash joins in NonStop SQL/MP, and compares the hash join algorithm with other join strategies available to the NonStop SQL/MP optimizer. The article also discusses the performance benefits of hash joins and lists conditions under which a hash join is likely to help. The article can be read by anyone familiar with relational databases and the SQL programming language. It may be of particular interest to database designers and database administrators concerned with high-performance query processing.

## Comparison of OLTP and Decision Support Processing Requirements

NonStop SQL has been successfully used in many online transaction processing (OLTP) applications. Such applications are characterized by a large number of small or medium-size transactions that are performed in parallel. A typical example is an automated teller machine (ATM) application in which multiple clients (the ATMs) simultaneously generate independent requests and require short response times. Here, processing a large number of separate requests in parallel, provides load balancing and efficient utilization of system resources. Fast responses to individual requests are possible because each request typically involves a small result set that can be directly accessed via indexed columns. Joins that process large amounts of data are not used extensively in OLTP applications.

Database applications for decision support systems (DSS) use parallel processing in a different way. Unlike OLTP, DSS applications typically handle only a few requests at a time, but each of these is likely to be very large and require considerable system resources. Here, inter-query parallelism, alone, will not provide effective load balancing and resource utilization. To achieve this, parallel processing must also be carried out within a query (Moore and Sodhi, 1990). Such *intra-query* parallelism more fully exploits the resources provided by a multiprocessor, multidisk system and improves performance times on large queries. Hash join execution plans that employ intra-query parallelism are described later in the article.

In addition to using intra-query parallelism, performance improvements on large queries can be achieved by maximizing the use of main memory for complex operations like joins. Main memory can be used to cache rows that are needed multiple times during a join operation. If random access is necessary for a join, table data can often be sequentially read into memory, so that the random access is carried out in memory space rather than through repeated disk accesses. The hash join algorithm described in this article was designed with these considerations in mind. It can be used for intra-query parallelism, it uses application memory to cache rows that are likely to be accessed multiple times, and it uses a simple and efficient hash access structure to quickly perform random accesses on data in memory.

The usefulness of such a hash join algorithm can be illustrated with a simple example. Suppose a DSS database contains a large SALESHISTORY table and a small STORES table, with the columns and primary keys shown in Figure 1.

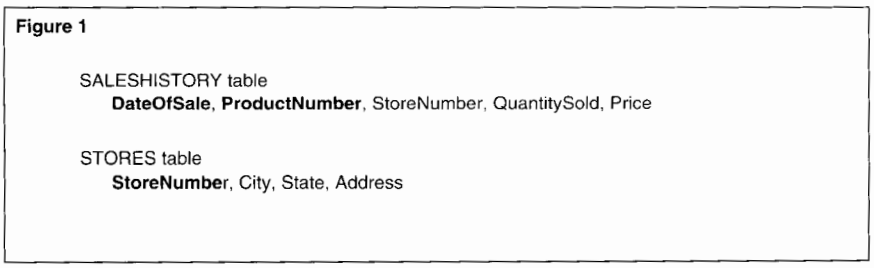



**Figure 1.**
*Columns and primary keys (in bold) for the SALESHISTORY and STORES tables.*

A typical DSS question requiring a join of data from both tables might be "How much revenue came from sales at stores in California over the last three months?" Expressed as a query in SQL, this would be

```
SELECT SUM(QuantitySold × Price)
FROM SALESHISTORY, STORES
WHERE SALESHISTORY.StoreNumber
    = STORES.StoreNumber AND
  STORES.State = "CA" AND
  SALESHISTORY.DateOfSale BETWEEN
    (CURRENT YEAR TO DAY –
       INTERVAL "3" MONTH) AND
    (CURRENT YEAR TO DAY);
```

There are many possible execution plans for this query. Here, it is assumed that the SALESHISTORY table is read row by row for the three months of interest, and that for each SALESHISTORY row, the STORES table is accessed, the StoreNumbers matched, and the state evaluated. For a disk-resident table, each access costs a message to the disk process. Given a large number of sales in a three-month period, accessing the STORES table from the disk process, even assuming that many rows were cached in memory, would involve a considerable cost in messages. The NonStop SQL/MP algorithm for a hash join avoids such costs by building a temporary table in application memory. Thus, in the example, STORES would be accessed as a memory-resident table, and almost all messages to the disk process would be eliminated.

A hash join execution plan is often the most efficient choice for a query in NonStop SQL/MP, especially for large joins between tables of very different sizes, as in the preceding example. The following sections describe and compare join algorithms in some detail. Readers less interested in the internal design of join algorithms and the description of hash join execution plans can skip directly to the section "Performance Benefits" near the end of the article.

**Figure 2**

TABLE1

| COL1 | COL2 |
|---|---|
| 1 | 1 |
| 2 | 2 |
| 3 | 0 |
| 3 | 3 |
| 4 | 4 |
| 6 | 6 |
| 7 | 7 |
| 9 | 9 |

TABLE2

| COL1 | COL2 |
|---|---|
| 1 | 1 |
| 3 | 0 |
| 3 | 9 |
| 4 | 4 |
| 5 | 5 |
| 7 | 7 |
| 8 | 8 |

Join query

```
SELECT *
FROM TABLE1, TABLE2
WHERE TABLE1.COL1 =
      TABLE2.COL1;
```

Join result

1, 1, 1, 1
3, 0, 3, 0
3, 0, 3, 9
3, 3, 3, 0
3, 3, 3, 9
4, 4, 4, 4
7, 7, 7, 7

**Figure 3**

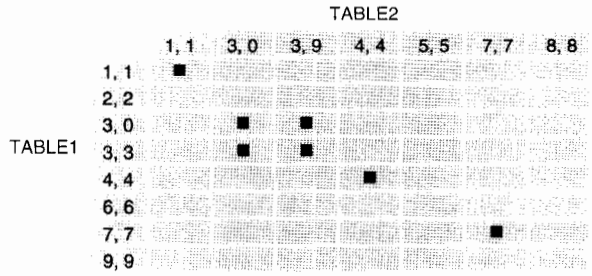



**Figure 4**

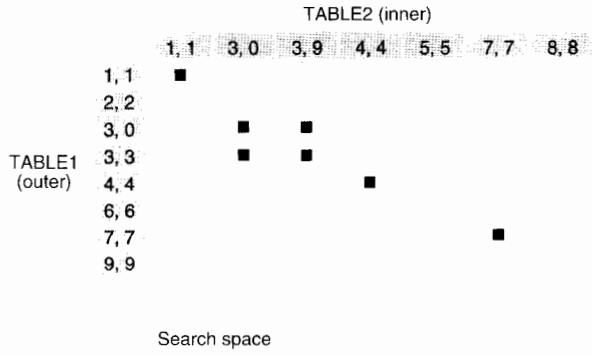



**Figure 2.**
*Sample tables, join query, and join result.*

**Figure 3.**
*Cartesian product of TABLE1 and TABLE2 and qualifying result rows from join query.*

**Figure 4.**
*Search space and join result of a nested-loop join.*

## Overview of Join Algorithms

A relational join is a subset of the Cartesian product of two relational tables. The Cartesian product is the combination of every row in the first table with every row in the second table. The subset is determined by a join predicate. Figure 2 shows two tables, a simple join query between the tables, and the join result. Figure 3 graphically represents the Cartesian product of the two tables and the outcome of the join query.

Before introducing the different join algorithms, it is necessary to define some terms. Most of the join algorithms discussed here apply only to *equijoins*. An equijoin has a join predicate that specifies an equality between columns, as in the predicate TABLE1.COL1 = TABLE2.COL1 in the query in Figure 2. The columns in an equijoin predicate are called the *join columns*. The values in these columns are the *join values*. In the following discussion, the *search space* of a join algorithm consists of the combinations, or cells, in a Cartesian product that the algorithm must evaluate in order to arrive at the final join result. The main purpose of a good join algorithm is to keep the search space small and not try unnecessary combinations. The following subsections describe three basic types of join algorithms: nested-loop join, merge join, and partitioned join.

### Nested-Loop Join

The simplest join algorithm, the nested-loop join, evaluates the entire Cartesian product of two tables by reading every row of one table, the inner table, for each row of the other, the outer table.[1] Those combinations that are qualified by the join predicate are retained as result rows, and all other combinations of the Cartesian product are discarded. As shown in Figure 4, the search space of a nested-loop join is the entire Cartesian product of the tables in the join.

[1] In NonStop SQL/MP EXPLAIN plans, the term *nested join* is used to refer to both a nested-loop join and, as defined later, a nested index join.

In Figure 4, the Cartesian product of the two tables is represented by 56 cells. The join result consists of 7 rows, identified by cells with black squares. Thus, the nested-loop join visits 49 cells that are not part of the join result.

## Merge Join

A merge join algorithm can dramatically reduce the search space of an equijoin. To apply a merge join, both tables must be ordered by their join columns. The join result can then be found by searching diagonally through a matrix representing the Cartesian product of the tables. Figure 5 illustrates a merge join for the tables and query given in Figure 2. Cells containing black squares are part of the join result. White cells constitute the search space of the join.

The query in Figure 2 specifies that an equijoin be carried out on COL1 of both tables. Conveniently for the merge join illustrated in Figure 5, the rows in both tables happen to be in ascending order by COL1. The problem for merge joins is that tables are not always ordered on their join columns. One way to overcome this is to sort the data and perform a merge join on the result. This is called a sort-merge join.

In Figure 5, the search space consists of 16 cells. Only 9 of the cells are not included in the join result.

## Partitioned Join

In a partitioned join, values in the join column (or columns) of the inner table are partitioned into sections. In this approach, given a join value from the outer table, it is only necessary to search the section of the inner table that encompasses this value. This is illustrated in Figure 6, in which the inner table, TABLE2, is partitioned into three sections. In the first section, values in the join column are less than or equal to 3. The second section contains join values 4 through 6, and the third section contains join values of 7 or more.

**Figure 5**

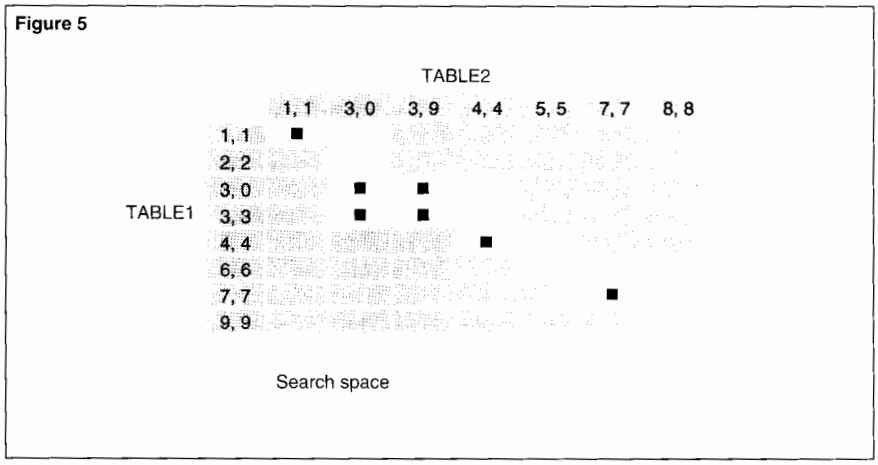
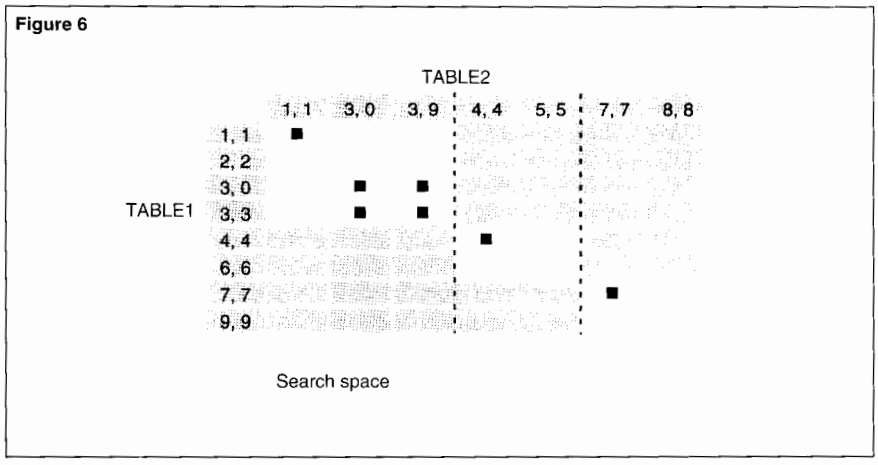



**Figure 6**




In Figure 6, the partitioned join only reaches 13 cells not included in the join result, making it somewhat less efficient than a merge join when no sort is required and far more efficient than a nested-loop join. The concept of using a partitioned search space is the basis for both the nested index join and the hash join in NonStop SQL/MP. In the implementation of the hash join, partitioning is also used to handle memory overflow and to distribute join processing across CPUs in the execution plan for a hash-partitioned join (described later in the article).

**Figure 5.**
*Search space of a merge join.*

**Figure 6.**
*Search space of a partitioned join.*

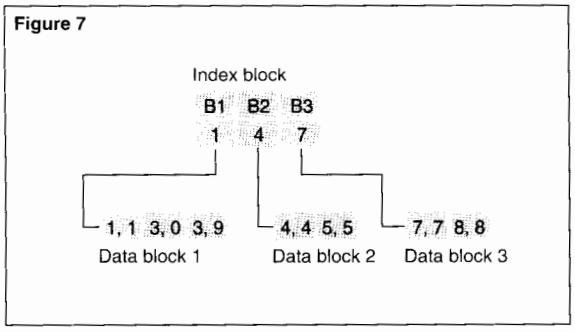



**Figure 7.**
*Partitioning of a table through B-tree indexing.*

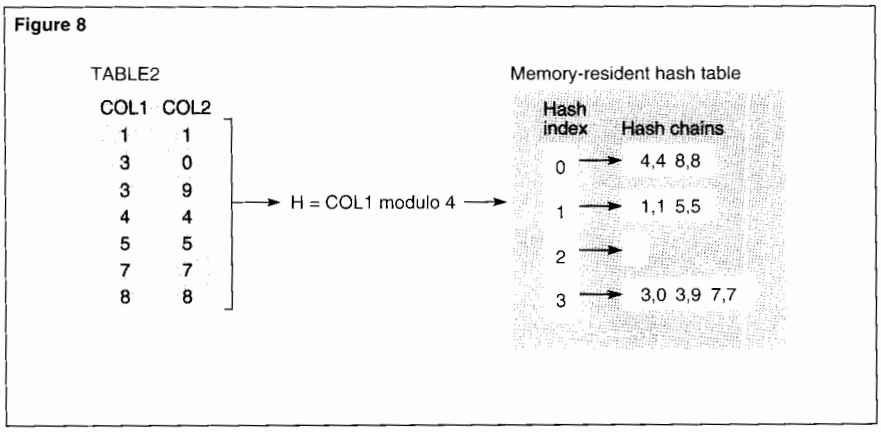



**Figure 8.**
*Building-phase of a hash join: mapping rows from TABLE2 to a memory-resident hash table.*

***Nested Index Join.*** There are a number of ways to partition a table into sections. For example, in a key-sequenced table in NonStop SQL/MP, a B-tree index structure makes it possible to find the block containing a row with a given key value. This is a form of partitioning in which each section consists of the range of key values contained in an individual disk block. In Figure 7, a B-tree index partitions TABLE2 into three sections (data blocks) that match the three sections of TABLE2 in Figure 6.

When a partitioned join uses a B-tree index on the join columns of the inner table, for each row of the outer table, the B-tree index of the inner table identifies the block or blocks that may contain the matching join value. Only these data blocks, rather than the entire table, need to be scanned for matching rows. In this article, a partitioned join that uses a B-tree index is called a nested index join.

An ad hoc query can use any column or group of columns in a table as a join column. Since it is often too expensive to create a permanent index structure for every possible join column, let alone all possible combinations of join columns, there will be cases where a nested index join is either impossible or inefficient. One way to avoid a nested-loop join and obtain the efficiency of a partitioned join is to create a temporary index for the join, as in the case of a NonStop SQL/MP hash join.

***Simple Hash Join.*** A simple hash join is a partitioned join in which a memory-resident hash table is used to partition the inner table, rather than the disk-resident B-tree index of a nested index join. The hash index has as its hash key the join columns of the inner table, and it contains all columns from the table that are needed to form the join result. A simple hash join executes in two phases, a building phase and a probing phase.

In the building phase, the inner table is read from disk and inserted into the memory-resident hash table, using the join column or columns as the hash key. During this phase, rows and columns that are not needed to calculate the join result can be eliminated. Figure 8 illustrates the building phase of a simple hash join that uses the hash function

H = COL1 modulo 4

where H is the hash value and COL1 is the join value submitted to the function.

In Figure 8, TABLE2 is mapped to a memory-resident hash table. The table consists of a hash index and hash chains. Values in the index come from the hash function. Each distinct hash value is associated with a single hash chain containing rows from TABLE2. Conceptually, mapping a row from TABLE2 to the hash table involves two steps: (1) using the hash function to derive a hash value for the value in COL1 and (2) writing all column values for the row into the corresponding hash chain.

In building the memory-resident hash table, all rows of the inner table that have the same join value are read into the same hash chain, although a chain can contain more than one join value. Each chain is equivalent to a separate section in a partitioned join. During the probing phase of a simple hash join, for each row of the outer table, it is only necessary to scan a single hash chain of the memory-resident hash table.

Once the inner table in a hash join has been read into the memory-resident hash table, the probing phase of the join can begin. Figure 9 illustrates the probing phase of a hash join with TABLE1 as the outer table and TABLE2 as the memory-resident inner table.

The basic steps for achieving the join result in Figure 9 are:

1. For each row in TABLE1, use the hash function to derive the hash value corresponding to the value in COL1.
2. Using the hash index to the memory-resident hash table, locate and read the hash chain corresponding to the derived hash value.
3. Return for inclusion in the join result any rows that match the join value in COL1 and meet other selection criteria in the join query.

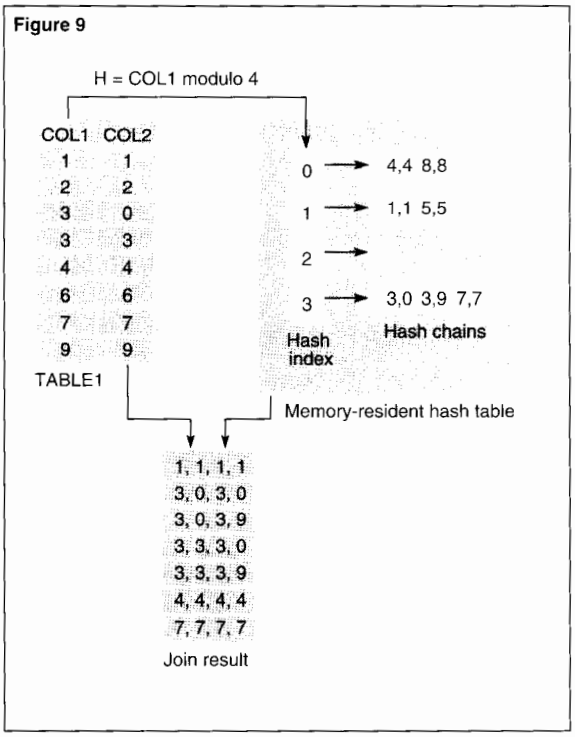
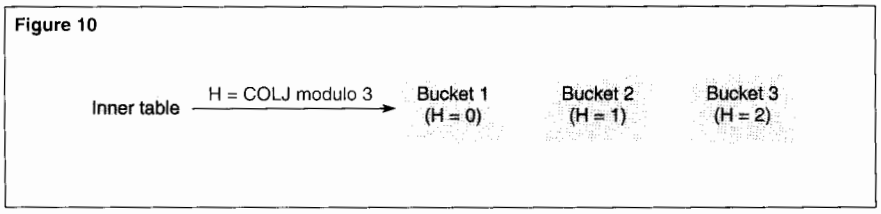



**Figure 9.**
*Probing phase of a hash join and join result.*

## Hash Joins When the Inner Table Is Too Large for Memory

A major problem for simple hash joins is that there is not always enough memory to hold the memory-resident hash table. The following subsections describe three types of hash joins for handling memory limitations: the grace join; the hybrid hash join, which is an improvement on the grace join; and the adaptive hash join, which improves on the hybrid hash join and is the approach implemented in NonStop SQL/MP.



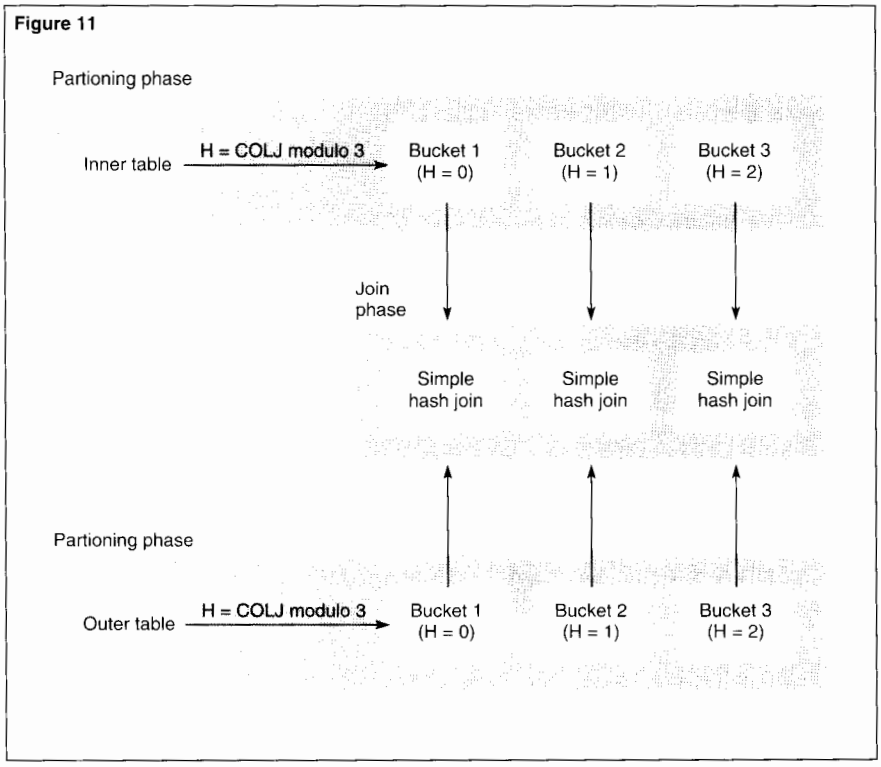




**Figure 10.**
*Grace join: assignment of rows from the inner table to buckets.*

**Figure 11.**
*Partitioning and join phases of a grace join.*

## Grace Join

To deal with memory limitations, a grace join (Kitsuregawa, Nakayama, and Takagi, 1989) consists of two phases, a partitioning phase and a join phase. In the partitioning phase, a hash function is used to divide both the inner and outer tables into a number of *buckets* that reside in temporary files on disk. In the join phase, each inner-table bucket is read into a memory-resident hash table and a simple hash join is performed with the corresponding bucket from the outer table.

***The Partitioning Phase of a Grace Join.*** At the beginning of the partitioning phase, a grace join calculates the number of inner-table buckets it will need, based on the amount of memory available and the size of the inner table. The calculations assume equal bucket sizes and a constant amount of available memory. The goal is to create as few buckets as possible and still have each bucket fit in memory during the join phase. For example, suppose the inner table contains 27 megabytes of data and there is sufficient application memory to hold a 10-megabyte hash table. In this case, partitioning will create three bucket files, each of which is expected to receive 9 megabytes of data.

Given the required number of buckets, a hash function can be defined and rows from the inner table assigned to buckets, as illustrated in Figure 10.

Figure 10 shows an inner table partitioned into three buckets. COLJ of the table is assumed to be the join column. For each row in the table, the value in COLJ is submitted to the hash function

H = COLJ modulo 3

and the row then assigned to the bucket that corresponds to the resulting hash value.

Once the inner table has been partitioned into buckets, the same hash function used for the inner table is used to partition the outer table. This concludes the partitioning phase. For each inner-table bucket on disk, there is now a corresponding outer-table bucket with the same hash value and the same range of possible join values.

***The Join Phase of a Grace Join.*** In the join phase of a grace join, a simple hash join is performed on each pair of corresponding buckets from the inner and outer tables, as illustrated in Figure 11. The final join result is achieved by combining the results of the individual bucket joins.

***Limitations of the Grace Join Algorithm.*** A grace join makes the following crucial assumptions:

- It has a valid estimate of the size of the inner table.
- The distribution of join values in the inner table is not skewed, so that each bucket file receives approximately the same number of rows.
- The amount of memory available does not change throughout the join.

Under production conditions, it is difficult to make sure that even one of these assumptions holds. As a result, an unmodified grace join algorithm is not practical in most circumstances.

The hybrid hash join algorithm described in the next section is limited by the same assumptions as a grace join. It is not until the adaptive hash join of NonStop SQL/MP that these limitations are addressed.

## Hybrid Hash Join

A grace join reads all rows from the inner and outer tables and writes them back out to temporary files on disk for bucket-partitioning. It then has to read the same data back into memory to carry out the join. The hybrid hash join (DeWitt and Gerber, 1985; Schneider and DeWitt, 1989), is based on the grace join, but uses memory more efficiently by performing a simple hash join on one pair of inner and outer partitions during the partitioning phase. In the first part of the partitioning phase, instead of partitioning the entire inner table into buckets on disk, rows belonging to one section are assigned to a memory-resident hash table as soon as they are read into memory. This is illustrated in the upper portion of Figure 12, in which inner-table rows whose join column (COLJ) has a hash value of 0 are immediately read into a memory-resident hash table.

In the second part of the partitioning phase, the outer table is partitioned into sections. Rows from the section with the same hash value as the section of the inner table in the memory-resident hash table are directly joined with rows in the hash table. Following the partitioning phase, the join phase of a hybrid hash join is the same as for a grace join. Figure 12 illustrates the partitioning and join phases of a hybrid hash join.

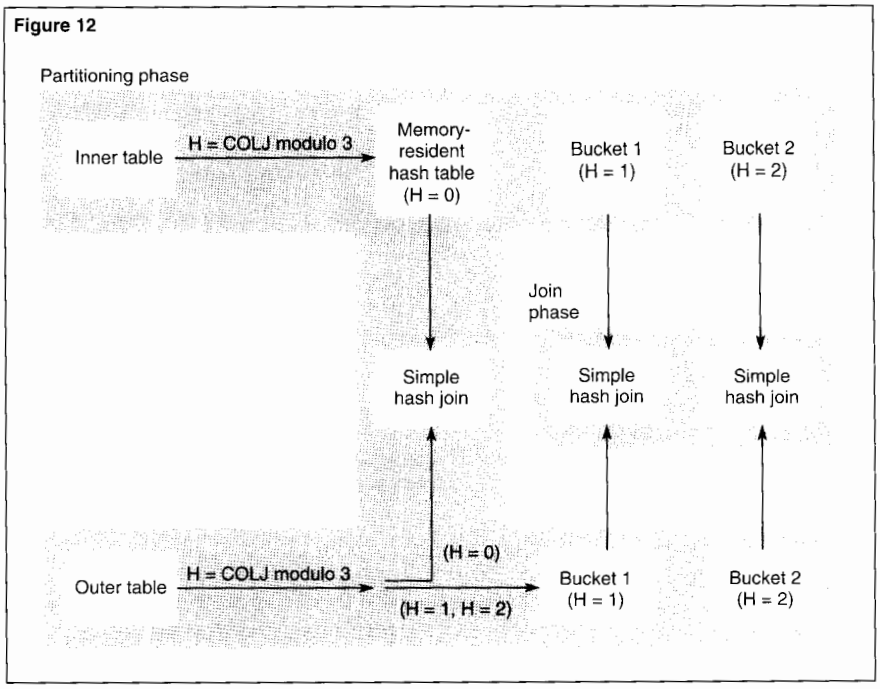
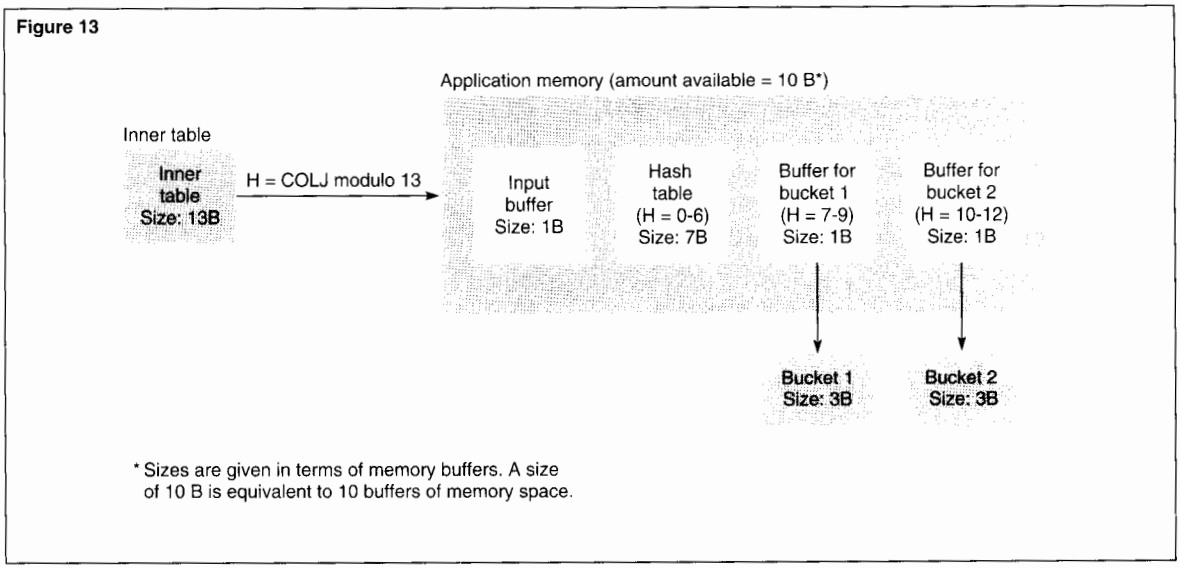



**Figure 12.** *Partitioning and join phases of a hybrid hash join.*

As a comparison between Figures 11 and 12 shows, if the inner and outer tables are only partitioned into a small number of buckets, a hybrid hash join is likely to provide significantly better performance than a grace join. In the grace join of Figure 11, three buckets for each table are written to disk and then read back into memory for simple hash joins. In the hybrid hash join in Figure 12, the initial sections from the inner and outer tables are joined directly in memory, without being written to disk and reread.

**Figure 13.**
*Hybrid hash join: partitioning the inner table into a larger memory-resident section and two smaller buckets.*




A feature of the hybrid hash join that provides for efficient memory usage is that the initial, memory-resident section of the inner table can be a different size than the buckets that are written to disk (see Figure 13). If available memory at the outset of the partitioning phase is relatively large compared to the size of the inner table, the join algorithm can generate a memory-resident hash table that is larger than the buckets written to disk. If available memory is small, relative to the size of the table, the size of the hash table can be made smaller than the remaining buckets. In calculating memory allocations, the join assigns all non-memory-resident buckets the same amount of data and assumes that available memory will remain constant throughout the partitioning and join phases. Figure 13 illustrates the use of memory space in partitioning the inner table.

In Figure 13, the inner table contains enough data to fill 13 memory buffers. In application memory, 10 buffers are available for use in partitioning the table. One of the buffers must be reserved for receiving input from the inner table, leaving 9 buffers for building the memory-resident hash table and receiving data for buckets on disk. To reduce I/O to buckets on disk, a hybrid hash join allocates a separate buffer for each bucket. Multiple rows of data can be read to a buffer and then written to disk when the buffer is full. Figure 13 shows the optimal allocation of memory space, given the size of the inner table and available memory. Seven buffers are allocated for the hash table, and two buffers are allocated to provide for two buckets. Each bucket is to receive three buffers of data.

Note that if only five buffers of memory were available in Figure 13, there would only be four buffers for the hash table and output to individual buckets. In this case, a partitioning based on two buckets would not work. Two memory buffers would have to be used for output to the two buckets and two buffers would be used for the initial memory-resident hash table. Correspondingly, two buffers of data from the inner table would go to the initial hash table and five-and-a-half buffers of data would have to go to each of the two buckets on disk. However, this would exceed the amount of memory available during the join phase, when each bucket has to be read into the memory-resident hash table. A successful partitioning would have to use three buckets, with three memory buffers used for output to buckets and one buffer assigned to the hash table. Each bucket would then receive four buffers of data.

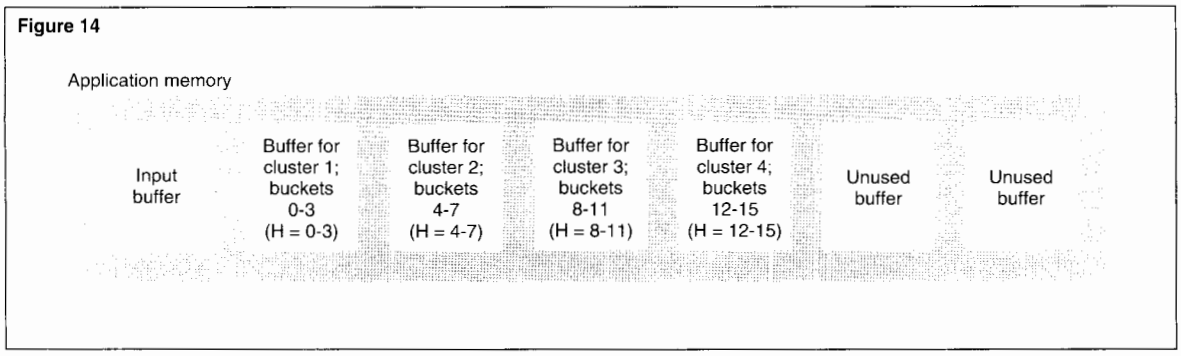



**Figure 14.**
*Adaptive hash join: partitioning the inner table into buckets and clusters (based on H = COLJ modulo 16).*

## Adaptive Hash Join

A hybrid hash join is very efficient when the size of the inner table and the amount of memory available are known before the join starts. This information is necessary for determining the size and number of inner-table partitions that are to be created. Optimally, partitions are as large as possible for the available amount of memory. In many practical applications, however, only a rough estimate of table size is possible and the amount of memory available during the join may vary as the join competes with other processes for memory in its CPU. The adaptive hash join in NonStop SQL/MP uses several strategies to modify the hybrid hash join and provide an algorithm that is robust when the available memory changes dynamically and when there are errors in estimating table size.[2] Sketches of the major strategies follow. They cover

- Allocation and deallocation of buffers for the memory-resident hash table.
- Dynamic adjustment of buffer size.
- Regulating the size of disk files for clusters.
- Using hash loop joins to deal with memory shortages.

For greater detail on the adaptive hash join, see Zeller and Gray (1990).

***Dynamic Adjustment of Buffer Space for the Memory-Resident Hash Table.*** In a hybrid hash join, the amount of memory allocated to the initial memory-resident hash table is determined before partitioning, and cannot change. Throughout partitioning of the inner table, rows can be read into the hash table. In an adaptive hash join, no memory is specifically allocated for the hash table and the inner table is fully partitioned before the hash table is created. As illustrated in Figure 14, partitioning is carried out in terms of buckets and groups of buckets called *clusters*.

In Figure 14, each cluster is made up of 4 buckets. For the example shown, the NonStop SQL/MP optimizer has determined that 16 buckets, making up four clusters, will be adequate for executing the join. Initially, a single buffer is assigned to each cluster. Additional buffers available in memory are unused.

[2] An adaptive hash join can be considered a form of hybrid hash join. When an adaptive hash join is used in NonStop SQL/MP, EXPLAIN plans show the selection of a hybrid hash join.

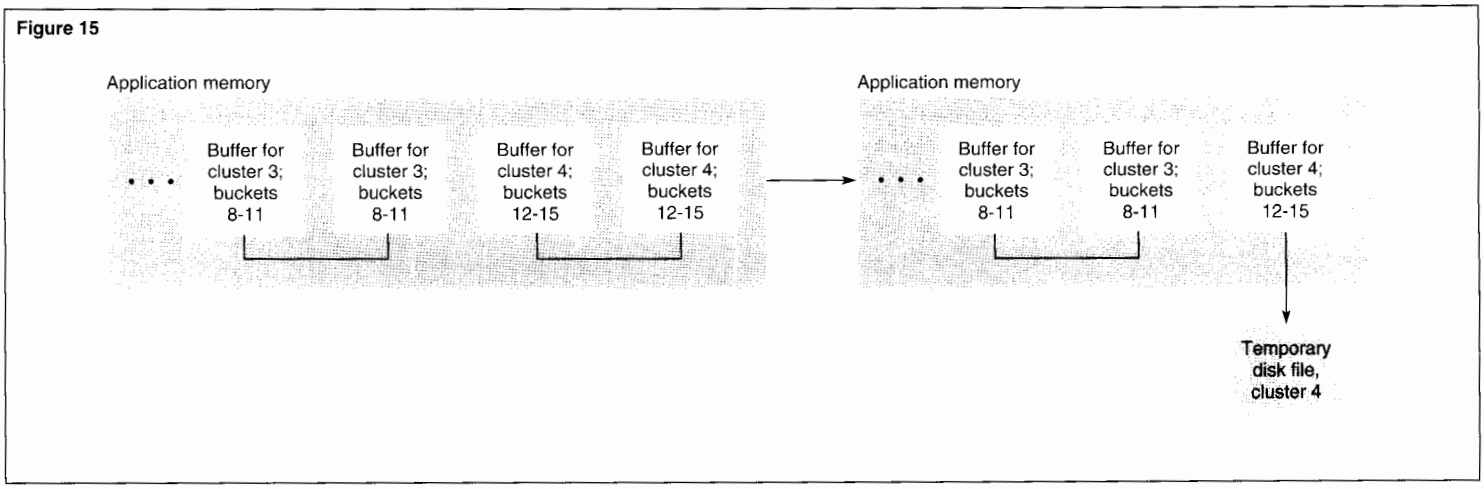



**Figure 15.**
*Adaptive hash join partitioning as buffers overflow and available memory changes.*

Following the initial assignment of buffer space for partitioning the inner table, an adaptive hash join attempts to respond to changes in available memory in a way that maximizes the number of sequential buckets and clusters that stay in memory. When partitioning of the inner table is complete, rows from the memory-resident buckets stay in place as they are linked into the hash chains of a memory-resident hash table. Figure 15 continues the example given in Figure 14 and shows how sequential clustering is maintained as buffers become full and available memory changes.

On the left side of Figure 15, cluster 3 overflowed its original buffer and a new buffer was brought in from available memory for additional cluster-3 rows. Similarly, cluster 4 overflowed its original buffer. It received the last unused buffer available for use in partitioning. Subsequently, as shown on the right side of Figure 15, the amount of memory available to the join was reduced, leaving only three buffers for rows belonging to cluster 3 and cluster 4. In response, the adaptive hash join wrote both buffers of cluster 4 to disk. One of these buffers is retained as an output buffer to cluster 4 on disk. At the end of partitioning, any rows in the buffer will be written to disk; cluster-4 rows will not be included in the initial memory-resident hash table.

Cluster 3 retains its two buffers. If partitioning were to stop at this point, all rows in cluster 1 through cluster 3 would be included in the initial memory-resident hash table. If partitioning continued without additional memory becoming available and cluster 3 overflowed its second buffer, both buffers would be written to disk, as in the case of cluster 4, and cluster 3 would not participate in the initial hash table. If additional memory did become available, a third memory buffer would be added for cluster 3, and it could still be included in the hash table.

***Dynamic Adjustment of Buffers Used for Output to Disk.*** An adaptive hash join can dynamically respond to memory shortages by reducing the size of the buffers it uses. During partitioning, there must be at least one buffer in memory for each cluster. As long as enough memory is available, an adaptive hash join uses large (28-kilobytes) buffers to provide efficient disk I/O for clusters that are written to disk; if available memory becomes too small, buffer size is reduced so that there can be as many buffers as there are clusters.

***Splitting Clusters Before Disk Files Grow Too Large for Available Memory.*** In the join phase, the most efficient way to join clusters from the inner and outer tables is to carry out a simple hash join with the inner-table cluster resident in memory. An adaptive hash join attempts to produce clusters small enough for memory by setting up a sufficient number of clusters at the outset of partitioning, based on available memory and the estimated size of the inner table. However, during partitioning, available memory may decline or the inner table may turn out to be unexpectedly large. To prevent this from resulting in clusters that are too big for memory, an adaptive hash join monitors cluster size during partitioning and dynamically splits clusters when they grow too large.

When a cluster is split, it is divided into two new sections based on its component buckets. Each section has its own output buffer in memory and its own file space on disk.

***Using a Hash-Loop Join When Inner-Table Disk Files Are Too Large for Memory.*** In some cases, it is not possible to prevent the disk file for an inner-table cluster from growing too large for available memory. This may happen when there is an extremely skewed distribution of join values in the inner table, when the table is far larger than expected, or when there is an extreme shortage of memory.

Although a simple hash join is highly efficient when a cluster fits in memory, it causes extensive page faulting and becomes inefficient when a cluster does not fit in memory. As a result, when a cluster is too large for memory, an adaptive hash join uses a *hash-loop join* in place of a simple hash join.

In a hash-loop join, as many blocks of data as possible are read from an inner-table cluster into a memory-resident hash table and a simple hash join is performed with the corresponding cluster of the outer table. When this join is completed, the next set of blocks from the inner-table cluster is read into the memory-resident hash table and a second join is performed with the outer-table cluster. This process is repeated until the entire inner-table cluster has been joined with the outer-table cluster.

With each loop, a hash-loop join can read multiple blocks into a small hash table. This results in far fewer disk accesses than would be necessary with a nested-loop join. A hash-loop join has no overflow conditions and can be very efficient when a simple hash join is not practical.

## Parallel Execution of Hash Joins in NonStop SQL/MP

NonStop SQL/MP provides sequential execution plans for the simple and adaptive hash joins described earlier. In addition, it offers two parallel hash join execution plans (Zeller, 1990), one that performs parallel joins based on the individual partitions of a partitioned table, and another, the hash-partitioned join, that uses a hash function to partition the inner and outer tables and then executes joins on the corresponding sections.

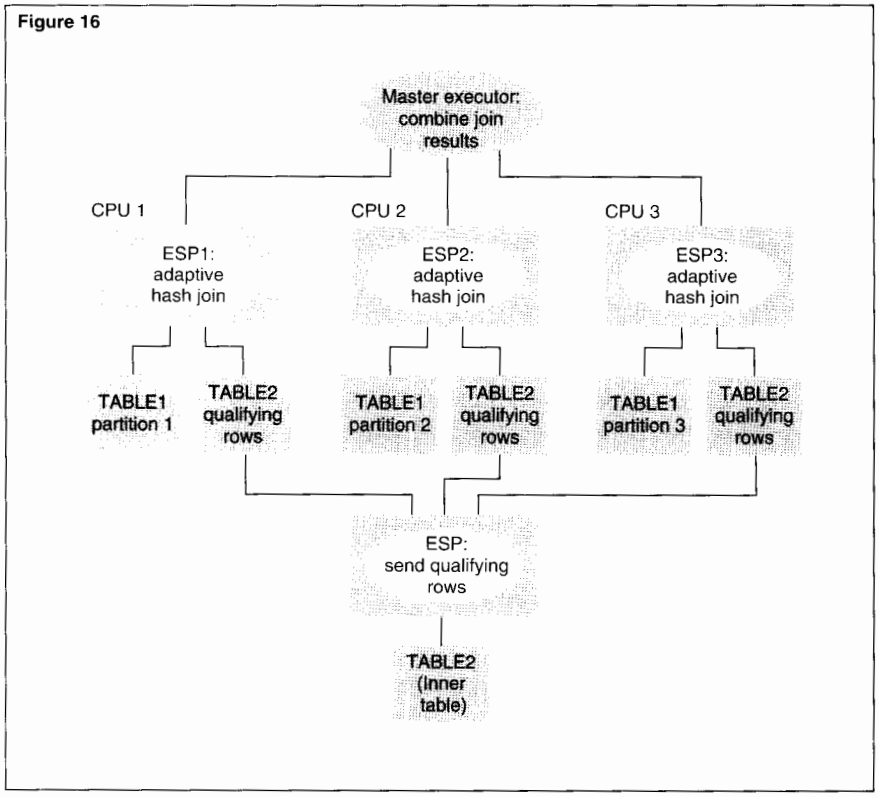



**Figure 16.**
*Parallel hash join on a partitioned table.*

## Parallel Hash Join Based on the Partitions of a Partitioned Table

This hash join execution plan always uses a partitioned outer table. The inner table can also be partitioned, but does not have to be. Under the plan, hash joins are executed in parallel to join each partition of the outer table with a separate copy of all qualifying rows from the inner table. To carry out the joins, an executor server process (ESP) is created for each partition of the outer table. The ESP for a partition resides in the CPU that controls access to the partition's disk volume. The results of the join for each partition are combined to yield the final join result. Figure 16 shows ESPs executing adaptive hash joins in parallel on a partitioned table.

In Figure 16, TABLE1, a partitioned table, is to be joined with TABLE2, the inner table. Selection predicates from a query are applied to TABLE2, and an ESP sends a separate copy of the result to each ESP responsible for a join. These ESPs then execute in parallel, each ESP performing an adaptive hash join on its partition and the rows from the inner table. At the end, the master executor combines the results from each separate join to produce the final join result.

If both the inner and outer tables are partitioned, the partitions of the inner table can be read in parallel for the selection of qualifying rows. Copies of the results from all inner partitions are sent to each ESP responsible for a join, where the results are combined and further processing is the same as for an unpartitioned table.

By using separate copies of rows from the inner table, the parallel execution of a hash join on a partitioned table provides greater parallelism and generates less contention for the inner table than the comparable execution plan for a nested index join. In many cases, this results in better performance than parallel execution of a nested index join.

Parallel execution of a hash join on a partitioned table is most efficient when the inner table is small relative to the partitions of the outer table. If the inner table is relatively large, sending copies of all qualifying rows to all CPUs that control access to an outer-table partition may be less efficient than using a hash-partitioned join.

Figure 17

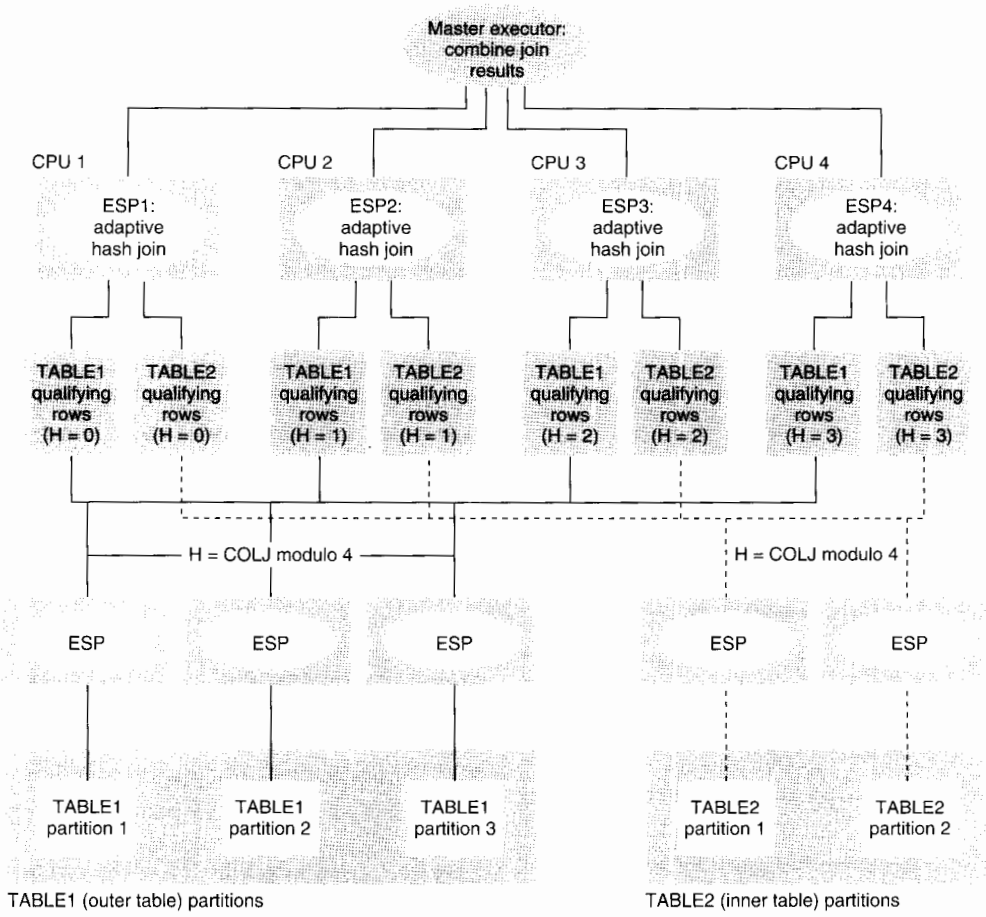
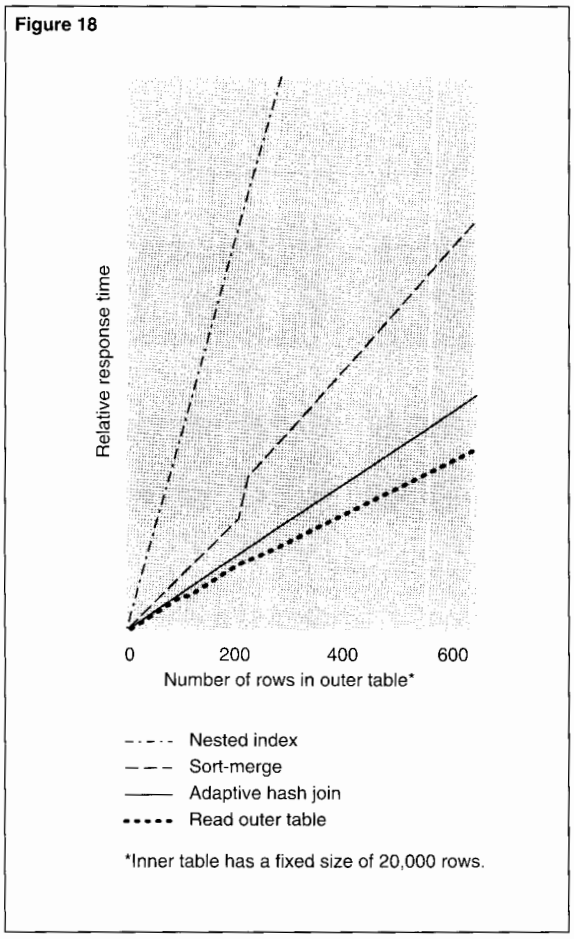



**Figure 17.**
*Hash-partitioned join with four CPUs.*

## Hash-Partitioned Join

In the current NonStop SQL/MP implementation of a hash-partitioned join, a hash function is used to partition the inner and outer tables into separate sections for each CPU in the system. The same hash function is used to partition both the inner and outer tables. Once the tables have been partitioned, hash joins execute in parallel to join corresponding sections. To carry out the joins, a separate ESP is created in each CPU. At the end, the master executor combines the outcomes of the individual joins into a final join result. The tables participating in a hash-partitioned join can be partitioned or unpartitioned tables. Figure 17 illustrates a hash-partitioned join of two partitioned tables on a system with four CPUs.

**Figure 18.**
*Comparison of relative response times for three join algorithms.*




## Performance Benefits

On the Wisconsin-Benchmark, a widely used benchmark for evaluating SQL query performance, response times for hash joins were from 12 percent to 60 percent lower than for either nested index joins or sort-merge joins. In competition with nested index joins, hash joins are faster primarily because they eliminate random disk accesses and reduce messaging to the disk process. Hash joins perform better than sort-merge joins mainly because they avoid intermediate sorts and therefore require less CPU time and generate less message traffic. As a further benefit, when a large table is joined with a smaller table, a hash join needs less temporary disk space than a sort-merge join.

The sidebar "When is a Hash Join Likely to Help?" lists several conditions under which a hash join is likely to provide performance benefits. A hash join is usually most advantageous when an ad hoc query requires a join between a large table and a much smaller table. Figure 18 shows relative response times for a hash join, a nested index join, and a sort-merge join on a series of joins between a small inner table and an outer table of varying sizes. On all joins, the inner table had a fixed size of 20,000 rows. The size of the outer table was varied on each trial, beginning with 5,000 rows and increasing to 600,000 rows. Both the inner and outer tables conformed to the standard definition of a table for the Wisconsin-Benchmark.

## Using EXPLAIN Plans to Evaluate Hash Joins

NonStop SQL/MP provides a CONTROL QUERY HASH JOIN statement for enabling or disabling the use of hash joins in an application. The default condition is for CONTROL QUERY HASH JOIN to be ON. Users can evaluate the benefits of hash joins on individual queries by generating EXPLAIN plans using the NonStop SQL/MP EXPLAIN utility and the NonStop SQL/MP conversational interface (SQLCI) in conjunction with CONTROL QUERY HASH JOIN statements. To do this, perform the following steps:

1. Using SQLCI with CONTROL QUERY HASH JOIN on, generate EXPLAIN plans for queries that are likely candidates for hash joins.
2. Identify queries for which the optimizer chooses hash joins.
3. Generate EXPLAIN plans for these queries with CONTROL QUERY HASH JOIN off.
4. Compare the execution costs in EXPLAIN plans for hash joins with the estimated costs of EXPLAIN plans for other joins on the same queries.

For more detailed information on using EXPLAIN plans and tuning query performance, see the *NonStop SQL Query Guide* (1994).

## Conclusion

Hash joins are a new feature of NonStop SQL/MP. They are available in both sequential and parallel execution plans. Hash joins outperform sort-merge joins and nested index joins in many situations and are particularly useful with decision support applications and ad hoc join queries. The adaptive hash join in NonStop SQL/MP dynamically adjusts to changes in available memory during execution and performs efficiently in the face of unexpectedly large tables and highly skewed table values. Based on the Wisconsin-Benchmark, NonStop SQL/MP hash joins can improve performance by 12 percent to 60 percent in the appropriate circumstances.

### When is a Hash Join Likely to Help?

A hash join is likely to provide performance benefits under the following conditions:

- When no index is usable, since a nested index join is not possible and a merge join would require sorts of the input tables.
- When the tables of a join are of different sizes, since only the smaller table needs to be stored in memory.
- When memory in multiple CPUs can be utilized, since the execution plans for parallel hash joins are very efficient.
- When many rows from the inner table qualify for the join result, since the overhead of building the memory-resident hash table is then acceptable.
- When a small subset of the rows in a large inner table can efficiently be selected for inclusion in the hash table.

#### References


DeWitt, D. and Gerber, R. 1985. Multiprocessor Hash-based Join Algorithms. *Proceedings of the 11th Very Large Data Base (VLDB) Conference.*

Kitsuregawa, M., Nakayama, M., and Takagi, M. 1989. The Effect of Bucket Size Tuning in the Dynamic Hybrid Grace Hash Join Method. *Proceedings of the 15th VLDB Conference.*

Moore, M. and Sodhi, A. 1990. Parallelism in NonStop SQL Release 2. *Tandem Systems Review*. Vol. 6, No. 2. Tandem Computers Incorporated. Part no. 46987.

*NonStop SQL Query Guide*. 1994. Tandem Computers Incorporated. Part No. 93964.

Schneider, D. and DeWitt, D. 1989. A Performance Evaluation of Four Parallel Join Algorithms in a Shared-nothing Multiprocessor Environment. *Proceedings of the Association for Computing Machinery (ACM) Special Interest Group on Modification Of Data (SIGMOD) Conference.*

Zeller, H. 1990. Parallel Query Execution in NonStop SQL. *Proceedings of the IEEE Spring COMPCON.*

Zeller, H. and Gray, J. 1990. An Adaptive Hash Join Algorithm for Multiuser Environments. *Proceedings of the 16th VLDBConference.*



#### Acknowledgments

Thanks to Diana Shak, Anoop Sharma, and Jay Vaishnav for designing and implementing much of the support for the new join algorithm in the SQL compiler and for improving its functionality. Thanks also to the reviewers of this article for their many helpful suggestions and to Susanne Englert for providing performance data.



**Hansjorg Zeller** joined Tandem in 1988, initially on temporary assignment to a joint project between Tandem and the University of Stuttgart, Germany. Since then, he has been responsible for the design and implementation of hash joins in NonStop SQL/MP. He has also worked on the NonStop SQL executor and on a project to improve name binding and reduce recompilation overhead.

# NonStop ODBC Server

Microsoft's ODBC interface and the Microsoft/Sybase DBLIB interface are widely used application program interfaces (APIs) for client/server database applications. The Tandem™ NonStop™ ODBC Server (NSODBC) product gives applications written to either interface open and transparent access to NonStop SQL/MP databases. This includes popular off-the-shelf applications such as Access and PowerBuilder and custom-built applications developed using tools such as Visual Basic. All such applications can take advantage of Tandem and NonStop SQL/MP features for high availability, distributed data, and parallel execution.

The ODBC interface is based on standards work and is specified as part of Microsoft's Open Database Connectivity (ODBC) product. ODBC is currently available for SQL applications that run under Microsoft Windows. In the future, it will also be available for applications that run on Windows NT, Macintosh, or other client platforms.

NSODBC is specially architected to establish client connections quickly and provide high-performance NonStop SQL/MP execution. This allows NSODBC to be used for both OLTP and decision support applications, as well as for applications that build and issue queries and generate reports.

The first sections of this article discuss openness in client/server computing and provide an overview of ODBC and the role it plays in supporting openness. The remaining sections describe NSODBC architecture and functionality, with special emphasis on NSODBC features for high performance.

## Client/Server and Openness

The distinguishing feature of a client/server architecture is that functionality is divided between client processes and server processes that communicate using an agreed upon protocol. Typically, the client and server processes are located on different computers and communicate over a local-area or wide-area network. The basic protocol is for a client to send a request to a server and wait for a reply. One popular form of client/server is client/database server (C/DBS), shown in Figure 1. The client sends an SQL statement and the server executes the statement and replies with the results. The Tandem NonStop ODBC Server, the Tandem SQL Server Gateway, and the DAL Server are examples of C/DBS in a Tandem environment.[1]

[1]For a detailed description of NSODBC, see the *NonStop ODBC Server Manual* (1994); for detailed information on the Tandem SQL Server Gateway, see the *Tandem SQL Server Gateway Manual* (1989). The DAL Server is discussed in Schlansky and Schrengohst (1993).

The pros and cons of client/server in general have been widely discussed elsewhere (see, for example, Cooperstein, 1992, and Rohner, 1994). Here, the focus is on *openness* in client/server computing and the way ODBC and NSODBC contribute to it. Openness, in this context, refers to the ability of different products, usually from different vendors, to interconnect with each other and to replace each other on a network. Openness is important to client and server vendors because it extends the use of their products and it is important to client/server users because it extends their implementation choices.

On the client side, an application typically communicates with a server through what in ODBC terminology is called a *driver*. A driver consists of a library of code and supports an API. To send messages to a server and receive messages back, the client application must be coded to the driver's API and make calls to specific functions or procedures in the driver. The driver communicates with a server using a particular *formats and protocol* (FAP). The FAP describes permissible message formats and allowable sequences of messages. A single vendor often supplies both the driver and server, with the consequence that both the API and FAP are proprietary. In such cases, an application developed for use with one proprietary server cannot be used with another vendor's server unless it is ported to a different API and uses a driver that supports the second vendor's FAP.

A client/server architecture is conducive to openness. If clients and servers support the same FAP, there is *open interoperability*: clients can freely interconnect with servers and use one server in place of another. If the API used in an application is supported by drivers on different platforms, there is *open portability*: the application can be ported from one client platform to another without change. Figure 2 illustrates open portability. Figure 3 illustrates open interoperability.

In Figure 2, application A on the PC uses API 1. It is directly portable to the UNIX workstation, since driver 2 on the workstation supports API 1.

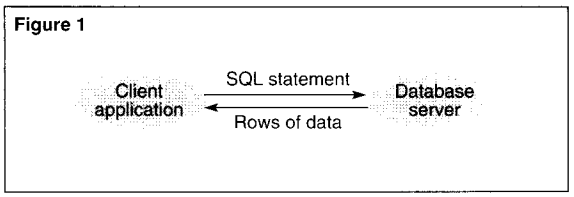



**Figure 1.**
*Client/database server architecture.*

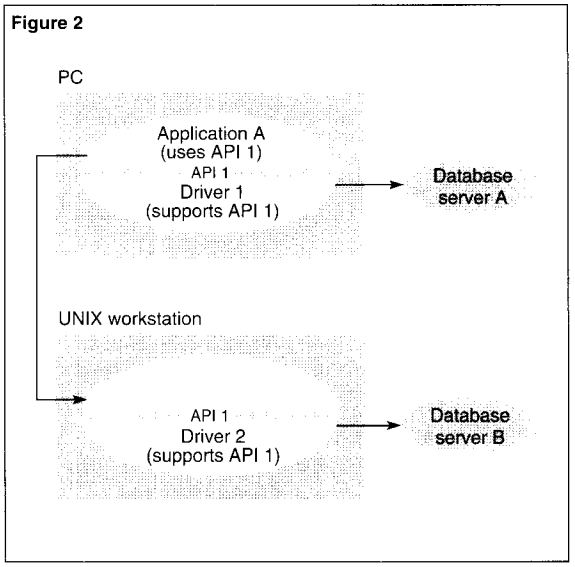



**Figure 2.**
*Open portability.*

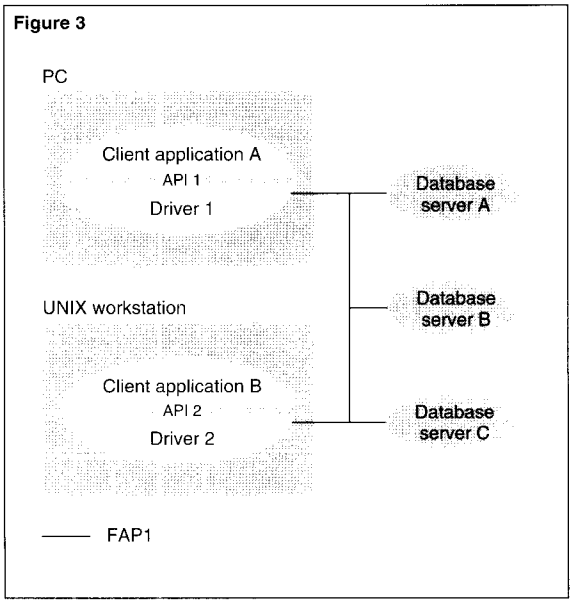
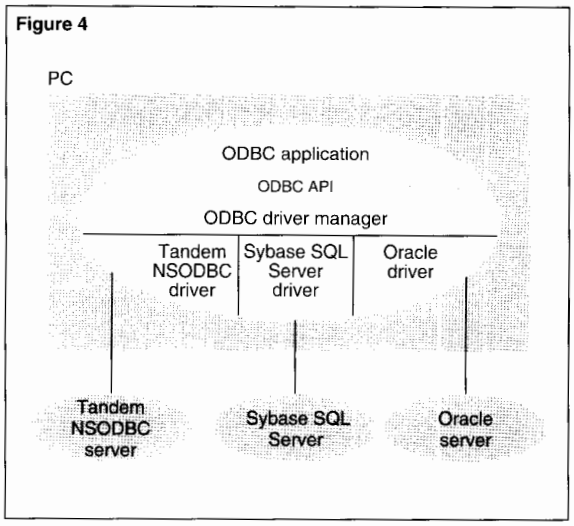



**Figure 3.**
*Open interoperability.*

**Figure 4.**
*ODBC architecture.*




In Figure 3, applications on a PC and a UNIX workstation communicate with database servers A, B, and C. All three database servers use FAP 1. Application A, on the PC, uses API 1 and driver 1. Since driver 1 uses FAP 1, application A is fully interoperable with the three database servers. Application B, on the UNIX workstation, uses API 2 and driver 2. Since driver 2 uses FAP 1, application B can also interoperate with all three database servers. Note that the ability of an application to interoperate with database servers is determined by the FAP its driver uses, and not by the application's API. As described in the next section, ODBC provides both open portability and a form of open interoperability for C/DBS applications.

## ODBC

Microsoft Windows is currently the most popular platform for applications with a graphical user interface (GUI) and many database vendors, Oracle and Sybase among them, have provided APIs and drivers so that Windows applications can be written to access their databases. However, because such APIs are proprietary and different from one another, it has been difficult to develop Windows applications that could access databases from multiple vendors. To address this problem, Microsoft developed its ODBC product to provide a single API for database services. The ODBC API is based on standards work by the SQL Access Group and X/Open,[2] but also contains a number of extensions. The API is described in the next section. For more detailed information on ODBC, see the ODBC *Programmer's Reference* (1992).

Applications that use the ODBC API will have open portability across Windows, Windows NT, Macintosh, and other platforms. In addition, as server vendors supply ODBC drivers, ODBC clients will have a form of open interoperability: without modification, any ODBC client application will be able to connect with a wide range of database servers. Industry acceptance of ODBC has been substantial, with more than 75 vendors pledging support and many already delivering products. Almost all major application and tool vendors have made a commitment to use the ODBC API; almost all major database vendors have made a commitment to provide drivers so that ODBC clients can connect to their database systems. NSODBC provides ODBC connectivity to Tandem NonStop SQL/MP databases, and makes it possible for a large number of ODBC query, report, and application-development tools to be used with NonStop SQL/MP.

Figure 4 illustrates the ODBC architecture. The ODBC API is supported by one or more ODBC drivers and an ODBC driver manager that handles loading and binding to the drivers. Applications use the API without knowing the particular driver that is handling connectivity to the database server.

[2]For information on the work of X/Open, see *Data Management: SQL Call Level Interface (CLI)* (1993).

Most database vendors have developed their own ODBC drivers for connecting to their database servers. Since the FAP used by a driver is almost always proprietary, different drivers must be used for connecting to different database servers, as shown in Figure 4.

When an application calls the API SQLConnect function, which is used to establish a connection to a server, the driver manager loads the appropriate driver and passes the SQLConnect call to it. Most subsequent API calls to process SQL statements are then passed directly to the driver. The driver is entirely responsible for connecting to a server and communicating with it in order to execute SQL statements. ODBC does not specify how a driver works or what FAP is used to communicate with servers.

## ODBC API

ODBC's most significant feature is its API, which specifies

- A call level interface (CLI) of C functions.
- A syntax definition for SQL statements.
- Sequencing rules for CLI calls.

An application can conform to the CLI at any of three levels. There are also three conformance levels for the SQL syntax specification. Table 1 summarizes the API components.

Specifying the CLI and SQL syntax in terms of levels is intended to promote portability and interoperability, since it is easier for clients and servers to agree on a particular intermediate level than to have to agree on the entire specification. Microsoft recommends level 1 CLI functions and core SQL syntax as most appropriate for interoperability and portability. This means that ODBC drivers and database servers should support at least this much, if they are to promote interoperability, and that client applications should require no more than this if they are to be portable and interoperable. Most client and server vendors supporting ODBC have announced that they will comply with this recommendation. While ODBC promotes a common API, it also has facilities for supporting extensions added by server vendors. Client applications can make use of these extensions if open interoperability with database servers is not a concern.

The Tandem NSODBC Server product supports more than level 1 CLI functions and more than core SQL syntax, and thus enables a large set of ODBC applications to openly use NonStop SQL/MP on Tandem systems. One of the features in extended SQL that NSODBC supports is stored procedure execution. In addition, NSODBC supports API extensions that allow applications to utilize specific Tandem features such as creating a partitioned table.

**Table 1.**
Components and conformance levels of the ODBC API.

| | |
|---|---|
| **I. Call level interface (CLI) conformance levels** | |
| Core | Connect to a database, define and execute SQL statements, manage cursors, supply parameters, read result values and status information, and manage transactions. |
| Level 1 | CORE plus: retrieve SQL catalog information, read and set options for the driver and server, and handle very large data values. |
| Level 2 | Level 1 plus: scrollable cursors, retrieve SQL catalog information for stored procedures, and additional extensions. |
| **II. SQL syntax conformance levels** | |
| Minimum | Includes: create and drop table, insert, update, delete; select (without subqueries or aggregation); one data type: character. |
| Core | Minimum plus: views and indexes, grant and revoke, select with subqueries and aggregation; numeric data types. |
| Extended | CORE plus: stored procedure execution, outer join, datetime, and several built-in functions. |
| **III. Sequencing rules for CLI calls** | |
| | Order in which CLI calls can be executed. For example, one must connect to a database (CLI call SQLConnect) before executing SQL statements. Most sequencing rules depend on the SQL statements being executed. For example, one cannot redefine the select statement for a cursor when the cursor is open. |

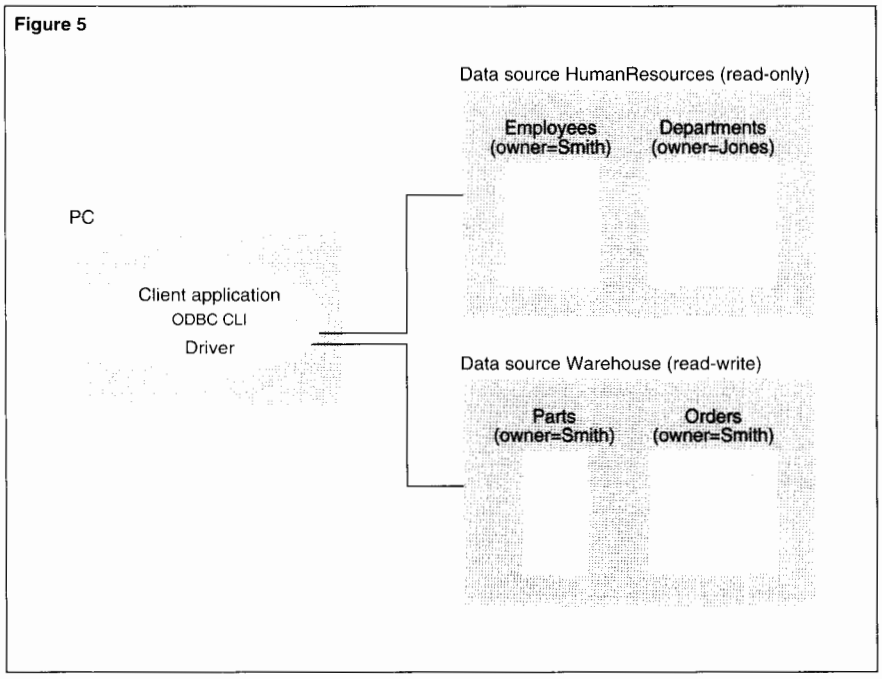



**Figure 5.**

*Client connections to two data sources.*

## Reconciling Differences Between Database Servers

One of the major challenges in designing an effective client API for ODBC was the need to address the many differences between vendors in respect to proprietary APIs. For example, every vendor's client API has a unique mechanism for connecting to a database. For one product, a client first connects to a database server and then selects the database it wants to use. For another product, the client connects to a database directly. Additionally, the format of database names may vary from vendor to vendor; naming may be different in the maximum length and the set of characters allowed. Another type of example concerns the way information about SQL objects is maintained in the database. Every vendor uses a set of SQL tables, called an SQL catalog, to hold the information, but each vendor's SQL catalog structure is unique. The following sections describe ODBC approaches to reconciling vendor differences in the two examples given. Similar approaches are used in other cases.

### Connecting to a Database

To provide a uniform approach to database connections, ODBC uses the simple abstraction of a *data source* to encompass different vendor notions of what a database is and how clients connect to it. From the perspective of an ODBC application, it always connects to a data source and does not need to be concerned about whether there is a database, a database server, or something else at the other end of the connection. In some cases, the data source may in fact be a NonStop SQL/MP database, in others, it may be a Sybase SQL Server, an Oracle database, or the database of some other vendor. In all cases, the data source contains a set of SQL objects such as tables and views and supports user names and other standard database features.

The abstraction of a data source provides a uniform way of viewing database connections, but does not in itself provide a way to set or alter features of a connection that may be different from one vendor to the next. To accomplish this, ODBC provides CLI functions for establishing connections and configuring some connection attributes. For example, the CLI SQLConnect function is used to establish connections between a client application and a database server. SQLConnect's parameters include a data source name, a user name, and a password. The SQLSetConnectionOption function is used to configure run-time options, such as automatically committing a transaction for each SQL statement executed.

In addition to using CLI functions for configuring connection attributes, ODBC uses the configuration file ODBC.INI to associate specific information necessary for establishing a connection with individual data source names. The ODBC.INI file is used to bind data source names to particular database servers and to provide such information as the network address of a data source, proprietary verbs to be used in establishing connections, and the name of the ODBC driver to be used for the data source.

Figure 5 illustrates connections to two data sources, HumanResources and Warehouse. HumanResources has two tables, Employees, created by user Smith, and Departments, created by user Jones. Data source Warehouse has two tables, both created by user Smith. Regardless of the server vendor and the way the server accesses the database, from the perspective of an ODBC client application it is simply connecting to data source HumanResources or data source Warehouse.

ODBC uses the abstraction of a data source as more than simply a means of connecting to a database. In addition to being used in the configuration of database attributes, an ODBC data source can be configured for the type of access it allows and for performance features, transaction modes, a default logon name, and other attributes. For example, in Figure 5, HumanResources is limited to read-only access; users cannot modify its tables. Data source Warehouse allows read-write access; SQL tables in Warehouse can be both read and modified.

## Retrieving SQL Catalog Information

SQL database systems from most vendors allow client applications to use SQL statements to obtain information from their catalog tables, but SQL catalog structure varies widely over different vendors. To address these differences, ODBC provides a set of CLI functions for retrieving SQL catalog information. For example, a client can use the CLI function SQLTables to list the names of all tables in a given data source. In addition, parameters of the SQLTables function make it possible to qualify the table names returned. For example, SQLTables could be limited to retrieving only the tables owned by Smith.

ODBC's use of CLI functions provides a uniform interface for access to catalog information, but it means that applications give up the ability to use SQL statements in order to obtain catalog information. For example, an ODBC application using CLI functions cannot use a SQL join query to obtain composite information from two or more catalog tables.

Figure 6

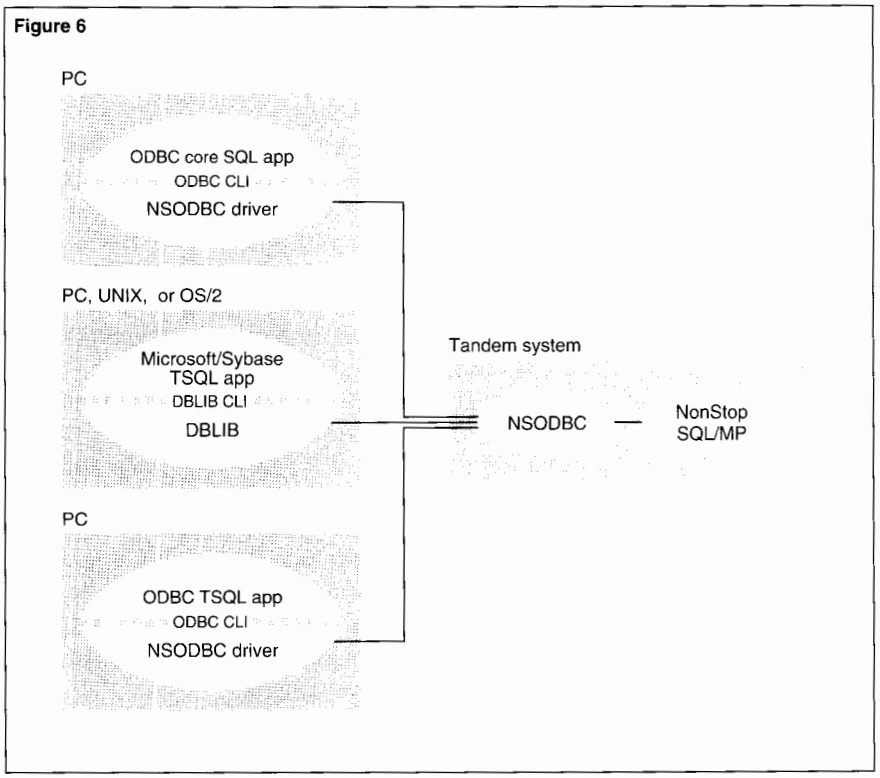



**Figure 6.**
*The three types of NSODBC clients.*

## NSODBC Overview

The discussion now moves to the Tandem NonStop ODBC Server, which provides transparent access to NonStop SQL/MP for ODBC and DBLIB applications. The following subsections describe the architecture and major functional aspects of NSODBC. The closing portion of the article focuses on design features for optimizing NSODBC performance.

## NSODBC Clients

NSODBC is designed to support ODBC applications that run in a Windows environment and use the ODBC CLI and SQL syntax specifications, Microsoft/Sybase DBLIB applications that use the Transact SQL (TSQL) dialect of SQL, and a special class of ODBC applications that use the ODBC CLI, but instead of ODBC SQL syntax, use TSQL. Figure 6 illustrates the three types of NSODBC clients.

For ODBC clients, NSODBC supports all CLI level 1 functions and some level 2 functions, and it supports core SQL syntax plus some features of extended SQL.

Microsoft/Sybase DBLIB applications use TSQL and run in Windows, DOS, UNIX, and OS/2 environments. In the past, support for DBLIB applications was provided by the Tandem SQL Server Gateway product. NSODBC is intended as a replacement for the Tandem SQL Server Gateway and provides superior performance.

To make it easier for DBLIB applications to migrate to ODBC, NSODBC supports clients that use the ODBC CLI and TSQL. Such applications only need to migrate to the ODBC CLI initially, and can then migrate to ODBC's core SQL later. Although ODBC and DBLIB differ in many ways and have distinct database models, SQL syntaxes, SQL catalog structures, and data types, NSODBC uses the same underlying mechanisms to support both environments.

## NSODBC Object Naming and Catalogs

NSODBC's main task is to be an SQL gateway and translate between a client's SQL environment and the Tandem NonStop SQL/MP environment (for a discussion of SQL gateways, see Slutz, 1990). Translation is required for SQL syntax, data types, error messages, object names, database models, SQL catalog structures, and other items. The translation of database models and object names is most relevant to this article (for other aspects of translation, see the *NonStop ODBC Server Manual*, 1994).

For translation between database models, NSODBC associates an ODBC data source or a Sybase SQL Server database with a single Tandem NonStop SQL/MP catalog. All objects in the NonStop SQL/MP catalog appear in the corresponding data source or SQL Server database.

***Logical Names.*** Name translation is necessary for mapping the logical names of objects in an ODBC data source or SQL Server database to Tandem or NonStop SQL/MP object names. When an ODBC or DBLIB application creates a database object such as a table or view, the name assigned to the object within the application is called the logical name. For example, suppose an ODBC data source called HumanResources is associated with the Tandem NonStop SQL/MP catalog \N.$V1.HUMANRES. If the user of an ODBC application creates a table named Employees in data source HumanResources, this has the effect of creating a NonStop SQL/MP table with an NSODBC assigned name, for example, \N.$V.HR.EMP. Employees is the table's logical name within the ODBC data source, and \N.$V.HR.EMP is the table's Tandem name.

The distinction between logical names and Tandem names also applies to user names. For example, the user name specified in the CLI SQLConnect function of an ODBC application is a logical name and must be translated to a NonStop Kernel user name before the user can access the desired NonStop SQL/MP database. When a client connects to NSODBC, all access to NonStop SQL/MP objects is made through the corresponding NonStop Kernel user name. One consequence of this is that the NonStop Kernel user name must be authorized in advance to access NonStop SQL/MP objects.

***NSODBC Catalogs.*** To map between logical names and Tandem names, NSODBC maintains a set of mapping tables. Every row in a mapping table pairs a logical name with the corresponding Tandem name. When an ODBC or DBLIB application uses a logical name, NSODBC retrieves the row with the logical name and maps it to the required Tandem name.

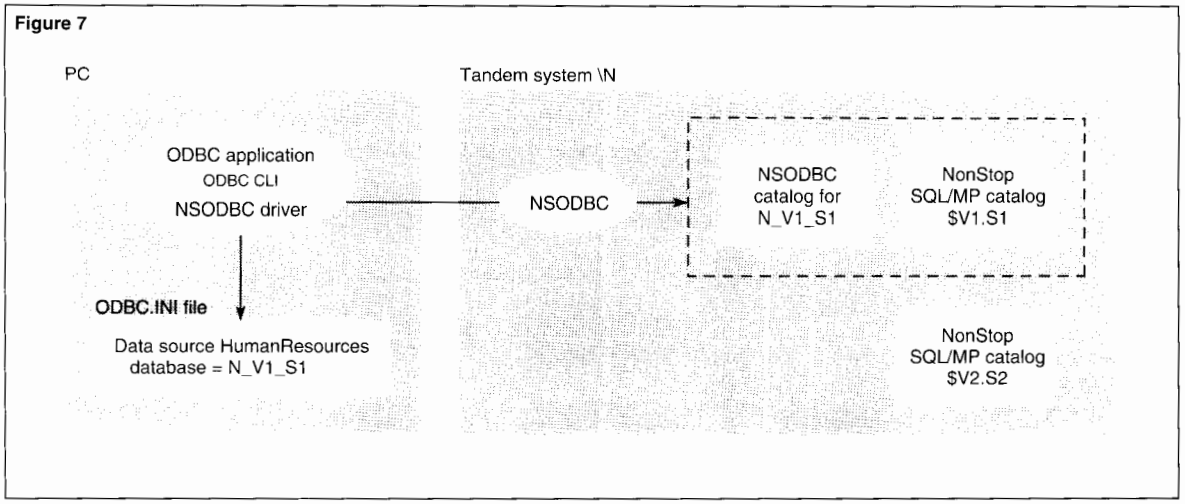



**Figure 7.**
*Access to a NonStop SQL/MP catalog requires an NSODBC catalog.*

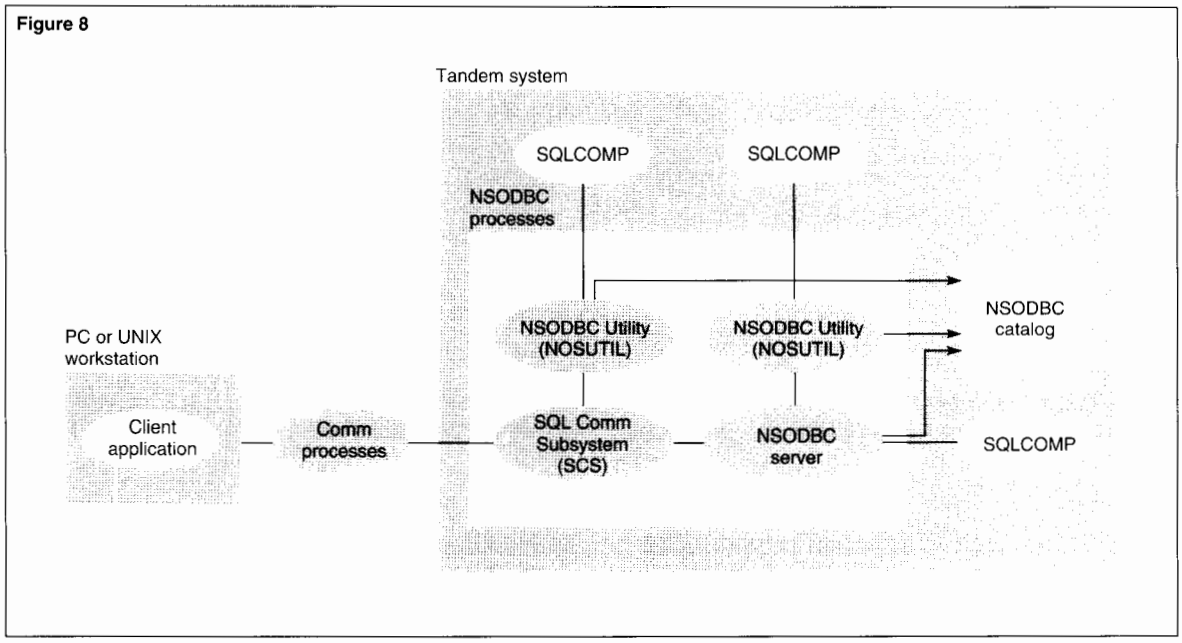
The set of tables that contain mappings for logical and Tandem names is called an NSODBC catalog. As discussed later, an NSODBC catalog also contains configuration and user profile information. NSODBC catalogs are created by the NSODBC utility (NOSUTIL) process. When a client application issues a CREATE DATABASE statement to NSODBC, NSODBC calls NOSUTIL to create an NSODBC catalog for the database; if the database does not already exist, NSODBC also creates a NonStop SQL/MP catalog for the database. NSODBC catalogs can also be created for existing NonStop SQL/MP catalogs by running NOSUTIL directly. The tables in an NSODBC catalog are NonStop SQL/MP tables and, by convention, are registered in the NonStop SQL/MP catalog they are associated with.

A NonStop SQL/MP catalog cannot be accessed as a data source or SQL Server database unless its NonStop SQL/MP catalog has an NSODBC catalog associated with it. This is illustrated in Figure 7. In the figure, \N is a Tandem system that contains two NonStop SQL/MP catalogs, $V1.S1 and $V2.S2. Since NonStop SQL/MP catalog $V1.S1 has an NSODBC catalog associated with it, an ODBC or DBLIB client can be configured to access catalog $V1.S1 and its objects as a data source or SQL Server database. In the figure, data source HumanResources on the ODBC client is configured as database N_V1_S1, which gives it access to NonStop SQL/MP catalog $V1.S1 and its objects through the NSODBC catalog for N_V1_S1. Objects in NonStop SQL/MP catalog $V2.S2 are not accessible to clients because there is no associated NSODBC catalog.

When NSODBC is used, in addition to NSODBC catalogs, there must also be an NSODBC system catalog to correspond to the NonStop SQL/MP system catalog. The NSODBC system catalog is like other NSODBC catalogs, except that it contains added tables with information about users and about databases and their corresponding NonStop SQL/MP catalogs.

When a client connects to NSODBC, a session is started and continues until the client disconnects. The session's environment is initialized according to the attributes of a profile associated with the user. The profile information is stored in the NSODBC system catalog and includes the user's NonStop Kernel user name, the user's default database, the user's database access mode (read-only or read-write), transaction isolation level, performance levels, resource limits, and trace settings.

**Figure 8.**
*NSODBC process structure.*




## NSODBC Process Structure

In addition to a driver on an ODBC client's platform, the NSODBC product consists of a set of processes that run on the Tandem NonStop Kernel and a set of NSODBC catalogs. The primary functions of the NSODBC processes are to handle connections with clients, execute NonStop SQL/MP statements on behalf of clients, and maintain the NSODBC catalogs. Figure 8 shows the basic NSODBC process structure. The NSODBC processes are described in the following subsections.

***NSODBC Server Processes.*** NSODBC server processes do the basic SQL gateway work. An NSODBC server process translates a client's SQL dialect to NonStop SQL/MP, translates logical names to Tandem names using the NSODBC catalogs, executes NonStop SQL/MP statements, and translates data values, error codes, and catalog structures. Every client connection is serviced by a separate context-sensitive single-threaded NSODBC server process. NSODBC server processes are NonStop SQL/MP application processes; each one has its own NonStop SQL/MP Compiler (SQLCOMP) process and, when needed, its own NonStop SQL/MP Catalog Manager (SQLCAT) process (not shown in Figure 8).

***SQL Communication Subsystem Processes.*** The primary function of an SQL Communication Subsystem (SCS) process is to manage connections between clients and NSODBC server processes. In doing this, an SCS process (1) maintains communications transport protocol stacks, currently either TCP/IP or named pipes, (2) provides a uniform communication interface for the NSODBC server processes, and (3) manages NSODBC server processes by starting and stopping individual NSODBC server processes and assigning them to client connections. SCS processes are multithreaded. An SCS process maintains a thread for receiving connection requests at its network address and an additional thread for each client connection it establishes.

As a major part of providing fast client connections, SCS can start and initialize pools of NSODBC server processes before client connection requests arrive. This avoids the creation of a new NSODBC server process for every connection request and significantly improves client connection times.

The function of an SCS process in NSODBC is very similar to that of the Transaction Delivery Process (TDP) in the Tandem Remote Server Call (RSC) product (discussed in Iem and Kocher, 1992). An SCS process is driven by a configuration file that is similar to a Pathway configuration file. The SCS configuration file contains the network address on which the SCS process listens for client connections, the name of the NSODBC server's object file, the NSODBC server classes that run under the process, and the attributes of each server class. Many SCS processes can run simultaneously on the same Tandem host, each configured for a specific transport protocol and a unique network name.

***NSODBC Utility Processes.*** NSODBC Utility (NOSUTIL) processes are used to create, drop, and for most purposes, update NSODBC catalogs. Some NSODBC catalog update operations are directly carried out by NSODBC server processes. NOSUTIL processes also maintain user configuration attributes in the NSODBC catalogs.

Every NSODBC server has its own NOSUTIL process for providing catalog services. This process is started and stopped as needed, since most client operations do not require its use. In addition, every SCS process has a separate NOSUTIL process, which it calls to obtain the name of the server class to assign to a client connection and for additional user information. A NOSUTIL process can also be run as a standalone process from a TACL™ (Tandem Advanced Command Language) prompt. NOSUTIL processes are NonStop SQL/MP application processes; each has its own SQLCOMP process and, when needed, its own SQLCAT process (not shown in Figure 8).

## NSODBC Server Classes

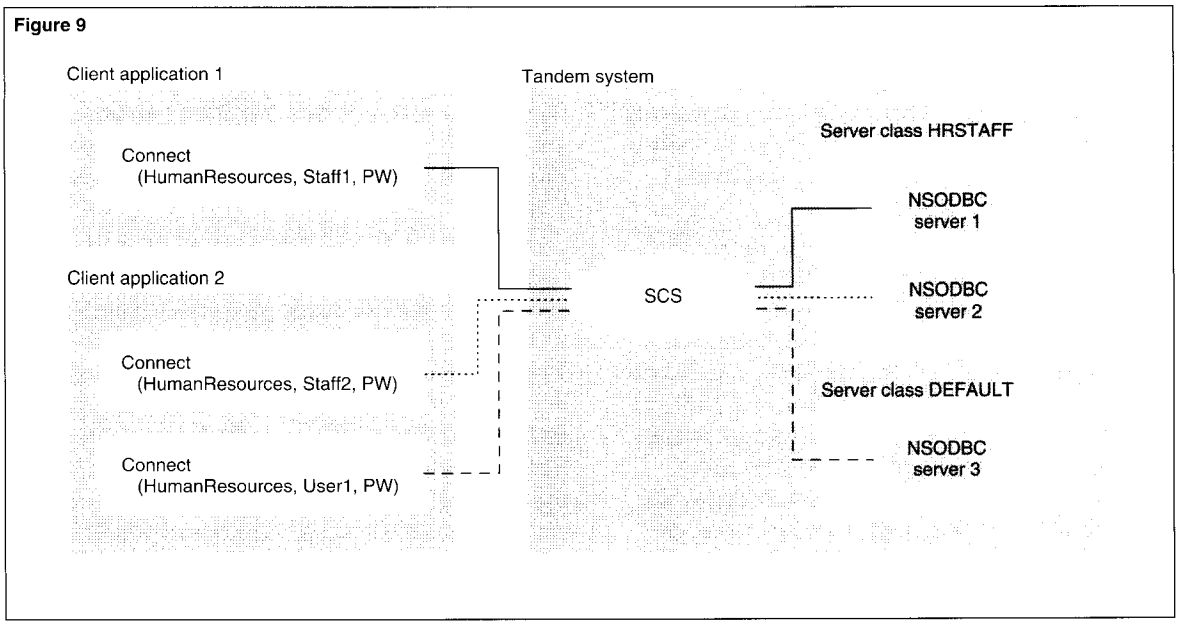
NSODBC server processes can be grouped together as NSODBC server classes under SCS. The primary function of NSODBC server classes is to avoid the creation of server processes in response to client connection requests and to preserve a cache of previously compiled SQL statements so that they are available from one client session to another. NSODBC server classes are central to establishing client connections quickly and reducing SQL execution times in NSODBC.

An individual SCS process can maintain multiple named server classes and one default server class. All NSODBC server processes belonging to a named server class execute under the same NonStop Kernel user name. Server processes belonging to the default server class can execute under different NonStop Kernel user names. Server classes are configured in the SCS configuration file, which specifies attributes for each server class, such as the number of NSODBC server processes that are to be prestarted and available for client connections and the maximum number of processes that are to be allowed under the server class.

***Named Server Classes.*** A named server class is beneficial for an individual or a group of users that initiate frequent NSODBC sessions and are likely to issue similar SQL statements from one session to the next. As an example, a named class called HRSTAFF might be configured if, in the course of a day, employees in a large human resources (HR) department repeatedly initiate NSODBC sessions and issue similar types of queries against the Employees table. In assigning attributes to HRSTAFF in the SCS configuration file, a system administrator would be likely to configure SCS to keep a pool of several NSODBC server processes available at all times for HRSTAFF connections, with a high limit on the total number of NSODBC server processes allowed under HRSTAFF.

The HR staff in the example can benefit from a named server class because prestarted server processes under HRSTAFF will provide rapid connection times and the collective use of similar queries will make reduced SQL execution times possible through NSODBC SQL statement caching (described later under "SQL Execution Times and Statement Caching").

**Figure 9.**
*Assignment of NSODBC server processes to clients.*




***Default Server Classes.*** A default server class is maintained for users that are not associated with a named server class. As in the case of a named server class, the default server class can be configured so that a specified number of NSODBC server processes are always kept prestarted and available for client connections, up to a specified maximum number of processes. Server processes in the default server class may execute under a different NonStop Kernel user name with each connection, in which case it may not be possible to take advantage of NSODBC SQL caching. As a result, establishing a connection to a server process and executing SQL may be slower than with a process in a named server class. However, even in the default server class, the benefits of caching are available if the same user connects to the default server class repeatedly.

Figure 9 illustrates the assignment of NSODBC server processes to client connections. In the figure, client application 1 sends a connection request for Staff1 to the Tandem system. SCS receives the request, finds that logical user name Staff1 is associated with server class HRSTAFF and assigns the connection to prestarted NSODBC server 1. Application 2 issues two connection requests, one for user Staff2 and the other for user User1. SCS finds that user Staff2 is associated with server class HRSTAFF and assigns Staff2 to NSODBC server 2. SCS finds that the logical user name User1 is not associated with a named server class, so it assigns the connection to a server in server class DEFAULT.

## Client Connections

A client connects to NSODBC using a network name, such as an internet protocol (IP) address for TCP/IP. The network name is part of the data source configuration in an ODBC client's ODBC.INI file. On the Tandem side, the network name is in the configuration file for an SCS process. The client's connection request arrives as a message to the SCS process configured to listen on the specified network name.

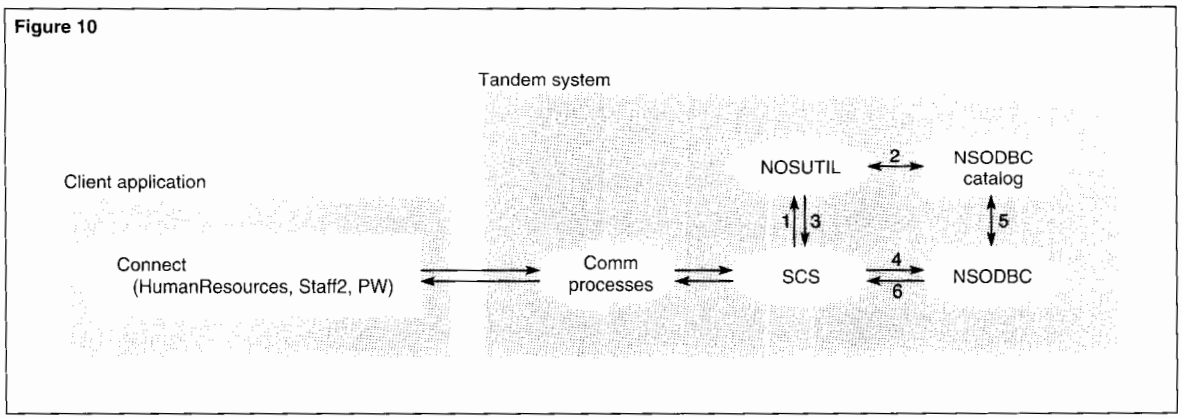



**Figure 10.**
*Client connection to NSODBC.*

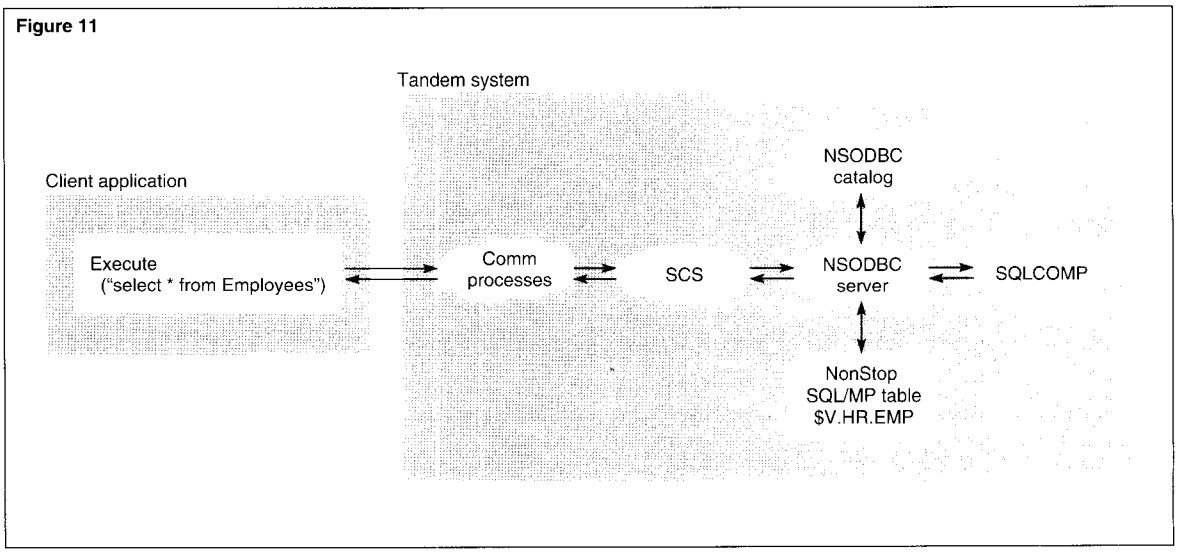
As illustrated in Figure 10, when a client connection arrives at SCS,

1. SCS extracts the logical user name in the connection request and passes it to NOSUTIL.
2. NOSUTIL accesses the NSODBC catalog to find the NonStop Kernel user name and the server class name corresponding to the logical user name.
3. NOSUTIL sends the NonStop Kernel user name and server class name back to SCS.
4. SCS selects an NSODBC server in the server class and passes the connection request to it.
5. The selected NSODBC server processes the connection request. The server uses the password supplied with the connection request to authenticate the NonStop Kernel user name associated with the request and then runs under that name. It accesses the NSODBC catalog to obtain profile information about the user and initializes itself accordingly. Depending on the user's profile, the server either runs in core mode, to support an ODBC client using core SQL, or in TSQL mode, to support a client using TSQL.
6. When the NSODBC server completes initialization, it sends a message to SCS that the connection was successful.

   SCS forwards this information to the client.

Connection processing involves substantial work and can be very time consuming. As described earlier, server classes, pools of prestarted and preinitialized NSODBC server processes maintained by SCS, are the primary means of reducing NSODBC connection times. In addition, several caching features have been implemented in order to further reduce connection times. To avoid repeating steps 1 through 3 in Figure 10 for every connection request, SCS caches the server class information it receives from NOSUTIL. To avoid obtaining user profile information at step 5 for every client connection, NSODBC server processes cache the profile information they receive from the NSODBC catalog. Thus, if the same client repeatedly connects to NSODBC, only steps 4 and 6, and password authentication at step 5, are necessary.

**Figure 11.**
*Executing an SQL query.*




## NonStop SQL/MP Execution

An NSODBC server process executes NonStop SQL/MP statements on behalf of its client. Figure 11 illustrates the flow of control for executing an SQL query.

In the figure, the client sends a message to NSODBC containing the SQL statement "select * from Employees." An NSODBC server parses the statement and reads the NSODBC catalog tables to map the logical table name Employees to the corresponding NonStop SQL/MP table name $V.HR.EMP. The server then forms a NonStop SQL/MP select statement using $V.HR.EMP and issues a PREPARE statement to invoke the NonStop SQL/MP compiler (SQLCOMP) and compile the query. The compiled statement is executed and the resulting rows of data are sent to SCS and returned to the client.

To improve SQL execution times, an NSODBC server caches object-name mapping information obtained from the NSODBC catalog. As described in a later subsection, it can also be configured to cache and reuse compiled NonStop SQL/MP statements.

## NSODBC Performance Features

NSODBC needs to perform well for both OLTP and decision support workloads. OLTP is characterized by many small transactions, with each transaction containing a few simple SQL statements that only modify or retrieve a few rows at a time. Decision support is characterized by moderately-to-highly complex queries that can generate large result sets and may take hours to execute. To support both types of workloads, NSODBC has been designed to establish connections quickly and to provide fast SQL execution and rapid data transfer back to the client. Performance in the first two areas is critical for OLTP. Optimal performance in all three areas, but particularly in data transfer to the client, is required for decision support.

## Connection Times

The response time for establishing a connection is the total time it takes for a client to connect to an NSODBC server process. This is dependent on the client's machine, the configuration of the LAN, the time needed for SCS to assign an NSODBC server process, the time it takes to start and initialize an NSODBC server process, if this is required, and the time needed for the server process to obtain client configuration information and validate the connection request. Since NSODBC can maintain server classes of preinitialized NSODBC server processes, starting and initializing a new NSODBC process is often unnecessary. Further, NSODBC server processes cache client configuration information, so this information does not need to be retrieved again when a server is used consecutively by clients that connect under the same logical name.

Connection times over TCP/IP have been obtained for clients on a SUN Sparc2 workstation and on a PC with an Intel 80486, 50MHz CPU. For a prestarted, preinitialized NSODBC server with client information already in cache, connection times were about 250 milliseconds or less. This contrasts with connection times of up to several seconds when SCS starts an NSODBC server and its SQLCOMP process on demand and the NSODBC server has to be initialized for the connection.

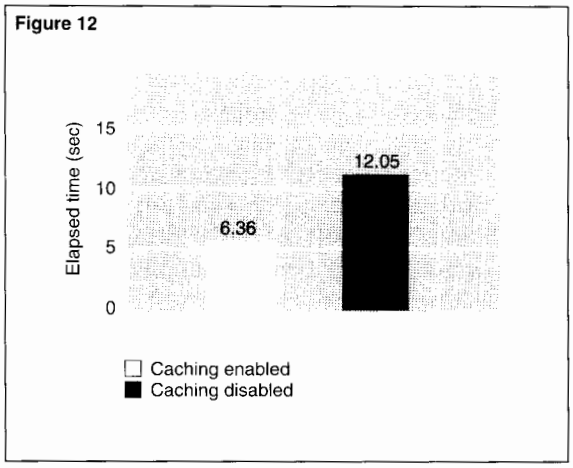
## SQL Execution Times and Statement Caching

An NSODBC server processes each SQL statement from a client by translating it into a NonStop SQL/MP statement, compiling it, and executing it. NSODBC SQL execution time is the total time required for all three operations. Compilation time is often the major part of SQL execution time. Compiling a NonStop SQL/MP statement requires opening and reading the NonStop SQL/MP catalog tables relevant to each referenced object, generating an execution plan for the statement, and creating any temporary tables that will be needed at execution time.

Compiling a NonStop SQL/MP statement can take hundreds of milliseconds, compared to an execution time of a few milliseconds for most OLTP statements. In many applications, the same set of SQL statements are executed repeatedly, so having to compile a statement every time it occurs can be costly. To avoid repeated compilations, NSODBC provides a configuration option called SQL statement caching. When statement caching is enabled, all SQL SELECT, UPDATE, INSERT, and DELETE statements that an NSODBC server sends for compilation are compiled using an SQL PREPARE statement and kept in the server's cache with the statement text as a key. Before sending a PREPARE statement, the server checks its cache to see if the statement is already compiled. If it is, the server executes the compiled version directly.

An NSODBC server maintains its cache of compiled SQL statements between connections. Thus, a series of clients running under the same NonStop Kernel user name and reusing the same set of ODBC servers can execute the same SQL statements indefinitely, without recompilation. As a safeguard to protect the integrity of compiled statements in cache, an NSODBC server process purges its statement cache as soon as an event, such as the execution of a NonStop SQL/MP CONTROL statement, occurs that could threaten the integrity of its cached compilations.

**Figure 12.**
*Effect of statement caching on SQL execution time.*




In order to increase the likelihood of a compiled statement being found in cache, cached statements are *parameterized*. This means that certain literal constants in an SQL statement are replaced with unnamed dynamic parameters. For example, if a client sent an NSODBC server the following SQL statement

```
SELECT NAME, SALARY
   FROM EMPLOYEE
   WHERE EMP_DEPT=6405
```

the statement would be translated into NonStop SQL/MP, parameterized, compiled, and stored in cache as equivalent to

```
SELECT NAME, SALARY
   FROM EMPLOYEE
   WHERE EMP_DEPT=?
```

It would then be executed with the parameter set to 6405.

If two client SQL statements translate to the same parameterized SQL statement, they can both execute the same compiled statement. Thus, if an NSODBC server process had the preceding query in cache and then received the query

```
SELECT NAME, SALARY
   FROM EMPLOYEE
   WHERE EMP_DEPT=9999
```

it would be able to execute the cached query directly, with its parameter set to 9999.

The performance benefits of statement caching can be significant. For example, a study of one application found that total SQL execution time for a transaction containing 19 SELECT statements and 1 UPDATE statement was reduced by almost 50 percent when statement caching was enabled (see Figure 12).

Another series of measurements were made on an OLTP workload in which each transaction contained three UPDATE statements and one INSERT statement. In this case, statement caching reduced SQL execution times by more than 50 percent, from an average of 0.71 seconds per transaction to 0.30 seconds.

## Data Transfer Rates

For queries that return a large amount of data to the client, the data transfer rate becomes an important performance issue. The effective data transfer rate is a function of message size and the number of messages sent between client and server, and the amount of traffic on the LAN. To increase data transfer rates, NSODBC implements *bulk fetch* (described below) and allows message flows to the client to overlap with NonStop SQL/MP execution.

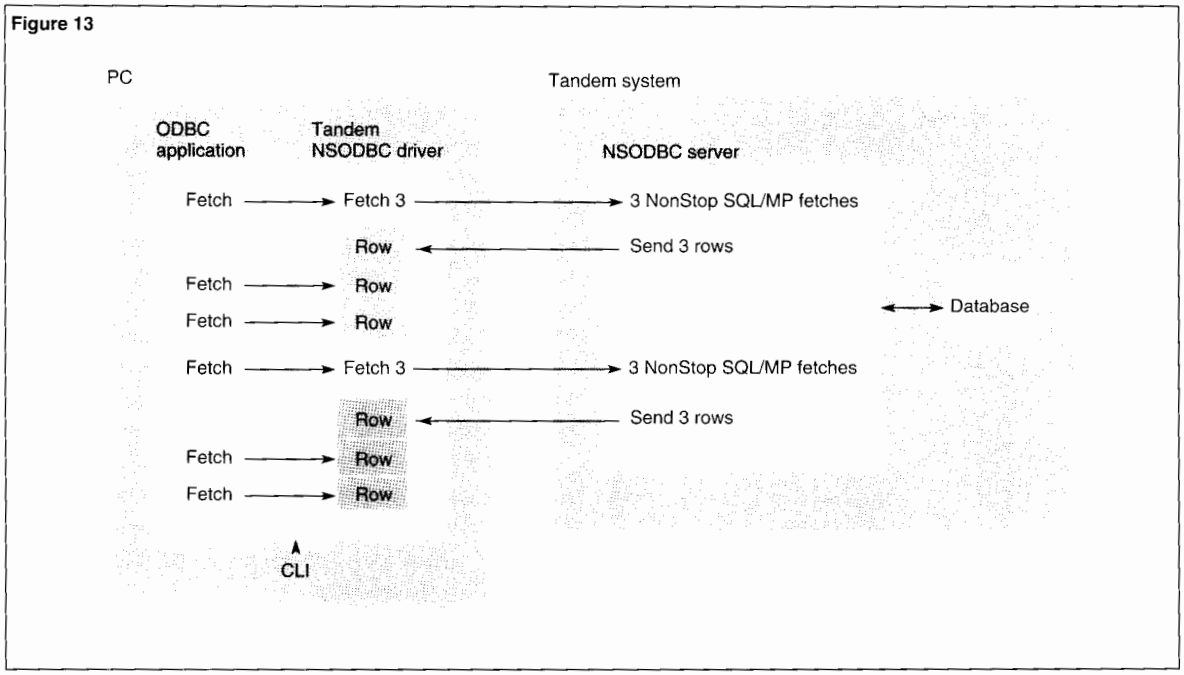



**Figure 13.**
*Bulk fetch for a core SQL client.*

In NonStop SQL/MP, to read row values from a SELECT statement into application variables, a series of SQL FETCH statements associated with a cursor is used to retrieve one result row at a time from the database. However, in a client/server environment it is not efficient to send data row by row over a network. As mentioned above, to improve data transfer rates in a client/server environment, NSODBC implements bulk fetch. When bulk fetch is used, multiple result rows can be returned in each message to the client. Bulk fetch can only be used when the client application does not modify the fetched rows with a cursor-positioned UPDATE or DELETE statement. Bulk fetch is always used for TSQL applications, because TSQL does not support cursors. It is provided as a configuration option for ODBC core SQL clients.

When bulk fetch is used for an ODBC client, on the first fetch associated with a cursor, the Tandem ODBC driver calculates the number of result rows it can fit into its internal buffer, currently set at 4 kilobytes, and issues a request to the NSODBC server process to fetch that many rows. The server returns the rows to the ODBC driver's buffer on the client system. After the first fetch, succeeding client fetches obtain rows directly from the ODBC driver's buffer until the last row in the buffer has been fetched. At this point, the ODBC driver issues another bulk fetch to the NSODBC server process. Figure 13 illustrates bulk fetch with an ODBC driver buffer that can hold three rows at a time.

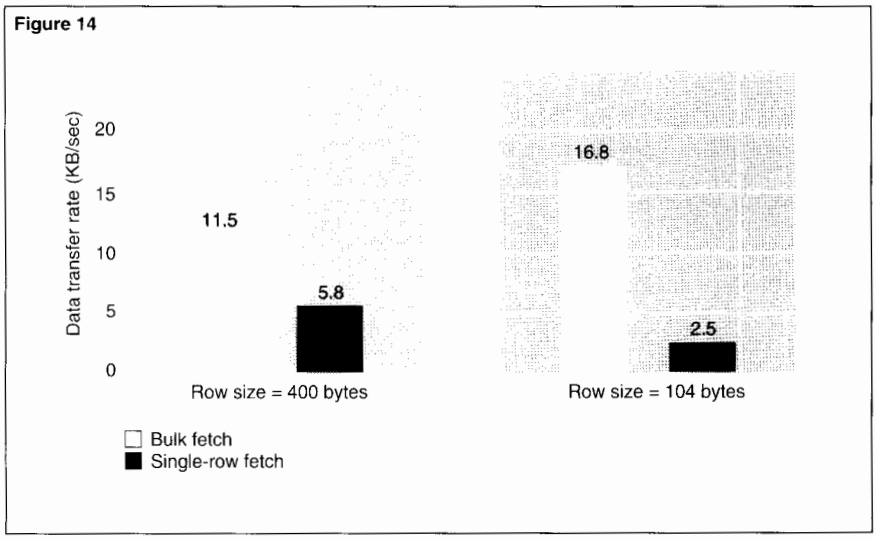



**Figure 14.**

*Comparison of data return rates for bulk fetch and single-row fetches.*

Figure 14 compares data transfer rates for bulk fetch and single-row fetches for a client PC with an Intel 80486, 50MHz CPU. In the first comparison, with result rows of 400 bytes, the data transfer rate for bulk fetch was almost twice that of single-row fetches. In the second comparison, with result rows of 104 bytes, the data transfer rate for bulk fetch was more than six times that of single-row fetches. The benefits of bulk fetch are greater when rows are small, since this allows more rows per message to be fetched into the ODBC driver's buffer.

## NSODBC Configuration

NSODBC is provided with default settings that allow it to be used almost immediately after installation with a minimum of configuration. For example, it comes with a default server class and a default user profile that uses the default server class. Clients can connect using special logical user names that are derived from existing NonStop Kernel user names and do not require configuration of the NSODBC catalogs.

Of course, NSODBC can also be configured to take advantage of features such as named server classes and SQL statement caching. In some cases, a useful strategy is to start an application using NSODBC default settings and, as experience is gained, gradually change the configuration to add features and improve performance.

For ODBC clients, the NSODBC product includes a Windows application and additional ODBC applications to assist in installing and configuring the Tandem driver and establishing connections to NSODBC server processes. These also provide measurement of connection response times and, for purposes of debugging, ODBC activity tracing.

## Managing NSODBC

NSODBC can be managed in a number of ways. A Tandem system administrator can define and alter user profiles, use NOSUTIL commands to verify the integrity of NSODBC catalogs, and use the Tandem Subsystem Control Facility (SCF) product to query SCS for the state of server classes and NSODBC server processes. In addition, NSODBC provides a tracing facility that can obtain information from SCS and NSODBC server processes. This information includes the current state of a process, the operations it performs, and the data it handles.

Attributes in user profiles that are relevant to management functions include the maximum allowable execution cost for an SQL statement and the maximum size of the SQL statement cache. The maximum execution cost for SQL statements is useful for resource governing. Before execution of an SQL statement, the maximum allowed cost is compared with the NonStop SQL/MP compiler's cost estimate for the statement. If the compiler's estimated cost is greater than allowed, the statement is not executed and an error message is returned.

NSODBC supports a pass-through feature that allows an application, running in either TSQL or ODBC core SQL mode, to include NonStop SQL/MP statements for execution. The NonStop SQL/MP statements are executed by NSODBC server processes directly, without translation, and make it possible to use special NonStop SQL/MP features such as parallel execution and partitioned tables. Pass-through statements can also be used to alter NSODBC server attributes and execute NOSUTIL commands.

## Conclusion

ODBC specifies a widely endorsed API for connecting to a database and executing SQL. ODBC drivers support the API and provide transparent connectivity to a wide variety of proprietary database servers. This promotes both open portability and open interoperability for applications.

NSODBC provides access to NonStop SQL/MP on Tandem systems for both ODBC and DBLIB applications in an easily managed environment. A number of NSODBC features are designed to optimize performance, which allows NSODBC to be used for both OLTP and decision support applications.

### References


Cooperstein, H. 1992. An Overview of Client/Server Computing on Tandem Systems. *Tandem Systems Review*. Vol. 8, No 3. Tandem Computers Incorporated. Part no. 89803.

*Data Management: SQL Call Level Interface (CLI)*. 1993. X/Open Document No. P303.

Iem, M. and Kocher, T. 1992. Implementing Client/Server Using RSC. *Tandem Systems Review*. Vol. 8, No 3. Tandem Computers Incorporated. Part no. 89803.

*NonStop ODBC Server Manual*. 1994. Tandem Computers Incorporated. Part no. 106609.

*Programmer's Reference*. 1992. Microsoft Open Database Connectivity Software Development Kit. Version 1.0.

Rohner, T. 1994. Extending the Client/Server Model With Object-Oriented Technology. *Tandem Systems Review*. Vol. 10, No 1. Tandem Computers Incorporated. Part no. 104396.

Schlansky, W. and Schrengohst, J. 1993. The DAL Server: Client/Server Access to Tandem Databases. *Tandem Systems Review*. Vol. 9, No 1. Tandem Computers Incorporated. Part no. 89804.

Slutz, D. 1990. Gateways to NonStop SQL. *Tandem Systems Review*. Vol. 6, No 2. Tandem Computers Incorporated. Part no. 46987.

*Tandem SQL Server Gateway Manual*. 1989. Tandem Computers Incorporated. Part no. 46070.



### Acknowledgments

We would like to thank the reviewers of this article for their valuable comments and the entire NSODBC team, including development, QA, product management, publications, and performance, for their efforts.



**Haleh Mahbod** is a member of the SQL-connectivity development group and worked on development of the NSODBC server process, focusing on performance features. She joined Tandem in 1984 and was a member of the NonStop SQL development group from 1985 to 1992. She helped design and implement the NonStop SQL compiler and the NonStop SQL preprocessor.

**Donald Slutz** has worked at Tandem for ten years on NonStop SQL and SQL connectivity. He previously worked at a database startup company and at IBM Research on database systems and performance modelling and analysis.

# Enhancing Availability, Manageability, and Performance With NonStop TM/MP

To achieve continuous availability in online transaction processing (OLTP) applications, users must be able to rely on supporting system software such as the Tandem™ TMF™ (Transaction Monitoring Facility). By providing transaction management and protecting the integrity of user data, the TMF subsystem plays a critical role among Tandem's OLTP software products. If TMF becomes unavailable, even briefly, the OLTP applications that depend on it cannot function, and end users temporarily cannot carry on their business.

The NonStop™ Transaction Manager/Massively Parallel (TM/MP) subsystem, the new version of TMF, includes several innovative design features that improve its availability, make it easier to manage, and enhance its performance. To be released with the D30 release of the NonStop™ Kernel, NonStop TM/MP is an entirely new product, not an enhancement or upgrade of TMF. Existing user applications will, however, be able to use NonStop TM/MP without alteration. In addition, TMF will continue to be supported under D20, and NonStop TM/MP will include migration software to provide an orderly transition from one subsystem to the other.

This article describes the new features of NonStop TM/MP and discusses three areas in which NonStop TM/MP offers significant advantages over TMF: continuous availability, operability and manageability, and performance. The article includes two scenarios illustrating some of those advantages.

Readers should be familiar with the concepts and capabilities of TMF in particular or transaction processing managers in general. For readers unfamiliar with these concepts, the following section provides a brief review.

## Review of TMF Concepts and Capabilities

Fundamental to both the TMF and NonStop TM/MP subsystems is a programmatic construct called a *transaction*. A transaction is an explicitly delimited operation, or set of related operations, that changes the content of a database from one consistent state to another.

The database operations within a transaction are treated as a single unit. Either all of the changes performed by the transaction are made permanent (the transaction is committed) or none of the changes is made permanent (the transaction is aborted). If a failure occurs during the execution of a transaction, whatever partial changes were made to the database are undone automatically, thus leaving the database in a consistent state.

Before a transaction permanently commits its changes to the database, information about the database rows or records affected by the transaction is written to an *audit trail.* An audit trail is a series of files containing TMF audit and control records.

The TMF autorollback feature, called *volume recovery* in NonStop TM/MP, recovers data volumes that had one or more tables or files open when a media or system failure occurred. It is automatically invoked whenever the subsystem is started or whenever a data volume is enabled. The autorollback or volume recovery process uses information in the audit trail to reapply database changes that were not yet committed or aborted at the time of the failure.

The TMF rollforward feature, called *file recovery* in NonStop TM/MP, can recover a database table or file that was accidentally purged, reestablish the structural integrity of a damaged database table or file, or return the database to a consistent state as of a particular point in time. It uses online dumps (archived copies of database tables or files) and audit dumps (archived copies of audit trail files) to restore the specified tables and files to a consistent state. The operator explicitly invokes rollforward or file recovery for particular database tables or files (when needed).

## New Features in NonStop TM/MP

The new features in NonStop TM/MP enhance several aspects of transaction management. In particular, NonStop TM/MP does the following:

- Supports large database configurations, high transaction throughput, and fast recovery through the use of multivolume audit trails, overflow audit volumes, and restore audit volumes.
- Allows users to reconfigure virtually all aspects of the NonStop TM/MP environment dynamically without stopping the subsystem or any applications using it.
- Allows users to monitor and manage the transaction management environment through the use of a graphical user interface (GUI).

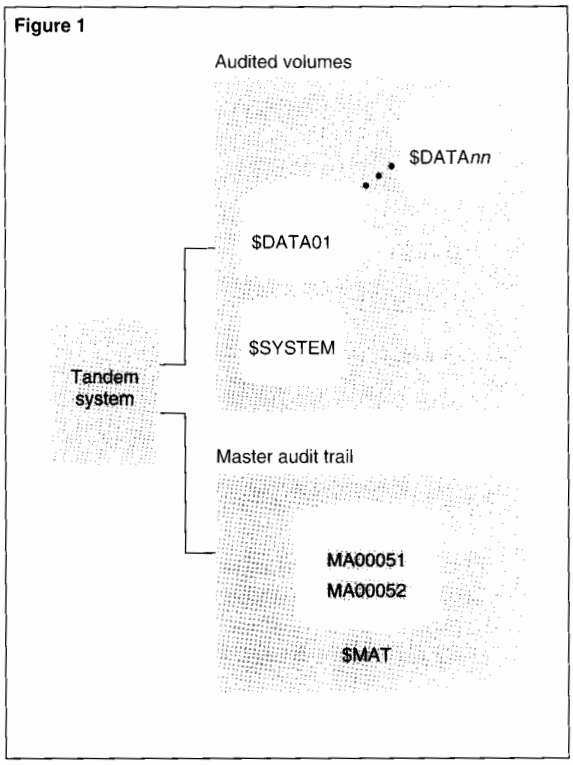



**Figure 1.**
*TMF environment. For the master audit trail volume ($MAT), MINFILES=2 and MAXFILES=5.*

### Multivolume Audit Trails

NonStop TM/MP supports significantly larger application throughput than does the preceding TMF subsystem; it also enables growing applications to remain available continuously. Each TMF or NonStop TM/MP system has one master audit trail (MAT) and can include up to 15 auxiliary audit trails. The TMF subsystem requires each audit trail to exist on a single disk volume, as shown in Figure 1. With NonStop TM/MP, each configured audit trail can occupy up to 16 disk volumes. Figure 2 shows an example of a NonStop TM/MP environment with a two-volume audit trail.

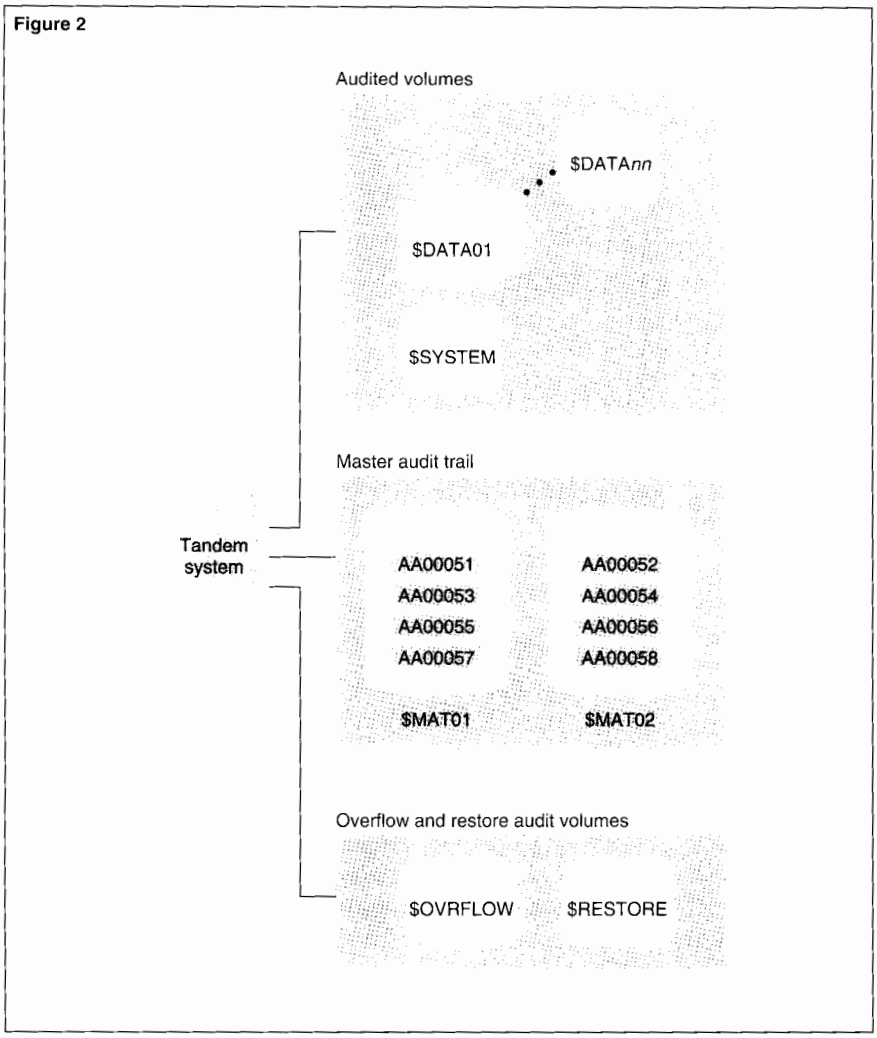



**Figure 2.**
*NonStop TM/MP environment. For the master audit trail volumes ($MAT01 and $MAT02), FilesPerVolume=4.*

Previously, a large application generating hundreds of kilobytes of audit information per second during peak hours was physically limited to a maximum of 16 audit trail volumes (assuming the maximum-sized environment of a MAT and 15 auxiliary audit trails). NonStop TM/MP increases that limit to 256 volumes (again assuming the maximum-sized environment), thereby accommodating virtually unlimited growth in business activity. For information about configuring audit trails, refer to the *NonStop TM/MP Configuration and Planning Guide* (1994), a new manual for the NonStop TM/MP subsystem.

## Overflow Audit Volumes

When configuring a NonStop TM/MP audit trail, users can designate one or more disk volumes to be used for audit if all of the active audit trail files become filled. These are called *overflow audit volumes.* There can be up to 16 such volumes per audit trail and they can be any disk volumes in the system (including data volumes or active audit volumes).

If a TMF audit trail becomes full, the subsystem and all applications using it cease to function. That situation cannot happen easily with NonStop TM/MP. When a NonStop TM/MP audit trail reaches a user-configurable overflow threshold, the subsystem automatically copies the oldest audit trail file from its active audit volume to an overflow audit volume. NonStop TM/MP then renames the copied file and makes it available for receiving new audit information.

## Restore Audit Volumes

When configuring a NonStop TM/MP audit trail, users can also designate one or more disk volumes, called *restore audit volumes*, to be used for receiving copies of audit trail files that must be restored from audit dumps as part of a file-recovery operation. There can be up to 16 such volumes per audit trail and they can be any disk volumes in the system (including data volumes or active audit volumes).

Previously, the operator could redirect audit restoration to some degree. With NonStop TM/MP, the operator can add and delete restore audit volumes dynamically and thereby completely avoid any space or head contention on an active audit volume.

### Dynamic Online Reconfiguration

Users can reconfigure almost all aspects of the NonStop TM/MP environment online without stopping the subsystem or any applications using it. Once NonStop TM/MP is started, the operators can dynamically add or delete active audit volumes, overflow audit volumes, restore audit volumes, and data volumes. They can also change the number of audit trail files per volume, move data volumes from one audit trail to another, and alter the configured value of any control threshold. In a TMF environment, similar types of changes require, at the least, stopping and restarting the subsystem.

Online reconfiguration allows operators to respond quickly and easily to changing or unexpected conditions, thus avoiding circumstances that might otherwise adversely affect application performance. Only two operations cannot be performed after NonStop TM/MP is started: changing the size of the audit trail files and adding or deleting an auxiliary audit trail.

Although the user should choose a reasonable audit-trail file size when initially configuring the NonStop TM/MP environment, making an error is not necessarily critical. If the user specifies a file size that is too small, rollover from one audit trail file to another will occur more frequently than it would otherwise (as will audit dumping, if enabled). Specifying a file size that is too large generally does not present a problem. (It could, for example, cause audit dumps to exceed a single tape reel.) If the configured file size is too small, the user can compensate subsequently for the miscalculation by dynamically increasing the number of files per volume or by explicitly disabling and enabling audit dumping. If the configured file size is too large, the user can explicitly force rollovers, when desired, by issuing TMFCOM NEXT AUDITTRAIL commands (and thereby contain audit dumps to a single tape reel).

The subsequent adding of auxiliary audit trails is a more serious issue, but one that can be avoided. The user can, for example, initially establish one or more minimally-configured auxiliary audit trails (one active audit volume, two audit files per volume) with no data volumes attached to them.

### GUI for Managing NonStop TM/MP

TM View, the NonStop TM/MP GUI, provides operators with a visual means of interpreting the ongoing status of the TM/MP subsystem, including icons for controlling important aspects of the NonStop TM/MP environment. TM View complements the TMFCOM interactive command interface, providing an alternative approach to performing many query and control operations.

The TMF subsystem informs operators of subsystem status solely by displaying character output in response to TMFCOM INFO and STATUS commands. With TM View, operators can intuitively understand NonStop TM/MP status just as one does by looking at the indicators and gauges on an automobile dashboard. One bar graph, for example, shows operators how much of the audit trail is currently in use. They can tell at a glance if the present transaction workload is pushing the audit trail capacity toward the overflow threshold or, beyond that, toward the point at which NonStop TM/MP will not allow new transactions to begin.

## NonStop Availability

If the TMF subsystem is down, all applications that use it are also down. In a production environment, it is imperative that users' business applications, and all subsystems that support them, be operational virtually all the time. NonStop TM/MP provides five significant advantages over TMF with regard to subsystem availability:

- Fewer CPU halts.
- Fewer cold loads.
- Fewer SYSGEN operations.
- Online reconfiguration.
- Audit trail overflow.

## CPU Halts

TMF halts occur rarely; most occur because the Transaction Monitor Process (TMP), a component of TMF, detects an internal error. Although the backup TMP takes over, all other processes in the halted TMP's CPU are abruptly terminated, and the processor remains unavailable until it can be reloaded.

The NonStop TM/MP TMP does not halt its CPU when an internal error occurs; it thereby increases CPU availability and reduces CPU reloads. If an internal error occurs, the primary TMP process issues an ABEND (abnormal end) procedure call, the backup TMP process automatically takes over, and the primary TMP's CPU is unaffected.

## Cold Loads

A TMF crash occurs when any audit trail goes down or the TMP process pair is lost due to double CPU failures or nonrecoverable internal errors. Previously, the subsystem would become nonfunctional after a TMF crash, and the operator had to perform a cold load to restart it. In the meantime, all applications that used the TMF subsystem were also down.

If NonStop TM/MP crashes, the operator merely issues a START TMF command (after the problem is resolved) to restart the subsystem. An application using the Tandem Pathway transaction processing system, for example, can survive the crash with only a brief interruption of transaction services if the application is designed to retry operations.

A common cause of TMF crashes, for example, is the accidental issuance of a PUP DOWN command for an active audit trail volume. With NonStop TM/MP, that can never happen, because PUP DOWN commands are rejected for active audit volumes.

## SYSGEN Operations

Most TMF interim product modification (IPM) releases include the TMP program. When installing a new IPM on a TMF system, the system administrator must perform a SYSGEN operation and a cold load to replace the TMP. This necessitates stopping the subsystem and all applications that use it.

With NonStop TM/MP, users can install new IPMs while maintaining maximum availability of their applications. A SYSGEN operation and cold load are required only to replace the TMF library (TMFLIB) or monitor (TMFMON) process. Those two program files, however, are seldom included in IPMs. To replace the TMP, the operator merely duplicates the new TMP object file from the IPM to the TMF subvolume, issues a STOP TMF, ABRUPT command, and then issues a START TMF command. (One can do this during off hours with minimal impact on the user applications).

## Online Reconfiguration

As mentioned earlier, dynamic online reconfiguration in NonStop TM/MP increases the availability of the underlying system software. To add new data volumes to a TMF configuration, the operator must perform a clean shutdown of all applications that use TMF, stop and restart TMF, and then restart the applications to get TMF to recognize the new volumes and allow them to participate in transaction activity. Further, to move a data volume from one audit trail to another, the operator must perform the following steps:

1. Stop all applications that use TMF.
2. Issue a STOP TMF command.
3. Issue an INITIALIZE TMF command.
4. Reconfigure the subsystem.
5. Issue a START TMF command.
6. Make new online dumps of all files protected by TMF.
7. Restart the applications.

With NonStop TM/MP, on the other hand, system administrators can connect new physical disk drives to the system, take existing drives down for maintenance, change volume definitions (add or delete active audit, overflow audit, restore audit, or data volumes), and move database volumes from one audit trail to another without stopping the subsystem or interrupting application activity.

Furthermore, with NonStop TM/MP, operators know how much of the audit trail disk space is in use at any given time. In addition, operators can dynamically expand or contract the capacity of the audit trail in response to changing conditions. If they notice that a particular trail is nearing the overflow threshold, for example, they can add another active audit volume or increase the number of audit trail files per volume.

### Audit Trail Overflow

As mentioned earlier, the TMF subsystem and all applications using it cease to function when an audit trail becomes full. TMF allocates audit trail files as they are needed; it does not know ahead of time if there will be enough space on the audit trail volume for the next file. If other files are stored on a TMF audit trail volume, for example, they could eliminate space needed by TMF under maximum operating conditions.

With NonStop TM/MP, the system will rarely cease to function from a lack of audit trail space. In addition to using overflow audit volumes, NonStop TM/MP preallocates the necessary space on all active audit volumes for the designated number of files per volume. If the user accidentally specifies a files-per-volume value that is too large, NonStop TM/MP will refuse to perform the operation and instruct the user to enter a smaller value. This practice of preallocating audit trail files guarantees that the maximum number of audit trail files configured by the user will always be available for use. In Figure 2, the environment has been configured with two audit trail volumes and four files per volume. Once configured, space for all eight files always exists.

## Operability and Manageability

NonStop TM/MP includes many design features that greatly facilitate its operability and manageability. They fall into five general categories:

- Audit trail management.
- Data volume management.
- Volume and file recovery.
- Disaster recovery.
- Online monitoring.

### Audit Trail Management

On occasion, the TMF MINFILES and MAXFILES configuration parameters have been accidentally misused to specify a greater number of audit trail files than can fit on a single disk volume. NonStop TM/MP eliminates those parameters. Instead, NonStop TM/MP has users explicitly specify the number of audit files per active audit volume, the audit trail file size, an overflow threshold value, and a *begin transaction disable* threshold.

The overflow threshold specifies the percentage of audit trail use, from 50 to 100 percent, at which the oldest audit trail files will be copied to an overflow audit volume. The *begin transaction disable* threshold specifies the percentage of audit trail use, from 50 to 100 percent, at which the starting of new transactions will be prohibited.

As mentioned previously, NonStop TM/MP greatly enhances an operator's ability to monitor and manage audit trails. It automatically preallocates the maximum configured number of audit trail files, and the TM View GUI dynamically tracks and displays audit trail space usage.

## Data Volume Management

The TMF subsystem assumes that all disk drives on the system other than the configured audit trail volumes are audited data volumes. One consequence of this assumption is that, whenever the operator starts the subsystem, TMF attempts by default to perform autorollback on virtually every disk drive on the system, even those on which no audited activity has occurred. (Each time the subsystem is started, the user is required to tell TMF which disk drives not to start, and the system manager must explicitly exclude those drives from the TMF configuration.)

When configuring a TMF audit trail, the user explicitly designates which data volumes are associated with it. (In addition, one audit trail must include the USAGE * attribute, indicating that it will be responsible for all other disk volumes on the system.) These definitions are permanent; they cannot be altered except by stopping the applications, stopping and initializing TMF, reconfiguring the audit trails, and restarting TMF and the applications. Although the user can employ constructs called phantom drives to add disk drives to the system, or move volumes around within the system, TMF will not recognize a new or moved volume until TMF has been stopped and restarted.

With NonStop TM/MP, data volumes are added explicitly after the subsystem is running. The user can dynamically add or delete data volumes, or move them from one audit trail to another, without stopping the subsystem or interrupting application activity. Furthermore, NonStop TM/MP immediately recognizes new or moved disk volumes without the subsystem having to be stopped and restarted.

Because TMF uses static bit maps, the total number of disk processes per CPU that can participate in transaction activity is limited to 62. Because the primary and backup disk processes are all represented in this count, this translates to a limit of 31 data volumes per CPU. NonStop TM/MP, however, uses linked lists, thereby imposing no limit on the number of data volumes per CPU.

## Volume and File Recovery

By default, TMF messages indicating the results of autorollback and rollforward recovery are sent only to the operator console. If the console is busy, these messages can roll off the screen and be lost. In addition, if the CPU in which a recovery process is executing fails before recovery is complete, the recovery process must start all over again from the beginning. For example, if a rollforward recovery operation had been working for several hours when its CPU suddenly failed, all its work would be lost, and a new rollforward process would have to start at the beginning again.

NonStop TM/MP sends the results of both volume recovery and file recovery to the Event Management Service (EMS) log, which provides a permanent record of all event messages. In addition, all operations are assigned identifiers to support status monitoring and EMS event filtering. The operators can direct audit file restoration to disk volumes other than the active audit volumes and thereby avoid head contention with the generating of new audit information.

If the CPU in which a file recovery process is running fails, the TMP automatically starts another file recovery process in another CPU. The new process fetches the latest restart position from disk and continues the recovery operation with no more than 10 minutes' duplication of work already done.

Finally, for large file recovery operations involving the mounting of many audit dump and online dump tapes, operators can use the PLAN ONLY option of the RECOVER FILES command to obtain a list of all the required tapes. This option allows them to fetch all necessary tapes before initiating the actual file recovery operation, rather than repeatedly fetching individual tapes in response to tape-request messages.

### Disaster Recovery

With TMF, critical subsystem configuration information is maintained both in the audit trail and on the $SYSTEM.TMF subvolume. If the local system fails, it is difficult to recover the database on a different node using TMF dumps. (This kind of TMF recovery is not supported by Tandem.) The only viable way to recover the database under these circumstances is to use RDF™ (Remote Duplicate Database Facility) software, which requires a remote backup system that physically duplicates the primary production system.

With NonStop TM/MP, most state information is maintained in the audit trail rather than in the $SYSTEM.TMF subvolume. This makes it possible to recover from the loss of the $SYSTEM volume, audit trail disk volumes, or the entire system. Tandem supports the recovery of NonStop TM/MP databases on nodes other than the primary production system. The procedures for doing so are documented in the *NonStop TM/MP Operations and Recovery Guide* (1994), a new manual for the NonStop TM/MP subsystem.

### Online Monitoring

Previously, users could monitor the health of the TMF subsystem by running several utility and management applications, including the TMFCOM process, ViewPoint™ operations console facility, File Utility Program (FUP), and Disk Space Analysis Program (DSAP). Additionally, the spooler might be used to monitor the outcome of autorollback or roll-forward recovery.

The new TM View interface graphically displays a variety of dynamically-updated NonStop TM/MP health indicators. It also displays the status of long-running transactions, allows operators to log information before executing critical commands, and makes it possible to monitor and manage multiple NonStop TM/MP systems from a single console.

## Enhanced Performance

Because it was developed as a new product, NonStop TM/MP includes many design features that greatly increase its performance over that of the previous TMF subsystem. The most significant performance improvements fall into three categories:

- Audit trail throughput.
- Network throughput.
- Volume and file recovery.

### Audit Trail Throughput

For several reasons, NonStop TM/MP generates audit data more efficiently, with less interference from other NonStop TM/MP activity, than does TMF. When two active audit volumes are configured for the same NonStop TM/MP audit trail, for example, they are used alternately for the currently active audit trail file. (The first audit trail file is on the first volume, the second file on the second volume, the third file on the first volume, and so forth.) When there are three or more active audit volumes, they are used in a round-robin fashion. This means that, if audit dumping is enabled, the audit dump process will not compete with audit creation for use of the disk head.

Similarly, when a disk drive other than one containing an active audit volume is configured as a restore audit volume, audit restoration for file recovery will not compete with audit creation for use of the disk head. In addition, NonStop TM/MP uses a new algorithm that makes rollover from one audit trail file to the next virtually invisible from a performance standpoint.

## Scenario 1: The Need for More Audit Trail Space

**Problem:** The audit trail volume is nearly full, because the user's business has expanded, and, as a result, the database has grown and transaction activity has increased faster than originally anticipated. Assume that the TMF MAXFILES attribute value or the NonStop TM/MP FilesPerVolume attribute value has already been raised to the point at which the entire audit trail volume is being used.

**Response Under TMF:** The operators must shut down all applicaions that use the TMF subsystem, issue a STOP TMF command, initialize and reconfigure the subsystem with an auxiliary audit trail, issue a START TMF command, and then restart the applications.

**Response Under NonStop TM/MP:** The operators add another active audit volume, by issuing an ALTER AUDITTRAIL command, with no interruption of application activity.

## Scenario 2: The Need to Add Data Volumes

**Problem:** One or more disk volumes have been added to the system and must now be added to the TMF or NonStop TM/MP configuration as audited data volumes.

**Response Under TMF:** The operators must shut down all applications that use the TMF subsystem, issue a STOP TMF command, issue a START TMF command, and then restart the applications. If the new data volumes are to be associated with an audit trail other than the one for which USAGE * is configured, the operators must explicitly add the data volumes to the configuration while the subsystem is topped.

**Response Under NonStop TM/MP:** The operators add the data volumes to the NonStop TM/MP configuration, by issuing an ADD DATAVOLS command, with no interruption of application activity.

### Network Throughput

Users should experience an improvement in network transaction performance with NonStop TM/MP because of three enhancements. NonStop TM/MP provides a new transaction state-change algorithm, a reduction in the number of interprocessor messages, and the *boxcarring* of network messages (buffering what used to be many individual intersystem messages into a single intersystem message).

### Volume and File Recovery

With NonStop TM/MP, file recovery is performed as fast as, or faster than, creation of the original audit data, and volume recovery will not take more than 10 minutes. A major design goal for NonStop TM/MP was for volume recovery and file recovery to perform 10 to 20 times faster than their TMF counterparts. Some preliminary testing, however, has shown volume recovery to perform even more than 20 times faster than autorollback. In addition, if either the volume recovery or file recovery process fails before it is completed, no more than 10 minutes of work previously done will have to be repeated when the process is subsequently restarted.

## The Role of NonStop TM/MP in OLTP

NonStop TM/MP is an integral part of Tandem's massively parallel approach to OLTP. It can monitor thousands of complex transactions sent by hundreds of users to a common database. The database can be distributed over scores of disks on a system or across many systems in a network. The database can consist of tables and views created by the NonStop SQL relational database management system, files created by the Enscribe record manager, or any combination thereof.

Moreover, NonStop TM/MP provides a foundation relied on by other Tandem software products that contribute to transaction processing solutions. TMF is required by two existing Tandem products, NonStop SQL and RDF, and is compatible with, or transparent to, all others.

NonStop SQL/Massively Parallel (SQL/MP) will use NonStop TM/MP to protect SQL catalogs and the file labels of tables, views, indexes, and programs; previous releases of NonStop SQL use TMF. Although users can choose whether or not individual tables and views are to be audited, all tables and views must reside on audited volumes because their file labels are audited. The design of NonStop TM/MP makes possible several availability features introduced in SQL/MP, notably physical database operations such as partitioning and moving tables online. Troisi (1994) discusses these operations in the article "NonStop Availability and Database Configuration Operations" in this issue of the *Tandem Systems Review*.

The RDF product, which provides disaster recovery for OLTP production databases, also uses TMF. RDF monitors all database changes audited by a TMF master audit trail on a primary system and applies those changes to a copy of the database on a remote duplicate backup system. The existing version of RDF will continue to use TMF. A version to be released in the near future, however, will be compatible with both TMF and NonStop TM/MP.

In addition, several new products, some still being developed and others soon to be released, also require NonStop TM/MP. NonStop TUXEDO, for example, is a UNIX transaction processing monitor that is being adapted to run on NonStop Kernel systems in the Open System Services (OSS) environment. It will use NonStop TM/MP to manage transaction activity and maintain database consistency.

## Conclusion

NonStop TM/MP, a cornerstone of Tandem's massively parallel approach to OLTP, provides several new facilities that enhance the availability, manageability, and performance of production database applications. Its most significant new features are multivolume audit trails, overflow audit volumes, improved recovery performance, dynamic online reconfiguration, and a GUI. Existing applications can use NonStop TM/MP, and take advantage of its enhancements, immediately and without any alteration.

### References

*NonStop TM/MP Configuration and Planning Guide.* 1994. Tandem Computers Incorporated. Part no. 85811.

*NonStop TM/MP Operations and Recovery Guide.* 1994. Tandem Computers Incorporated. Part no. 81472.

Troisi, J. 1994. NonStop Availability and Database Configuration Operations. *Tandem Systems Review*. Tandem Computers Incorporated. Part no. 104400.

**Mala Chandra** was the program manager responsible for the development of NonStop TM/MP. Since joining Tandem in 1982, she has worked in education, field support, marketing, and development.

**Dave Eicher** has been a software technical writer at Tandem since 1980. During the past two years he participated in the team effort to design and write the new manuals set for NonStop TM/MP.

# RDF Enhancements for High Availability and Performance

The Tandem™ RDF™ (Remote Duplicate Database Facility) product is used to monitor changes made to a database on a local system and maintain a duplicate copy of that database on a remote system. Because RDF applies changes to the database on the remote system as soon as they are detected, it keeps the database on the backup system continuously up to date with changes made by a business application on the primary system. This makes it possible to switch business operations from the primary system to the backup system in a matter of minutes, should a sudden disaster cause the primary system to fail. Additionally, if a planned shutdown of the primary system is necessary, business applications on the primary system can be stopped and immediately restarted on the backup system to access the replicated database.

This article provides a general overview of RDF and describes important availability and performance enhancements made in the two most recent interim product modifications (IPMs), AAP (March, 1993) and AAQ (June, 1993). It then covers future enhancements such as replication of a database to multiple backup sites and a new means of providing stable access for query processing on the database at the backup system. Finally, the article presents an amended procedure for carrying out a planned switchover from a primary system to a backup system—a useful facility for ensuring little interruption to the business application, even during major hardware reconfigurations at the primary site or a move of the primary data center.

For additional information on RDF, two earlier articles on the subject have been published in the *Tandem Systems Review*. Guerrero (1991) provides a general overview, while Senf and Jongma (1992) provide a more specialized discussion, titled "RDF Synchronization." Both articles remain useful, although not fully up to date. This article does not assume prior knowledge of RDF. The reader should, however, be familiar with the Tandem TMF™ (Transaction Monitoring Facility) product.

## Overview of RDF

RDF maintains a replicated database by monitoring changes made to audited files[1] on RDF-protected volumes on the primary system and applying those changes to corresponding volumes on the backup system. More specifically, on the primary system, an RDF *extractor* process reads the master audit trail (MAT), a log maintained by TMF of all database transactions that affect audited files, and sends any audit associated with RDF-protected volumes to an RDF

[1]Throughout this article, *file* is used to refer both to audited files created under the Tandem Enscribe record management system and to tables created through the Tandem NonStop™ SQL relational database management system.

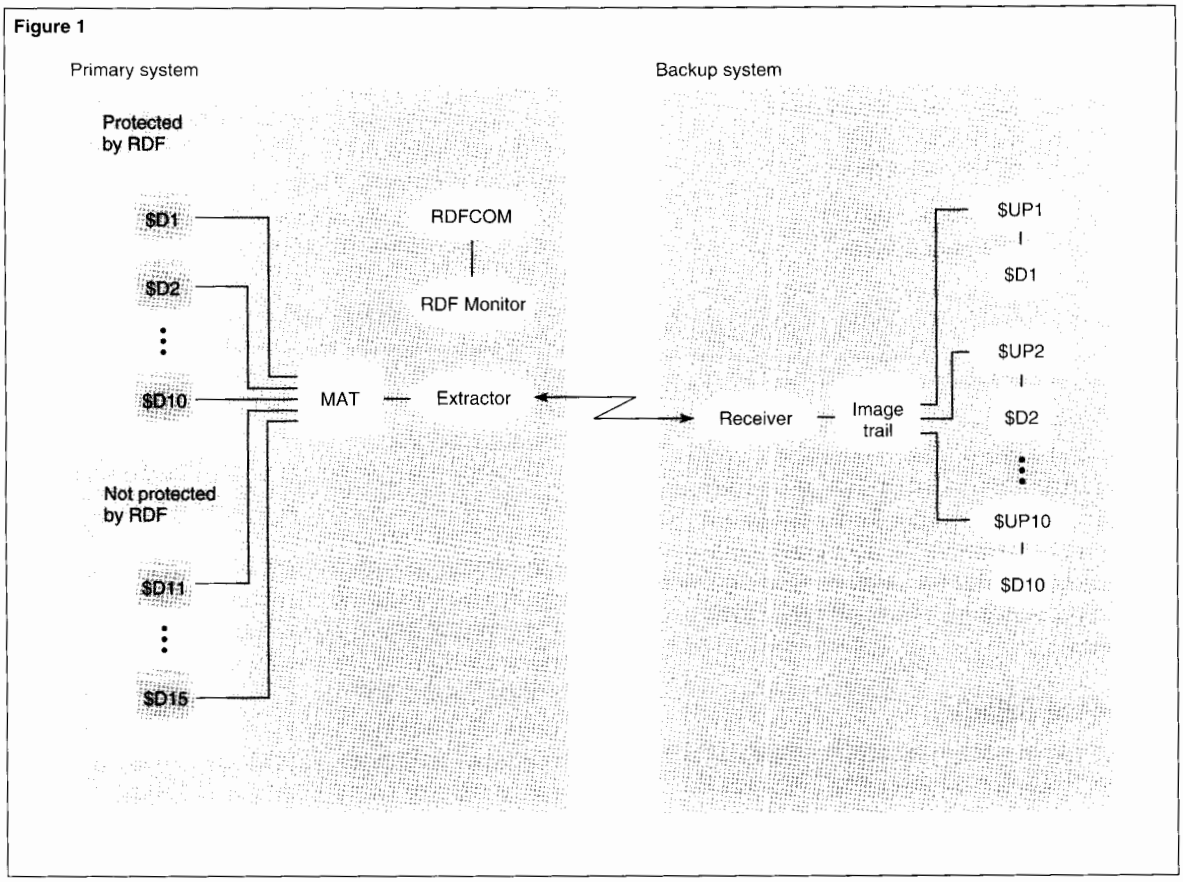



**Figure 1.**
*Basic RDF configuration.*

*receiver* process on the backup system. The receiver process writes audit from the extractor to an *image trail*. RDF *updater* processes on the backup system read the audit from the image trail and apply only audit associated with committed transactions to the backup database. Audit associated with aborted transactions on the primary system is never applied to the database on the backup system. Each RDF-protected volume on the primary system has its own updater process on the backup system that is responsible for applying audit to the corresponding volume on that backup system. Figure 1 illustrates a basic RDF configuration.

In Figure 1, there are 15 audited volumes on the primary system, $D1 through $D15. However, only volumes $D1 through $D10 are configured for RDF protection. Audit for all 15 volumes is sent to the MAT file. The RDF extractor process reads the MAT file and sends audit associated with volumes $D1 through $D10 to the RDF receiver process on the backup system. In the figure, the receiver writes the audit to a single image trail. As described later, IPM AAQ makes it possible to have multiple image trails for storing audit.

Updaters $UP1 through $UP10 read the image trail to find audit associated with committed transactions and apply the audit to volumes $D1 through $D10 on the backup system. For example, when updater $UP1 finds committed audit for files associated with volume $D1 on the primary system, it applies the audit to the corresponding files on $D1 on the backup system.

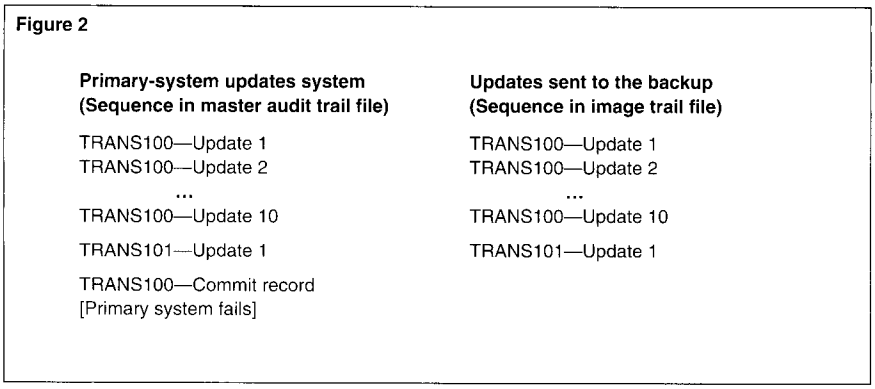



**Figure 2.**

*Audit on the primary and backup systems at the time of a primary system failure.*

Figure 1 shows two processes on the primary system that have not been mentioned, RDFCOM and the RDF monitor. RDFCOM provides the user interface for issuing RDF commands. The RDF monitor responds to user commands specified through RDFCOM, such as start and stop operations, and it monitors all other RDF processes.

## Unplanned Outages

An unplanned outage typically occurs as the result of a sudden disaster that prevents the database on the primary system from being used. The classic purpose of the RDF product is to make rapid recovery possible by maintaining a replicated database on a backup system. When the primary system is unexpectedly affected by a disaster, one can shift operations to the replicated database on the backup system after having the RDF updaters bring the backup database to a consistent state. This is done by starting RDFCOM on the backup system and executing an RDF *takeover* operation.

An RDF takeover operation ensures that all audit associated with committed transactions, and whose commit audit records are actually in the image trail, are applied to the backup database. If the status of a transaction is unknown on the backup system because the commit or abort record was not sent by the time the disaster brought the primary system down, the transaction is considered to be aborted, and no audit associated with the transaction is applied to the backup database.

Because only audit for committed transactions is applied to the backup database, when an updater reads an audit record, it must determine whether the record belongs to a committed transaction. Under normal conditions, if the outcome of the transaction is not in the image trail, the updater simply waits until the expected commit or abort record arrives. If, however, the primary system has unexpectedly been brought down because of a disaster, the outcome of some transactions may never be known, as illustrated in Figure 2.

In the example of Figure 2, a disaster has brought down the primary system immediately after the commit record for transaction 100 was written in the MAT, but before the RDF extractor was able to send the commit record to the backup system. For transaction 101, a single update was logged in the MAT and sent to the backup system, but the primary system was brought down before the transaction was resolved. Under normal processing, an updater on the backup system would read update 1 for transaction 100 and wait for the commit or abort record to arrive at the backup system before determining if it should apply the record. In this example, however, the outcomes of transactions 100 and 101 would never be known at the backup system.

The RDF takeover operation prevents an updater from waiting indefinitely for an outcome that will never arrive. When the command for a takeover is given, updaters treat all transactions whose outcomes are not available as aborted transactions. This guarantees that only transactions known with certainty to have been committed on the primary system are applied to the backup database, and that all audit associated with those transactions is applied. Therefore, in the example of Figure 2, no audit associated with transactions 100 and 101 is applied to the backup database.

Typically, the extractor sends audit within a second after it has been entered in the MAT file, so that a minimum number of transactions are lost when a disaster strikes down the primary system. The RDF takeover operation is discussed further in Guerrero (1991); recovering lost transactions is discussed under "Recovering Updates Not Transmitted During a System Failure" in Senf and Jongma (1992).

### Planned Outages

RDF can be very useful when a planned shutdown of the primary system is necessary. For example, one may need to bring the system down for the purpose of installing new hardware, performing periodic maintenance, or for any number of other reasons. When faced with such a situation, it may be undesirable to stop the business application for a length of time. Using RDF, it is only necessary to stop the business application momentarily, carry out an RDF switchover operation, and restart the business application on the backup system (see "Carrying Out a Planned Switchover," later in this article). When the primary system is ready for use again, one can use RDF to bring the primary system's database up to date with changes made to the backup database while the primary system was shut down. When the primary system's database has caught up with the database on the backup system, one can perform another switchover operation from backup system to primary system and then restart the business application again on the primary system using the original configuration.

## Complete Restartability After a Failure

Prior to the AAP IPM, if an unexpected event, such as a double CPU failure or system failure, caused either the RDF extractor or receiver process to stop unexpectedly, the cost of recovery was high. If the failure occurred during normal processing, one had to stop the business application, stop TMF, resynchronize the databases, initialize RDF, restart TMF, and then restart RDF.[2] Resynchronizing the databases meant making a copy of the database on the primary system and restoring it on the backup system. For systems with even a moderately-sized database, this meant significant outage time for the business application.

If the backup system failed during an RDF takeover operation, the consequences were serious. Without manual intervention by a Tandem RDF support specialist, the takeover could not be restarted.

[2]Under the first C30 version of RDF, the TMF product had to be reinitialized before RDF could be initialized. This required stopping the business application and taking new online dumps of the database. Since the AAJ IPM in 1991, reinitialization of TMF has not been required. Now, to initialize RDF, one must only stop the TMF product momentarily and initialize RDF at the TMF shutdown timestamp.

The AAP IPM makes RDF fully restartable if an RDF process is unexpectedly stopped during either normal or takeover processing. Now, if a failure causes an RDF crash during normal processing, one just needs to issue the RDFCOM START RDF command. If a failure occurs during RDF takeover processing, one simply needs to reissue the RDFCOM TAKEOVER command. To achieve restartability, the AAP IPM completely revised the way RDF maintained *context*, or restart information, on disk.

## Changes in Maintaining Context

The RDF extractor maintains restart information in a context record contained in a context file on the primary system; the receiver and the individual updater processes maintain their context records in a context file on the backup system. When RDF is started, each process uses its context record to determine where to begin. Specifically, the extractor uses its context to determine where to begin reading in the MAT; the receiver uses its context to determine where in the image trail to write new audit sent by the extractor; and each updater uses its context to determine where in the image trail to start reading audit for possible application to the backup database.

Prior to the AAP IPM, when the extractor and receiver processes started, each read its context record from disk and then immediately deleted it. Each process, while running, maintained context in memory while running, but not on disk. When RDF was subsequently shut down, such as in response to an RDFCOM STOP RDF command, each process wrote its latest context from memory to disk and then completed its shutdown. If, however, one of these processes unexpectedly stopped, it would not have been able to write its latest context from memory to disk. This implementation made it easy for RDF to determine the nature of its last shutdown. If the context record for either process was missing when one attempted to start RDF, then the last shutdown resulted from an unexpected stoppage.

Because the extractor and receiver context records shared no common information that would make it possible to determine whether their context records were in harmony with each other, and because database corruption could result if RDF was restarted when these context records were not in harmony, each process deleted its context record on disk, after starting, in order to prevent RDF from subsequently being restarted after an unexpected stoppage.

Now, from the AAP IPM onward, neither the extractor nor the receiver deletes its context record after starting. Moreover, the receiver frequently updates its context record on disk with new restart information. With this restart information available on disk, RDF can be restarted after any type of unexpected stoppage without requiring database resynchronization or RDF reinitialization.

***Extractor Context Under AAP IPM.*** As stated above, the extractor no longer deletes its context record when starting. Instead, it reads its context record during startup to determine its starting position in the MAT and then immediately writes that record back out to disk, turning on a new CRASHOPEN flag in the context record. When shutting down, the extractor updates its context and turns this CRASHOPEN flag off. Thus, if the extractor stops unexpectedly, its CRASHOPEN flag on disk will still be turned on, thereby preserving the information that the process had failed the last time it was running. As with all previous releases, the extractor does not update its context on disk while running.

***Receiver Context Under AAP IPM.*** The AAP IPM added several new fields to the receiver's context record. The MAT position of each audit record is now sent with that record to the backup system. This enables the receiver to keep track of the MAT position of the last complete audit record it received, and a field in its context record preserves this information. It also has a CRASHOPEN flag.

The CRASHOPEN flag works in the same way as the corresponding flag does in the extractor's context record. When the receiver starts, it reads its context from disk to determine where in the image trail to start writing new audit and then it immediately writes the record back to disk, turning on the CRASHOPEN flag. When the receiver shuts down, it updates its context on disk to preserve restart information and it also turns off the CRASHOPEN flag. Therefore, if the receiver stops unexpectedly, the CRASHOPEN flag will still be turned on in the receiver's context record on disk. As stated earlier, the receiver updates its context record on disk frequently.

## RDF Recovery After an Extractor Failure on the Primary System

If a double CPU failure or some other event causes the extractor to stop unexpectedly, the RDF monitor shuts down all remaining RDF processes. Once the problem on the primary system has been corrected, RDF can be restarted with the RDFCOM START RDF command.

When RDFCOM starts work on this command, it first checks the CRASHOPEN flags in the context records of the extractor and the receiver. In this case, it finds that the extractor's CRASHOPEN flag is on, indicating that the extractor had stopped unexpectedly the last time it ran. RDFCOM then initiates a recovery operation to reset the extractor's restart information. This operation entails obtaining the receiver's context record and updating the extractor's restart position, based on the MAT location of the last audit record written in the image trail. Therefore, when the extractor is restarted, it begins reading audit in the MAT based on the position of the last audit record to be stored in the image trail. This provides an absolute guarantee that no audit is skipped when RDF is restarted. When the recovery operation has been completed, RDFCOM proceeds with the normal START RDF operation.

## RDF Recovery After a Receiver Failure on the Backup System

If a double CPU failure, a system crash, or some other problem causes the receiver to stop unexpectedly, the RDF monitor shuts down all remaining RDF processes. Once the problem has been corrected and the backup system is operating normally, RDF can be restarted with the RDFCOM START RDF command. When RDFCOM starts work on this command, it first checks the CRASHOPEN flags in the context records of the extractor and the receiver. In this case, it finds that the receiver's CRASHOPEN flag is on, indicating that the receiver had stopped unexpectedly the last time it ran. RDFCOM then performs a different set of recovery operations.

First, because the extractor's context may no longer be correct, RDFCOM resets the extractor restart position in the MAT, following the same protocol used when restarting after an unexpected extractor stoppage. Second, it adjusts the latest image file in the image trail, possibly even reducing that file's EOF. Third, it creates a new image file, so that when the receiver is restarted, it will write new audit it receives from the extractor into this file. Finally, it may need to adjust the restart positions of some updaters. Having completed these operations, RDFCOM proceeds with the normal START RDF operation.

### Recovery After an RDF Takeover Failure

Prior to the AAP IPM, the takeover operation was not restartable, and to restart one needed the manual intervention of an RDF support specialist. With the AAP IPM, the RDF takeover operation is restartable. If an RDF takeover is not completed, for whatever reason, one merely needs to reissue the RDFCOM TAKEOVER command.

The AAP IPM has also improved the information logged at the end of the takeover operation. Previously, the outcome of a takeover operation could be ambiguous, and one could have thought the outcome successful when it was not. To provide a completely clear indication of the outcome, a new TAKEOVERCOMPLETED flag was added to the updater context record. This flag is only turned on when the updater is involved in an RDF takeover operation and has processed all audit in the image trail for which it is responsible. Therefore, if the TAKEOVERCOMPLETED flag of every updater is turned on at the end of the operation, the takeover was a success. If one or more flags are not turned on, the takeover operation was not completed successfully.

In preparing to log its shutdown message at the end of the takeover operation, the receiver examines the context records of the updaters. If this new TAKEOVERCOMPLETED flag is turned on for every updater, the receiver indicates in its shutdown message that the takeover succeeded. If, however, it finds that one or more TAKEOVERCOMPLETED flags did not get turned on (e.g., one or more updaters were unexpectedly stopped), the receiver's shutdown message indicates that the takeover only partially succeeded.

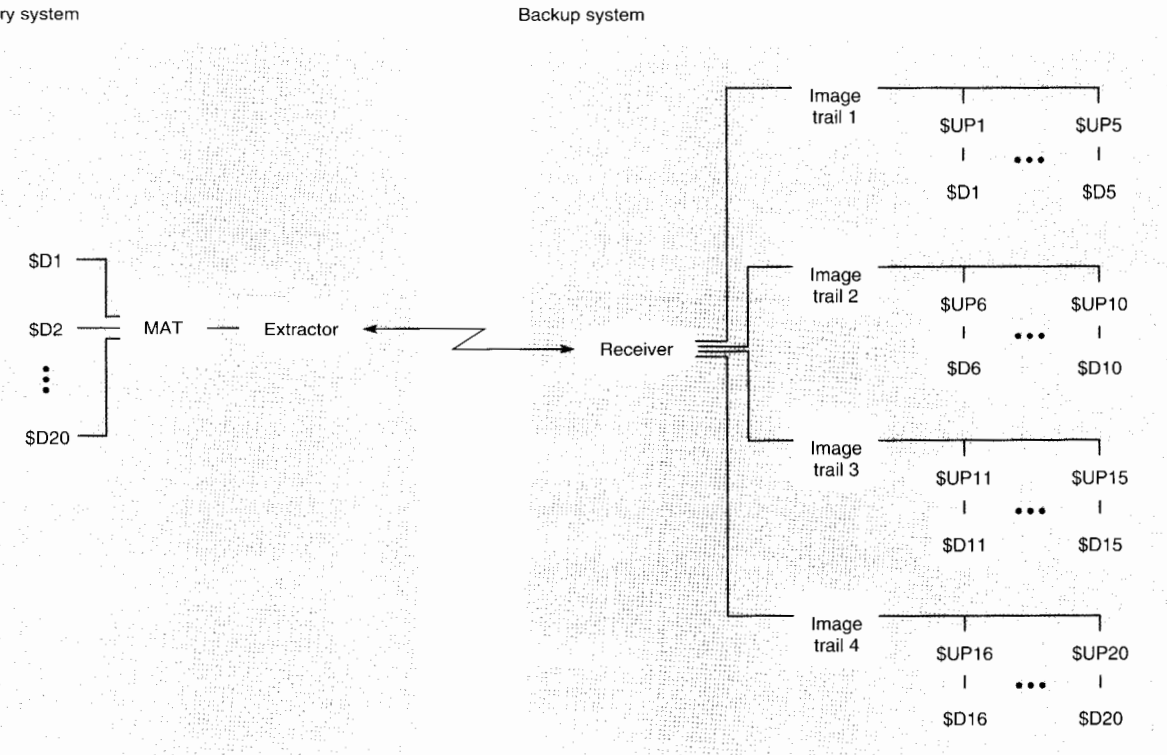
If a takeover is partially completed, the database manager simply needs to reissue the RDFCOM TAKEOVER command. This starts a new RDF Monitor on the backup system, which in turn starts a new receiver and a full set of updaters. For those updaters that had previously completed their takeover processing, their restart locations are already at the end of the image trail, and they will shut down immediately. The updaters that still have processing to do will resume at their last restart locations. When all updaters have stopped, the receiver will once again check their context records and again log the outcome of the operation. If another updater again stopped unexpectedly, the takeover operation can be restarted.

## Improved Updater Performance

The current release of RDF, the AAQ IPM, significantly improves updater performance when a large number of primary-system volumes are protected by RDF. Previously, if one used RDF to protect 15 to 20 volumes on a primary system that had a high audit generation rate, the maximum combined throughput rate for the updaters on the backup system was limited to approximately 45 kilobytes per second. If additional volumes required RDF protection, the combined updater throughput rate declined significantly. Thus, to achieve a reasonable updater throughput rate when the audit generation rate on the primary system was high, the maximum practical number of volumes that could be protected by RDF was limited to roughly 20, even though RDF allows 64 volumes to be configured.

This limit in updater performance was due to excessive contention on the image-trail volume. Previously, only a single image trail existed, all audit sent to the receiver was written to that trail, and each updater had to search that trail for audit it needed to apply. Consider an example in which 20 volumes on a primary system are protected by RDF. The rate at which the updaters read new audit is extremely high, and each updater must compete against the other 19 updaters attempting to read new audit. If more updaters are added, the contention is that much greater and the combined updater throughput is further decreased.

Figure 3



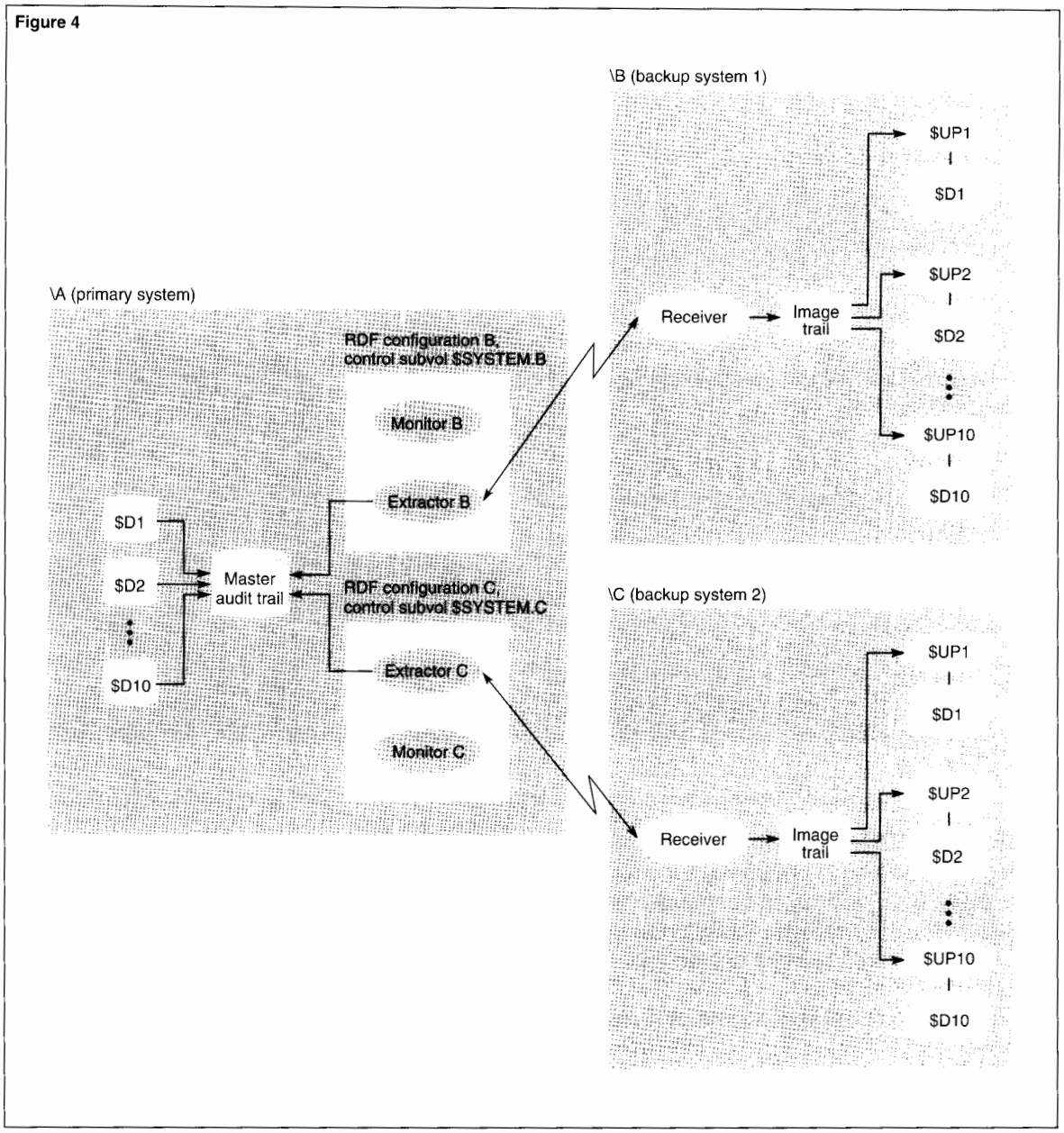

The AAQ IPM eliminates this contention by making it possible to configure up to seven image trails, each on a separate volume. With multiple trails, one can assign a different set of updaters to each trail, thereby reducing the number of readers of an image trail. With the contention eliminated, the combined updater throughput is significantly increased. Figure 3 illustrates an RDF configuration with multiple image trails.

In Figure 3, 20 volumes on the primary system, $D1 through $D20, are protected by RDF. On the backup system, there are 20 updaters and 4 image trails, with 5 updaters to each image trail. In this configuration, only 5 updaters contend with each other for access to their specific image trail. With a single image trail, all 20 updaters would be in contention.

Using a pair of Cyclone™ systems, measurements were taken to determine the increase in updater performance. The maximum rate at which the extractor sent audit to the receiver was limited to roughly 120 kilobytes per second. This figure represented the maximum theoretical rate at which updaters might be able to apply audit, because they can only process as fast as the audit is written to the image trail. As stated previously, the combined updater throughput for an RDF configuration with 20 updaters on a single image trail was roughly 45 kilobytes per second. With 12 updaters on a single trail, the combined throughput rose to roughly 88 kilobytes per second. With 8 updaters on a single trail, the rate rose to roughly 110 kilobytes per second. The cost of the receiver managing either one trail or up to seven trails was found to be virtually the same.

**Figure 3.**
*RDF configuration with multiple image trails.*

**Figure 4.**
*Primary database with two independent backup databases.*




Therefore, if one were to spread 20 updaters over two image trails, one should attain a combined updater throughput rate that is at least two times faster than the old rate with this many updaters. More significantly, the practical limit to the number of volumes that can be protected by RDF is raised from 20 to 64. If one needs 60 volumes to be RDF protected with a higher updater performance rate, one should configure six image trails, with 10 updaters configured per trail.

## Forthcoming RDF Features

At present, a database on a given local system can only be protected by a single RDF system that replicates that database at a single remote system. A future RDF IPM will make it possible to replicate a database on multiple backup systems. In addition, this next version of RDF will provide triple contingency support. Triple contingency support is meant for users who cannot afford to run their database application without RDF protection. If they lose their primary system to some disaster, they need to be able to restart their business application at the backup system and have it protected by a new RDF system with the shortest possible delay. Also, a new means of achieving stable access for query processing on the backup system will be provided.

### Multiple Backup Databases

Some RDF installations use their backup database to handle a high volume of query processing. Multiple independent backup databases make it possible to improve query response times by distributing queries over several systems. Another benefit of two or more independent backup systems is that if one backup system fails, the primary database is still fully replicated on another system that can take over database operations if necessary.

Figure 4 illustrates an environment in which a database on a primary system is replicated on two backup systems by two independent RDF systems.

### Triple Contingency Support

This future IPM will also provide triple contingency support for those who cannot afford to run their business application without RDF protection after they have had to perform an RDF takeover. For example, suppose one were running an RDF system from node \A to node \B, and \A was suddenly destroyed by a natural disaster. After completing an RDF takeover at \B, one would want to restart the business application at \B and configure a new RDF system to replicate database changes at \B to a new backup database on node \C.

To achieve this currently, one must first perform the costly task of synchronizing the databases by moving a copy of the database on \B to \C. If the database is even moderate in size, the time required for database synchronization can be very long, and one cannot start the business application until synchronization is complete.

With triple contingency support, one could use RDF to achieve rapid database synchronization, so that one could restart one's business application with full RDF protection very quickly after an RDF takeover operation. Essentially, one would configure two independent RDF systems to protect a database, with each RDF system replicating the database changes on its own backup node. In our example, one would have one RDF system running from \A to \B (configuration 1) and another running from \A to \C (configuration 2).

If \A subsequently fails, one would execute the RDF takeover operation at \B and at \C. Because the two RDF systems run independently from each other, however, the probability is high that, at the time when \A crashed, one backup database might be ahead of the other. This could happen because some audit may have reached one of the two backup systems but not the other. In the example, the extractor of configuration 1 might have just read and sent new audit to its receiver, but the disaster struck before the extractor of configuration 2 could read and send the same audit. In this situation, some audit would be missing at \C.

For triple contingency support, several new RDFCOM commands will be provided that will (1) move the missing audit from the one backup system to the other, (2) fix up context, and (3) enable a second takeover operation at the node whose database is behind that on the other node. In our example, the audit would be moved from \B to \C. When the second takeover is completed at \C, the database at \C will be synchronized with that at \B. Depending on how much audit is missing between the two nodes after the takeover operations, the time required to execute the new RDFCOM commands and the second RDF takeover operation might involve only a few minutes. Thus, with triple contingency support, one could restart one's business application with full RDF protection in a very short period of time.

### Stable Access on the Backup Database

Currently, because updaters apply committed audit to the backup database independently from each other, query processing on the backup system's database can be done only with browse access. To achieve stable access, only two reliable ways are available. The most reliable way is to stop TMF on the primary system and wait for RDF to shut down. The less reliable way is to disable transactions on the primary system and wait for all updaters to reach the end of the image trail. Both methods, however, require stopping new transactions at the primary system.

The future IPM will provide a new means of achieving stable access for query processing on the backup system, one that will not interfere with database activity on the primary system. The RDFCOM STOP UPDATE command will be enhanced to accept a timestamp. If one issues the RDFCOM STOP UPDATE command without a timestamp, the command will work the same way it always has. If, however, one includes a timestamp in the command, the updaters will apply all audit that was committed on the primary system up to the time specified in the timestamp and then shut down. They will not apply any audit that was committed after that timestamp. This guarantees that when the updaters have all stopped, they will have all processed the same transactions, and the database will be available for stable access. When updating is restarted, the updaters will apply all committed audit they could not apply the last time because those transactions were committed after the specified timestamp.

## Carrying Out a Planned Switchover

Guerrero (1991) describes how to carry out a planned RDF switchover on the backup system and then return processing to the primary system. The revised and updated switchover procedure given below should be used in its place. In the steps given for carrying out the procedure, \A is the primary system and \B is the backup system. It is assumed that a planned shutdown of \A is necessary, for example to install new hardware, and that one wants the business application to be down for as little time as possible.

The revised switchover procedure is as follows:

1. Stop the business application that accesses the primary database.
2. Issue the TMFCOM STOP TMF command on the primary system (\A).

   TMF will stop as soon as all outstanding database transactions have either been committed or aborted. When it stops, TMF writes a shutdown record to the audit trail. Subsequently, RDF will shut down when all updaters on the backup system (\B) reach the TMF shutdown record in \B's image trail file. At this point, the two databases have identical contents.

3. If TMF has been running on the backup system, issue the TMFCOM STOP TMF command.

   The stoppage is only momentary, and stopping TMF generates a TMF shutdown record in the audit trail on the backup system.
4. Restart TMF on the backup system.
5. Use the FUP ALTER or SQLCI ALTER TABLE command to turn on audit flags for files in the backup database.

   Once the business application starts to use the backup database, files in the database must be audited so that TMF protection applies to them. A predefined obey file for turning on audit flags in the backup database is highly recommended. It makes this task fast and simple.
6. Restart the business application to access the database on \B.
7. Once \A is back online, restart TMF on \A.
8. Turn off audit flags for all RDF-protected database files on \A. This prepares the files for the RDF updaters when running from \B to \A. As at Step 5, a predefined obey file is highly recommended.
9. On \B, issue the RDFCOM INITIALIZE RDF command, specifying the TMF shutdown timestamp obtained in Step 3.

   This initializes the RDF extractor on \B to begin reading the MAT file at the first record after the TMF shutdown record generated in Step 3. Since this record in the MAT file was created before the business application generated any updates on \B, the extractor is guaranteed not to miss any relevant audit.
10. Configure RDF to run from \B to \A.
11. Issue the RDFCOM START RDF command. RDF now sends database changes from \B to \A.
12. When the extractor on \B has reached the point where it is sending current audit to \A, stop the application on \B and carry out another planned switchover on \A.

The planned switchover repeats Step 1 through Step 6, except that this time, the business application is switched back to \A and RDF once again runs from \A to \B, as at the beginning.

## Conclusion

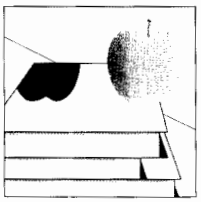
The Tandem Remote Duplicate Database Facility (RDF) product makes it possible to maintain a backup database for use in the event the primary database becomes unavailable. Enhancements to RDF since its initial C30 release provide faster update performance on the backup system during normal processing and faster takeover operations when there are unplanned outages. In a future version, RDF will make it possible to have multiple independent backup systems, triple contingency support, and a new means of attaining stable access on the backup system's database.

#### References

Guerrero, Jorge. 1991. RDF: An Overview. *Tandem Systems Review*. Vol.7, No. 2. Tandem Computers Incorporated. Part no. 65248.

Senf, Wouter and Jongma, Frans. 1992. RDF Synchronization. *Tandem Systems Review*. Vol.8, No. 2. Tandem Computers Incorporated. Part no. 69848.

**Malcolm Mosher, Jr.**, joined Tandem in 1982 as a software developer. He joined the RDF development team in 1988 and has been responsible for all RDF design and implementation since 1992. In addition, Malcolm is responsible for much of the design and implementation of the new high-performance Tandem NonStop TM/MP backout and volume recovery processes, as well as the low-level code of the new TM/MP File Recovery process. He has a Ph.D. in Egyptology from the University of California, Berkeley (1990).



## Tandem Education

The following paragraphs provide highlights of the latest education courses offered by Tandem. To sign up for a class or to order an independent study program (ISP), users should call 1-800-621-9198. Full descriptions of all available courses and ISPs appear in the *Tandem Education Course Catalog* and on InfoWay.

### Integrity FT Architecture and Product Overview

This one-day lecture-and-lab course is designed for those interested in a thorough understanding of the design, theory of operation, features, and functions of Tandem's Integrity FT systems. Students learn the essentials of the triple modular redundancy (TMR) architecture of Tandem's Integrity FT system and become familiar with the communications, database, and language products offered on the Integrity FT system. Students also review the features and functions of the NonStop-UX operating system, Tandem's implementation of UNIX System V, Release 4.

### Integrity System Management Suite (ISMS)

This is a one-day lecture-and-lab course meant for UNIX analysts, support engineers, and system administrators. The course provides the best introduction to ISMS, a state-of-the-art graphical user interface (GUI) used to manage hardware components, monitor system resources, monitor the diagnostic event log, administer System Exerciser and Confidence Tests (SECT), and read postscript documents including the NonStop-UX manual pages. Extensive lab exercises reinforce the skills needed to administer an Integrity FT System using ISMS. After completing this course, students are well-equipped to use the ISMS interface in their administrative activities.

### Introduction to Enterprise Internetworking

This independent study program (ISP) provides a comprehensive introduction to new networking architectures and products. In this ISP, the student learns about the hardware, software, and types of applications that make up the enterprise internetworks of the 1990s. The program traces the evolution of networks from hierarchical, mainframe-centered environments to fully distributed internetworks of local and wide area networks. Upon completing this ISP, the student understands the reasons for internetwork evolution and for the advent of the client/server model. The student also understands the impact of desktop computing and the management challenges associated with heterogeneous networks. The time required to complete the program is about four hours.

*The Technical Information and Education department is an annotated list of new Tandem education courses and consulting and information services, as well as other technical information of interest to Tandem users.*

### SNAX/APN System Management

This independent study program acquaints the student with IBM's Advanced Program-to-Program Communication (APPC) and IBM's Advanced Peer-to-Peer Networking (APPN) architectures, as they apply to Tandem's SNAX/APN family of products. Through self-paced study and practical review exercises, the student masters the concepts and architectural principles behind IBM's new peer-to-peer SNA, Tandem's SNAX/XF, and related SNAX/APN and SNAX/APC features. After completing this program, the student has the technical foundation necessary to install and configure Tandem's SNAX/APN and related product features. The estimated time required to complete this program is 30 hours.

### NonStop SQL Query Analysis and Optimization

This two-day lecture-and-lab course is designed to help users master the art of advanced query analysis and optimization. Students learn how to analyze a query and rewrite queries to influence the optimizer. Lab exercises help students hone their query writing skills. After completing this course, users know how to find needed information about query performance and how to test their queries before they go into the production environment.

## Tandem Professional Services

Tandem Professional Services provides a variety of service programs and on-site technical consulting services designed to help users gain optimum performance, as well as the greatest possible benefit, from their systems. Following are brief descriptions of new professional services and programs offered by Tandem. For more information, users should contact their local Tandem representative.

### Availability Practice Guide

To help maximize the availability of Tandem NonStop systems, Tandem has produced the Availability Practice Guide, which shows how to reduce computing costs and improve system and staff productivity. The guide contains a wealth of proven procedures, written by Tandem availability experts, that can help users get the most from their Tandem investment. Each practice reduces computing costs by eliminating or greatly reducing unplanned or planned outage minutes.

The first edition of the Availability Practice Guide is for all NonStop users and includes topics covering management guidelines, production operation guidelines, writing efficient command files, general system management guidelines, and development and database management guidelines. To receive a free copy of the guide, users should call their Tandem representative.

### Custom Availability Support Program

Currently available for Himalaya servers, the Custom Availability Support (CAS) program provides qualified Tandem users with guaranteed system availability of at least 99.9 percent, and covers hardware, system software, and operations. To qualify for CAS support and the availability guarantee, users must first undergo an extensive qualification review of their system and operations environment. They must then make improvements to their system/environment as identified in the qualification review summary document.

Users may utilize either their own resources or Tandem professional services to meet the CAS high-availability specification. Once a user is qualified, Tandem will provide ongoing maintenance services to ensure that the high availability goal is continuously met. If the user fails to meet the 99.9-percent availability goal during a calendar quarter (due to the failure of Tandem hardware, Tandem system software, or Tandem certified operating procedures), hardware and software maintenance fees for that period will be credited.

### Multivendor Network Solutions Services

This is a program designed to help users meet their business networking needs and keep pace with the demands of evolving network technologies. The program makes available a full range of services for evaluating, managing, and improving existing networks and designing, installing, certifying, and supporting new networks. Users can choose a service or combination of services that suits their needs. Tandem's expert staff and partners will ensure that the user's network offers the performance and availability that the user's critical client/server applications require. The program's strategies and designs adhere to open systems standards and will also deliver the Tandem Trusted Network often requested by client/server users.

### Multivendor Solutions Support

The Multivendor Solutions Support program consolidates support to a single source, for every aspect of a heterogeneous, networked computing environment. The program gives users access to Tandem's expert support staff and the services of Tandem's support partners, whose expertise covers the range of today's diverse hardware and software environments.

After an initial consultation, Tandem organizes an appropriate support team from among Tandem's partners to meet the user's requirements. From then on, Tandem becomes the single source for support and takes responsibility for managing overall problem management and each partner's delivery of hardware and software support services. In its management role, Tandem handles all aspects of problem resolution and escalation: coordinating service personnel, following up to make sure that the problem is satisfactorily resolved, and if necessary, escalating the problem to other partners for additional help.

### Tandem System Access Service

This service provides users with access to a predefined, fully operational Tandem system in the event of a disaster. The service is designed to fit into the user's disaster recovery plan by ensuring rapid restoration of the user's critical computer applications. The service provides for system delivery, installation, and 30-day usage. As part of the service, Tandem includes Application Migration Planning for Emergencies, a 1–2 day session with the user designed to ensure that this service will meet the user's mission critical computing requirements.

### Integrity Starter Kit Service

This service provides users with hands-on training in a fast-paced, interactive workshop to help them quickly and effectively integrate Integrity FT systems and NR servers into their computing environment. A combination of system installation and on-site services help users optimize their investment in their new UNIX system. Led by a Tandem expert, users learn the basics about system installation, configuration, and administration, and are given the context for understanding the system's full functionality. The workshop combines demonstrations and participatory exercises to help users feel comfortable working with their new system.

### CA-SESMAN Startup Service

This service helps users install and integrate the CA-SESMAN access control and security product into their business environment. The Tandem systems consultant assists the user's operations and security staff in installing and configuring CA-SESMAN for a core set of applications and users. The user's staff is trained in starting up CA-SESMAN, configuring it, managing it, making full use of its capabilities, and shutting it down gracefully. The consultant leaves behind a list of tasks the user should perform to continue integrating CA-SESMAN into the user's environment. The service typically requires three days.

### Performance Review and Analysis Service—Core Service

This is an economy version of the comprehensive Performance Review and Analysis packaged service. This service typically requires eight days, and covers optimization and balancing of CPUs, memory, cache, files, and disc I/O. Unlike the comprehensive service, this service does *not* cover in-depth optimization of many of the other major subsystems. As part of this service, a report is provided with recommendations for performance improvements.

### Tandem Annual Services

Tandem Annual Services are professional services that focus on specific, recurring tasks that all users need to perform. These services can be purchased on a one-time basis, but are more beneficial when contracted for annually. The annual services currently available are:

- Change Management, which tracks current Tandem software enhancements, IPMs, bulletins, and alerts, and recommends those appropriate to the user. This service also includes implementation plans and telephone assistance.
- SYSGEN Definition, which analyzes proposed configuration changes and provides recommendations for correcting discrepancies. This service also provides an installation plan and telephone assistance.
- Technical Consultation, which consists of tailored consultations to assist the user's systems management and operations staff in building practical knowledge to improve their use of Tandem systems.
- Problem Determination, which provides initial diagnosis of problems suspected of involving Tandem equipment.

## CD Read Documentation

Tandem CD Read provides a complete set of NonStop Kernel documentation on a single CD-ROM disc. New CD Read discs containing the latest documentation are shipped by first class mail to all CD Read subscribers. To obtain a replacement for a missing shipment or to place an order for CD Read, users may call 800-243-6886.

### CD Read, Version C30_10_1 and Version D20_00_1

*February 1994*

The C30_10_1 and D20_00_1 discs contain complete sets of NonStop Kernel documentation, over 350 manuals and the entire set of softdocs. (Listings of manuals that are new or have changed can be found on the CD Read disc itself in the "About the Softpubs Library" document.) In addition, the D20_00_1 disc includes new screen fonts for Macintosh users and new files for PC users running Windows software.

# TandemSystemsReview*Index*

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

| | | | |
|---|---|---|---|
| Volume 1, No. 1 | Fall 1983 | Volume 2, No. 2 | Spring 1984 |
| Volume 2, No. 1 | Winter 1984 | Volume 2, No. 3 | Summer 1984 |

As of this issue, 26 issues of the *Tandem Systems Review* have been published:

| | | | | | |
|---|---|---|---|---|---|
| Volume 1, No. 1 | Feb. 1985 | Volume 5, No. 1 | April 1989 | Volume 9, No. 2 | Spring 1993 |
| Volume 1, No. 2 | June 1985 | Volume 5, No. 2 | Sept. 1989 | Volume 9, No. 3 | Summer 1993 |
| Volume 2, No. 1 | Feb. 1986 | Volume 6, No. 1 | March 1990 | Volume 9, No. 4 | Fall 1993 |
| Volume 2, No. 2 | June 1986 | Volume 6, No. 2 | Oct. 1990 | Volume 10, No. 1 | Jan. 1994 |
| Volume 2, No. 3 | Dec. 1986 | Volume 7, No. 1 | April 1991 | Volume 10, No. 2 | April 1994 |
| Volume 3, No. 1 | March 1987 | Volume 7, No. 2 | Oct. 1991 | Volume 10, No. 3 | July 1994 |
| Volume 3, No. 2 | Aug. 1987 | Volume 8, No. 1 | Spring 1992 | | |
| Volume 4, No. 1 | Feb. 1988 | Volume 8, No. 2 | Summer 1992 | | |
| Volume 4, No. 2 | July 1988 | Volume 8, No. 3 | Fall 1992 | | |
| Volume 4, No. 3 | Oct. 1988 | Volume 9, No. 1 | Winter 1993 | | |

The articles published in all 30 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 | Publication date | Part number |
|---|---|---|---|---|---|
| **APPLICATION DEVELOPMENT AND LANGUAGES** | | | | | |
| A New Design for the PATHWAY TCP | R. Wong | TJ | 2,2 | Spring 1984 | 83932 |
| An Overview of Client/Server Computing on Tandem Systems | H. Cooperstein | TSR | 8,3 | Fall 1992 | 89803 |
| An Introduction to Tandem EXTENDED BASIC | J. Meyerson | TJ | 2,2 | Spring 1984 | 83932 |
| Application Code Conversion for D-Series Systems | K. Liu | TSR | 9,2 | Spring 1993 | 89805 |
| Application Profile: Storing Macintosh Graphics on the Tandem 5200 Optical Storage Facility | D. Broyles | TSR | 9,3 | Summer 1993 | 89806 |
| Automating Call Centers With CAM | W. Choi | TSR | 10,2 | April 1994 | 104398 |
| Basic Uses and New Features of Extended GDS | A. Hotea | TSR | 10,1 | Jan. 1994 | 104396 |
| Debugging TACL Code | L. Palmer | TSR | 4,2 | July 1988 | 13693 |
| Designing and Implementing a Graphical User Interface | S. Wolfe | TSR | 9,3 | Summer 1993 | 89806 |
| Designing Client/Server Applications for OLTP on Guardian 90 Systems | W. Culman | TSR | 8,3 | Fall 1992 | 89803 |
| Extending the Client/Server Model With Object-Oriented Technology | T. Rohner | TSR | 10,1 | Jan. 1994 | 104396 |
| Implementing Client/Server Using RSC | M. Iem, T. Kocher | TSR | 8,3 | Fall 1992 | 89803 |
| 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 |

| Article title | Author(s) | Publication | Volume, Issue | Publication date | Part number |
|---|---|---|---|---|---|
| **APPLICATION DEVELOPMENT AND LANGUAGES** *(cont.)* | | | | | |
| PATHWAY IDS: A Message-level Interface to Devices and Processes | M. Anderton, M. Noonan | TSR | 2,2 | June 1986 | 83937 |
| The RESPOND OLTP Business Management System for Manufacturing | H. Bolling, W. Bronson | TSR | 9,1 | Winter 1993 | 89804 |
| 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 DAL Server: Client/Server Access to Tandem Databases | W. Schlansky, J. Schrengohst | TSR | 9,1 | Winter 1993 | 89804 |
| 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 |
| **CLIENT/SERVER** | | | | | |
| An Overview of Client/Server Computing on Tandem Systems | H. Cooperstein | TSR | 8,3 | Fall 1992 | 89803 |
| Application Profile: Storing Macintosh Graphics on the Tandem 5200 Optical Storage Facility | D. Broyles | TSR | 9,3 | Summer 1993 | 89806 |
| Client/Server Availability | A. Wood | TSR | 10,2 | April 1994 | 104398 |
| Designing and Implementing a Graphical User Interface | S. Wolfe | TSR | 9,3 | Summer 1993 | 89806 |
| Designing Client/Server Applications for OLTP on Guardian 90 Systems | W. Culman | TSR | 8,3 | Fall 1992 | 89803 |
| Extending the Client/Server Model With Object-Oriented Technology | T. Rohner | TSR | 10,1 | Jan. 1994 | 104396 |
| Gateways to NonStop SQL | D. Slutz | TSR | 6,2 | Oct. 1990 | 46987 |
| Implementing Client/Server Using RSC | M. Iem, T. Kocher | TSR | 8,3 | Fall 1992 | 89803 |
| NonStop ODBC Server | H. Mahbod, D. Slutz | TSR | 10,3 | July 1994 | 104400 |
| The DAL Server: Client/Server Access to Tandem Databases | W. Schlansky, J. Schrengohst | TSR | 9,1 | Winter 1993 | 89804 |
| **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 |
| Basic Uses and New Features of Extended GDS | A. Hotea | TSR | 10,1 | Jan. 1994 | 104396 |
| Changes in FOX | N. Donde | TSR | 1,2 | June 1985 | 83935 |
| Connecting Terminals and Workstations to Guardian 90 Systems | E. Siegel | TSR | 8,2 | Summer 1992 | 69848 |
| Expand High-Performance Solutions | D. Smith | TSR | 9,3 | Summer 1993 | 89806 |
| 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 |

| Article title | Author(s) | Publication | Volume, Issue | Publication date | Part number |
|---|---|---|---|---|---|
| **DATA MANAGEMENT** | | | | | |
| A Comparison of the B00 DP1 and DP2 Disc Processes | T. Schachter | TSR | 1,2 | June 1985 | 83935 |
| A New Hash-Based Join Algorithm for NonStop SQL/MP | H. Zeller | TSR | 10,3 | July 1994 | 104400 |
| An Overview of NonStop SQL/MP | F. Ho, R. Jain, J. Troisi | TSR | 10,3 | July 1994 | 104400 |
| 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 |
| Enhancing Availability, Manageability, and Performance With NonStopTM/MP | M. Chandra, D. Eicher | TSR | 10,3 | July 1994 | 104400 |
| 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 |
| 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 Availability and Database Configuration Operations | J. Troisi | TSR | 10,3 | July 1994 | 104400 |
| NonStop ODBC Server | H. Mahbod, D. Slutz | TSR | 10,3 | July 1994 | 104400 |
| 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 |
| Online Information Processing | J. Viescas | TSR | 9,1 | Winter 1993 | 89804 |
| 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 |
| 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 |
| **DECISION SUPPORT SYSTEMS** | | | | | |
| An Overview of NonStop SQL/MP | F. Ho, R. Jain, J. Troisi | TSR | 10,3 | July 1994 | 104400 |
| NonStop ODBC Server | H. Mahbod, D. Slutz | TSR | 10,3 | July 1994 | 104400 |
| Online Information Processing | J. Viescas | TSR | 9,1 | Winter 1993 | 89804 |
| The DAL Server: Client/Server Access to Tandem Databases | W. Schlansky, J. Schrengohst | TSR | 9,1 | Winter 1993 | 89804 |
| The RESPOND OLTP Business Management System for Manufacturing | H. Bolling, W. Bronson | TSR | 9,1 | Winter 1993 | 89804 |
| **OBJECT-ORIENTED TECHNOLOGY** | | | | | |
| Extending the Client/Server Model With Object-Oriented Technology | T. Rohner | TSR | 10,1 | Jan. 1994 | 104396 |

| Article title | Author(s) | Publication | Volume, Issue | Publication date | Part number |
|---|---|---|---|---|---|
| **OPERATING SYSTEMS** | | | | | |
| Application Code Conversion for D-Series Systems | K. Liu | TSR | 9,2 | Spring 1993 | 89805 |
| 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 |
| Migration Planning for D-Series Systems | S. Kuukka | TSR | 9,2 | Spring 1993 | 89805 |
| 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 D-Series Guardian 90 Operating System | W. Bartlett | TSR | 9,2 | Spring 1993 | 89805 |
| 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 | 83934 |
| The NonStop Himalaya K10000 Interprocessor Bus | R. Jardine, S. Hamilton, K. Krishnakumar | TSR | 10,2 | April 1994 | 104398 |
| The Tandem Global Update Protocol | R. Carr | TSR | 1,2 | June 1985 | 83935 |
| **PERFORMANCE AND CAPACITY PLANNING** | | | | | |
| A Performance Retrospective | P. Oleinick, P. Shah | TSR | 2,3 | Dec. 1986 | 83938 |
| Buffering for Better Application Performance | R. Mattran | TSR | 2,1 | Feb. 1986 | 83936 |
| Capacity Planning Concepts | R. Evans | TSR | 2,3 | Dec. 1986 | 83938 |
| Capacity Planning With TCM | W. Highleyman | TSR | 7,2 | Oct. 1991 | 65248 |
| C00 TMDS Performance | J. Mead | TSR | 4,1 | Feb. 1988 | 11078 |
| Credit-authorization Benchmark for High Performance and Linear Growth | T. Chmiel, T. Houy | TSR | 2,1 | Feb. 1986 | 83936 |
| Debugging Accelerated Programs on TNS/R Systems | D. Cressler | TSR | 8,1 | Spring 1992 | 65250 |
| DP2 Performance | J. Enright | TSR | 1,2 | June 1985 | 83935 |
| Estimating Host Response Time in a Tandem System | H. Horwitz | TSR | 4,3 | Oct. 1988 | 15748 |
| Expand High-Performance Solutions | D. Smith | TSR | 9,3 | Summer 1993 | 89806 |
| 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 |
| Implementing a Systems Management Improvement Program | J. Dagenais | TSR | 9,4 | Fall 1993 | 89807 |
| Improved Performance for BACKUP2 and RESTORE2 | A. Khatri, M. McCline | TSR | 1,2 | June 1985 | 83935 |
| Improving Performance on TNS/R Systems With the Accelerator | M. Blanchet | TSR | 8,1 | Spring 1992 | 65250 |
| MEASURE: Tandem's New Performance Measurement Tool | D. Dennison | TSR | 2,3 | Dec. 1986 | 83938 |
| Measuring DSM Event Management Performance | M. Stockton | TSR | 8,1 | Spring 1992 | 65250 |
| Message System Performance Enhancements | D. Kinkade | TSR | 2,3 | Dec. 1986 | 83938 |
| Message System Performance Tests | S. Uren | TSR | 2,3 | Dec. 1986 | 83938 |
| Network Design Considerations | J. Evjen | TSR | 5,2 | Sept. 1989 | 28152 |
| NonStop NET/MASTER: Configuration and Performance Guidelines | M. Stockton | TSR | 9,4 | Fall 1993 | 89807 |
| 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 |

| Article title | Author(s) | Publication | Volume, Issue | Publication date | Part number |
|---|---|---|---|---|---|
| **PERFORMANCE AND CAPACITY PLANNING** *(cont.)* | | | | | |
| 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 |
| RDF Enhancements for High Availability and Performance | M. Mosher | TSR | 10,3 | July 1994 | 104400 |
| 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 |
| 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 |
| 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 |
| Application Profile: Storing Macintosh Graphics on the Tandem 5200 Optical Storage Facility | D. Broyles | TSR | 9,3 | Summer 1993 | 89806 |
| 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 |
| Terminal Selection | E. Siegel | TSR | 8,2 | Summer 1992 | 69848 |
| 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 |

| Article title | Author(s) | Publication | Volume, Issue | Publication date | Part number |
|---|---|---|---|---|---|
| **PROCESSORS** | | | | | |
| Fault Tolerance in the NonStop Cyclone System | S. Chan, R. Jardine | TSR | 7,1 | April 1991 | 46988 |
| A Hardware Overview of the NonStop Himalaya K10000 Server | C. Kong | TSR | 10,1 | Jan. 1994 | 104396 |
| NonStop CLX: Optimized for Distributed On-Line Transaction Processing | D. Lenoski | TSR | 5,1 | April 1989 | 18662 |
| NonStop VLX Hardware Design | M. Brown | TSR | 2,3 | Dec. 1986 | 83938 |
| Overview of Tandem NonStop Series/RISC Systems | L. Faby, R. Mateosian | TSR | 8,1 | Spring 1992 | 65250 |
| The High-Performance NonStop TXP Processor Transaction Processing | W. Bartlett, T. Houy, D. Meyer | TJ | 2,1 | Winter 1984 | 83931 |
| The NonStop Himalaya K10000 Interprocessor Bus | R. Jardine, S. Hamilton, K. Krishnakumar | TSR | 10,2 | April 1994 | 104398 |
| 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** | | | | | |
| Basic Uses and New Features of Extended GDS | A. Hotea | TSR | 10,1 | Jan. 1994 | 104396 |
| Building Open Systems Interconnection with OSI/AS and OSI/TS | R. Smith | TSR | 6,1 | March 1990 | 32986 |
| Connecting Terminals and Workstations to Guardian 90 Systems | E. Siegel | TSR | 8,2 | Summer 1992 | 69848 |
| Implementing Client/Server Using RSC | M. Iem, T. Kocher | TSR | 8,3 | Fall 1992 | 89803 |
| 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 |
| Terminal Selection | E. Siegel | TSR | 8,2 | Summer 1992 | 69848 |
| The OSI Model: Overview, Status, and Current Issues | A. Dunn | TSR | 5,1 | April 1989 | 18662 |
| **SYSTEM MANAGEMENT** | | | | | |
| Configuring Tandem Disk Subsystems | S. Sitler | TSR | 2,3 | Dec. 1986 | 83938 |
| Data Replication in Tandem's Distributed Name Service | T. Eastep | TSR | 4,3 | Oct. 1988 | 15748 |
| Enhancements to TMDS | L. White | TSR | 3,2 | Aug. 1987 | 83940 |
| Event Management Service Design and Implementation | H. Jordan, R. McKee, R. Schuet | TSR | 4,3 | Oct. 1988 | 15748 |
| Implementing a Systems Management Improvement Program | J. Dagenais | TSR | 9,4 | Fall 1993 | 89807 |
| Instrumenting Applications for Effective Event Management | J. Dagenais | TSR | 7,2 | Oct. 1991 | 65248 |
| Introducing TMDS, Tandem's New On-line Diagnostic System | J. Troisi | TSR | 1,2 | June 1985 | 83935 |
| Measuring DSM Event Management Performance | M. Stockton | TSR | 8,1 | Spring 1992 | 65250 |
| Network Statistics System | M. Miller | TSR | 4,3 | Oct. 1988 | 15748 |
| NonStop NET/MASTER: Configuration and Performance Guidelines | M. Stockton | TSR | 9,4 | Fall 1993 | 89807 |
| NonStop NET/MASTER: Event Management Architecture | M. Stockton | TSR | 9,4 | Fall 1993 | 89807 |
| NonStop NET/MASTER: Event Processing Costs and Sizing Calculations | M. Stockton | TSR | 9,4 | Fall 1993 | 89807 |
| Overview of DSM | P. Homan, B. Malizia, E. Reisner | TSR | 4,3 | Oct. 1988 | 15748 |
| SCP and SCF: A General Purpose Implementation of the Subsystem Programmatic Interface | T. Lawson | TSR | 4,3 | Oct. 1988 | 15748 |
| RDF: An Overview | J. Guerrero | TSR | 7,2 | Oct. 1991 | 65248 |
| RDF Enhancements for High Availability and Performance | M. Mosher | TSR | 10,3 | July 1994 | 104400 |
| RDF Synchronization | F. Jongma, W. Senf | TSR | 8,2 | Summer 1992 | 69848 |

| Article title | Author(s) | Publication | Volume, Issue | Publication date | Part number |
|---|---|---|---|---|---|
| **SYSTEM MANAGEMENT** *(cont.)* | | | | | |
| 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 | Publication date | 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** | | | | | |
| Application Profile: Storing Macintosh Graphics on the Tandem 5200 Optical Storage Facility | D. Broyles | TSR | 9,3 | Summer 1993 | 89806 |
| 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 |
| **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 |
| **CAM** | | | | | |
| Automating Call Centers With CAM | W. Choi | TSR | 10,2 | April 1994 | 104398 |
| **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 |
| **DAL SERVER** | | | | | |
| The DAL Server: Client/Server Access to Tandem Databases | W. Schlansky, J. Schrengohst | TSR | 9,1 | Winter 1993 | 89804 |

| Article title | Author(s) | Publication | Volume, Issue | Publication date | Part number |
|---|---|---|---|---|---|
| **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 |
| **DSM** | | | | | |
| Data Replication in Tandem's Distributed Name Service | T. Eastep | TSR | 4,3 | Oct. 1988 | 15748 |
| Event Management Service Design and Implementation | H. Jordan, R. McKee, R. Schuet | TSR | 4,3 | Oct. 1988 | 15748 |
| Instrumenting Applications for Effective Event Management | J. Dagenais | TSR | 7,2 | Oct. 1991 | 65248 |
| Measuring DSM Event Management Performance | M. Stockton | TSR | 8,1 | Spring 1992 | 65250 |
| Network Statistics System | M. Miller | TSR | 4,3 | Oct. 1988 | 15748 |
| Overview of DSM | P. Homan, B. Malizia, E. Reisner | TSR | 4,3 | Oct. 1988 | 15748 |
| SCP and SCF: A General Purpose Implementation of the Subsystem Programmatic Interface | T. Lawson | TSR | 4,3 | Oct. 1988 | 15748 |
| Tandem's Subsystem Programmatic Interface | G. Tom | TSR | 4,3 | Oct. 1988 | 15748 |
| Using the Subsystem Programmatic Interface and Event Management Services | K. Stobie | TSR | 4,3 | Oct. 1988 | 15748 |
| VIEWPOINT Operations Console Facility | R. Hansen, G. Stewart | TSR | 4,3 | Oct. 1988 | 15748 |
| Writing Rules for Automated Operations | J. Collins | TSR | 7,2 | Oct. 1991 | 65248 |
| **DYNAMITE** | | | | | |
| An Introduction to DYNAMITE Workstation Host Integration | S. Kosinski | TSR | 1,2 | June 1985 | 83935 |
| The DYNAMITE Workstation: An Overview | G. Smith | TSR | 1,2 | June 1985 | 83935 |
| **ENABLE** | | | | | |
| The ENABLE Program Generator for Multifile Applications | B. Chapman, J. Zimmerman | TSR | 1,1 | Feb. 1985 | 83934 |
| **ENCOMPASS** | | | | | |
| The Relational Data Base Management Solution | G. Ow | TJ | 2,1 | Winter 1984 | 83931 |
| **ENCORE** | | | | | |
| The ENCORE Stress Test Generator for On-line Transaction Processing Applications | S. Kosinski | TJ | 2,1 | Winter 1984 | 83931 |
| **ENSCRIBE** | | | | | |
| Converting Database Files from ENSCRIBE to NonStop SQL | W. Weikel | TSR | 6,1 | March 1990 | 32986 |
| **EXPAND** | | | | | |
| Expand High-Performance Solutions | D. Smith | TSR | 9,3 | Summer 1993 | 89806 |
| **FASTSORT** | | | | | |
| FASTSORT: An External Sort Using Parallel Processing | J. Gray, M. Stewart, A. Tsukerman, S. Uren, B. Vaughan | TSR | 2,3 | Dec. 1986 | 83938 |

| Article title | Author(s) | Publication | Volume, Issue | Publication date | Part number |
|---|---|---|---|---|---|
| **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 |
| **GDS** | | | | | |
| Basic Uses and New Features of Extended GDS | A. Hotea | TSR | 10,1 | Jan. 1994 | 104396 |
| **GUARDIAN 90** | | | | | |
| Application Code Conversion for D-Series Systems | K. Liu | TSR | 9,2 | Spring 1993 | 89805 |
| 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 |
| Migration Planning for D-Series Systems | S. Kuukka | TSR | 9,2 | Spring 1993 | 89805 |
| 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 D-Series Guardian 90 Operating System | W. Bartlett | TSR | 9,2 | Spring 1993 | 89805 |
| 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 | 83934 |
| The Tandem Global Update Protocol | R. Carr | TSR | 1,2 | June 1985 | 83935 |
| **HIMALAYA** | | | | | |
| A Hardware Overview of the NonStop Himalaya K10000 Server | C. Kong | TSR | 10,1 | Jan. 1994 | 104396 |
| The NonStop Himalaya K10000 Interprocessor Bus | R. Jardine, S. Hamilton, K. Krishnakumar | TSR | 10,2 | April 1994 | 104398 |
| **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 |

| Article title | Author(s) | Publication | Volume, Issue | Publication date | Part number |
|---|---|---|---|---|---|
| **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 NET/MASTER** | | | | | |
| NonStop NET/MASTER: Configuration and Performance Guidelines | M. Stockton | TSR | 9,4 | Fall 1993 | 89807 |
| NonStop NET/MASTER: Event Management Architecture | M. Stockton | TSR | 9,4 | Fall 1993 | 89807 |
| NonStop NET/MASTER: Event Processing Costs and Sizing Calculations | M. Stockton | TSR | 9,4 | Fall 1993 | 89807 |
| **NONSTOP ODBC SERVER** | | | | | |
| NonStop ODBC Server | H. Mahbod, D. Slutz | TSR | 10,3 | July 1994 | 104400 |
| **NONSTOP SQL/MP** | | | | | |
| A New Hash-Based Join Algorithm for NonStop SQL/MP | H. Zeller | TSR | 10,3 | July 1994 | 104400 |
| An Overview of NonStop SQL/MP | F. Ho, R. Jain, J. Troisi | TSR | 10,3 | July 1994 | 104400 |
| 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 Availability and Database Configuration Operations | J. Troisi | TSR | 10,3 | July 1994 | 104400 |
| 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 |
| **NONSTOP TM/MP** | | | | | |
| Improvements in TMF | T. Lemberger | TSR | 1,2 | June 1985 | 83935 |
| Enhancing Availability, Manageability, and Performance With NonStop TM/MP | M. Chandra, D. Eicher | TSR | 10,3 | July 1994 | 104400 |
| 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 |
| **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 |

| Article title | Author(s) | Publication | Volume, Issue | Publication date | Part number |
|---|---|---|---|---|---|
| **POET** | | | | | |
| Designing Client/Server Applications for OLTP on Guardian 90 Systems | W. Culman | TSR | 8,3 | Fall 1992 | 89803 |
| **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 |
| RDF Enhancements for High Availability and Performance | M. Mosher | TSR | 10,3 | July 1994 | 104400 |
| RDF Synchronization | F. Jongma, W. Senf | TSR | 8,2 | Summer 1992 | 69848 |
| **RESPOND** | | | | | |
| The RESPOND OLTP Business Management System for Manufacturing | H. Bolling, W. Bronson | TSR | 9,1 | Winter 1993 | 89804 |
| **RSC** | | | | | |
| Implementing Client/Server Using RSC | M. Iem, T. Kocher | TSR | 8,3 | Fall 1992 | 89803 |
| **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 |
| **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 |
| **TNS/R** | | | | | |
| Debugging Accelerated Programs on TNS/R Systems | D. Cressler | TSR | 8,1 | Spring 1992 | 65250 |
| Improving Performance on TNS/R Systems With the Accelerator | M. Blanchet | TSR | 8,1 | Spring 1992 | 65250 |
| Overview of Tandem NonStop Series/RISC Systems | L. Faby, R. Mateosian | TSR | 8,1 | Spring 1992 | 65250 |

| Article title | Author(s) | Publication | Volume, Issue | Publication date | Part number |
|---|---|---|---|---|---|
| **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 |

# TandemSystemsReview*OrderForm*

Use this form to order new subscriptions, change subscription information, and order back issues.

☐ I am a Tandem customer. My Tandem sales representative is _____.

☐ I am not a Tandem customer and am enclosing a check or money order for the requests indicated on this form. (Subscriptions are $75 per year and each back issue is $20. Make checks payable to Tandem Computers Incorporated.)

## Subscription Information

☐ New subscription

☐ 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 Issue Requests

Number of copies

***Tandem Systems Review***

_____ Vol. 1, No. 1, Feb. 1985
_____ Vol. 1, No. 2, June 1985
_____ Vol. 2, No. 1, Feb. 1986
_____ Vol. 2, No. 2, June 1986
_____ Vol. 2, No. 3, Dec. 1986
_____ Vol. 3, No. 1, March 1987
_____ Vol. 3, No. 2, Aug. 1987
_____ Vol. 4, No. 1, Feb. 1988
_____ Vol. 4, No. 2, July 1988
_____ Vol. 4, No. 3, Oct. 1988
_____ Vol. 5, No. 1, April 1989
_____ Vol. 5, No. 2, Sept. 1989
_____ Vol. 6, No. 1, March 1990
_____ Vol. 6, No. 2, Oct. 1990
_____ Vol. 7, No. 1, April 1991
_____ Vol. 7, No. 2, Oct. 1991
_____ Vol. 8, No. 1, Spring 1992
_____ Vol. 8, No. 2, Summer 1992
_____ Vol. 8, No. 3, Fall 1992
_____ Vol. 9, No. 1, Winter 1993
_____ Vol. 9, No. 2, Spring 1993
_____ Vol. 9, No. 3, Summer 1993
_____ Vol. 9, No. 4, Fall 1993
_____ Vol. 10, No. 1, Jan. 1994
_____ Vol. 10, No. 2, April 1994
_____ Vol. 10, No. 3, July 1994

***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

For questions or ordering information, call 800-473-5868 in the U.S. and Canada or +1-408-285-0665 in other countries.

Send this form to:

Tandem Computers Incorporated
*Tandem Systems Review*, Loc 208-65
10400 Ridgeview Court
Cupertino, CA 95014-0723
FAX: +1-408-285-0840

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

Menu sequence: Marketing Information → Literature Orders → Technical Marketing Pubs (TSR)

▲ FOLD

▲ FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

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

POSTAGE WILL BE PAID BY ADDRESSEE

**TANDEM SYSTEMS REVIEW**
LOC 208-65
TANDEM COMPUTERS INCORPORATED
19333 VALLCO PARKWAY
CUPERTINO, CA 95014–9862

▼ FOLD

▼ FOLD

# TandemSystemsReview*ReaderSurvey*

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

1. How useful is each article in this issue?

*Product Update*

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

*An Overview of NonStop SQL/MP*

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

*NonStop Availability and Database Configuration Operations*

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

*A New Hash-Based Join Algorithm for NonStop SQL/MP*

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

*NonStop ODBC Server*

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

*Enhancing Availability, Manageability, and Performance With NonStop TM/MP*

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

*RDF Enhancements for High Availability and Performance*

25 ☐ Indispensable 26 ☐ Very 27 ☐ Somewhat 28 ☐ Not at all

*Technical Information and Education*

29 ☐ Indispensable 30 ☐ Very 31 ☐ Somewhat 32 ☐ Not at all

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

33 ☐ Overview discussions of new products and enhancements
34 ☐ Performance and tuning information
35 ☐ High-level overviews on Tandem's approach to solutions
36 ☐ Application design and customer profiles
37 ☐ Technical discussions of product internals
38 ☐ Strategic information and statements of direction
39 ☐ Other _____

3. Your title or position:

40 ☐ President, VP, Director 41 ☐ Systems analyst 42 ☐ System operator
43 ☐ MIS manager 44 ☐ Software developer 45 ☐ End user
46 ☐ Other _____

4. Your association with Tandem:

47 ☐ Tandem customer 48 ☐ Tandem employee 49 ☐ Third-party vendor 50 ☐ Consultant
51 ☐ Other _____

NAME

COMPANY NAME

ADDRESS

7/94

▲ FOLD

▲ FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

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

POSTAGE WILL BE PAID BY ADDRESSEE

**TANDEM SYSTEMS REVIEW**
LOC 208-65
TANDEM COMPUTERS INCORPORATED
19333 VALLCO PARKWAY
CUPERTINO, CA 95014–9862

▼ FOLD

▼ FOLD

TANDEM

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



Part No. 104400

7/94 Printed in USA