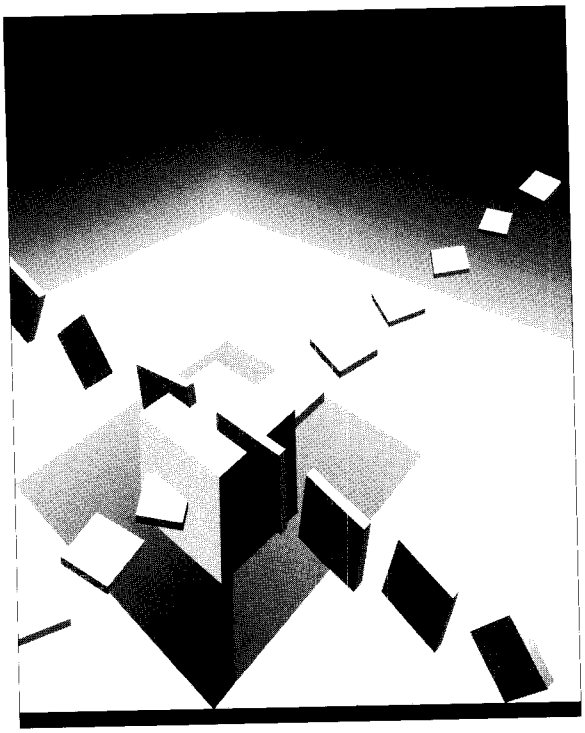
T A N D E M

SYSTEMS REVIEW

VOLUME 6, NUMBER 2

OCTOBER 1990



NonStop SQL Release 2 Overview

Performance • Parallelism

NonStop SQL Release 2 Benchmark

Online Reorganization

Outer Join • NonStop SQL Gateways

Online Batch Processing

Volume 6, Number 2, October 1990

Editorial Director
Susan W. Thompson

Editor
Anne Lewis

Associate Editors
Steven Kahn
Mark Peters

Technical Advisors
Mark Anderton
Terrye Kocher
Mike Noonan

Assistant Editor
Sarah Rood

Electronic Publishing
Annie F. Valva

Art Director
Janet Stevenson

Cover Art
Niklas Hallin

Illustrations
Laurie Scott
Cynthia Moore

Circulation
Cathy Gerrity

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

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

Subscription additions and changes: As of the March 1990 issue, subscriptions to the *Tandem Systems Review* must be approved by a Tandem representative. Complete the customer portions of the order form at the back of this copy and send the form to your local Tandem sales office.

Comments: The editors welcome suggestions for content and format. Please send them to the *Tandem Systems Review*, LOC 216-05, 18922 Forge Drive, Cupertino, CA 95014.

Editor's Preface

With Release 2 of the NonStop™ SQL relational database management system, Tandem™ systems can simultaneously run batch jobs, ad hoc queries, and online transaction processing (OLTP) applications against a single, large database. Release 2 meets the growing demand for a relational database that can manage large amounts of data (100 gigabytes or larger) and at the same time satisfy all types of processing demands.

This is the *Tandem Systems Review*'s second issue featuring NonStop SQL. Volume 4, number 2 (July 1988) introduced NonStop SQL as a new product and discussed its most important features.

The first article in this issue, by Pong, is an overview of the new release. It briefly describes the features introduced with Release 2 and guides the reader to the other articles in this issue.

By exploiting the Tandem multiprocessor architecture, Release 2 of NonStop SQL simultaneously executes portions of an SQL query on several processors, reducing its execution time to a fraction of what it would be on a single processor. In addition, NonStop SQL can execute parallel queries and batch jobs on the database with little impact on the response time of concurrently running OLTP applications.

The article by Englert and Gray is the first of two articles that describe the performance benefits of parallel query execution in Release 2. The authors describe in detail how parallel query execution can provide speedup and scaleup for batch and query processing. They also outline new Tandem system features that support parallelism in NonStop SQL by optimizing sequential processing and mixed workload performance.

The second of the two performance articles, by Englert, Gray, Kocher, and Shah, describes the benchmark tests that demonstrate near-linear speedup and scaleup for SQL queries on a variety of Tandem systems. The article explains how the tests were run and describes the results of each query.

Release 2 of NonStop SQL manages parallel query execution by dividing an SQL query into smaller tasks and assigning the tasks to separate processors. The article by Moore and Sodhi describes how Release 2 implements parallel query execution. It suggests how Tandem users can configure their systems to take maximum advantage of this feature. Examples show how each basic NonStop SQL operation executes in parallel in a sample system configuration. The article also describes the parallel index maintenance feature.

An important element of managing a very large database is the ability to reorganize fragmented files without having to suspend access to online applications.Version C30 of the Guardian™ 90 operating system allows users to reorganize audited, key-sequenced files online, thus improving performance and recovering unused disk space. Smith's article explains how tables and files become disorganized, describes the impact on system performance, and outlines the methods available to reorganize them. It also describes how to control and tune online file reorganization.

With the outer join operation, a new feature in Release 2 of NonStop SQL, users can generate exception reports while benefiting from simpler SQL queries. The outer join operation combines rows from multiple tables and also preserves information that failed to qualify for the join. Thus, with a single outer join query, users can generate a complex exception report. The article by Vaishnav defines the basic concepts related to the outer join operation and compares the functions of inner join and outer join operations.

To make NonStop SQL available to users of PCs and workstations, Tandem is developing gateways to connect popular SQL applications to NonStop SQL. Tandem is working together with SQL database vendors such as Oracle, INGRES, and Microsoft/Sybase. Although a standard for SQL exists, each vendor's implementation of an SQL gateway differs. Slutz discusses general design issues for SQL gateways, the standardization efforts currently in progress, and solutions for gateway applications.

The final article in this issue discusses the importance of batch processing in the online computing enterprise. Keefauver describes the enhancements in Tandem batch processing and discusses the advantages of integrating batch processing with OLTP. The article also describes the software requirements for online batch processing, including I/O optimization, record locking, and transaction protection.

The last page of this issue is a customer survey. This questionnaire gives the *Tandem Systems Review* staff information about reader interests. Please take a few minutes to evaluate each article in this issue and indicate the subjects about which you would like to see more articles.

Susan W. Thompson
Editorial Director

An Overview of NonStop SQL Release 2

Release 2 of NonStop™ SQL, the Tandem™ relational database management system (RDBMS), allows users to run batch jobs and ad hoc queries, as well as online transaction processing (OLTP) applications, on a single database. It meets the growing demand for an RDBMS that can manage large databases (100 gigabytes or larger). By supporting concurrent batch, query, and online transaction processing, NonStop SQL Release 2 eliminates the need to operate one database for OLTP applications and a second database for batch and query processing. NonStop SQL Release 2 realizes the goal, increasingly sought after by users, of managing all basic business computing tasks on a single, up-to-date, enterprise-wide database.

Release 1 of NonStop SQL, which implemented the American National Standards Institute (ANSI) SQL standard on Tandem NonStop computer systems, was a fully distributed RDBMS (ANSI, 1986). It provided transparent access to both local and remote data. It also provided transaction protection for updates to data stored in a distributed network. Furthermore, benchmark tests demonstrated that NonStop SQL Release 1 was viable for production-scale OLTP (Tandem Performance Group, 1988). NonStop SQL achieved superior performance by integrating SQL with the Tandem file system and disk process, components of the Tandem Guardian™ 90 operating system (Borr and Putzolu, 1988). The articles in the July 1988 issue of the *Tandem Systems Review* (Vol. 4, No. 2) describe various aspects of NonStop SQL Release 1.

This article briefly describes the features introduced with Release 2 of NonStop SQL. It also discusses how those features contribute to meeting the goals of NonStop SQL Release 2. Finally, it guides the reader to the other articles in this issue of the *Tandem Systems Review*; those articles give detailed information about many of the features outlined here.

The Goals of NonStop SQL Release 2

Tandem had two major goals in developing Release 2 of NonStop SQL. First, DBMS users are increasingly choosing NonStop SQL to manage large databases. To meet this demand, NonStop SQL must offer batch and query performance that matches its proven OLTP performance. It must also provide the management tools to handle large tables.

Second, users are increasingly asking for a single DBMS to manage OLTP, batch, and query processing on a single database. Traditionally, users have one database for their OLTP applications and one or more databases for their query, batch and report generating applications. They must separate the databases so that the batch and report generation applications do not degrade the response time of the OLTP applications. The demands on corporate information have become even more complicated with the rise of information centers.

Maintaining separate databases has several drawbacks. First, one of the databases is always out of date. Consider, for example, a bank's 24-hour ATM system operating on an OLTP database. Typically, that database is a stripped-down version of the complete customer account database maintained by the batch system. A memo-posting mechanism transfers data from the OLTP database to the batch database. When the nightly batch run finishes, a similar mechanism strips the batch data and transfers it to the OLTP database. As long as new data is not transferred the instant each update occurs, at least one database system must operate on stale data.

Second, maintaining separate systems is expensive. Often, different vendors' hardware and software (DBMS) systems operate the different databases. Equipment and processing power may not be used as efficiently as they could be in a single-database installation. Also, communicating between the databases is costly. Users must maintain common data definitions (the databases must understand each other) and pay the costs of transferring information from one database to the other.

The Features of NonStop SQL Release 2

NonStop SQL Release 2 solves the dilemmas inherent in maintaining separate databases by allowing all three types of processing (online transaction, batch, and query) to execute concurrently on a single database. NonStop SQL Release 2 realizes both this goal and the first goal, managing large databases, by introducing features in three areas:

- Performance.
- Operability and manageability.
- Compatibility with the ANSI standard.

If OLTP, batch, and query processing are to coexist on a single database, batch jobs and queries must perform well without degrading OLTP performance. Intra-query parallel processing and parallel index maintenance, two features introduced with NonStop SQL Release 2, significantly enhance batch and query processing and also benefit OLTP performance.

To maintain performance in a mixed workload environment, the system must be able to balance properly the various demands being made on it. Typically, it must give priority to online transactions and allow batch jobs to run in the background. A new disk process algorithm ensures that large batch jobs and queries do not take over the system and interfere with OLTP response times.

Also, NonStop SQL must offer users tools that manage and operate a large, distributed database without taking the database offline. The online reorganization and partition split features simplify database management by performing these operations online; they keep the database continuously available to OLTP and batch applications.

Finally, because SQL is becoming an industry-wide standard, NonStop SQL must extend its compatibility with the versions of SQL provided by other vendors. NonStop SQL Release 2 has become more compatible with the ANSI standard by supporting null values, the UNION operator, the outer join operator, and the DATETIME and INTERVAL data types.

Performance

NonStop SQL Release 2 significantly improves performance in batch and query processing, OLTP, and mixed workload environments. Features that benefit batch jobs and queries include intra-query parallelism and improved buffering for update operations. Features that enhance OLTP performance include better optimization of the SQL OR operator and support of non-unique clustering keys. Parallel index maintenance offers advantages to both OLTP and batch processing. Finally, a new load-balancing feature allows mixed workloads of OLTP and batch processing to operate concurrently.

Intra-query processing offers the potential for scaleup and speedup.

Parallel Query Execution

Intra-query parallelism, introduced with NonStop SQL Release 2, significantly improves the response times of batch and ad hoc query processing. When the user invokes intra-query parallelism, NonStop SQL Release 2 divides a large query into smaller parts and evaluates all the parts in parallel. This feature takes advantage of the Tandem multiprocessor architecture by distributing the parts of a query among the processors in the Tandem system.

NonStop SQL Release 1 provided parallelism for OLTP applications by distributing each online transaction to a separate processor. It did not divide an individual transaction among multiple processors. This type of parallelism, called *inter-query parallelism,* is not adequate for batch jobs and ad hoc queries, which are usually much larger than online transactions.

Intra-query parallelism has two advantages over traditional sequential processing; it offers the potential for *scaleup* and *speedup. Scaleup* allows users to maintain the same response time for a query when the database size increases. To achieve scaleup, users increase the amount of system hardware in proportion to the size of the job. For example, by doubling the number of processors and disks, users can maintain the same execution time for a query on a database that has doubled in size.

Speedup allows users to reduce the response time of a query by adding more system hardware to the system. For example, by doubling the number of processors and disks, users can reduce the response time of a query by half (assuming that the database size remains constant.) In an audited benchmark, NonStop SQL Release 2 has demonstrated that the intra-query parallelism feature provides both linear scaleup and linear speedup for the basic SQL operations, including the SELECT, UPDATE, DELETE, aggregate, and join operations.

Three articles in this issue of the *Tandem Systems Review* discuss various aspects of intra-query parallelism. Scaleup and speedup are described in "Performance Benefits of Parallel Query Execution and Mixed Workload Support in NonStop SQL Release 2" (Englert and Gray, 1990). The benchmark tests are described in "The NonStop SQL Release 2 Benchmark" (Englert et al., 1990). The parallel processing feature is described in "Parallelism in NonStop SQL Release 2" (Moore and Sodhi, 1990).

Parallel Index Maintenance

NonStop SQL Release 2 further augments the OLTP performance of Release 1 by updating indexes in parallel. With parallel index maintenance, NonStop SQL can update 10 indexes in approximately the same elapsed time it takes to update one index (as long as the user has assigned each index to a separate disk volume). This reduces the elapsed time of online transactions that update tables containing many indexes.

Because multiple indexes are no longer a performance liability for OLTP, application designers can define multiple indexes on a table to benefit batch and query processing. For example, by taking advantage of the increased number of indexes, application designers can reduce the number of times the application performs sorting, a time-consuming process. Parallel index maintenance is described elsewhere in this issue of the *Tandem Systems Review* (Moore and Sodhi, 1990).

OR Operator

NonStop SQL Release 2 provides another OLTP performance enhancement by improving the optimization of queries that use the OR operator. In NonStop SQL Release 1, a query that contained the OR operator often resulted in a full table scan of the affected tables. Since table scans are time-consuming operations for large tables, OLTP applications had to minimize the use of the OR operator. To avoid the OR operator, application designers had to devote more effort to writing application programs.

NonStop SQL Release 2 can rapidly evaluate queries that use the OR operator if the predicates involved reference index columns. Consider the query:

```
SELECT * FROM T
WHERE COL1 = 10 OR COL2 = 20
```

If COL1 and COL2 are the prefix of two different indexes, NonStop SQL uses the indexes to retrieve (first) all the rows that satisfy the predicate COL1 = 10 and (second) all the rows that satisfy the predicate COL2 = 20. Evaluating the query in this fashion is much more efficient than scanning the entire table.

Non-Unique Clustering Keys

The addition of non-unique clustering keys provides yet another OLTP performance enhancement. Clustering permits related rows of a table to be stored physically close to one another, which allows the system to retrieve them quickly. In NonStop SQL Release 1, only the primary keys of a key-sequenced table were clustered. However, primary keys must be unique and many applications do not have unique key values. NonStop SQL Release 2 allows non-unique key values to be clustered by attaching unique, system-generated values to the non-unique keys.

Batch Updates

NonStop SQL Release 1 introduced virtual sequential block buffering (VSBB) for reading NonStop SQL tables (Pong, 1988). This feature allows the disk process to select rows and project fields before it returns a block of qualified rows to the application. NonStop SQL Release 2 extends the concept of VSBB to insert and update operations.

By buffering insert and update operations on NonStop SQL tables, NonStop SQL Release 2 significantly reduces the amount of messages and overhead data passed from the file system to the disk process. Because of the VSBB feature, the efficiency of insert and update operations has increased by up to 30 percent in NonStop SQL Release 2.

Balancing OLTP and Batch Processing

As the number of OLTP applications grows, business enterprises increasingly want to process their OLTP, batch, and query applications on the same computer hardware, using the same database, and at the same time. In a traditional, multiple-database installation, OLTP and batch applications do not interfere with one another because they execute on different databases. In a single-database installation, applications having different priorities compete for the same processing resources and need access to the same database. NonStop SQL Release 2 handles these competing demands by balancing processor loads according to the priority of the requesting applications.

Before NonStop SQL Release 2, a problem could arise when an application asked the disk process to retrieve data. (Traditionally, the disk process executes at a high priority even on behalf of a low-priority request.) A low-priority batch process executing in a lightly loaded CPU could send multiple requests to a disk process executing in another CPU. The low-priority requests could keep the disk process busy and slow down any OLTP process executing in this other CPU.

In NonStop SQL Release 2, the disk process includes a new scheduling algorithm that prevents low-priority batch and query applications from adversely affecting the performance of high-priority OLTP applications. In an internal Tandem benchmark, a background batch job did not affect the performance of online transactions. (This enhancement affects all requests to the disk process; thus, it applies to applications that use the Tandem Enscribe record management system as well as those that use NonStop SQL.) The new scheduling algorithm is described elsewhere in this issue of the *Tandem Systems Review* (Englert and Gray, 1990).

Operability and Manageability

As the corporate database grows in size and becomes increasingly distributed, it also becomes increasingly difficult to handle. Managing a complex, distributed network of databases can be human-resource intensive and, as a result, error prone. To make this task easier, NonStop SQL Release 2 has implemented several database management features, including online reorganization, semi-online partition split, and enhanced node autonomy.

Online Reorganization

As update and delete operations modify a database, the database can become fragmented. Over time, a fragmented database can degrade the performance of the applications that access it. To maintain satisfactory application performance, the database administrator must periodically reorganize the database. For many applications (such as 24-hour-a-day banking), it is especially desirable to be able to reorganize the database without taking it offline.

Online reorganization is a new disk process feature that allows a key-sequenced NonStop SQL table or a key-sequenced Enscribe file to be reorganized online. During the reorganization, online and batch applications can maintain full read and write access to the data.

To minimize the impact the online reorganization may have on the performance of online and batch applications, users can specify the rate at which the reorganization is performed. Furthermore, users can suspend and restart an online reorganization; these features allow users to suspend the reorganization during periods of heavy system load and continue the reorganization later, when the system load is lighter. Online reorganization is described elsewhere in this issue of the *Tandem Systems Review* (Smith, 1990).

Partition Split

As the size of a database grows, more partitions may be needed to accommodate the new data. For example, application activity may cause the first partition of a three-partition table to become full. In many DBMSs, the database administrator must unload the table, define a new table with four partitions, and load the unloaded data onto the new table. Unloading and loading a large table can take many hours or even days. This is unacceptable to applications that require extended operations (24 hours a day, 7 days a week).

NonStop SQL Release 1 helped to alleviate this problem by letting users add a new partition to the end of a table. NonStop SQL Release 2 lets users split a partition into two partitions while allowing read access to all partitions of the table and write access to all partitions of the table not involved in the partition split. If necessary, NonStop SQL can also move the affected data to the new partition. With this feature, the database designer does not have to worry about defining a table with enough partitions to accommodate anticipated growth.

Enhanced Node Autonomy

A DBMS should be flexible enough to obtain access to a distributed database even when a node in the database network is not available. A DBMS that implements the principle of node autonomy is restricted only when it cannot retrieve the requested data (or part of the data) because the particular node on which that data resides is not available. With node autonomy, a query such as an SQL SELECT statement can execute without error as long as all SQL database objects (table partitions, indexes, and catalogs) that contribute to the result of the query are available at execution time.

NonStop SQL Release 1 provided two forms of node autonomy. If a query access plan required the use of an index and the index was not available at execution time, NonStop SQL still returned the result of the query as long as the base table was available (Pong, 1988). Furthermore, a query compiled successfully as long as the referenced table was registered in a catalog that was available. The query compiled even if different partitions of the table were registered in other catalogs that were not available.

Full read and write access is maintained during online reorganization.

NonStop SQL Release 2 enhances these forms of node autonomy by allowing a query to execute even when a table partition specified in the query is not available at execution time (as long as that partition does not contribute data to the result of the query). Furthermore, if the access path requires the use of an index, NonStop SQL Release 2 allows the query to execute even when the primary partition of the index is not available. Again, the primary partition of the index must not contribute data to the result of the query. (In NonStop SQL Release 1, the query would have failed under these conditions.) With NonStop SQL Release 2, Tandem achieves nearly complete node autonomy.

Consider, for example, a query that specifies P1 as the first partition of a table that contains three partitions, P1, P2, and P3, and the first keys of the partitions as 1, 100, and 200, respectively:

```
SELECT * FROM P1
WHERE PRIMARY_KEY
BETWEEN 150 AND 200
```

Suppose partition P1 is not available at execution time. In NonStop SQL Release 1, this query would fail even though P1 does not contribute to the result of the query. In NonStop SQL Release 2, the query executes successfully.

The only requirement in NonStop SQL Release 2 is that all partitions required by the query access plan must be available. Consider the query:

```
SELECT * FROM P1
WHERE PRIMARY_KEY < 100
OR PRIMARY_KEY > 200
```

Suppose partition P2 is not available at execution time. If NonStop SQL chooses a query access plan that involves scanning the entire table, the query will fail because partition P2 is not available.

ANSI Compatibility

When NonStop SQL Release 1 was developed, ANSI SQL 86 had not yet become an industry standard. As a result, there were several minor incompatibilities between NonStop SQL and the ANSI SQL standard. NonStop SQL Release 2 eliminates several of these incompatibilities. NonStop SQL Release 2 also introduces features that will become part of the ANSI SQL 2 standard.

Compatibility with ANSI SQL 86

NonStop SQL Release 2 supports null values (ANSI, 1986). In SQL, a null value represents missing or unknown information. For example, an EMPLOYEE table may contain a field called SPOUSE_NAME. However, a given employee may not have a spouse. To represent the nonexistent spouse, NonStop SQL fills in the field with a null value. In essence, a null value allows a database to model real-life situations in which missing information is commonplace. With NonStop SQL Release 2, applications can define and manipulate unknown values in an industry-standard fashion.

NonStop SQL Release 2 also supports the UNION operator, another ANSI SQL 86 feature (ANSI, 1986). The UNION operator allows users to combine the results of two or more SELECT statements that have the same number of SELECT list entries and equivalent data types. NonStop SQL Release 2 makes it easy to fragment and distribute data across the system because users can recombine the data easily with the UNION operator.

Compatibility with ANSI SQL 2

NonStop SQL Release 2 also includes features that will be part of the future ANSI SQL 2 standard (ANSI, 1989). These new features are the LEFT JOIN operator and the DATETIME and INTERVAL data types.

With the LEFT JOIN operator, an application can preserve information from one or more joined tables without having matched that information in the joined columns. The LEFT JOIN operator is described elsewhere in this issue of the *Tandem Systems Review* (Vaishnav, 1990).

The new DATETIME and INTERVAL data types simplify the manipulation of date and time values, a welcome feature for application programmers. NonStop SQL Release 2 supports arithmetic operations on date, time, and interval data. It also includes several specialized functions that help to simplify the manipulation of these data. Finally, to allow for different national preferences, NonStop SQL supports the USA, European, and ANSI formats for displaying date and time values.

Conclusion

NonStop SQL Release 1 provided users with a superior, distributed, ANSI-compatible RDBMS that performed well enough to support production-level OLTP. NonStop SQL Release 2 applies the power of parallel processing to batch and query processing, making it an ideal DBMS for large, enterprise-wide databases. Also, NonStop SQL Release 2 allows OLTP, batch processing, and query processing to coexist in a single database system without adversely affecting OLTP applications.

By introducing the online reorganization and partition split features, NonStop SQL Release 2 makes it simpler and easier to maintain large databases. NonStop SQL also shows its commitment to ANSI standards by implementing ANSI SQL 2 features such as the LEFT JOIN operator and the DATETIME and INTERVAL data types. With Release 2, NonStop SQL offers businesses a DBMS that can manage all of their important information processing needs on a single database of record.

References

ANSI. 1986. *X3H2 Data Base Languages.* (January 1986.) American National Standards Institute.

ANSI. 1989. *X3H2 Data Base Languages.* (July 1989.) American National Standards Institute.

Borr, A. and Putzolu, F. 1988. High Performance SQL Through Low-Level System Integration. *Proceedings of SIGMOD 88.* ACM.

Englert, S. and Gray, J. 1990. Performance Benefits of Parallel Query Execution and Mixed Workload Support in NonStop SQL Release 2. *Tandem Systems Review.* Vol. 6, No. 2. Tandem Computers Incorporated. Part no. 46987.

Englert, S. et al. 1990. NonStop SQL Release 2 Benchmark. *Tandem Systems Review.* Vol. 6, No. 2. Tandem Computers Incorporated. Part no. 46987.

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.

Pong, M. 1988. NonStop SQL Optimizer: Query Optimization and User Influence. *Tandem Systems Review.* Vol. 4, No. 2. Tandem Computers Incorporated. Part no. 13693.

Smith, G. 1990. Online Reorganization of Key-Sequenced Tables and Files. *Tandem Systems Review.* Vol. 6, No. 2. Tandem Computers Incorporated. Part no. 46987.

Tandem Performance Group. 1988. Tandem's NonStop SQL Benchmark. *Tandem Systems Review.* Vol. 4, No. 1. Tandem Computers Incorporated. Part no. 11078.

Vaishnav, J. 1990. The Outer Join in NonStop SQL. *Tandem Systems Review.* Vol. 6, No. 2. Tandem Computers Incorporated. Part no. 46987.

Acknowledgments

I would like to thank the members of the NonStop SQL development team for their contributions to an outstanding release of NonStop SQL. Special thanks are also due to the reviewers of this article for providing technical and editorial suggestions.

Mike Pong is manager of the SQL Compiler Group in the Transaction Networks Division. He is the designer of the NonStop SQL optimizer. Before joining the SQL Compiler Group, Mike designed and implemented the autorollback feature of DP1 TMF.

Performance Benefits of Parallel Query Execution and Mixed Workload Support in NonStop SQL Release 2

Release 2 of the Tandem™ NonStop™ SQL distributed relational database management system transparently and automatically implements parallelism within individual SQL statements. By exploiting the Tandem multiprocessor architecture, NonStop SQL simultaneously executes portions of an SQL query on many processors, reducing its execution time to a fraction of what it would be on a single processor. At the same time, new Guardian™ 90 operating system support for mixed workloads allows NonStop SQL to execute parallel queries and batch jobs on a production-level database with little impact on the response time of concurrently running online transaction processing (OLTP) applications.

With parallel query execution, users can increase the speed of a NonStop SQL query almost linearly by adding processors and disk drives to their system. This performance benefit, called *speedup*, helps users to meet the increasing demand for up-to-date information. Parallelism also helps users to manage a growing database. When a batch job increases in size, users can keep its processing time constant by adding proportionately more equipment to their system. This performance benefit, called *scaleup*, is especially useful for batch jobs restricted to a fixed execution time (such as an overnight shift).

This is the first of two articles describing the performance benefits of parallel query execution in NonStop SQL Release 2. This article discusses the reasons for implementing parallel query execution, explains speedup and scaleup, and describes OLTP scaleup (provided by NonStop SQL Release 1). Next, the article describes in detail how parallel query execution provides speedup and scaleup for batch and query processing. Finally, the article outlines new Tandem system features that support parallelism in NonStop SQL by optimizing sequential processing and mixed workload performance.

The second article, "The NonStop SQL Release 2 Benchmark," describes the benchmark tests performed by Tandem staff that demonstrate near-linear speedup and scaleup for NonStop SQL queries on a variety of Tandem systems.

Why Tandem Developed Parallel Query Execution

Release 1 of NonStop SQL offered users a distributed relational database system that could support high-performance OLTP on a production-scale database. NonStop SQL was the first SQL system to offer distributed data (with node autonomy); distributed, fault-tolerant execution; and distributed transaction processing. It was integrated with the Tandem software environment, including the Guardian 90 operating system, Transaction Monitoring Facility (TMF™) logging system, Pathway transaction processing system, and Pathmaker™ application generator. Because of its low-level integration with Guardian 90 (and its integration with other Tandem software), NonStop SQL Release 1 provided outstanding performance for OLTP applications. Benchmark tests demonstrated its performance at over 200 DebitCredit transactions per second (tps) (*NonStop SQL Benchmark Workbook*, 1987).

Tandem had two reasons for implementing parallel query execution in Release 2 of NonStop SQL. First, users are increasingly choosing NonStop SQL for large databases (100 gigabytes or larger). Without parallelism, a query scanning a database at one megabyte per second would take about one day to search a 100-gigabyte database. With parallelism, the query can execute in less than an hour. Large databases require parallel execution for scanning data as well as for utilities such as building indexes and loading, dumping, and reorganizing data.

Second, users increasingly want to establish a single database of record that supports concurrent OLTP, batch processing, and query processing. OLTP systems have many processors and disks to support many small transactions doing random I/O. By automatically converting SQL statements to parallel execution, NonStop SQL can apply this OLTP hardware to a batch job running against the online database. Furthermore, by bringing batch jobs and queries from the information center (which contains stale data) into the online environment, NonStop SQL provides access to current data and satisfies the performance requirements of a single database of record. With the new mixed workload support in NonStop SQL Release 2, a low-priority batch job will not significantly degrade the response time of a simultaneously executing OLTP application.

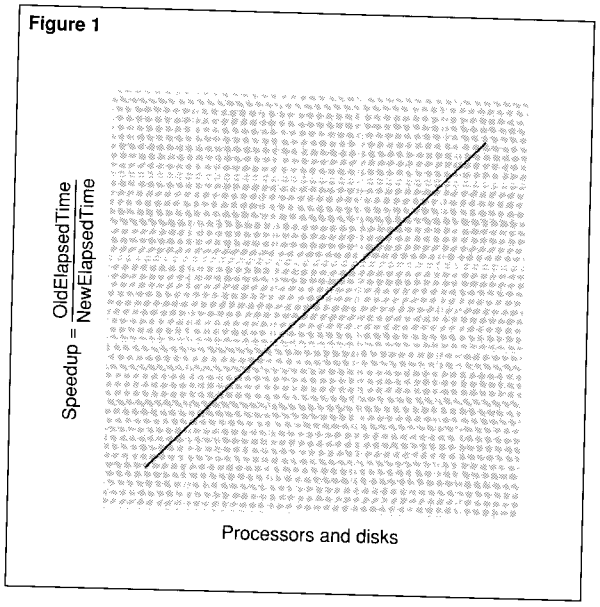
The Uses of Parallelism: Speedup and Scaleup

A multiprocessor system can be used to divide a big task into many smaller ones and solve them in parallel. Parallelism has two applications: speedup and scaleup. Speedup allows a task to be completed more quickly by breaking it into many smaller tasks. Scaleup allows a large task to be completed in the same time as a small one by using proportionately more processing power.

For example, consider a manufacturing application that runs a batch job every night to perform materials resource planning (MRP). The user faces a speedup problem if the MRP batch run takes thirty hours on a single processor. A parallel processing system with ten processors should be able to run the job in three hours. The user faces a scaleup problem if the batch job now doubles in size (because new products are added). With parallel processing, if the user adds another ten processors and a corresponding number of disk drives to the system, the MRP job should continue to execute in three hours.

Users with OLTP applications that grow also face a scaleup problem. If a company doubles the number of customers it serves, its order-entry OLTP system must process twice as many transactions per second. A parallel processing system should be able to double the size of its terminal network, processors, and database to meet the increased demand.

Figure 1.
This is a good speedup curve showing linear speedup in elapsed time as more processing elements are applied to the job.



How Speedup and Scaleup Differ

Speedup and scaleup are closely related concepts, but they differ in interesting ways. A system or an application can have good scaleup properties but no opportunities for speedup.

When users apply parallelism to speedup, they trade time for money by buying more equipment to finish a job more quickly. Successful speedup gives a linear (even) tradeoff between time and money; twice as much equipment produces an answer in half as much time. In some cases, users have already bought the hardware for OLTP, so batch programs can run at times of light load to get "free" speedup.

When users apply parallelism to scaleup, they can save money by adding new processors and storage modules to their system as the demand grows. In contrast, a system that does not exploit parallelism cannot be expanded incrementally. To improve performance, users must replace their entire hardware investment with new equipment. Thus, a system that scales up has significant financial benefits over traditional system designs.

The challenge is to design a system that automatically gives linear speedups and scaleups. There are exceptions; some tasks do not decompose even when parallelism is available. However, users can get linear speedups and scaleups when querying large tables using any of the SQL relational operators: SELECT, INSERT, UPDATE, DELETE, project, aggregate, and join. Evidence of linear speedups and scaleups was given by the Gamma System at the University of Wisconsin (DeWitt et al., 1988) and by the benchmark results described in "The NonStop SQL Release 2 Benchmark," the companion article in this issue of the *Tandem Systems Review* (Englert et al., 1990).

Defining Speedup

Speedup measures how much faster a parallel multiprocessor system completes a task than a single processor. It is defined by the formula:

$$\text{Speedup} = \frac{\text{OldElapsedTime}}{\text{NewElapsedTime}}$$

As users add more processors and disks, the new elapsed time for the job should be proportionately less than the old time. Figure 1 shows an ideal speedup curve, in which speedup increases in linear fashion.

Speedup and OLTP. Most systems do not display speedup because they do not exploit parallelism. (See Figure 2.) For example, Release 1 of NonStop SQL did not automatically decompose relational operations into smaller jobs that could be executed independently on multiple processors and disks. Instead, each individual job used a single processor and a single disk at a time. Therefore, NonStop SQL Release 1 displayed a speedup of one for a single job no matter how many processors and disks were added to the system.

Because NonStop SQL Release 1 focused on OLTP performance, it obtained parallel execution by running many independent transactions in parallel. Parallelism is explicit in OLTP applications, which consist of many small jobs. Moreover, dividing an individual transaction is impractical. When a small job is decomposed into parallel units of work, the system may spend more processing time starting and distributing the work than executing the transaction.

Speedup for Queries and Batch Jobs. Unlike an online transaction, an individual query or batch job is often large enough to benefit from parallel execution. Release 2 of NonStop SQL provides near-linear speedups for queries and batch jobs by automatically executing individual SQL statements in parallel.

Startup, Interference, and Skew. Typically, even parallel systems do not have linear speedups because of problems with *startup*, *interference*, and *skew*. (See Figure 3.) Startup problems occur because parallel processors take more time to begin working on a job than a single processor, just as a large group of people takes longer to begin a shared project than a small group.

After the processors start working, they can interfere with one another or queue behind a bottleneck. For example, in shared-memory multiprocessors, memory or software interference can cause a six-processor system to have only three times the power of a single processor. If the system grows beyond a certain size, adding an additional processor may introduce more interference than its processing power contributes to the system, causing the system to slow down rather than speed up.

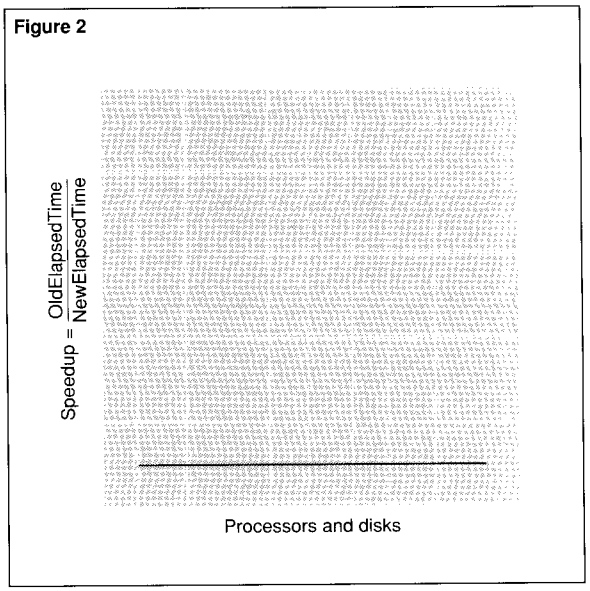


Figure 2.
This is a bad speedup curve typical of systems without parallelism. The application uses only one processor and disk at a time no matter how many are added, so there is no speedup at all.

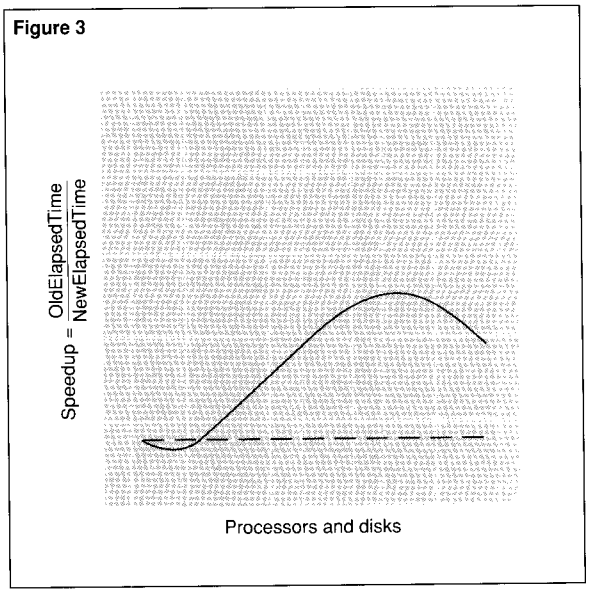


Figure 3.
This is another kind of bad speedup curve. The initial slowdown is due to parallelism startup costs; the nonlinear then diminishing speedup is due to interference problems, skew problems, or both.

Figure 4.
This is a good batch scaleup curve showing constant processing time as proportionately more processing elements are applied to a proportionately larger job.

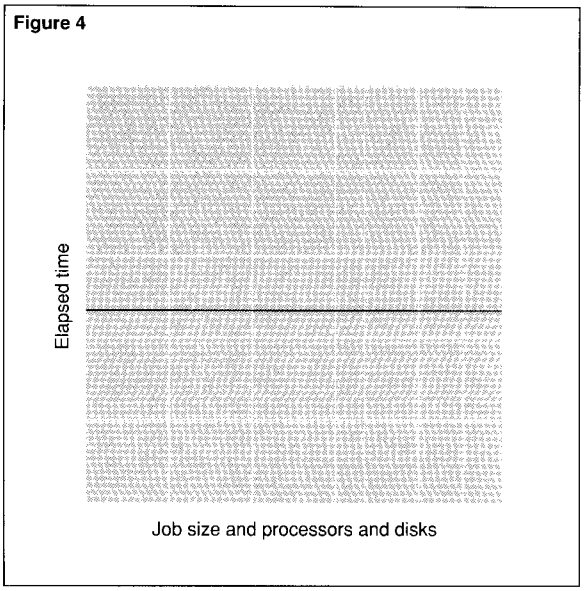
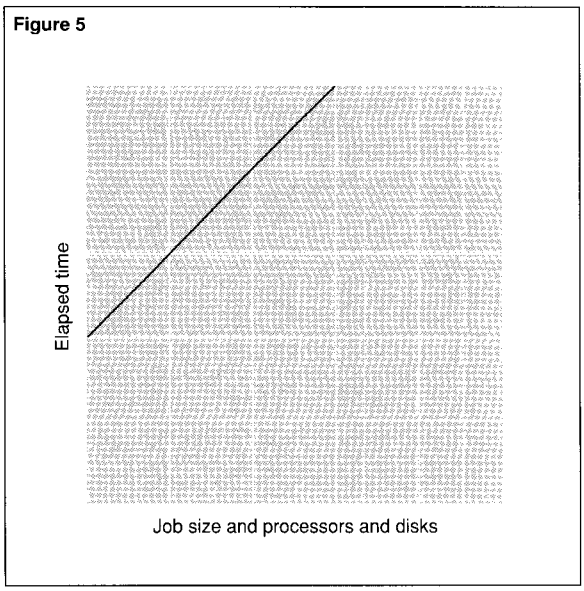


Figure 5.
This is a bad scaleup curve showing that as the job grows, the elapsed time grows even though more processing and disk elements are applied to the job.



Skew problems arise when a job is divided into such tiny units that the startup and processing variance is larger than the processing time for each unit. After skew begins to dominate, subdividing a job further does not increase the speedup. Startup, interference, and skew problems are inherent in parallel systems (Smith et al., 1989). Therefore, the speedup curve shown in Figure 3 ultimately flattens out and begins to curve downward.

Defining Scaleup

Scaleup measures the degree to which a parallel system can manage a growing workload. Scaleup has two forms: one applies to batch processing, the other to OLTP. Both types of scaleup postulate that the workload and the system grow in proportion to one another, but the two scaleup types differ in their goals. Batch scaleup should keep a constant elapsed processing time for a large job. OLTP scaleup should increase the system's transaction throughput (transactions per second) while keeping response times the same. Scaleup is defined by the formulas:

$$\text{Batch Scaleup} = \frac{\text{NewElapsedTime}}{\text{OldElapsedTime}}$$

and

$$\text{OLTP Scaleup} = \frac{\text{NewThroughput}}{\text{OldThroughput}}$$

Good batch scaleup numbers for an n-processor, n-disk system are close to 1. Good OLTP scaleup numbers for such a system are close to n.

Batch scaleup is a requirement for any application that must execute within a fixed time period (such as the nightly eight-hour graveyard shift). As the batch job grows, the system must continue to execute it within the batch window.

Figure 4 shows an ideal batch scaleup curve; the processing time remains constant as the job and system grow proportionately in size. Figure 5 shows a more typical batch scaleup curve. As the job grows, the processing time grows.

Figure 6 uses the specific example of the NonStop SQL Release 1 DebitCredit OLTP workload benchmark to illustrate linear OLTP scaleup of a system (*NonStop SQL Benchmark Workbook*, 1987). The transaction throughput (transactions per second) grows in linear fashion as the number of terminals, processors, and disks increases. (OLTP scaleup curves are similar to speedup curves, except that the abscissa indicates transaction throughput rather than speedup.)

Previous benchmark tests have shown that Release 1 of NonStop SQL provided near-linear scaleup for OLTP applications (Tandem Performance Group, 1988). To achieve this goal, NonStop SQL provides *inter-transaction parallelism*, in which many relatively simple transactions execute in parallel.

A system with good speedup on a large job will probably have good batch scaleup on smaller jobs. With scaleup, one must consider how large the job can grow before the system reaches a bottleneck or system limit. Ideally, one should hit an economic barrier long before the system hits a software or hardware limit.

Batch Speedup and Scaleup

NonStop SQL Release 2 exploits the Tandem parallel architecture to provide *intra-transaction parallelism*, in which a single SQL operation executes in parallel on many processors. Intra-transaction parallelism gives near-linear speedups and scaleups for batch SQL operations.

SQL is a nonprocedural, set-oriented data manipulation language. SQL operations are built on the following basic operations:

- Select all rows in a set that satisfy a predicate.
- Project (remove) certain fields from all rows in a set.
- Aggregate all values in a set (compute a function on the values in a set; for example, find their count, average, sum, minimum, or maximum value).
- Join the rows in two sets on some attribute to form a new set.

Figure 6

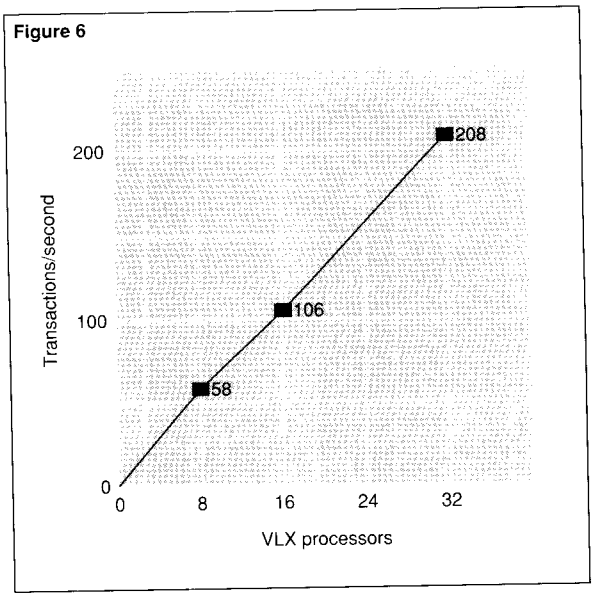


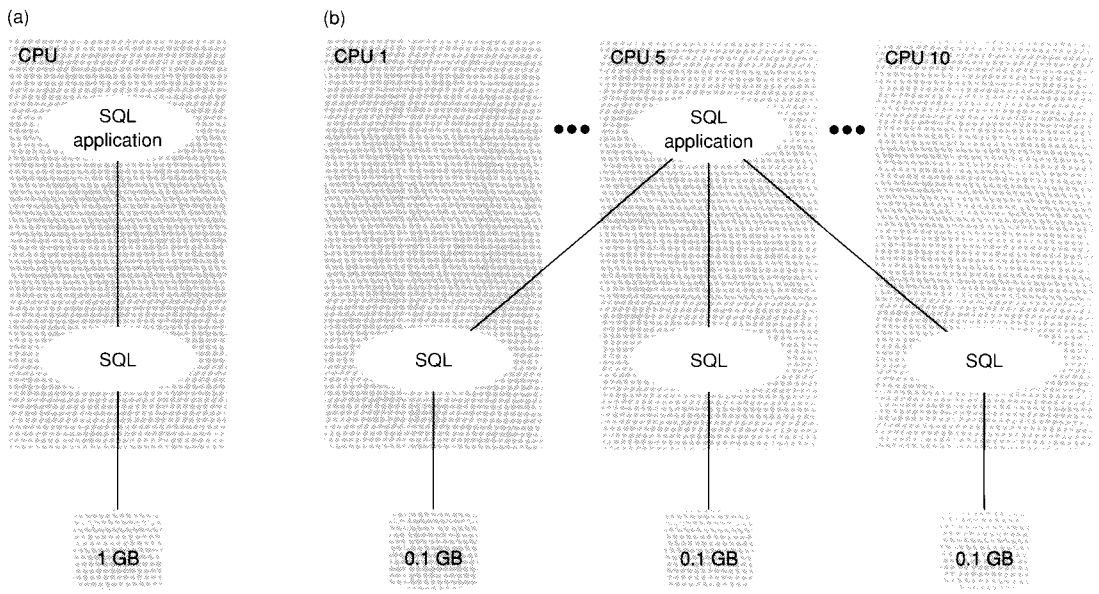
Figure 6.
OLTP scaleup of NonStop SQL Release 1 on the Debit-Credit OLTP workload. As processors grew from 4 to 32, the database grew from 3.2 million rows to 25.6 million rows, and the terminal network grew from 320 terminals to 2560 terminals. A 26-GB history table was also maintained. The throughput of the system scales almost linearly with the workload.

Each of these operations produces a new set that can be fed into other operations or deleted, updated, or inserted into an existing table. The power of the relational model derives from its ability to arbitrarily compose relational operators. If a system can give linear speedup and scaleup for each of these operators, then it should give linear speedup and scaleup for any combination of them.

Figure 7.

A typical speedup design. (a) The 1-GB table is stored on a single disk, and the SELECT, project, aggregate, or other relational operator is executed by a single SQL executor on a single processor and disk. (b) The same table is partitioned among 10 disks on 10 processors. The application running in one of the processors transparently accesses all 10 disks in parallel (one SQL executor server per disk). The parallel execution should run 10 times faster than the serial execution.

Figure 7



Partitioning the Database

Before NonStop SQL can execute an SQL operation in parallel, users must partition the table being operated on. Consider a 1-gigabyte table consisting of 10,000,000 rows, each 100 bytes long. Users could store the table on a single disk accessed by a single processor or, to achieve parallelism, they could partition it equally among ten disks and processors. If the database system gives location transparency, the partitions are invisible to the application program. The application can run in any of the processors and access the partitioned table as a single logical table. (See Figure 7.)

Now consider a SELECT operation requiring a complete scan of the table. The NonStop SQL layer of the application program automatically spawns an executor server process in each CPU that issues disk-process scan requests for the local partition. The NonStop SQL system scans all ten disks in parallel, achieving a speedup of ten. More information about how NonStop SQL implements parallel processing appears elsewhere in this issue of the *Tandem Systems Review* (Moore and Sodhi, 1990).

NonStop SQL achieves linear scaleup in a similar way. Suppose the database grows by a factor of ten, from a 1-gigabyte table comprising 10,000,000 rows to a 10-gigabyte table comprising 100,000,000 rows. As the table is scaled up, it is spread among ten disks and processors. (See Figure 8.) By operating on all ten disks in parallel, the NonStop SQL system can scan the 10-gigabyte database in the same time it took to scan the original 1-gigabyte database.

As these examples show, parallelism in NonStop SQL depends on two key features. First, users must partition the data horizontally among multiple disks and processors. Second, when the query is invoked, NonStop SQL must subcontract the execution of the relational operator to each processor (and table partition). This method works with the aggregate, project, UPDATE, and DELETE operators just as it does with SELECT.

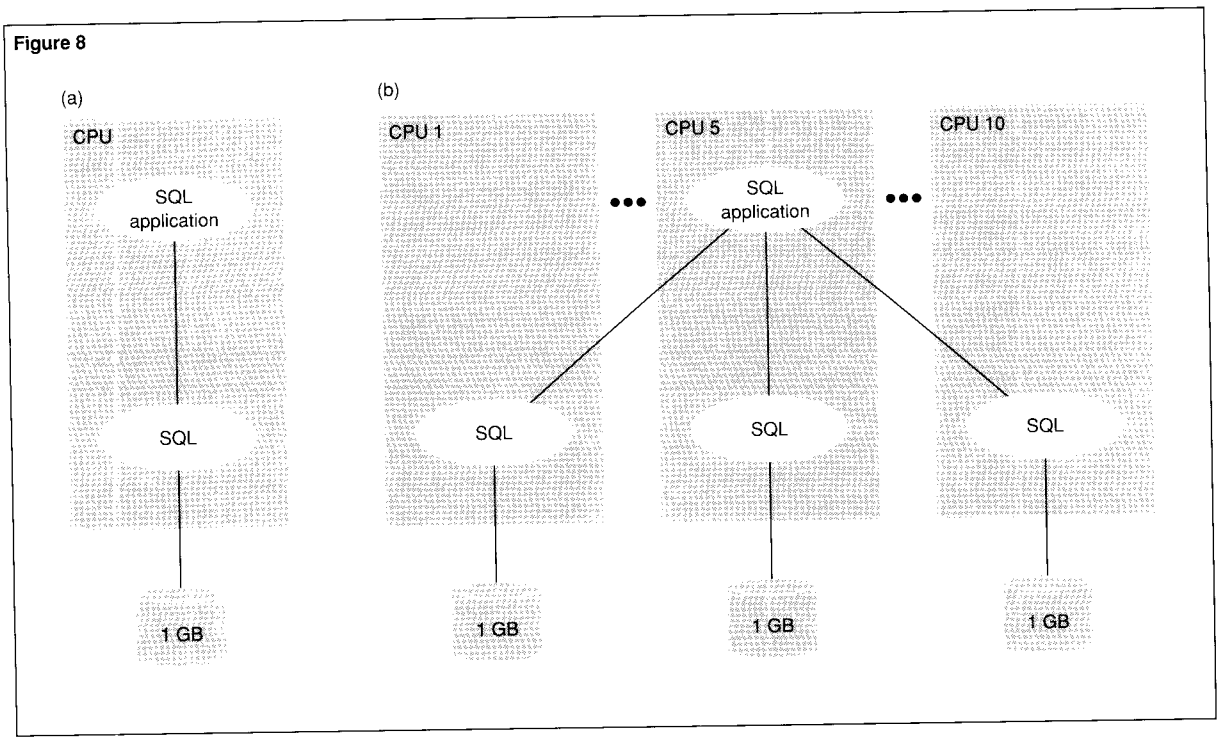


Figure 8.
A typical batch scaleup design. (a) The 1-GB table is stored on a single disk, and the SELECT, project, aggregate, or other relational operator is executed by a single SQL executor on a single processor and disk. (b) The same table has grown to 10 GB; it is partitioned among 10 disks on 10 processors. The application running in one of the processors transparently accesses all 10 disks in parallel (one SQL executor server process per disk). The parallel execution should run just as fast on the 10-GB table as the serial execution runs on the 1-GB database.

The INSERT operator adds one complication. If the target of the insert is an entry-sequenced table (a common format for intermediate tables and query answer sets), the end of the sequential table is a location of concentrated access (a hotspot) and consequently a potential bottleneck. To avoid the bottleneck, users can create a partitioned entry-sequenced table; NonStop SQL directs the inserts to the end of each local partition rather than the global end-of-file. If the target table is partitioned to match the partitioning of the query executors, NonStop SQL divides the hotspot among the executors and eliminates the bottleneck.

Even with parallel query execution, bottlenecks can occur elsewhere in the system. For example, if the answer set of a SELECT, project, or join operation is directed to an application, the overall system speed is limited by the speed with which the application can read and process the answer set.

The COBOL, C, or Pascal application can become the ultimate bottleneck because it runs on a single processor. Typically, NonStop SQL uses predicates to filter out most rows in a table, so the system does not reach this bottleneck until it has achieved considerable parallelism. Nevertheless, the application bottleneck remains a barrier to transparent parallelism. If this barrier is a problem, users must partition the application into several parallel applications, each working on a partition of the answer set (Reuter et al., 1989). To avoid the bottleneck internally, NonStop SQL horizontally partitions all intermediate answers among the processors and disks so that no bottlenecks occur within the processing of an SQL statement.

Parallel Join Operations

NonStop SQL uses a similar scheme to apply parallelism to join operations. NonStop SQL has three parallel join strategies. As shown in research prototypes (DeWitt et al., 1986; DeWitt et al., 1988), these techniques give both linear speedups and linear batch scaleups for join operations. The three parallel join strategies are described in detail elsewhere in this issue of the *Tandem Systems Review* (Moore and Sodhi, 1990).

The first and most common strategy applies when the tables being joined are already partitioned in the same way and the join is on a prefix of the tables' primary keys. Suppose the outer table is the item master of an invoice application and the inner table comprises the item details. The physical database design will probably cause corresponding partitions of the item details and item master to be located on the same disk. (The primary key of the item detail table will be a prefix of or the same as the primary key of the item master table.) In this case, each individual processor and disk execute one portion of the join and no interprocessor communication or interference occurs.

The second join strategy applies when the inner table is small and joined on a key field and the outer table is large and partitioned. In this case, each partition of the large table performs a nested join in parallel with the small (inner) table.

The third strategy applies when both tables are large or no useful key fields are involved in the join. In this case, NonStop SQL uses a hash function to repartition the two tables among all local processors. When the partitioning is complete, the join is divided into many small joins that can be performed independently. NonStop SQL can perform both the partitioning and the subsequent joining in parallel. NonStop SQL usually performs the individual partition joins as sort-merge joins.

Parallelism in a Commercial Environment

The parallel strategies implemented by NonStop SQL are similar in spirit to those used by the Teradata machine (The Genesis of a Database Computer, 1984; *DBC/1012 Database Computer System Manual, Release 1.3*, 1985) and other database machines, including Gamma (DeWitt et al., 1986; Schneider and DeWitt, 1989), Bubba (Smith et al., 1989), and Prospect (Reuter et al., 1989). The algorithms are a subset of those used in the University of Wisconsin Gamma Database machine.

Database machines are designed for large batch jobs and queries only. Users must buy and maintain a separate general-purpose system to run the computer network, transaction monitor, application programs, and operator interface. This two-system approach is inconvenient, and if the general-purpose system cannot provide speedup and scaleup as the application grows, the database machine's speedup and scaleup benefits may be lost.

However, Tandem implemented the algorithms on a conventional, commercial multiprocessor system rather than a specialized database machine like the Teradata or Gamma system. NonStop SQL also provides full transaction integrity for applications and data distributed in a local area network or a wide area network. Because of its many capabilities, the NonStop SQL system can be used for OLTP, networking, and running application programs as well as for batch and query processing.

System Support for Parallel Query Performance and Mixed Workloads

NonStop SQL requires low-level system support to achieve good performance for parallel batch queries that involve sequential processing. Several features in the Guardian 90 operating system and disk process, including bulk read-ahead, sequential block buffering, and asynchronous sequential write-behind, provide that support.

In a mixed workload environment in which batch jobs run concurrently with OLTP applications, the system must prevent batch jobs from interfering with OLTP performance. Two Guardian 90 features, browse access locking and the new mixed workload enhancement, allow parallel queries to execute without adversely affecting OLTP applications.

Sequential Processing Performance Improvements

NonStop SQL Release 2 substantially improves sequential read and write performance. Sequential reading benefits because the disk process (a part of the Guardian 90 system) detects sequential access and performs asynchronous bulk read-ahead of data (up to 56-kilobyte transfers). Thus, the disk process rarely has to wait for physical disk reads to complete. By the time the disk process needs it, the data has already been read from disk into memory.

Similar logic applies to inserts and updates. The SQL executor transparently buffers sequential write operations into 4-kilobyte blocks and sends them to the disk process. This sequential block buffering typically reduces the number of messages for a sequential insert or cursor update by a factor of twenty or more (depending on the record size). Also, it preserves update consistency by locking key ranges of the target table.

After the disk process receives the sequential insert or update data, it generates a single log record to cover all the inserts or updates, saving messages to the TMF logging system. The disk process buffers the sequential updates (and inserts) in cache until the log record has been written to the audit trail and enough data has accumulated to allow a single, large asynchronous write of multiple blocks to disk. Because of this read-ahead and write-behind logic, the application and disk execution of sequential reading and writing are completely overlapped, a traditional kind of parallelism.

Support for Mixed Workloads

Parallel query execution can exploit all the processors in a global network. For example, on a database partitioned among nodes in London, New York, and Tokyo, NonStop SQL automatically processes queries in parallel at all three sites. This benefit can also be a problem if the processors are executing online transactions at the same time.

The sequential processing features of the disk process (discussed previously) limit the impact of batch queries on online transactions to some extent. In particular, they minimize the number of I/Os generated by sequential programs and prevent sequentially accessed data pages from flooding the disk cache. However, additional support for mixed workloads is essential.

Browse Access Locking in NonStop SQL. NonStop SQL goes to considerable lengths to allow parallel queries to execute with minimal disruption of online transactions. First, batch read queries can specify BROWSE ACCESS, which allows the query to access data without setting any interfering locks and without being stalled by the locks of other operations. Many batch reports can operate using browse access locking because they produce ad hoc or statistical reports and need only an approximate view of the database. Operations that need a consistent picture of the database can specify either STABLE ACCESS or REPEATABLE ACCESS as the locking option, but these forms of locking can cause the batch operations to delay online transactions.

Mixed Workload Enhancement. In the past, and in spite of the other features that support mixed workloads, batch NonStop SQL requests involving sequential access could still adversely affect a simultaneously executing OLTP workload. The problem was caused by an anomaly in preemptive priority schedulers such as the one used by the Guardian 90 operating system.

When users execute a batch application at low priority, the system services it only when no high-priority task is ready to execute in its processor. However, if a low-priority task asks a high-priority server to perform an operation, the request is executed at the server's high priority. This problem, known as *priority inversion*, can cause high-priority servers executing on behalf of low-priority requesters to monopolize the processor in which they run, delaying servicing of other high-priority jobs.

In particular, the priority inversion problem can occur with the Tandem Disk Process 2 (DP2), a high-priority disk server. A low-priority batch job in a lightly loaded processor can create many requests to a DP2 process executing in a busy processor used by high-priority OLTP jobs. Because DP2 has higher priority than any user process, it executes on behalf of the low-priority batch job, making the processor unavailable to the OLTP job.

The problem is exacerbated if the requests generated by the batch job require lengthy servicing by DP2. Requests such as SQL queries requiring full table scans not only make DP2's processor unavailable to high-priority jobs for relatively long periods, but also delay servicing of DP2 requests made by other processes. This is true even though the DP2 request queue is ordered by the priority of the requesting process, because a low-priority request may arrive and be serviced when the queue is momentarily empty.

With NonStop SQL Release 2, the priority inversion problem becomes critical because a single query can tie up several processors. This can occur when a query causes all partitions of a table to be scanned in parallel and several DP2 processes execute scan requests simultaneously.

Two new features in DP2 significantly reduce the impact of priority inversion and allow batch jobs to execute without severely degrading performance of high-priority OLTP jobs. First, DP2 delays initiating service to a low-priority requester if there are ready-to-execute jobs of higher priority in DP2's processor. This allows other processes in DP2's processor to perform their work and approximates uniprocessor priority scheduling. The DP2 request is processed only when there are no processes of higher priority on the processor's ready list.

Second, once DP2 begins processing a long-running request, it "comes up for air" every few records to see if high-priority processes need the processor or high-priority DP2 requests are pending. In either case, DP2 preempts processing of the current request, which allows the high-priority processes to run or frees DP2 to service the high-priority requests. In this way, low-priority batch jobs are serviced only if the concurrently running high-priority OLTP workload does not need processor and disk resources. Thus, large NonStop SQL queries executing in parallel on multiple processors become viable in a mixed workload environment.

Conclusion

NonStop SQL uses many techniques to detect and exploit parallelism, including parallel query plans and algorithms, requester-server structuring, sophisticated concurrency control, distributed transaction management, preemptive priority scheduling, read-ahead, and write-behind. With these features, NonStop SQL provides near-linear speedups and scaleups for batch jobs and queries. At the same time, it ensures that batch jobs and queries do not adversely impact high-priority, response time-critical OLTP applications, which makes NonStop SQL well suited for mixed workload environments.

To verify the performance of NonStop SQL, Tandem staff measured several sample queries on Tandem VLX and CLX systems. That benchmark test, which demonstrates near-linear speedup and scaleup for basic queries on a uniform database, is described in "The NonStop SQL Release 2 Benchmark," the companion to this article.

References

DBC/1012 Database Computer System Manual, Release 1.3. 1985. Part No. C10-0001-01. Teradata Corporation.

DeWitt, D. et al. 1986. Gamma — A High Performance Dataflow Database Machine. *Proceedings of the 12th VLDB (Very Large Data Base).*

DeWitt, D. et al. 1988. A Performance Analysis of the Gamma Database Machine. *Proceedings of the 1988 ACM SIGMOD Conference.*

Englert, S. et al. 1990. The NonStop SQL Release 2 Benchmark. *Tandem Systems Review.* Vol. 6, No. 2. Part no. 46987. Tandem Computers Incorporated.

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

NonStop SQL Benchmark Workbook. 1987. Part no. 84160. Tandem Computers Incorporated.

Reuter, A. et al. 1989. *Progress Report #5 of Prospect Project.* Institute of Parallel and Distributed Super-Computers, University of Stuttgart.

Schneider, D. and DeWitt, D. 1989. A Performance Evaluation of Four Parallel Join Algorithms in a Shared-Nothing Multiprocessor Environment. *Proceedings of the 1989 ACM SIGMOD Conference.*

Smith, M. et al. 1989. An Experiment on Response Time Scalability. *Proceedings of the Sixth International Workshop on Database Machines.*

Tandem Performance Group. 1988. A Benchmark of NonStop SQL on the DebitCredit Transaction. *Proceedings of the 1988 ACM SIGMOD Conference.*

The Genesis of a Database Computer: A Conversation with Jack Shemer and Phil Neches of Teradata Corporation. 1984. *IEEE Computer.* Vol. 17, No. 11.

Susanne Englert has worked in Tandem's Performance Measurement and Analysis group in Software Development since 1986. She provided performance support for the Multilan product and NonStop SQL Release 2. Her most recent project involved extensive analysis and testing of the DP2 mixed workload enhancement.

Jim Gray works in the Software Development and has worked on the design and implementation of NonStop SQL. He has also worked on developing a word processor, system dictionary, parallel sort, and enhancements to the Encompass data management system. Jim is currently writing a book on transaction processing.

The NonStop SQL Release 2 Benchmark

Release 2 of NonStop™ SQL, the Tandem™ distributed relational database management system, transparently and automatically executes NonStop SQL queries in parallel on multiple processors. Audited benchmark tests have demonstrated that the parallel query execution implemented in Release 2 of NonStop SQL achieves near-linear speedup and scaleup for five different SQL query types on a uniform database. The benchmark queries included a SELECT that returned no data, an INSERT, an UPDATE, a representative aggregation function (AVG), and a join.

Speedup allows users to decrease the elapsed time of a job by adding processors and disks to their system. *Scaleup* allows users to process a large job in the same elapsed time as a small one by adding hardware to their system.

This is the second of two articles describing the performance benefits of parallel query execution in Release 2 of NonStop SQL. The first article, "Performance Benefits of Parallel Query Execution and Mixed Workload Support in NonStop SQL Release 2," explains how parallelism provides speedup and scaleup for SQL queries.

This article describes the benchmark tests run for Release 2 of NonStop SQL. It defines the components of the benchmark (database tables, query set, and hardware), explains how the tests were run, and describes the results of each query.

Defining the Benchmark

Parallel query execution benefits only certain types of queries and tables. In general, it can be applied to queries (including some joins) that require scans of large partitioned tables or partitioned index files. The purpose of the benchmark was to demonstrate the performance characteristics of parallel query execution using simple queries that satisfied these conditions and represented common types of batch operations.

The test query set included all the basic NonStop SQL relational operators. The organization of the tables allowed NonStop SQL to take the fullest advantage of parallel execution. To measure both speedup and scaleup, the Tandem staff performed each test on systems that grew incrementally in size.

The Database Tables

The benchmark database tables were modeled on the Wisconsin Database schema (Bitton et al., 1983). The rows were 200 bytes long, consisting of integer and character fields filled with random values. Figure 1 shows the definition of a single table with *n* rows.

In the complete definition, all fields were declared DEFAULT SYSTEM NOT NULL and the numerics declared UNSIGNED. Figure 1 omits these attributes for the sake of brevity. All the tables were transaction-protected, allocated with 4-kilobyte pages and large extents, and used row-granularity locking. (These features are defaults in NonStop SQL.)

Tandem constructed a data generator that builds the table partitions in parallel. The generator builds multi-gigabyte tables in less than an hour and uses a novel scheme to generate the random values (Englert and Gray, 1990).

The tables were uniform in the sense that all the partitions were of equal size, and the rows qualifying in the SELECT, UPDATE, INSERT, and join queries were distributed evenly across all partitions. This ensured, for example, that a query that scanned the entire table and updated 1 percent of all the rows spent about the same time on each partition and updated approximately 1 percent of the rows in each. As a result, all of the parallel executor processes would finish at roughly the same time. The tables were constructed with these properties in order to demonstrate the best-case performance of parallel query execution. Users with databases having very uneven distribution of data or widely varying partition sizes might want to consider reorganizing their tables to take fullest advantage of parallelism.

Figure 1

```
CREATE TABLE f1(  unique1     NUMERIC(8) ,   --  unique random n[0..n-1]
                  unique2     NUMERIC(8) ,   --  primary key, unique [0..n-1]
                  two         NUMERIC(8) ,   --  random [0..1]
                  four        NUMERIC(4) ,   --  random [0..4]
                  ten         NUMERIC(4) ,   --  random [0..10]
                  twenty      NUMERIC(8) ,   --  random [0..20]
                  onepct      NUMERIC(8) ,   --  random [0..(n/100)-1]
                  tenpct      NUMERIC(8) ,   --  random [0..(n/10)-1]
                  twenpct     NUMERIC(8) ,   --  random [0..(n/5)-1]
                  fiftypct    NUMERIC(8) ,   --  random [0..(n/2)-1]
                  hundpct     NUMERIC (8) ,  --  random [0..n-1]
                  odd1pct     NUMERIC (8) ,  --  random [1..(n/100)-1]
                  even1pct    NUMERIC (8) ,  --  2 x odd1pct
                  stringu1    CHAR (52) ,    --  random string
                  stringu2    CHAR (52) ,    --  random string
                  stringu3    CHAR (52) ,    --  random string
                  PRIMARY KEY unique2 );
```

Figure 1.
The definition of a table with n *rows.*

The Query Set

The query set consisted of five simple queries that included all the basic NonStop SQL operators. It was considered sufficient to demonstrate linear speedup and scaleup for the basic operations because more complex queries composed of several operations would retain the same speedup and scaleup properties.

- The SELECT 0% query scanned the entire table but did not return any data. It measured how quickly NonStop SQL can sequentially scan rows. The performance characteristics of a zero-selectivity table scan are important because all the other queries are based on this operation.
- The INSERT/SELECT 1% query scanned the entire table and inserted a random 1 percent of the rows into a target table. As well as measuring the cost of scanning the entire table, this query measured the additional overhead of returning data to the application and inserting it into a target table.
- The AVG query computed the average value of a field in the table. It tested the performance of aggregation functions.
- The UPDATE 1% query scanned the entire table and updated 1 percent of the rows at random. It measured the additional overhead of logging and locking updates.
- The join query joined a table with a copy of itself via the primary key. The join operation included a 1 percent selection and a 50 percent projection, which made the target table a manageable size (1 percent of the original table). The result of the join was inserted into a new table.

The use of parallel query execution was transparent to the queries. No special programming was required except for issuing a single NonStop SQL directive that causes the query optimizer to consider parallel execution in choosing its access plan.

The queries in the benchmark tests also had location transparency. The queries did not say where the tables were; they could, for example, have been partitioned between Tokyo, New York, and London. If a table being scanned had partitions in those three cities, NonStop SQL would create appropriate server executors to scan the data in parallel in each city (if that were the fastest way to get an answer).

The Hardware Environments

Tandem performed the queries in two contexts, one to measure speedup and one to measure scaleup. It performed the same queries in two hardware environments: the entry-level Tandem CLX™ system and the high-performance Tandem VLX™ system. Thus, the benchmark produced four curves for each query.

Each CLX/780 processor is rated at about 4 DebitCredit transactions per second (tps); the eight-processor CLX system is rated at 30 tps. Each VLX processor is rated at 7 tps; the sixteen-processor VLX system is rated at over 100 tps.

The CLX configuration consisted of two, four, or eight processors, each with 16 megabytes of memory and two mirrored pairs of data disks. The CLX disks each hold about 300 megabytes of formatted data.

The VLX configuration consisted of two, four, eight, or sixteen processors, each with 16 megabytes of memory. As with the CLX system, each processor had two mirrored pairs of data disks. The VLX disks each hold about 800 megabytes of formatted data.

In both cases, the first and second processors had an extra disk pair attached. (Each processor had a total of three disk pairs.) On the first processor, the extra disk stored programs ($SYSTEM). On the second processor, it stored the transaction log (audit trail). All disk caches were configured at 2 megabytes.

To simplify the measurements, Tandem kept all eight CLX processors and sixteen VLX processors (and their disks) attached at all times. For the two-processor tests, Tandem used only the first two processors and their disks. For the other tests, Tandem also used only the required number of processors. In every test, all disks were configured as mirrored volumes.

Table Sizes

Tandem used a fixed table size for the speedup tests. The table was successively partitioned among two, four, eight, and sixteen processors and disks. In each case, Tandem measured the elapsed time for each query and plotted the resulting speedup curves. The fixed-size tables were called F2, F4, F8, and F16.

Tandem used a fixed partition size for the scaleup tests. In each successive test, the table doubled in size. The tables had two, four, eight, and finally sixteen partitions. The growing tables were called G2, G4, G8, and G16.

Table 1 shows the number of rows per table. For example, table G16 on the VLX system had 16 million rows evenly divided among 16 mirrored disks attached to 16 VLX processors. Table G16 contained 3.2 gigabytes of data. Table F8 on the CLX system had 2,441,400 rows partitioned among 8 CLX processors and disks. Each partition contained 305,180 rows. Table F8 contained 488 megabytes of data.

Tandem chose these table sizes to allow all the tables, temporary results, and answers to fit on the disks at once. Tandem first built all the tables (F2 through F16, G2 through G16) and then ran the experiments. The chosen row counts allowed the entire test suite to run within a day, so that all the tests could be audited in a reasonable time. There were 5 queries and 14 tables (8 on the VLX system and 6 on the CLX system), totalling 70 tests in all. To complete the benchmark in a day, the average test had to run in a few minutes. As it turned out, however, it was possible to audit only 59 of the 70 tests.

Table 1.
Table sizes.

Type	VLX	CLX
F Tables	8,000,000 rows	2,441,440 rows
G Tables	1,000,000 rows/partition	420,000 rows/partition

The results of the INSERT queries were directed to an entry-sequenced table that was partitioned among the processors to allow parallel inserts. (To reduce the number of messages, the SQL executor buffers inserted rows into 4-kilobyte blocks before passing them to the disk process. This is called sequential block buffering of inserts.) Writes to disk also benefited from a feature known as sequential asynchronous write-behind, which permits the disk process to perform multiple logical writes as a single, large (up to 28-kilobyte) physical transfer.

Table 2.
Speedup ratios of all audited tests.

Partitions	SELECT 0%	INSERT/SELECT 1%	AVG	UPDATE	JOIN
VLX					
2	1.00	1.00	1.00	1.00	–
4	1.99	1.99	2.02	2.00	–
8	3.86	3.87	3.86	3.67	–
16	7.21	7.31	7.51	7.00	–
CLX					
2	1.00	1.00	1.00	1.00	1.00
4	1.95	1.98	1.97	1.86	1.99
8	3.70	3.67	3.86	3.49	3.82

Table 3.
Scaleup ratios of all audited tests.

Partitions	SELECT 0%	INSERT/SELECT 1%	AVG	UPDATE
VLX				
2	1.00	1.00	1.00	1.00
4	1.00	1.01	1.00	1.03
8	1.01	1.02	1.03	1.08
16	1.05	1.04	1.05	1.10
CLX				
2	1.00	1.00	1.00	1.00
4	0.99	1.01	1.01	0.98
8	1.02	1.06	1.02	1.04

Running the Benchmark Tests

The Tandem staff ran the tests as scripts fed to the NonStop SQL conversational interface (SQLCI). SQLCI optionally displays statistics on such metrics as elapsed time, CPU time, rows accessed, and messages sent. During the tests, the Tandem staff ran Measure™, the Tandem system performance monitor, to measure CPU, process, message, file, and disk activity. Codd and Date, Incorporated, audited the tests; Tables 2 and 3 are based on the table that appears in the auditor's report (Sawyer, 1989).

Because parallel query execution can be resource-intensive, users must explicitly request that NonStop SQL consider parallel query plans. The default option generates sequential plans, just as in Release 1 of NonStop SQL. Therefore, at the beginning of the test run, the script contained this directive:

```
CONTROL EXECUTOR PARALLEL
EXECUTION ON;
```

Thereafter, all compiled plans used multiple executors if that was the quickest way to get the answer. (The NonStop SQL optimizer would still use serial plans for queries such as single-row SELECT or UPDATE operations, which get no parallel speedup or scaleup. Users can turn off parallelism by executing the directive CONTROL EXECUTOR PARALLEL EXECUTION OFF;.)

The SELECT 0% Query

The first series of tests studied the speedup and batch scaleup of a zero-selectivity table scan. This query caused NonStop SQL to read all the rows in the table but return none to the application. The actual query was:

```
SELECT   *
FROM     =table
WHERE    hundpct > ?tablesize
         FOR BROWSE ACCESS;
```

The notation =table is a logical table name supported by NonStop SQL. The same query was run repeatedly, successively setting =table to F2, F4, F8, and F16 and to G2, G4, G8, G16. The term ?tablesize was a host-language variable. Before each run, it was set to the size of the table being scanned. Because there was no index on hundpct, the query required NonStop SQL to perform a full table scan. Because hundpct was always less than ?tablesize, this predicate was always false and the SELECT returned no rows.

To test speedup, Tandem ran the query on tables F2, F4, F8, and F16, as appropriate, on the VLX and CLX systems. Figure 2 shows the resulting speedup curves. To test scaleup, Tandem ran the query on tables G2, G4, G8, and G16. Figure 3 shows the resulting scaleup curves. Because the CLX system had only eight processors, queries were not run on tables F16 and G16 on the CLX system.

Figures 2 and 3 show a near-linear speedup and scaleup of table scans on both the CLX and VLX systems. Startup delays caused the slight nonlinearity of the CLX and VLX systems. Each job's elapsed time was only a few minutes at full speedup. When the entire job's elapsed time was that short, the startup time for the eight or sixteen NonStop SQL execution processes (a second or two per process) caused a slight slowdown of the overall job. The speedup and scaleup curves did not display interference or skew problems.[1]

[1]Englert and Gray (1990) address the topics of startup, interference, and skew problems in detail.

Figure 2

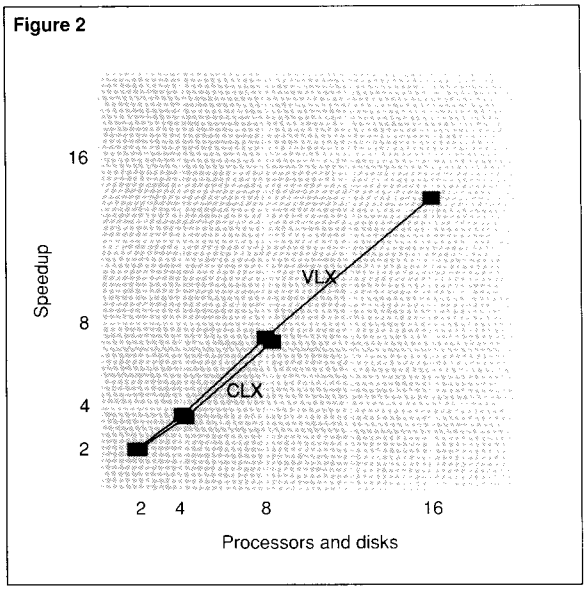


Figure 2.
Speedup curves for SELECT 0% on VLX and CLX.

Figure 3

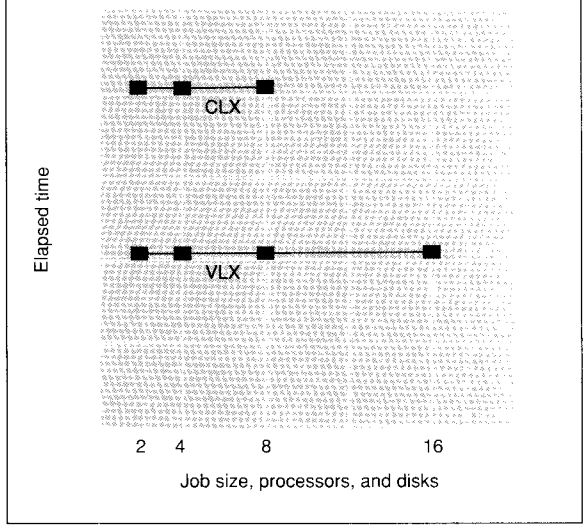


Figure 3.
Batch scaleup curves for SELECT 0% on VLX and CLX.

Figure 4.
Speedup curves for INSERT/SELECT 1% on VLX and CLX.

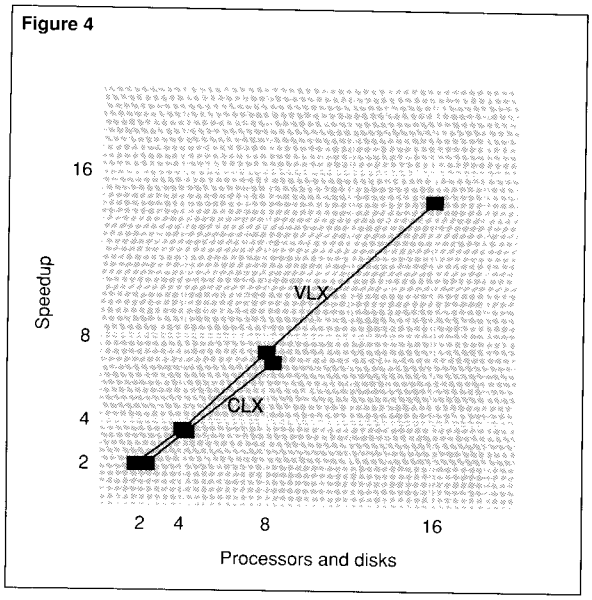
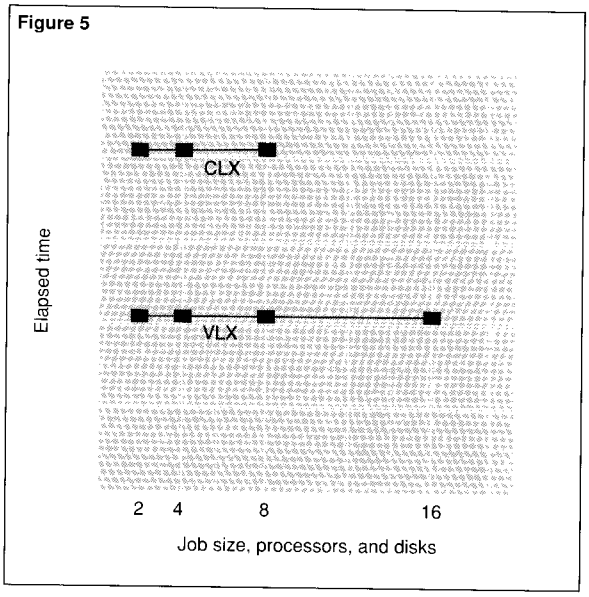


Figure 5.
Batch scaleup curves for INSERT/SELECT 1% on VLX and CLX.



These figures (as well as the figures that follow) treat the two-processor case as a speedup of two over the one-processor case, so all the speedup numbers are doubled. Also, the CLX scaleup tables appear above the VLX tables to indicate that a job runs about twice as fast on a VLX system as on a CLX system.

The INSERT/SELECT 1% Query

The next series of tests studied the speedup and batch scaleup of a 1 percent SELECT query that inserted its output into an entry-sequenced table. The result table was partitioned to exploit the parallel INSERT feature. The actual query was:

```
INSERT   INTO =result
SELECT   *
FROM     =table
WHERE    hundpct < (?tablesize/100)
         FOR BROWSE ACCESS;
```

The hundpct column took random values between 0 and the ?tablesize value, so this query selected a random 1 percent subset of the table. (Before the query was executed, the variable ?tablesize was set to the relevant table size in rows.) Tandem ran the query for all the tables on the VLX and CLX systems.

Figures 4 and 5 show near-linear speedup and scaleup curves for the SELECT/INSERT 1% query. The curves were virtually identical to those in Figures 2 and 3, except that here the speedup curve of the CLX system coincided with that of the VLX system. Also, the startup times caused less distortion of the scaleup curves than they did in the table scans because the startup time was a smaller fraction of elapsed time.

NonStop SQL chose an execution plan for the INSERT/SELECT 1% query that created an SQL server executor in each processor. (One server executor handled each partition in the table.) Each executor asked the corresponding disk process to perform a sequential scan of its partition. The disk process returned 1 percent of the partition's rows to the executor.

Next, the SQL executor sent the rows to be inserted into the local partition of the result table. Using sequential block buffering, it passed the rows to the disk process servicing the result table in blocks of about 20 (20 rows times 200 bytes per row fills up the 4-kilobyte buffer). The executor used only a single message for each block. The block of inserts generated a single log record when it arrived at the disk process. When several blocks of sequential inserts had accumulated in the disk's cache, all the blocks were written to disk in a single physical transfer.

The AVG Query

The third set of tests studied the speedup and batch scaleup of an AVG query, computing the average value of a numeric field in a table. The actual query was:

```
SELECT   AVG(onepct)
FROM     =table
         FOR BROWSE ACCESS;
```

Tandem ran the query for all the tables on the VLX and CLX systems. Figures 6 and 7 show near-linear speedup and scaleup curves for the AVG query. The execution plan directed each processor to compute the row sum and count of its partition. The application process combined all the individual computations to compute the global average.

The UPDATE Query

The fourth set of tests studied the speedup and batch scaleup of an UPDATE query. The query scanned the table in parallel and updated 1 percent of the rows, all within one transaction. The UPDATE query tested the ability of the transaction log to absorb the updates generated by eight CLX processors and sixteen VLX processors. The actual query was:

```
UPDATE   =table
SET      four = four + 117
WHERE    hundpct < (?tablesize/100);
```

Figure 6

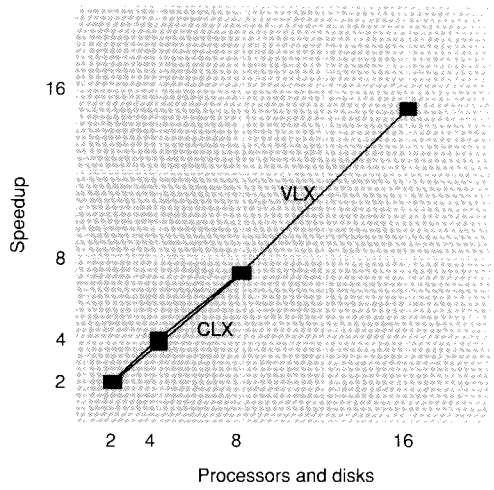


Figure 6.
Speedup curves for an AVG query on VLX and CLX.

Figure 7

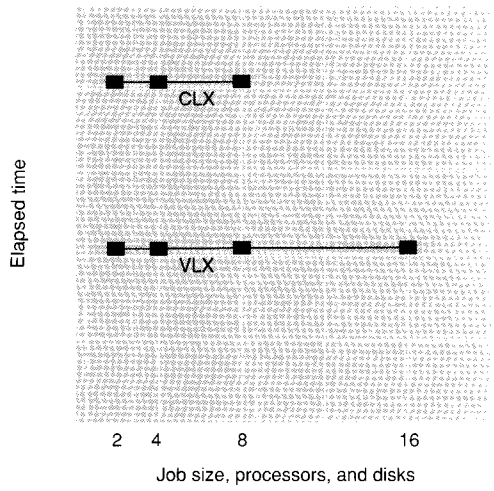


Figure 7.
Batch scaleup curves for an AVG query on VLX and CLX.

Figure 8.
Speedup curves for an UPDATE operation on VLX and CLX.

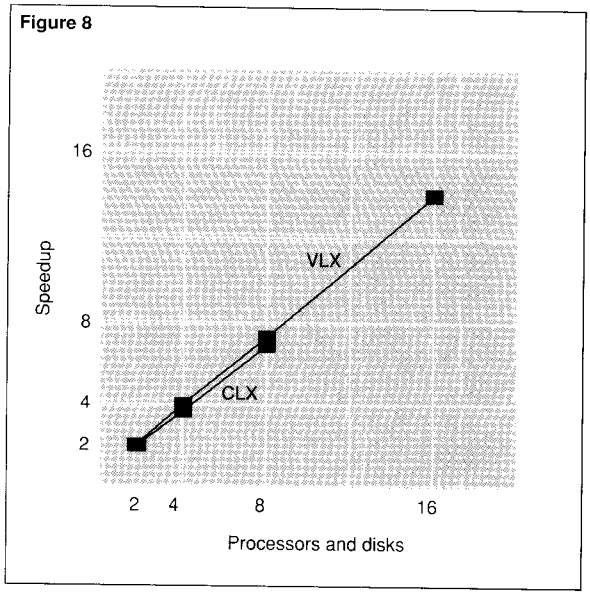
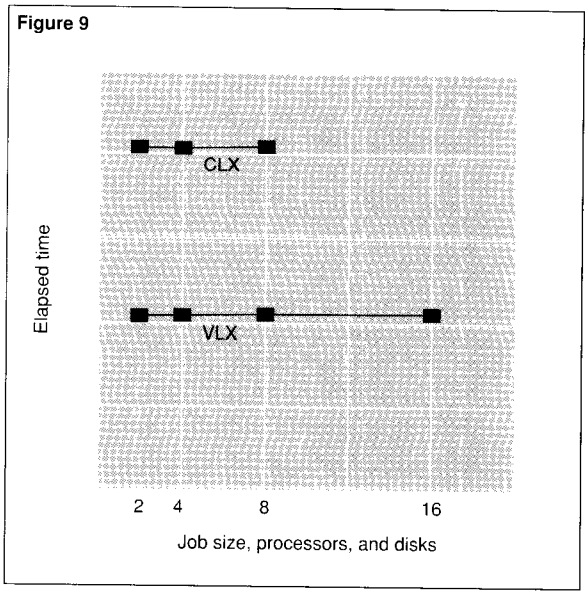


Figure 9.
Batch scaleup curves for an UPDATE operation on VLX and CLX.



Tandem ran the query for all the tables on the VLX and CLX systems. Again, Figures 8 and 9 show near-linear speedup and scaleup curves for the UPDATE query. The decline in the elapsed time for the four-processor CLX case fell within experimental error.

The UPDATE query ran as a single transaction. It defaulted to specifying the option STABLE ACCESS (all rows were locked when read), but locks were immediately released if the row was not updated. The query generated about 16 megabytes of log data on the F16 table. This did not saturate the log process and disk (in CPU 1). If the query had been a 100 percent UPDATE, the log process and disk would have had to absorb about 4 gigabytes of audit trail data. In that case, the query would have reached a bottleneck on the log, which would have limited the speedup and scaleup.

The UPDATE query showed the greatest nonlinearity of any of the tests: a 12.5 percent deviation at 16 processors. This indicated that the log processing (performed by CPU 1) slowed down the UPDATE query being performed by CPU 1. If and when this became a serious issue, the database manager could move the database disks from CPU 1 in order to devote CPU 1 to supporting the log activity.

Parallel Join Operations

The final tests studied the speedup and batch scaleup of a parallel join operation. In these tests, each of the tables F2, ..., F8 and G2, ..., G16 were joined with copies of themselves. The join operation occurred on the primary key (unique2).

To reduce the size of the join answer, the query added a selection expression that limited the qualifying rows to 1 percent of the table. The result of the join operation was placed in a partitioned, entry-sequenced target table. Figure 10 shows the actual query.

NonStop SQL executed the query in parallel by scanning the two tables (in parallel), filtering out the desired 1 percent of the rows. NonStop SQL joined each pair of partitions in parallel and inserted the result of that mini-join into the result table. The INSERT operation was similarly partitioned. This operation was completely parallel, so near-linear speedup and scaleup were expected.

The Tandem staff had planned to execute the join query for all tables on both the CLX and VLX systems. However, because of time constraints, the staff was able to execute and audit only the CLX speedup tests for the join query. Nevertheless, the tests that were completed indicated that the results of the remaining tests would have been similarly linear.

Figure 10

```
INSERT INTO = result
SELECT   one.unique1, one.unique2, one.two, one.four,
         one.ten, one.twenty, one.onepct, one.tenpct,
         one.twenpct, one.fiftypct, one.hundpct,
         one.odd1pct, one.even1pct, one.stringu1,
         two.stringu2, two.stringu3
FROM     =table1 one, =table2 two
WHERE    one.unique2 = two.unique2
AND      one.hundpct <= (?tablesize/100 -1)
AND      two.hundpct <= (?tablesize/100 -1)
         FOR BROWSE ACCESS;
```

Figure 10.
Actual text of the parallel join operation.

Figure 11

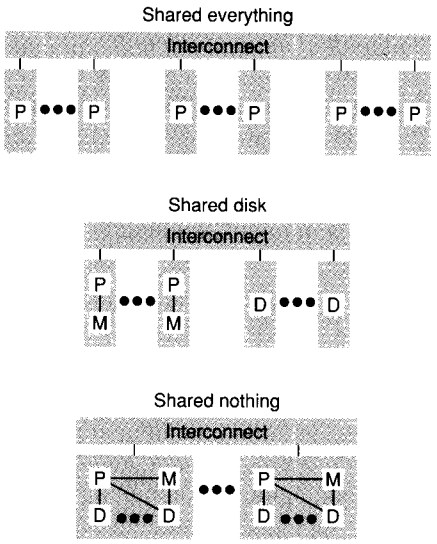


Figure 11.
Three classes of computer architectures showing sharing and interconnection of processors (P), memories (M), and disks (D).

Why NonStop SQL Gives Near-Linear Speedup and Scaleup

The results of this benchmark contradict the belief that multiprocessors do not give linear speedup or scaleup. (Designers have built multiprocessors for 30 years, and each one has had difficulty scaling past 10 or 20 processors.) The belief that multiprocessors cannot be scaled ignores the distinctions among three types of multiprocessor designs: conventional shared-memory (shared-everything), shared-disk, and the Tandem shared-nothing design (Bhide, 1988; Stonebraker, 1986). Figure 11 contrasts these three designs diagrammatically.

In a shared-everything design, all processors can access all memories and disks. The IBM 3090 system, which scales to six processors, typifies this design. A shared-everything design has inherent speedup and scaleup problems because all traffic must pass over the interconnect. The interconnect becomes a bottleneck due to contention or physical constraints (such as the speed of light) that limit the interconnect size.

To deal with the interconnect bottleneck, some designers have partitioned the memory among the processors (or groups of processors) but have continued to share the disks among all processors. The DEC VaxCluster typifies the shared-disk design. This design reduces many interconnect problems, but continues to have severe problems with disk cache interference.

A shared-nothing design completely partitions processors and disks, letting them communicate only through high-level (NonStop SQL-level or application-level) messages. This greatly reduces interconnect traffic and eliminates the cache invalidation problem. Tandem NonStop systems typify the shared-nothing design.

Typically, the DBMS must examine a great deal of data to produce one answer. A shared-nothing design moves the DBMS to the data rather than data to the DBMS. Only the answer returns to the application program. The test queries described in this article demonstrate the performance benefits of this design.

Previous research prototypes such as Gamma at the University of Wisconsin (DeWitt et al., 1986) and Bubba at MCC (Smith et al., 1989) have demonstrated near-linear speedup. Teradata has made similar demonstrations on special-purpose hardware (The Genesis of a Database Computer, 1984; *DBC/1012 Database Computer System Manual, Release 1.3*, 1985). Like Tandem NonStop systems, these systems are shared-nothing designs. The results obtained in this benchmark show that the ideas pioneered by these other groups can apply to a commercially available, general-purpose, shared-nothing system. Tandem believes that a shared-nothing architecture is the key to achieving near-linear speedup and scaleup.

Conclusion

The results of this benchmark, audited by Codd and Date, Incorporated (Sawyer, 1989), demonstrate that NonStop SQL Release 2 provides near-linear speedup and scaleup on the basic SQL relational operators. No skew or interference problems were observed in the benchmark. Minor startup problems did occur, but a solution to them is well understood. Tandem demonstrated speedup and scaleup on its CLX and VLX systems, using up to 16 processors. Except for log processing in the UPDATE query, no visible bottlenecks were noted.

This benchmark shows that with NonStop SQL Release 2, users can reduce query execution time on large databases or keep it constant on growing databases by adding hardware. This capability makes Tandem NonStop SQL ideally suited for batch and query workloads.

References

Bhide, A. 1988. An Analysis of Three Transaction Processing Architectures. *Proceedings of the 14th VLDB.*

Bitton, D. et al. 1983. Benchmarking Database Systems: A Systematic Approach. *Proceedings of the 9th VLDB.*

DBC/1012 Database Computer System Manual, Release 1.3. 1985. Part No. C10-0001-01. Teradata Corporation.

Dewitt, D. et al. 1986. Gamma - A High Performance Dataflow Database Machine. *Proceedings of the 12th VLDB.*

Englert, S. and Gray, J. 1990. *Generating Dense-Unique Random Numbers for Synthetic Database Loading.* Tandem Technical Report (in preparation). Tandem Computers Incorporated.

Sawyer, T. 1989. *Auditor's Report on NonStop SQL Release 2 Benchmark.* Codd and Date, Incorporated.

Smith, M. et al. 1989. An Experiment on Response Time Scalability. *Proceedings of the Sixth International Workshop on Database Machines.*

Stonebraker, M. 1986. The Case for Shared-Nothing. *Database Engineering.* Vol. 9.1.

The Genesis of a Database Computer: A Conversation with Jack Shemer and Phil Neches of Teradata Corporation. November 1984. *IEEE Computer.*

Acknowledgments

The parallel features of NonStop SQL were designed and implemented by Yung-Feng Kao (parallel partition inserts), Haleh Mahbod (SQL language), Mark Moore (parallel index maintenance), Carol Pearson (parallel sort), Mike Pong (parallel optimization), and Franco Putzolu and Amardeep Sodhi (parallel executor). The sequential read and write optimizations were designed and implemented by Andrea Borr (disk process) and Julia Lai and Harjit Sabharwal (file system). To the credit of all these people, the pre-alpha software that Tandem tested worked without problems.

The benchmark used the facilities of the Cupertino Benchmark Center. Steve Shugh was the liaison. Jeff Marturano and Phil Rose assembled the VLX and CLX systems. Ray Glasstone helped with some of the performance reports.

Tom Sawyer of Codd and Date, Incorporated, was the Auditor.

Susanne Englert has worked in Tandem's performance measurement and analysis group in Software Development since 1986. She provided performance support for the MULTILAN product and NonStop SQL Release 2. Her most recent project involved extensive analysis and testing of the DP2 mixed workload enhancement.

Jim Gray works in Software Development and has worked on the design and implementation of NonStop SQL. He has also worked on developing a word processor, system dictionary, parallel sort, and enhancements to the Encompass data management system. Jim is currently writing a book on transaction processing.

Terrye Kocher is currently a senior advisory analyst in Large Systems Marketing Support within the Tandem Customer Support Organization. Since joining LSMS in early 1988, she has been working in the areas of database and application development. Prior to joining CSO, Terrye spent six years with Tandem as a field analyst.

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

Parallelism in NonStop SQL Release 2

Release 2 of NonStop™ SQL, the Tandem™ relational database management system, introduces parallel query execution. NonStop SQL can automatically exploit the Tandem multiprocessor architecture by dividing a NonStop SQL query into smaller tasks and assigning the tasks to separate processors. This divide-and-conquer approach can improve the response time of all the basic SQL operations, including selections, insertions, updates, deletions, joins, and aggregate computations.

Databases containing many gigabytes of data are becoming increasingly common (Cassidy and Kocher, 1989). A business with a growing database must continue to perform batch jobs and generate reports in limited periods of time. Without parallel execution, this requirement becomes increasingly hard to satisfy. Buying multiple computers or a multiprocessor computer does not improve query performance if the database management system directs a single processor to execute the query. Users can redesign their batch applications so that separate processors execute parts of each task, but this alternative can be costly and hard to manage.

Parallel execution allows NonStop SQL to perform batch jobs, generate reports, and submit ad hoc queries on a production-scale database. By dividing a query among many processors, NonStop SQL can reduce enormously the elapsed time it takes to execute. By the same token, NonStop SQL can perform a task that has grown enormously (because the database has grown) in the same amount of time it performed the original task.

NonStop SQL Release 2 also uses parallel execution to maintain indexes. As a database increases in size, users must maintain multiple indexes to achieve acceptable OLTP and ad hoc query performance. In most database management systems, increasing the number of indexes also increases the cost of maintaining the indexes, and OLTP response time suffers.

NonStop SQL solves this dilemma by introducing parallel index maintenance. When a query causes a base table to be modified, NonStop SQL allows all the affected indexes to be modified in parallel (as long as each index is defined on a separate disk volume). Because this feature improves the response time of maintaining indexes, users can define multiple indexes on a table without adversely affecting OLTP performance.

This article describes how NonStop SQL Release 2 implements parallel query execution. The article also suggests how to configure a system to take advantage of parallel execution. Examples show how each basic NonStop SQL operation executes in parallel on a sample database. Finally, the article describes the parallel index maintenance feature in NonStop SQL. A discussion of the performance of parallel query execution appears in "Performance Benefits of Parallel Query Execution and Mixed Workload Support in NonStop SQL Release 2" in this issue of the *Tandem Systems Review* (Englert and Gray, 1990).

Overview of the Tandem Architecture

To take advantage of the parallel capabilities of NonStop SQL Release 2, users need to understand Tandem hardware architecture. A Tandem system consists of as many as 16 processors, each with its own memory. Users can connect multiple systems into a transparent network comprising over 4000 processors. (A Tandem network functions as if it were a single system.)

Typically, a disk volume consists of a mirrored pair of disk units. Each disk unit is connected to a pair of disk controllers, each of which is connected to a pair of processors. Thus, the system can access each disk unit through any of four physical paths. Each disk volume is managed by a pair of disk processes running in separate processors. However, at any given time, only one process in a pair has the primary responsibility for managing both units in a mirrored disk volume.

NonStop SQL allows a table or index to be partitioned across as many as 100 disk volumes (*NonStop SQL Conversational Interface Reference Manual*, 1989). In a partitioned table or index, the rows are physically distributed across multiple disk volumes based on a key range (key-sequenced tables only) or partition size (entry-sequenced and relative tables only). This is often called *horizontal partitioning*.

The benefits of parallelism in NonStop SQL depend largely on system configuration. It is important to balance the ratios of controller pairs to mirrored volumes and disk volumes to processors. Users should configure processors, controllers, and disks to allow simultaneous access to the table partitions and indexes needed by critical applications. Users can maximize parallelism by dedicating a controller pair per mirrored disk volume and by primaring at most one disk volume per processor. However, parallel query processing can still be very beneficial even if such an ideal configuration is not available.

Figure 1.
A disk configuration that allows maximum parallelism.

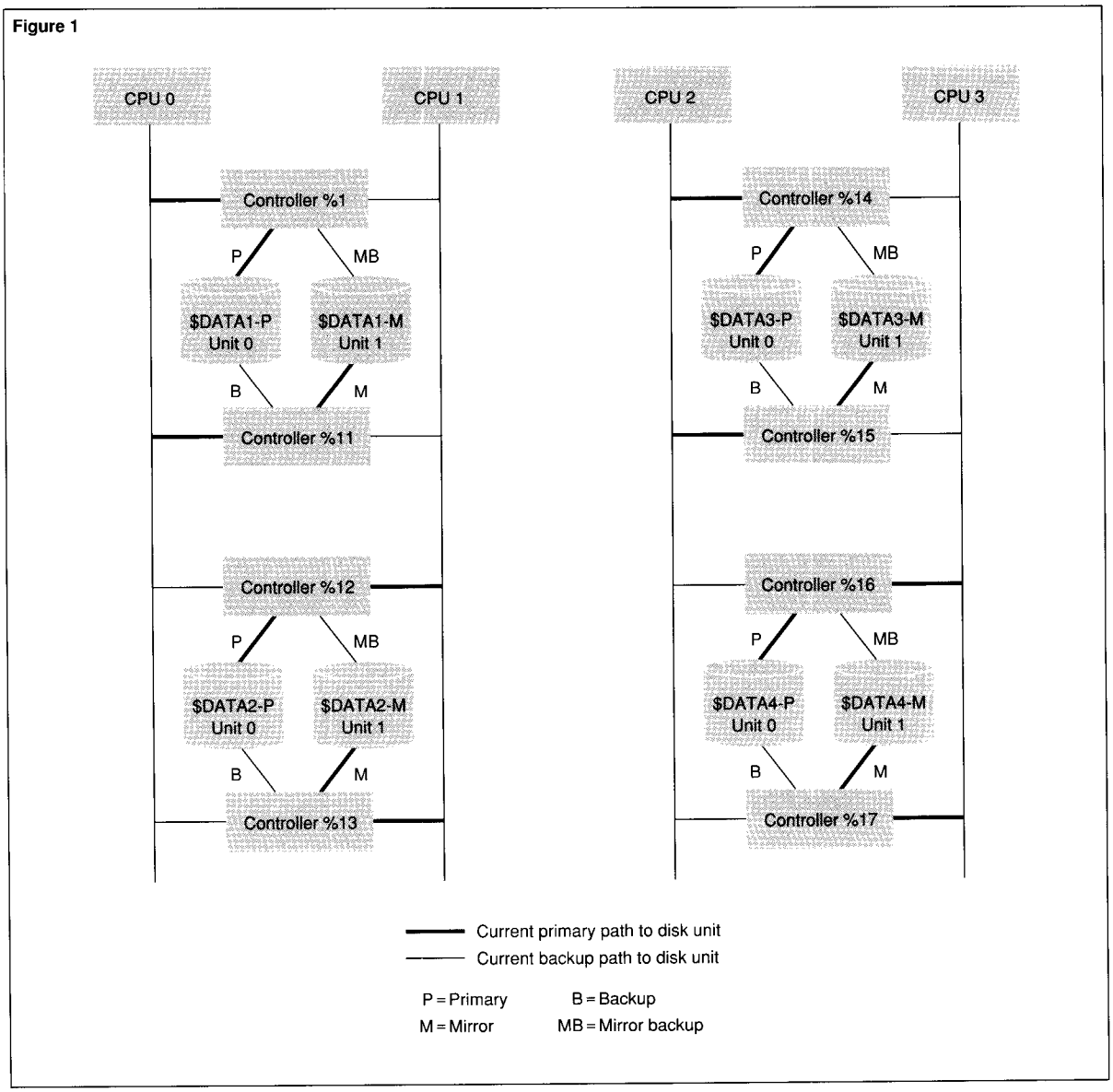


Figure 1 illustrates a configuration consisting of four processors, mirrored disk volumes, and controller pairs. Each disk volume is connected to a separate pair of controllers and controlled by a separate processor. The mirrored volumes are also configured so that the Tandem Disk Process 2 (DP2) can perform parallel reads and writes. The examples in this article use the configuration shown in Figure 1.

If users must reduce the cost of a configuration, they can use a weaker configuration at the expense of maximum parallelism or fault tolerance. For example, users can configure two mirrored volumes per controller pair (Sitler, 1986). This configuration takes advantage of parallelism because it avoids contention among processors and disk controllers. However, it has less fault tolerance than the configuration shown in Figure 1, and it does not allow DP2 to perform parallel reads and writes.

NonStop SQL comprises a conversational interface (SQLCI), SQL compiler and optimizer, SQL executor, SQL file system, catalog manager, and an extensive set of SQL utilities (Cohen, 1988). The SQL executor and file system are system library procedures; the other components are separate processes. Figure 2 illustrates the major components involved in the nonparallel execution of an ad hoc SQL query against a partitioned table.

NonStop SQL Release 2 Executor Architecture

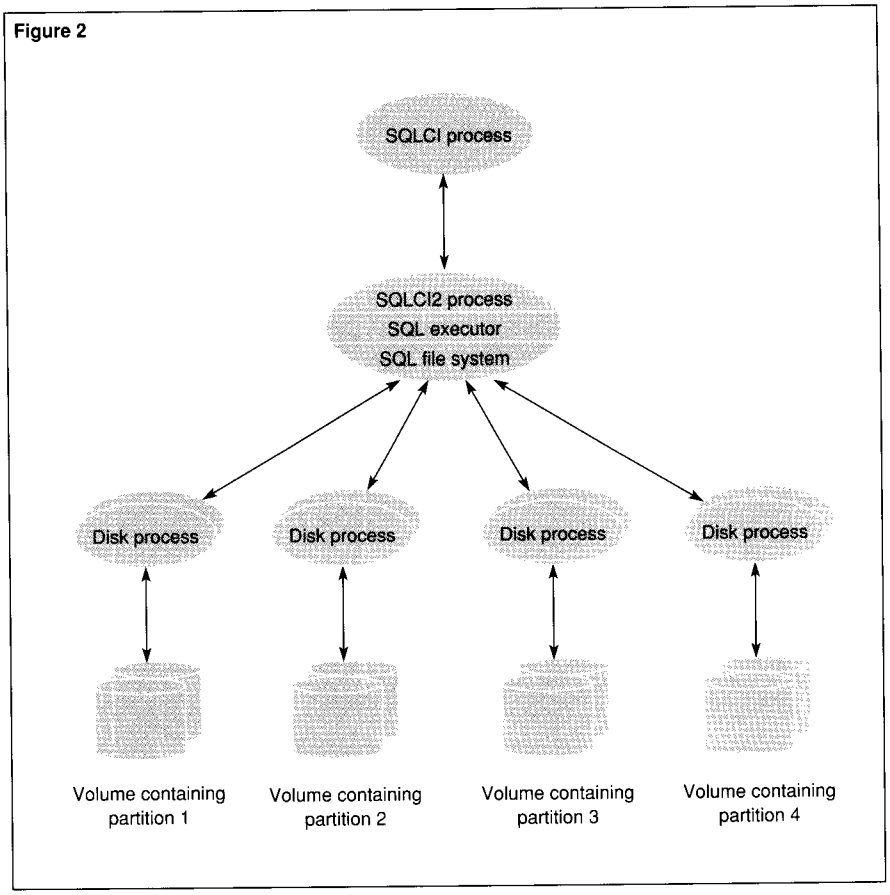
Release 2 of NonStop SQL introduces a new process, called an *executor server process* (ESP), that uses parallel processing in executing SQL statements. A *master executor* invokes several ESPs in parallel to process a statement or part of a statement. The master executor is the same as the NonStop SQL Release 1 executor, but it also starts, manages, and communicates with the ESPs.

In Release 1, the NonStop SQL optimizer determined (at compile time) the best access plan for processing a statement. In NonStop SQL Release 2, the optimizer also considers whether the system can process the entire statement or parts of the statement in parallel. The optimizer selects the parallel execution plan only if it is the best plan.

Partitioning a table and its indexes increases the likelihood that the optimizer will choose a parallel execution plan. The partitions may reside on one system or on many systems in a network.

If the optimizer determines that a statement would benefit from parallel execution, the master executor will assign one ESP process to each partition that must be accessed (as defined by the access plan). At run time, the master executor starts an ESP process in the current primary processor of each partition's disk volume (unless an existing ESP process can be used). Each ESP works only on the partition to which it is assigned.

Figure 2



The master executor simultaneously employs a number of ESPs to work in parallel on the chosen part of the statement. If there are n ESPs, the portion of the task processed by each ESP is $1/n$ and the ESPs can finish the task in $1/n$ of the time it would have taken a single process. For example, 10 ESPs can achieve a 90 percent time reduction. This performance benefit is called *speedup*. If a task increases by a factor of n, increasing the number of ESPs by the same factor allows NonStop SQL to process the task in the same elapsed time. This performance benefit is called *scaleup*. Speedup and scaleup are described in the article by Englert and Gray in this issue of the *Tandem Systems Review*.

Figure 2.
Major NonStop SQL components involved in the execution of a nonparallel ad hoc SQL query. Each volume is accessed serially during the execution of the query.

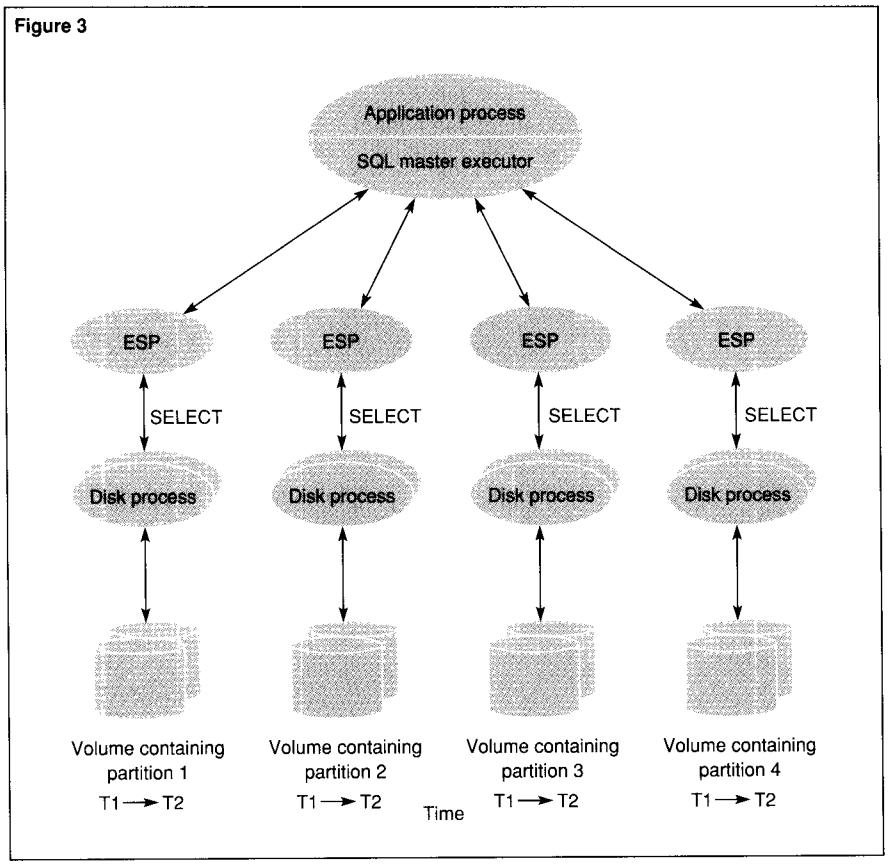


Figure 3.
Release 2 parallel executors architecture. Master executor and executor server processes (ESPs) perform a SELECT statement on a partitioned table (where all partitions are of equal size).

The master executor assigns the work to ESPs. The ESPs perform the work and return to the master executor either data or status information. The master executor processes that information and returns the end results to the user.

Figure 3 shows how the master executor assigns four ESPs to perform a SELECT statement on a table with four partitions. Each ESP selects data from one partition and returns it to the master executor.

An ESP returns status information to the master executor when it processes a DELETE statement. A separate ESP deletes data in each partition of the table. Depending on the outcome of the operation, it returns either success or error status to the master executor. If all ESPs return success status, the DELETE operation succeeds. If one or more ESPs return an error status, the master executor reports the error to the user and the DELETE operation fails.

ESPs are reusable. After they finish a task, the master executor can assign them to process other SQL statements or other parts of the same statement. Reusing an ESP saves the overhead of creating a new one each time one is needed.

Only the master executor that starts an ESP can use it. The master executor keeps as many as two ESPs in each CPU in the local system (the system in which the master executor is running). The master executor also maintains information about the use and status of the ESPs.

Repartitioning Data

Users can make the most efficient use of parallel query execution by partitioning their database tables (and indexes) across disk volumes. The best database configuration takes advantage of the system configuration illustrated in Figure 1. Each partition resides on a separate disk volume, and a separate processor has primary responsibility for each partition. When one ESP is assigned to each table partition, the ESPs can execute in parallel. The total time needed to finish a parallel task is the time needed to work on the largest partition.

Some tables may not be partitioned or may be partitioned in a way that does not facilitate parallel processing. In these cases, the optimizer can ask the NonStop SQL executor to *repartition* (reorganize) a copy of the data at run time. During repartitioning, NonStop SQL distributes the data over a set of temporary partitions. Each partition contains data that can be processed in parallel by a separate ESP.

The optimizer considers the cost of repartitioning and sorting the data when it selects the best access plan for the statement. If the optimizer chooses repartitioning, it asks that one temporary partition be created in each processor in the local system (the system in which the master executor will run). All the temporary partitions will reside in the local system.

At run time, the master executor starts an ESP (or uses an existing ESP) for each partition of the source table. The ESPs select tuples from the partition based on the predicates specified in the statement. NonStop SQL applies a hash function to the tuples to determine the target partition into which they will be inserted. A good hash function ensures almost equal distribution of the tuples among the partitions.

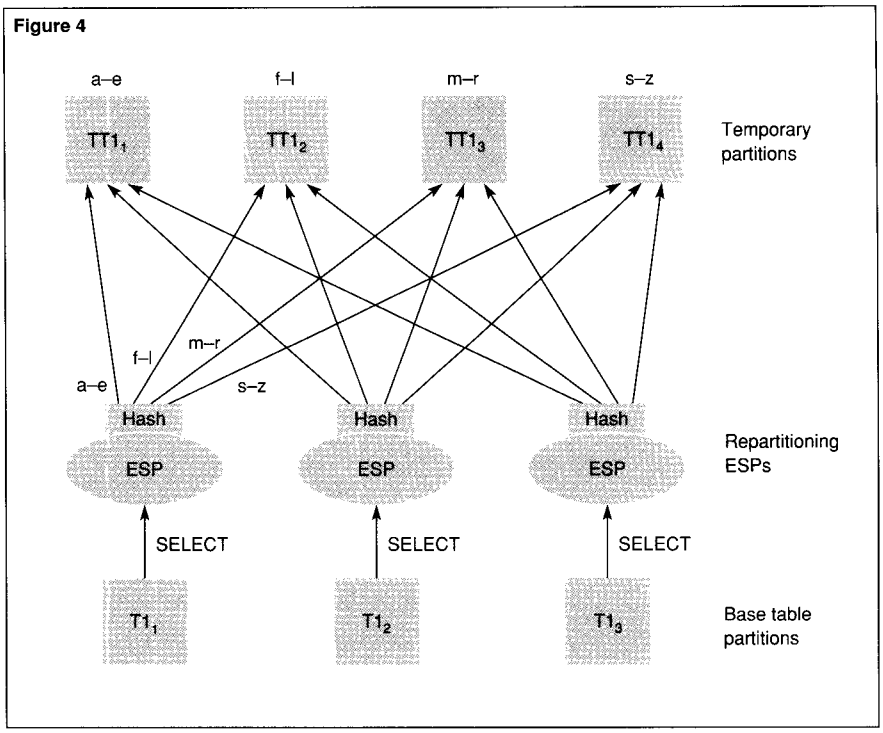
Usually, NonStop SQL needs to sort the resulting data, either to process a GROUP BY clause or to participate in a SORT MERGE join with another table. Using multiple SORTPROG processes, NonStop SQL simultaneously sorts the data in each of the temporary partitions.

NonStop SQL uses the temporary partitions for the subsequent steps in processing the statement. After the statement finishes executing, NonStop SQL keeps the partitions if the statement is embedded in a NonStop SQL application. This saves the overhead of creating them again if the statement is executed many times. However, the data is purged from the temporary partitions at the end of the statement execution. Every time the statement is executed, a fresh copy of data (as needed by that execution) is loaded into the partitions. When the application process terminates, the system automatically drops the partitions. If a user initiates the statement with dynamic NonStop SQL, the partitions are dropped after the statement finishes executing.

Consider a SELECT statement that contains a GROUP BY clause. If there is no index on the GROUP BY columns, NonStop SQL repartitions the data by placing all tuples that belong to a group into one partition. The GROUP BY columns are the criteria for repartitioning. NonStop SQL applies a hash function to these columns in each tuple to determine the partition into which that tuple is inserted.

Figure 4 illustrates the repartitioning of a table with three partitions into a temporary table with four partitions. Suppose the data is repartitioned on the basis of the first character of the column. The ESPs place column values with first characters from *a* through *e* in the first partition, *f* through *l* in the second, *m* through *r* in the third, and *s* through *z* in the fourth. (In real statements, NonStop SQL uses a more rigorous hash function than the one described here.)

Three ESPs perform the repartitioning in parallel. Each ESP selects a tuple and, based on the result of the hash function, inserts the tuple into one of the four target partitions.



NonStop SQL Release 2 Parallel Operations

To allow the NonStop SQL optimizer to choose a parallel execution plan, users must issue the CONTROL EXECUTOR PARALLEL EXECUTION ON compiler directive before compiling a query. (Users may not want parallel execution if maximizing throughput is more important than minimizing response time.) Users can choose whether or not to execute queries in parallel, but parallel index maintenance is automatic. That is, NonStop SQL always uses parallel index maintenance when multiple indexes need maintenance.

Figure 4.
Repartitioning data using a hash function.

Figure 5

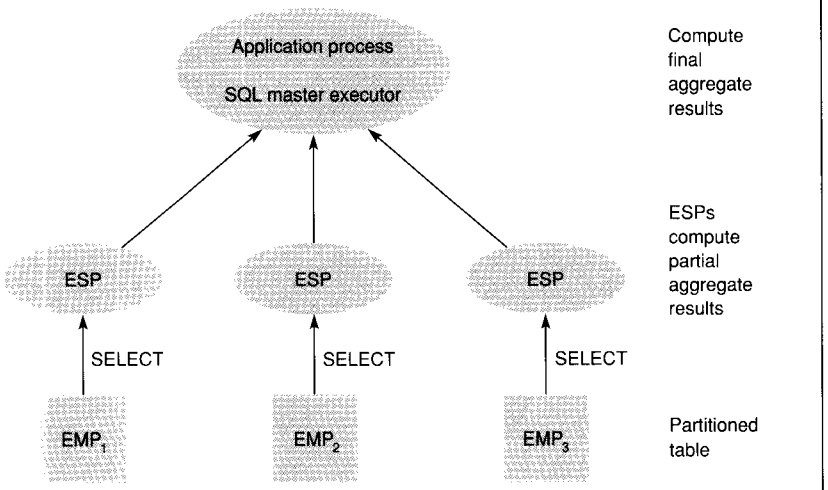


Figure 6

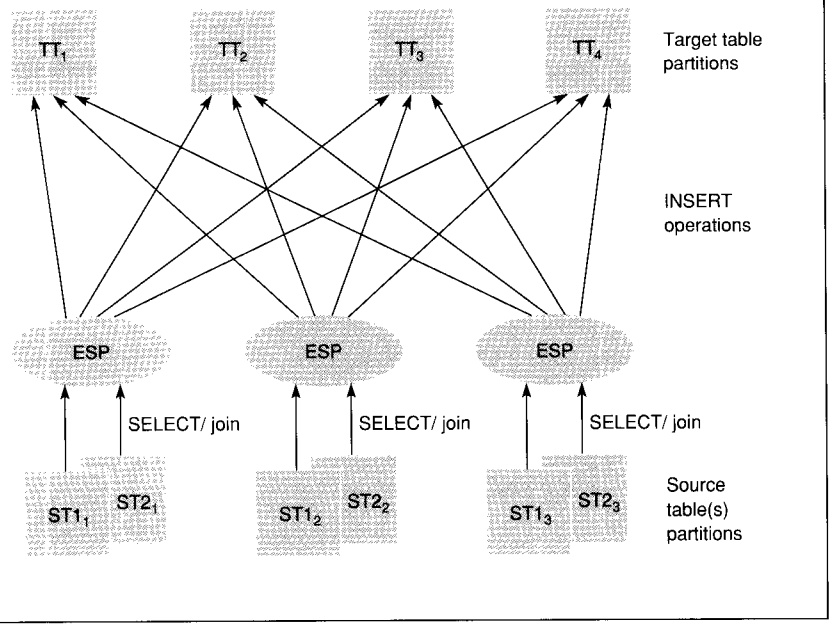


Figure 5.
Parallel aggregate evaluation.

Figure 6.
Parallel INSERT INTO ... SELECT FROM operation.

Parallel Aggregate Evaluation

Summary reports frequently use aggregate functions such as COUNT, SUM, and AVG. Often a query asks the database how many tuples a table contains before it lists them; the COUNT aggregate can count tuples. Report queries also use the SUM and AVG functions to find, for example, the total and average salary in a department.

Parallel ESPs can perform an aggregate evaluation on a partitioned table. (The best database configuration gives a separate processor primary responsibility for each partition.) Each ESP computes intermediate results for its partition and returns them to the master executor. The ESPs also return the number (count) of tuples that contributed to the aggregate. The master executor then computes the final value of the aggregate and returns it to the user. Figure 5 shows how NonStop SQL processes the following query on a table named EMP with three partitions (EMP_1, EMP_2, and EMP_3):

```
SELECT COUNT(*), AVG(commission)
  FROM EMP;
```

For the COUNT aggregate, each ESP returns the count of tuples in its partition. The master executor totals the counts to obtain the final value of COUNT for the entire table. For the AVG aggregate, each ESP returns the SUM of the non-null values in the commission column and the COUNT of the tuples with non-null commissions. The master executor adds up the SUM and COUNT values returned by the ESPs and then computes the final value by dividing the SUM by the total COUNT values.

In aggregate evaluation, NonStop SQL examines the entire table once. Therefore, NonStop SQL would not execute an aggregate query in parallel on a non-partitioned table. NonStop SQL never repartitions the source table to process only the aggregate functions in parallel. However, if a statement contains a GROUP BY clause and aggregate functions must be computed for the groups, the source table can be repartitioned.

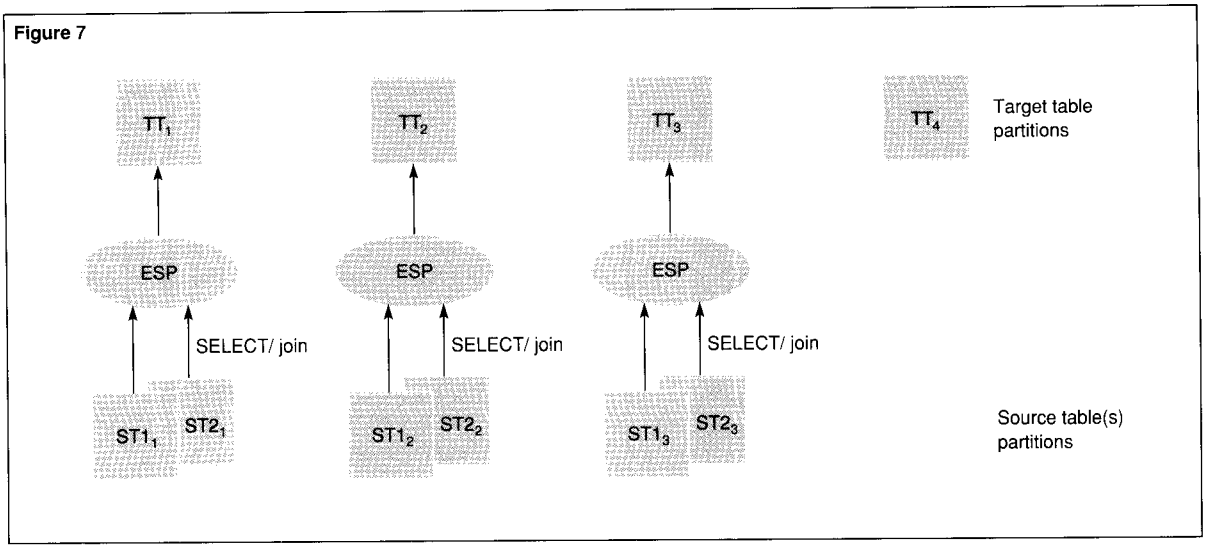


Figure 7.
Parallel INSERT INTO ... SELECT FROM operation for an entry-sequenced target table.

Parallel INSERT INTO ... SELECT FROM Statements

An INSERT INTO ... SELECT FROM statement loads a target table by selecting some or all tuples from a source table or a join of two or more tables. If the source table is partitioned, a separate ESP selects tuples from each partition. If a join of two or more tables selects the tuples, NonStop SQL uses a parallel join strategy to perform the join (whenever possible). In both cases, the ESPs insert the selected tuples directly into the target table.

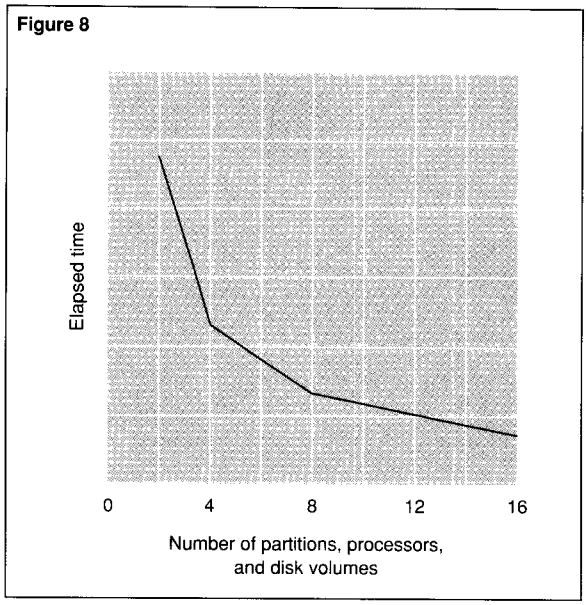
For INSERT INTO ... SELECT FROM statements, the best database configuration gives a separate processor primary responsibility for each affected partition in the source table and target table. To allow parallel execution of INSERT INTO ... SELECT FROM statements, the source table must be partitioned; the target table does not have to be partitioned.

Figure 6 shows three ESPs selecting tuples from one or more source tables and inserting them into a partitioned target table. The file system determines which partition receives each selected tuple. Because tuples selected by any ESP can belong in the same target partition, several ESPs may try to insert tuples into the same partition at the same time. In that case, the tuples are queued at the disk process of the partition, and the disk process inserts them in the order of their arrival. While a tuple waits in the disk process queue, the ESP that selected it is suspended until the tuple is inserted. This diminishes the parallel execution of the statement.

If the number of ESPs far exceeds the number of target table partitions, the disk process queues can grow large enough to hamper ESP activity and reduce parallelism. This is especially true if each ESP selects many tuples. However, if each ESP selects only a few tuples, the disk queues should remain small and the parallel ESPs will benefit the performance of the query.

The system can eliminate (or reduce) disk process queueing when the target table is a partitioned, unaudited, entry-sequenced table (SYNCDEPTH is set to 0, and SEQUENTIAL INSERT is ON). (See Figure 7.) Each ESP writes directly to one partition of an entry-sequenced table. If there are at least as many target partitions as source partitions, each ESP writes exclusively to one partition. If there are fewer target partitions than source partitions, the target partitions are assigned to ESPs in a round-robin manner. In that case, several ESPs may write to a target partition. A special protocol between the ESPs and the file system allows the ESPs to insert data into specified partitions. Also, it allows the ESPs to insert data into many partitions simultaneously, even though the first partition is not yet full. The end-of-file (EOF) for these entry-sequenced tables is the EOF of the last partition that contains data. If a partition becomes full, the file system inserts the data into a partition either succeeding or preceding the full one. The file system returns a "file is full" error only when all partitions are full.

Figure 8.
Results of an experiment using a parallel execution plan to update 1% of a 1.6 GB table.



Parallel Updates and Deletions

If an affected table (or an index defined on the table) is partitioned, NonStop SQL Release 2 can execute searched UPDATE and DELETE statements (those not associated with a cursor) in parallel, just as it can with other NonStop SQL statements. Thus, parallel execution can greatly reduce the elapsed time of a batch update or deletion.

The master executor assigns one ESP to each partition that must be accessed (as defined by the query access plan). Each ESP performs the update or deletion on its partition and returns an acknowledgement to the master executor. The parallel process shown in Figure 3 (the parallel execution of a SELECT statement on a table partitioned across four disk volumes) is essentially the same as a parallel update or deletion on a similarly partitioned table.

The NonStop SQL optimizer considers several factors in deciding that an UPDATE or DELETE statement is to be executed in parallel rather than serially. First, either the base table or an index defined on the table must be partitioned. Second, the UPDATE or DELETE must need to access multiple partitions. For example, assuming that a database and system configuration follows the one shown in Figure 1, consider the following table and index definitions:

```
CREATE TABLE $DATA1.X.T ( ..., A INT, ... );
CREATE UNIQUE INDEX $DATA2.X.I
   ON $DATA1.X.T ( A )
   PARTITION ($DATA3.X.I
                 FIRST KEY 100000,
              $DATA4.X.I
                 FIRST KEY 200000 );
```

The optimizer would not use parallelism to execute the statement DELETE FROM T because the qualifying rows the statement deletes reside in a single partition of the base table. (However, NonStop SQL would maintain the index in parallel.) In contrast, the optimizer would use parallelism to execute the statement DELETE FROM T WHERE A BETWEEN 78000 AND 142000 (depending on the execution cost) because executors would scan two partitions of INDEX X.I for qualifying rows.

Finally, the optimizer compares the costs of the serial and parallel execution plans. The optimizer chooses a parallel execution plan if the combined time needed to start up the ESPs and modify one partition is less than the combined time needed to modify all the affected partitions serially. Since minimal communication occurs between the master executor and ESPs for UPDATE and DELETE statements, communication costs are negligible.

Figure 8 shows the results of an experiment performed against a 1.6-gigabyte table using VLX™ processors. Tandem staff measured four system configurations consisting of 2, 4, 8, and 16 partitions, respectively. In each case, an UPDATE statement that affected 1 percent of the rows was issued against the table. The test results show that when an appropriate configuration is used, the parallel execution feature of NonStop SQL can greatly decrease the elapsed time required to perform batch UPDATE and DELETE statements. This experiment is described in detail elsewhere in this issue of the *Tandem Systems Review* (Englert et al., 1990).

Parallel Join Operations

Loosely speaking, a join is "a query in which data is retrieved from more than one table" (Date, 1984). Consider the following example:

```
SELECT EMPNAME, DEPTNAME
   FROM EMP, DEPT
   WHERE EMP.DEPTNO = DEPT.DEPTNO;
```

This query produces the names of employees and the departments they work in. The query selects employee information from the EMP table and department information from the DEPT table. It produces the results by matching the values of the DEPTNO column in the two tables. This column is called the *join column*. The matching condition, EMP.DEPTNO = DEPT.DEPTNO is called the *join predicate*. A join predicate with an equality comparison operator is called an *equijoin*.

Since equijoins produce results over matching values of join columns, NonStop SQL can divide the task into smaller pieces by repartitioning the tables, making sure that matching values of the join column are in corresponding partitions. By giving each ESP responsibility for one outer table partition and the corresponding inner table partition, NonStop SQL can join each pair (or set) of partitions in parallel. The total time to perform the join should equal the time it takes to perform a join for one set.

In some cases, the tables are already partitioned so that matching join column values are in corresponding partitions. These tables are *identically partitioned*; they do not have to be repartitioned.

Release 2 of NonStop SQL uses three parallel strategies to perform equijoins between two or more tables. The optimizer selects a strategy depending on the relative distribution of join column values among the tables, the availability of an index on the join column, and the relative sizes of the tables. The three strategies are based, respectively, on the following conditions:

- All tables are identically partitioned on the join columns.
- One table is partitioned; the other table is relatively small and has an index on the join columns.
- All tables either are not identically partitioned on the join columns or there is no index on the join columns.

Parallel Join Strategy 1. In the first parallel join strategy, the tables are partitioned identically on the join columns. Partitioned identically implies that the join columns are the key columns on which the partitioning is based. (Either the base table or an index can be partitioned.) Also, for all tables involved in the join, the corresponding partitions must have column values in the same range. This ensures that matching data for join operations is located only in the corresponding partitions.

For joins involving identically partitioned tables, a separate processor can have primary responsibility for each individual partition or each set of corresponding partitions. However, parallel performance declines if a single processor has primary responsibility for two partitions not involved in the same portion of the join. In that case, one processor becomes responsible for joining more than one set of partitions.

One ESP processes the join for each set of identical partitions. The ESP returns the selected (joined) tuples to the master executor, which returns them to the user. The ESPs collect a number of tuples in a buffer before returning them to the master executor. This technique limits the interprocess communication between the ESPs and the master executor. While the master executor returns the previously selected tuples to the user, the ESPs asynchronously process the join to produce the next set of tuples. For a join of tables with n partitions (in which each partition contains approximately the same number of rows), NonStop SQL spends approximately $1/n$ the time to process it in parallel as it would to process it serially.

Figure 9

Contents of Table T1 (Partitions $T1_1$, $T1_2$, $T1_3$)

$T1_1$		$T1_2$		$T1_3$	
key_col	col2	key_col	col2	key_col	col2
0	aaaa	100000	qqqq	200000	spdf
3	cccc	100003	rrrr	200004	fghi
6	cccc	120010	spdf	200010	fghd
14	spdf	121060	cccc	210555	rstu
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
99999	mous	199999	zbra	888888	graf

Contents of Table T2 (Partitions $T2_1$, $T2_2$, $T2_3$)

$T2_1$		$T2_2$		$T2_3$	
key_col	col2	key_col	col2	key_col	col2
0	aaaa	100000	qqqq	200000	spdf
4	cccc	111003	rrrr	200004	fghi
6	cccc	119999	rrrr	245678	dddd
14	graf	121060	cccc	256789	spdf
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
95566	kity	199999	zbra	999999	turk

Figure 10

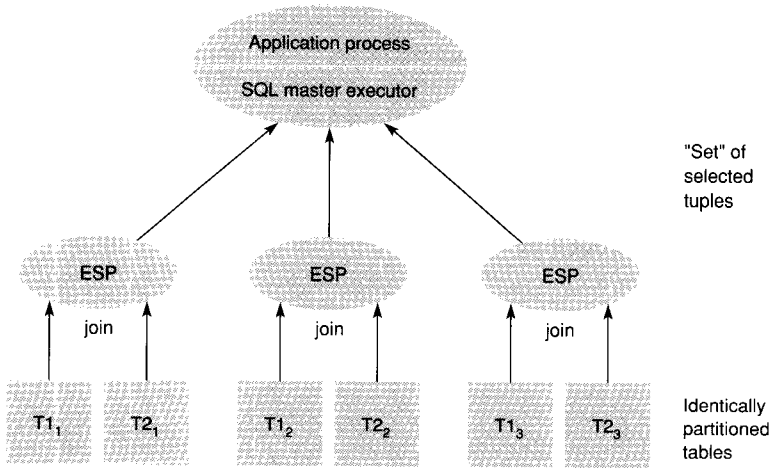


Figure 9.
Partitioned data.

Figure 10.
Parallel join strategy 1.

An exception to this asynchronous processing occurs if the join is performed within a user transaction and it requires locks. In that case, no row is returned to the user until all ESPs have replied to the master executor. Also, the next WRITEREAD operations to the ESPs do not occur until the current tuples have been returned to the user.

Figure 9 shows a two-table database partitioned for parallel join operations. Two tables, T1 and T2, have three partitions each (numbered 1 through 3). The tables are partitioned identically on the column KEY_COL. The KEY_COL values are less than 100,000 in the first partition, between 100,000 and 199,999 in the second, and greater than 199,999 in the third. Consider the following join statement:

```
SELECT T1.KEY_COL, T1.COL2, T2.COL2
   FROM T1, T2
   WHERE T1.KEY_COL = T2.KEY_COL
   FOR BROWSE ACCESS;
```

Assume that a cursor was declared for this statement and the user is issuing FETCH statements to retrieve tuples. As shown in Figure 10, the master executor employs three ESPs to process the join. One ESP joins partitions $T1_1$ and $T2_1$, a second ESP joins $T1_2$ and $T2_2$, and the third ESP joins $T1_3$ and $T2_3$.

The master executor issues NOWAIT requests to all ESPs to fetch the first set of tuples. As soon as an ESP returns the first batch of tuples, the master executor issues to that ESP another NOWAIT request to fetch the next batch of tuples. The ESP immediately starts to select the next batch of tuples. Meanwhile, the master executor returns one tuple (out of the batch last received) for each FETCH request from the user until no more tuples are left in that batch. When no ESP has any tuples to return, the FETCH returns with the "no more tuples" status.

In NonStop SQL Release 1, the join statement would always produce tuples in the same order (that of the KEY_COL). In NonStop SQL Release 2, the ESPs do not return the batches of tuples in any given order. Users should specify an ORDER BY clause on KEY_COL if they want the tuples to arrive in that order. Before the optimizer chooses a parallel join access plan, it considers the collective cost of performing the join and the ORDER BY sorting.

The optimizer is unlikely to use a parallel join access plan if the statement contains the FOR STABLE ACCESS clause. If a join statement contains the FOR STABLE ACCESS clause, each selected tuple must be locked until the ESP selects the next tuple. This limits the ESPs to returning one tuple per batch. Because the system locks only the last selected tuple in a batch, it would not provide stable access for the other tuples. For statements with the FOR STABLE ACCESS clause, the optimizer selects parallelism only if the ratio of selected tuples to the total number of tuples to be scanned is small.

Parallel Join Strategy 2. In the second parallel join strategy, one table is partitioned, the second table is much smaller than the first, and the smaller table has an index on the join columns. This join strategy uses one ESP for each partition of the large table. Each ESP performs the join between one partition of the large table and the entire small table. The small table may or may not be partitioned.

NonStop SQL can achieve maximum parallelism for this join strategy when separate processors have primary responsibility for each partition in both tables. (If the small table is not partitioned, one processor has primary responsibility for the whole table.)

Figure 11 shows three ESPs joining a large table divided into three partitions (T1) with a small table (T2) that is not partitioned. The ESPs and master executor communicate in the same way as in the first parallel join strategy.

Parallel Join Strategy 3. In the third parallel join strategy, the tables are not identically partitioned. This can happen in two cases. First, the join columns may not be part of any key. Second, if the join columns are the same as the key columns, the corresponding partitions of the tables involved in the join may not have the same key ranges.

Using the third parallel join strategy, NonStop SQL repartitions the tables on the join columns, creating an equal number of temporary partitions for each of the base tables. During the repartitioning, NonStop SQL applies the same hash function to all partitions of all tables to ensure that the same join column values fall into the respective partitions. NonStop SQL creates as many temporary partitions as there are CPUs

Figure 11

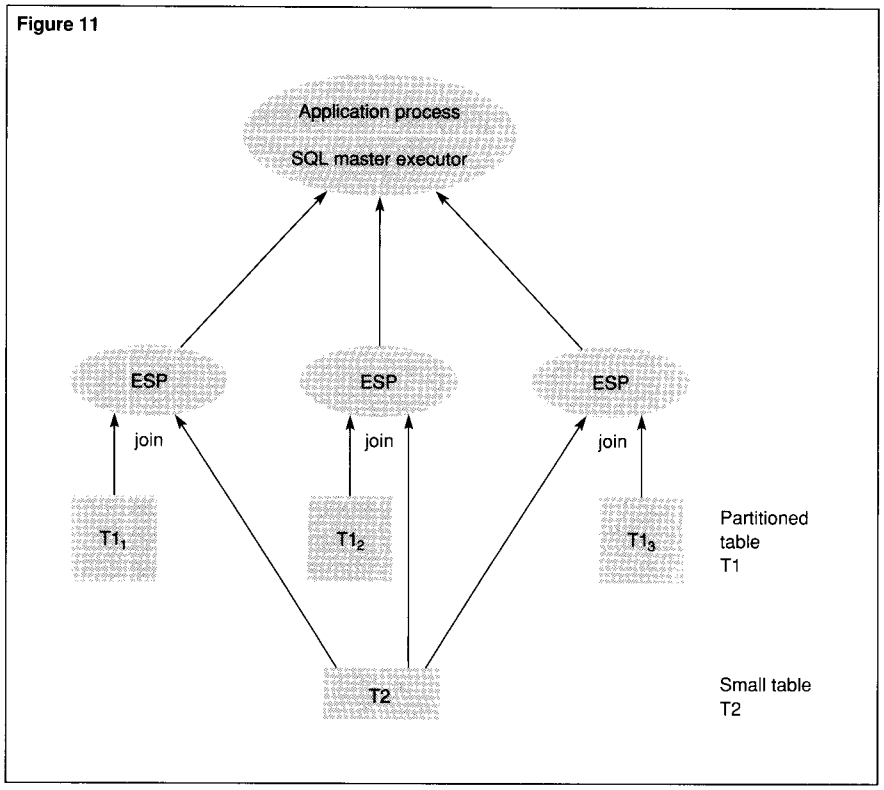


Figure 11.
Parallel join strategy 2.

available in the local system. If many CPUs are available, the temporary partitions can be relatively small; this permits greater parallelism and better response time.

Usually, the temporary partitions are entry-sequenced and NonStop SQL sorts the data to allow for SORT-MERGE join operations. ESPs process the join between the corresponding pairs of temporary partitions. Consider the following join statement:

```
SELECT T1.COL2, T1.KEY_COL,
       T2.KEY_COL
  FROM T1, T2
  WHERE T1.COL2 = T2.COL2;
```

Figure 12

Contents of Temporary Table TT1 (Partitions $TT1_1$, $TT1_2$, $TT1_3$, $TT1_4$)

$TT1_1$		$TT1_2$		$TT1_3$		$TT1_4$	
col2	key_col	col2	key_col	col2	key_col	col2	key_col
aaaa	0	fghd	200010	mous	99999	spdf	14
cccc	3	fghi	200004	qqqq	100000	spdf	120010
cccc	6	graf	888888	rrrr	100003	spdf	200000
cccc	121060	....	.....	rstu	210555	zbra	199999
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Contents of Temporary Table TT2 (Partitions $TT2_1$, $TT2_2$, $TT2_3$, $TT2_4$)

$TT2_1$		$TT2_2$		$TT2_3$		$TT2_4$	
col2	key_col	col2	key_col	col2	key_col	col2	key_col
aaaa	0	fghi	200004	qqqq	100000	spdf	200000
cccc	4	graf	14	rrrr	111003	spdf	256789
cccc	6	kity	95566	rrrr	119999	turk	999999
dddd	245678	....	.....	....	.....	zbra	199999
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Figure 13

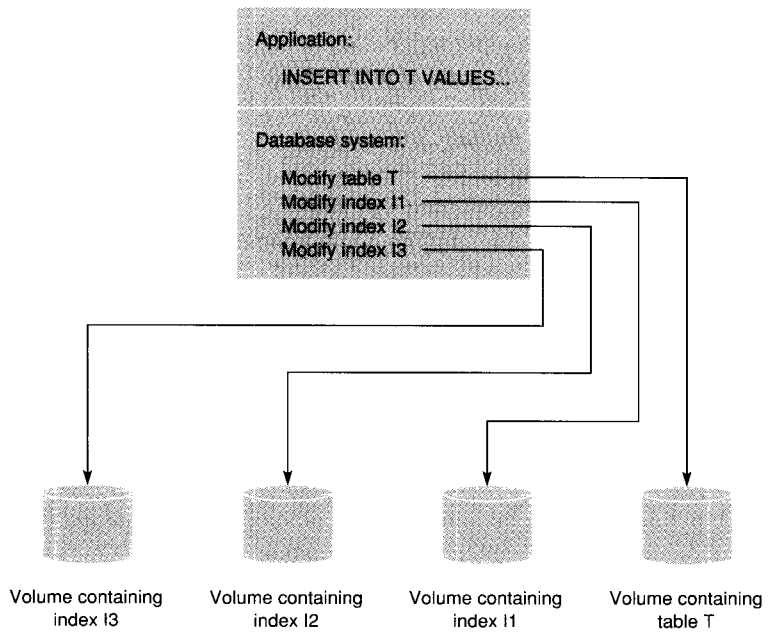


Figure 12. *Repartitioned and sorted data for parallel join strategy 3.*

Figure 13. *Modifying a base table and three indexes serially, as is done in most database systems.*

Assume that tables T1 and T2 are structured as shown in Figure 9 and that a cursor has been declared on the statement. At OPEN cursor time, NonStop SQL repartitions the base tables T1 and T2 into four temporary partitions each. (See Figure 4). This example is based on a system configuration of four processors, as shown in Figure 1.

NonStop SQL repartitions the tables serially (processing table T1 before table T2), but uses parallelism to repartition each individual table. For example, to repartition table T1, the master executor employs three ESPs to select tuples (one ESP for each original partition of table T1). The ESPs insert the tuples into one of the four temporary partitions based on the outcome of a hash function applied to the join column COL2. The ESPs select and insert the tuples in parallel. The master executor can use the same set of ESPs later to repartition table T2. If the data needs to be sorted after the repartitioning, the master executor uses four SORTPROG processes to sort the partitions in parallel.

Assume that the hash function distributes the data based on the first character of column COL2. The ESPs place all values starting with *a* through *e* in the first partition, *f* through *l* in the second, *m* through *r* in the third, and *s* through *z* in the fourth. Figure 12 shows the repartitioned and sorted data.

After the tables are repartitioned, NonStop SQL processes the join operations exactly as it does in the first parallel join strategy. As the FETCH cursor statements are issued, the master executor employs four ESPs to process the join operations on the repartitioned tables T1 and T2.

Parallel Index Maintenance

A relational database system can perform acceptably only if it has indexes. With indexes, the system often can locate all rows having a particular column value or range of values without having to scan the entire table. Even when a query requires the system to scan the entire table, the system can reduce the number of disk accesses by obtaining the data from an index that contains the desired columns (instead of from the base table). Indexes are also needed to impose a uniqueness constraint on a column or group of columns.

Unfortunately, indexes are costly to maintain, and creating too many indexes degrades OLTP performance. However, tables with too few indexes may cause batch and ad hoc query performance to suffer. To satisfy these competing constraints, database administrators must have intimate knowledge of the application load. They must choose carefully the base table columns on which they create indexes, and they must limit the total number of indexes on a table.

An index is a copy of a subset of columns in a base table. When an SQL query updates a base table column, the database system must also apply the update to all corresponding index columns. Most database systems update the base table and then update each index serially. Thus, each index adds to the response time whenever the system inserts, deletes, or updates data in the table. Figure 13 shows a database system inserting a row into a table with three indexes; the system performs the insert operations serially.

Release 2 of NonStop SQL introduces parallel index maintenance. If a change in a base table requires a corresponding change to multiple indexes, NonStop SQL changes the indexes in parallel. After completing the change in the base table, the NonStop SQL file system issues asynchronous update requests to each disk process controlling an affected index. The parallel index maintenance feature modifies multiple indexes in approximately the same elapsed time it would take to modify a single index (as long as NonStop SQL is running on a hardware configuration designed to maximize parallelism). To take advantage of parallel index maintenance, users should define a table's indexes on different disk volumes.

Figure 14 shows how NonStop SQL modifies three indexes in parallel. The illustration is based on the disk configuration shown in Figure 1. (The configuration of the disk volume in which the base table resides is not crucial for achieving best results.)

Figure 14

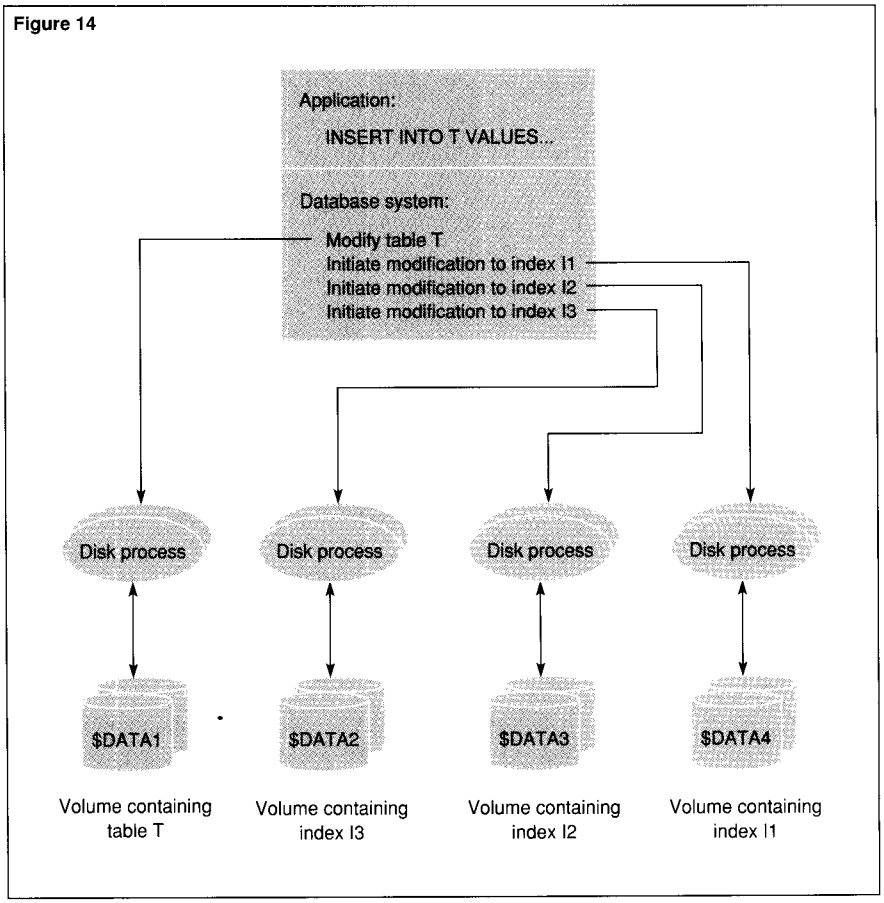


Figure 14.
Parallel index maintenance within NonStop SQL.

Figure 15.
Results of an experiment comparing the use of parallel index maintenance versus serial index maintenance when inserting or updating 10,000 rows in a table with four indexes. The percentages shown represent the reduction in the overall elapsed time for the transactions. The elapsed time required to perform index maintenance was reduced by over 60%.

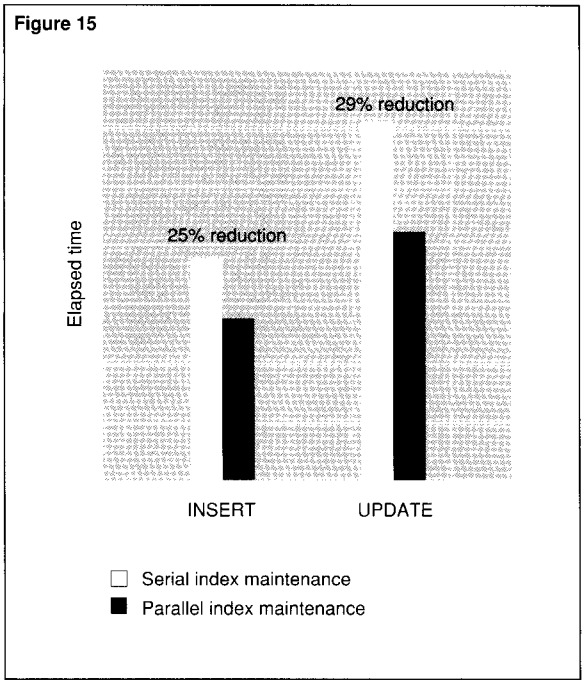


Figure 15 shows the results of an experiment comparing the serial and parallel insertion of 10,000 rows into a table with four indexes.The results demonstrate that NonStop SQL, operating on a disk configuration that promotes parallel index maintenance, significantly reduces the elapsed time of SQL operations in both batch and OLTP applications.

NonStop SQL automatically maintains indexes in parallel whenever multiple indexes are affected by an INSERT, UPDATE, or DELETE statement. The CONTROL EXECUTOR PARALLEL EXECUTION ON compiler directive does not affect the use of this feature. (NonStop SQL does not perform parallel index maintenance during a LOAD operation.)

Taking Advantage of Parallelism

If the user selects parallel execution, the NonStop SQL optimizer chooses a parallel execution plan whenever it would improve the response time of a query. The following actions can help users maximize the chance that NonStop SQL will use parallel execution.

Issue the CONTROL EXECUTOR PARALLEL EXECUTION ON compiler directive before compiling the NonStop SQL statement. This compiler directive allows the NonStop SQL optimizer to use parallelism. The optimizer weighs the performance benefits of a parallel execution plan against those of a serial plan. If the parallel plan improves performance, the optimizer automatically chooses it.

Whenever possible, use the BROWSE ACCESS or the REPEATABLE ACCESS option instead of the STABLE ACCESS option. These options sometimes allow parallel execution when the STABLE ACCESS option does not.

To encourage the use of parallelism for SELECT statements, do not perform the SELECT within a transaction unless the table is audited and locking is desired. The Transaction Monitoring Facility (TMF™) will not allow a row to be returned to the application if the master executor has any outstanding requests to ESPs.

For unaudited tables, use BROWSE ACCESS whenever possible. The optimizer will not choose the second parallel join strategy if there is a chance that ESPs will compete for the same locks.

For audited tables, use either BROWSE ACCESS, STABLE ACCESS with file locking, or REPEATABLE ACCESS whenever possible. The optimizer avoids parallel strategies (such as the second parallel join strategy) in which the ESPs may acquire excessive record locks.

When creating tables to store temporary query results, consider using partitioned, entry-sequenced tables. This table structure combines the advantages of writing entry-sequenced data with the benefits of parallelism. (The ESPs can write in parallel to multiple partitions.)

If a large partitioned table is frequently joined with a small table, partition the small table in the same way as the large table. NonStop SQL can use parallel execution more efficiently when it joins identically partitioned tables, even if one table is small enough to reside on a single disk volume.

Whenever possible, create partitioned indexes. When it executes a query, NonStop SQL can use parallelism to retrieve data from a partitioned index (if the desired columns reside in the index). This improves response time in two ways. First, NonStop SQL can retrieve data from an index more quickly than from the base table. Second, it can access many partitions in parallel more quickly than it can access data from the entire index serially.

Conclusion

The parallel execution feature introduced with Release 2 of NonStop SQL greatly improves the response time of batch operations, ad hoc queries, report generation, and OLTP applications. At the user's request, NonStop SQL can use parallel execution to perform all the basic SQL operations, including aggregate, SELECT, INSERT, UPDATE, DELETE, and join operations.

To maximize the benefits of parallel execution, users must properly configure their system's disks, controllers, and processors. The best configuration designates a separate pair of controllers and a separate processor for each disk volume. In addition, users should partition their tables across disk volumes in a way that takes maximum advantage of parallelism.

NonStop SQL Release 2 also introduces parallel index maintenance. By reducing the time it takes to maintain multiple indexes, this feature allows users to define more indexes across a wider range of columns. This simplifies database design and improves batch and ad hoc query performance without sacrificing OLTP response time.

References

Cassidy, J. and Kocher, T. 1989. NonStop SQL: The Single Database Solution. *Tandem Systems Review.* Vol. 5, No. 2. Part no. 28152. Tandem Computers Incorporated.

Cohen, H. 1988. Overview of NonStop SQL. *Tandem Systems Review.* Vol. 4, No. 2. Part no. 13693. Tandem Computers Incorporated.

Date, C.J. 1984. *An Introduction to Database Systems.* Volume 1. Addison-Wesley.

Englert, S. and Gray, J. 1990. Performance Benefits of Parallel Query Execution and Mixed Workload Support in NonStop SQL Release 2. *Tandem Systems Review.* Vol. 6, No. 2. Part no. 46987. Tandem Computers Incorporated.

Englert, S. et al. 1990. The NonStop SQL Release 2 Benchmark. *Tandem Systems Review.* Vol. 6, No. 2. Part no. 46987. Tandem Computers Incorporated.

NonStop SQL Conversational Interface Reference Manual. 1989. Part no. 22978. Tandem Computers Incorporated.

Sitler, S. 1986. Configuring Tandem Disk Subsystems. *Tandem Systems Review.* Vol. 2, No. 3. Part no. 83938. Tandem Computers Incorporated.

Acknowledgments

The authors wish to thank all the members of the NonStop SQL development group and other Tandem organizations who helped to add support for parallelism in NonStop SQL.

Mark Moore is a member of the NonStop SQL development group. He joined Tandem in 1981 as a member of the database group in Software Development. Mark began working on NonStop SQL in 1984 and helped design and implement the NonStop SQL File System.

Amardeep Sodhi is a member of the Selected Tools Vendors' (STV) program, working on the Oracle tools project. He joined Tandem in 1983 and was a member of the NonStop SQL development group from 1984 to 1989. He helped design and implement the NonStop SQL Executor and the expressions code generator of the NonStop SQL Compiler.

Online Reorganization of Key-Sequenced Tables and Files

With version C30 of the Tandem™ Guardian™ 90 operating system, users can reorganize audited, key-sequenced tables and files without having to stop or suspend access to their online applications. The online reorganization utility, part of the Tandem File Utility Program (FUP), operates on key-sequenced files created by the Tandem Enscribe record management system and key-sequenced tables created by the Tandem NonStop™ SQL distributed relational database management system.

Occasionally, tables or files need to be reorganized because online transactions and batch jobs alter the information in them, eventually causing them to become inefficient. Properly organized key-sequenced tables and files can improve the performance of online transaction processing (OLTP), batch, and query applications.

In the past, users had to weigh the benefits of reorganization against a major disadvantage: they had to interrupt the service to the application in order to reorganize a key-sequenced table or file. With a large table or file, the reorganization could make the system unavailable for hours or even days. This is not acceptable for OLTP systems in the 1990s, which must be continuously available to users.

With online reorganization, users can maintain their tables and files efficiently without disrupting OLTP applications. This article explains how tables and files become disorganized, describes the impact on system performance of disorganized tables and files, and outlines the methods available to reorganize them. It also describes how to control and tune online file reorganization. Online reorganization utility supports both NonStop SQL tables and Enscribe files; therefore, for clarity, this article uses the word *table* or *file* to identify the object being reorganized.

Key-Sequenced Table Disorganization

A key-sequenced table or file is *organized* when two conditions are met. First, the blocks containing the data records must be physically contiguous and arranged according to the primary record keys. Second, the blocks of both index and data records must be filled with data, leaving little or no wasted space. (See Figure 1.)

Over time, random insert, delete, and update operations can cause a key-sequenced file to become disorganized. There are two forms of disorganization: physical discontinuity and poor space utilization. Physical discontinuity arises when one data block follows another in logical or key value order, but the two blocks do not reside in physically adjacent locations on disk. Poor space utilization arises when many data and index blocks fail to contain the optimum number of records.[1] (See Figure 2.)

During online activity, the application may perform a random insert or update operation. The Tandem Disk Process 2 (DP2) may find that the block in which the record logically belongs has insufficient space to accommodate the added or updated record. In that case, DP2 must *split* the block by moving some records from the full block to an empty, available block. Because data blocks are chained by means of a double-linked list to allow both forward and reverse sequential processing, DP2 must write the new block and update the linked list. Random deletes can result in the deletion of the last remaining record in a block. In that case, DP2 must *collapse* the block by removing it from the chain. To complete a split or collapse operation, DP2 must modify multiple blocks, which increases the time the system takes to perform transactions.

Many systems other than those manufactured by Tandem collapse logically adjacent blocks when both blocks become less than half full. Collapsing partly full blocks can help control space utilization, but it can also impair performance in OLTP applications. For example, an OLTP application could perform a delete operations that causes a block to collapse and then perform an insert to the same logical block, causing it to split again.

[1]Schachter (1985) provides details on efficient use of buffered cache.

Figure 1

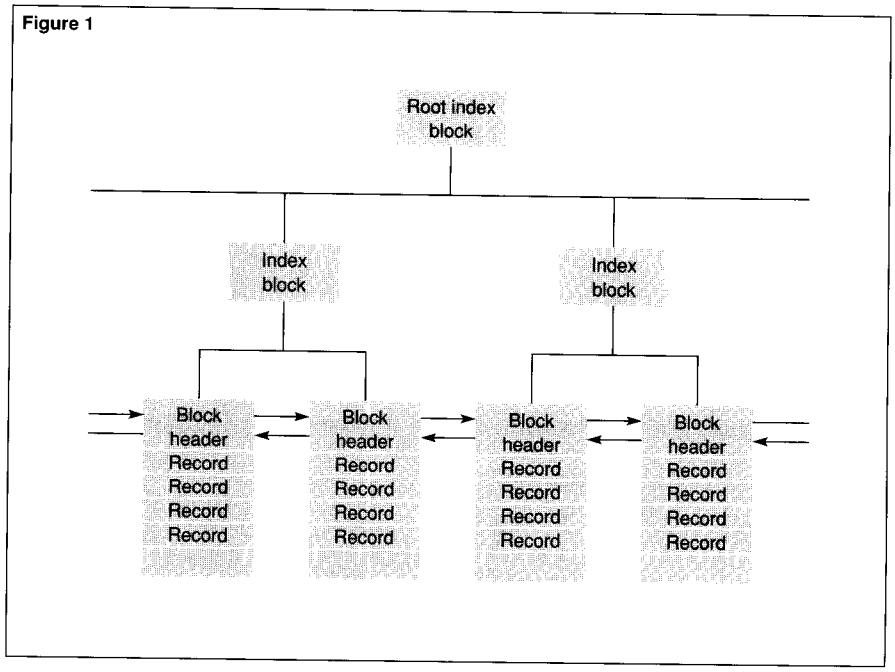


Figure 2

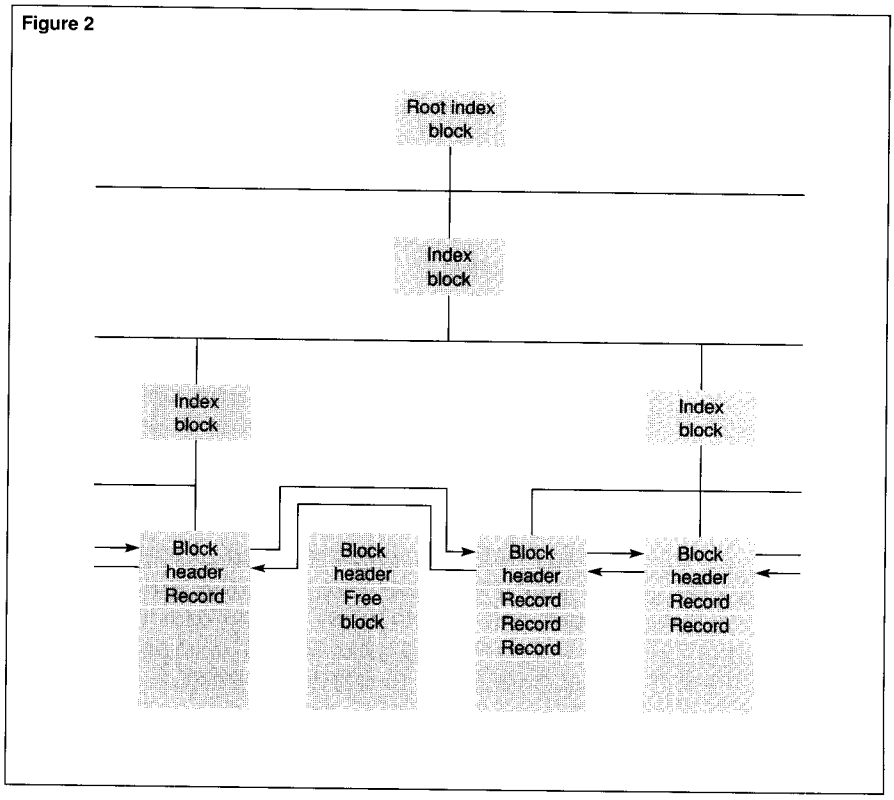


Figure 1.
An organized file. The data blocks are physically contiguous on disk. The amount of free space in the blocks is consistent. There are no empty blocks. The fewest possible index levels are used.

Figure 2.
A disorganized file. The data blocks are not physically contiguous on disk. The amount of free space in the blocks is not consistent. There are many empty blocks. The fewest possible index levels are not used.

Figure 3

Partial output of FUP INFO *filename*, STAT before RELOAD:

(a) EOF 100249600 (63.2% USED)

LEVEL	TOTAL BLOCKS	TOTAL RECS	AVG# RECS	AVG SLACK	AVG% SLACK
3	1	2	2.0	4039	99
2	2	231	115.5	1035	25
1	231	23622	102.3	1393	34
DATA	23622	519322	22.0	1263	31
FREE	618				
BITMAP	1				

Partial output of FUP INFO *filename*, STAT after RELOAD (SLACK set to 100%):

(b) EOF 68460544 (43.1% USED)

LEVEL	TOTAL BLOCKS	TOTAL RECS	AVG# RECS	AVG SLACK	AVG% SLACK
2	1	115	115.0	977	24
1	115	16597	144.3	247	6
DATA	16597	519322	31.3	77	2
FREE	0				
BITMAP	1				

Figure 3.
Partial output of the FUP INFO filename *STAT command. The significant numbers are the number of FREE blocks, average slack of DATA blocks, and average slack of INDEX blocks.*

To minimize the performance drawbacks of collapsing and splitting blocks, Tandem systems collapse blocks only when they become empty. However, when a table or file in any system becomes disorganized, the number of block splits and collapses rises, adversely affecting system performance.

Users may be able to reduce file disorganization by using sequential inserts (SETMODE 91, parameter = 3). Without the use of sequential inserts, blocks are split by placing about 50 percent of the records into the new block. With sequential inserts, DP2 leaves the block being split as full as possible and places the overflow records into the new block. (This feature is only useful if records are inserted sequentially.) Sequential inserts help to control space utilization, but do not always prevent physical discontinuity. Sequential inserts are described in detail in the September 1989 issue of the *Tandem Systems Review* (Keefauver 1989).

The Advantages of Reorganizing Tables

Most often, users reorganize tables or files to recover unused disk space. A disorganized key-sequenced table requires more disk space than an organized one because empty and partly filled blocks multiply as the table becomes physically disorganized.

An organized key-sequenced table improves the performance of sequential operations by reducing the number of I/Os and decreasing the time required to perform read operations. When logically contiguous data blocks are physically contiguous, disk access time is reduced because the disk arm moves a smaller distance between I/Os. Because each block in an organized file contains the optimum number of records, DP2 needs to scan fewer blocks to access a given amount of data. An organized file allows the NonStop SQL prefetch feature[2] to take full advantage of bulk transfers. The performance of utilities such as the Tandem Disk Compression Utility (DCOM) improves when key-sequenced files are organized, because there are fewer extents to be moved.

The performance of random access operations also improves after a table is reorganized. Reorganization reduces the number of index blocks, which improves DP2 cache utilization. It also reduces the number of index levels, which speeds up index searches.

How to Determine When a Table Is Disorganized

Two commands of the FUP INFO utility determine when a key-sequenced table or file is disorganized. The most commonly used command is FUP INFO *filename*, DETAIL. This command shows how full the table is (in percentage), based on the current End-Of-File and the maximum possible End-Of-File. When this value approaches 100 percent, the user must increase the maximum number of extents or reorganize the table to avoid an insertion failure.

The FUP INFO *filename*, STAT command shows the number of unused (free) blocks and the percentage of space used in data and index blocks. (See Figure 3.) A large number of free blocks or a low space utilization indicates that the table should be reorganized. Note that there is no current method that can be used to determine when data blocks are not contiguous.

[2]The prefetch feature is described in Borr (1988).

Reorganizing Tables with the LOAD Utilities

Before online reorganization was introduced, database managers could use two LOAD utilities to reorganize tables or files. The Tandem NonStop SQL Conversational Interface (SQLCI) LOAD utility reorganizes a NonStop SQL key-sequenced table. The FUP LOAD utility reorganizes an Enscribe key-sequenced file.

The use of these LOAD utilities has several disadvantages. First, application processes have no access to a table being reorganized until the LOAD is completed. Making a table inaccessible for an extended time is unacceptable in an OLTP environment that must be continuously available.

Second, these utilities require both a source file and a destination file. Users must copy the data to disk or tape before the LOAD can be accomplished. Loading a large, partitioned database usually requires a complex set of procedures that may take hours or days to complete.

In addition, during the LOAD, the Tandem Transaction Monitoring Facility (TMF™) does not protect files normally protected by TMF. If TMF is configured for full recovery, users must obtain a new TMF online dump after the LOAD is completed.

Reorganizing Tables Online

The C30 version of Guardian 90 offers the online RELOAD utility as an alternative to the LOAD utilities. Invoked with the FUP RELOAD command, the online RELOAD utility performs concurrently with application processing; it does not interrupt OLTP operations. By reorganizing the table in place (operating on one or two blocks of data at a time), it eliminates the need to copy the table. Because the online RELOAD utility audits all reorganization operations on TMF-protected tables and files, users do not have to perform a TMF online dump after the reorganization is completed.

Users can initiate the online RELOAD utility for any key-sequenced Enscribe file, Enscribe alternate key file, NonStop SQL base table, or NonStop SQL index table. RELOAD reorganizes only the specified file or table. For example, to reorganize both an Enscribe primary key file and its alternate key file, users must invoke RELOAD twice, once each for the primary and the alternate key files.

Comparing the RELOAD and LOAD Utilities

Both the LOAD utilities and the online RELOAD utility produce an organized key-sequenced table. Both types of utilities fill the table's data and index blocks based on a SLACK parameter specified by the user. (SLACK determines the amount of unused space left in each block after the reorganization.)

Both types of utilities organize the data blocks to be contiguous based on the primary key. However, the online RELOAD utility places all the data blocks together, followed by all the index blocks, whereas the LOAD utilities intersperse the index blocks among the data blocks. After the reorganization is completed, both types of utilities reset the End-Of-File to the first unused block and deallocate any unused extents.

Figure 4.
Online RELOAD timing.

Figure 4

File size	100249600	bytes		
Slack	100%	(default)		
Key length	226	bytes		
Record length	326	bytes		
Block length	4096	bytes		
Time required to complete RELOAD				
Rate:	10%	20%	50%	100%
RELOAD Time (hours)	17.1	9.5	4.4	2.2

Reorganizing Partitioned Tables

When the user issues a FUP RELOAD command for the primary partition of a table or file, the secondary partitions are not reorganized. The user must issue a separate FUP RELOAD command for each Enscribe or NonStop SQL partition. To initiate a RELOAD for an Enscribe secondary partition, the user must include the PARTOF parameter followed by the volume name of the primary partition. NonStop SQL partitions do not require the PARTOF parameter. Users can reload NonStop SQL or Enscribe partitions independently or concurrently.

Controlling the RELOAD in an Online Environment

Users can append several parameters to the FUP RELOAD command, including PARTOF, NEW, RATE, SLACK, DSLACK, and ISLACK. These parameters (together with other FUP commands that can be initiated while the reorganization is in progress) allow users to control the pace of the reorganization so that it does not interfere with OLTP and batch activity. By determining the amount of unused space the reorganized table will have, the parameters help users to adapt the table to various system loads and types of activity.

The RATE parameter, which controls the pace of the reorganization, allows the user to minimize the impact the RELOAD will have on OLTP applications. If the user specifies a RATE of 100 percent, the reorganization proceeds at full speed. If the user specifies a lower RATE, the reorganization rate is reduced via delays between each RELOAD operation.

Figure 4 shows the results of an online RELOAD. These results were obtained on a VLX system with XL80 disks with parallel writes on mirrored disk volumes. Actual results will depend upon system load factors, TMF configuration, and the hardware platform.

The SLACK parameters control the amount of unused space each block will have after the reorganization is completed. When setting values for these parameters, the user should consider the performance expectations for the file. The user can improve the performance of random inserts by increasing the DSLACK, which controls the slack in data blocks. A large DSLACK leaves space available to insert new records without having to split the block. The user can improve the performance of sequential operations by decreasing the DSLACK, which reduces the number of data blocks. To improve the performance of random queries, the user can decrease the ISLACK, which controls the slack in index blocks. Decreasing the ISLACK reduces the number of index levels in the file.

To obtain the current status of a reorganization, the user can issue a FUP STATUS *filename* command while the reorganization is in progress. (Control returns to the user after a RELOAD operation is initiated.) If the RELOAD has not finished, the status contains the time the RELOAD was initiated, the RATE and SLACK values, and the estimated completion percentage. If the RELOAD has been completed, the status also contains the completion time.

During periods of peak system activity, the user can suspend a RELOAD in progress by issuing a FUP SUSPEND *filename* command. The user can reinitiate the reorganization with the FUP RELOAD command. If the user does not specify values for RATE or SLACK, the RELOAD utility uses the values in effect at the time the RELOAD was suspended. If the user does specify a new RATE or SLACK, the suspended RELOAD will resume with the new values.

While the RELOAD is suspended, the part of the table that was reorganized can become disorganized again. When the user reissues the FUP RELOAD command, reorganization continues at the block where the suspended RELOAD stopped processing, leaving the first part of the table disorganized. To solve this problem, the user can use the NEW parameter in the FUP RELOAD command string, which causes the RELOAD to start again at the beginning of the table.

TMF Audit Considerations

The online RELOAD utility is implemented for audited key-sequenced tables and files only. The online RELOAD utility does not operate on key-sequenced tables or files that use index or data key compression. The RELOAD utility can generate a large amount of audit in a short time. A single RELOAD running at full speed can produce more than 90 megabytes of audit data per hour. If the TMF audit trail configuration cannot handle this amount of audit, it may cause a MAXFILES condition (suspension of transaction start). If this may be a problem, the user should set the RATE parameter to reduce the rate at which the audit trail is generated.

The Online RELOAD Server Process

The FUP RELOAD command initiates an online RELOAD server process (ORSERV), which processes the online reorganization. The ORSERV process initiates the RELOAD operation and controls the rate at which the RELOAD is accomplished. A separate ORSERV process manages each table reorganization in progress. The ORSERV process is also responsible for obtaining the status for a FUP STATUS command and suspending a RELOAD for a FUP SUSPEND command.

To facilitate the suspension of a RELOAD, ORSERV creates a key-sequenced status file, ZZRELOAD.ZZRELOAD. The status file contains a record for each file that is reloaded on the volume. The record contains the information necessary to restart a suspended RELOAD and obtain the status of the RELOAD. ORSERV creates one status file per volume. To purge the history of the RELOAD operations performed on a volume, the user can issue a FUP PURGE or FUP PURGEDATA command for the status file.

ORSERV Requests to DP2

The ORSERV process issues a series of online RELOAD requests to DP2 to accomplish the table reorganization. The first request tells DP2 to start processing at the beginning of the table. DP2 examines and reorganizes the first block in the table and returns its key value to ORSERV. With each subsequent request, DP2 examines and processes the next logical block until the table has been reorganized.

If the RATE is less than 100 percent, ORSERV delays between each online RELOAD request. The following formula determines the time used for the delay:

[(100 - RATE) / RATE] x *T*

where

T = the time to complete the last online RELOAD request.

Thus, if the user specifies a RATE of 50 percent, ORSERV pauses one delta before sending the next request. If the user specifies a RATE of 10 percent, ORSERV pauses 9 deltas before sending the next request. This throttling mechanism automatically adjusts the reorganization rate to changing system loads; the rate is fast under light loads and slower under heavy loads. This reduces conflict between the online reorganization and other batch jobs, thus increasing the effectiveness of the ORSERV process (Keefauver, 1989).

DP2 Support for Online RELOAD

New procedures have been added to DP2 to support online reorganization. The procedures determine the resources required to reorganize a file; they also perform the reorganization.

DP2 processes the online RELOAD in primary key order. It examines and moves each block until all the blocks are in consecutive order and filled based on the SLACK specified by the user. DP2 processes the data record blocks first, followed by the index record blocks. Each online RELOAD DP2 request performs one of four atomic operations:

- The *block swap* operation swaps the contents of two blocks. This operation moves blocks into consecutive order.
- The *block move* operation moves the contents of a block to an empty block. DP2 performs this operation instead of the block swap operation when the target block is empty.
- The *block merge* operation merges the contents of two blocks into one block. DP2 uses this operation to obtain the SLACK specified by the user.
- The *block split* operation divides the contents of a block into two blocks, preparing it for a block merge operation. DP2 uses this operation to obtain the SLACK specified by the user.

For each of these new operations, DP2 must prevent other requests from accessing the file for the duration of the operation. Users should be aware of the impact these operations have on system performance and response time. Users can set the RATE parameter to a lower value to lessen that impact.

When the compaction of the index blocks reduces the number of index levels needed in the B-Tree, DP2 executes a special operation that reduces the number of index levels. After the compaction is complete, the ORSERV process automatically returns unused disk space to the system by deallocating the unused file extents.

Conclusion

With the online reorganization utility, users of Enscribe and NonStop SQL systems can manage disk space resources and maintain high performance without reducing system availability. Users can initiate or resume the online reorganization without having to suspend OLTP applications. They can adjust the pace of a reorganization so that it does not interfere with OLTP or batch performance. Online table and file reorganization is a key component of continuous availability, a requirement for online enterprise computing in the 1990s.

References

Borr, A. 1988. Technical Paper: High-Performance SQL through Low-Level System Integration. *Tandem Systems Review.* Vol. 4, No. 2. Tandem Computers Incorporated. Part no. 13693.

Keefauver, T. 1989. Optimizing Batch Performance. *Tandem Systems Review.* Vol. 5, No. 2. Tandem Computers Incorporated. Part no. 28152.

Schachter, T. 1985. DP2 Key-sequenced Files. *Tandem Systems Review.* Vol. 1, No. 2. Tandem Computers Incorporated. Part no. 83935.

Acknowledgments

Franco Putzolu designed and implemented the online RELOAD utility and aided the author in the preparation of this article. Special thanks to Larry McGowan for his valuable assistance in the preparation of this article.

Gary S. Smith joined Tandem in 1985 with the Systems Support Group in Reston and currently works in the DP2 software development group.

The Outer Join in NonStop SQL

Release 2 of NonStop™ SQL, the Tandem™ relational database management system, offers a new SQL feature, the outer join. The outer join operation enhances the functionality of NonStop SQL in accordance with the specifications of the emerging ANSI SQL2 standard (ISO-ANSI, 1989). It is especially useful for generating exception reports.

NonStop SQL is often used to query databases for report generation. A NonStop SQL query involving a join can generate a complex report requiring data from multiple tables. However, a join can lose information because its result excludes rows that do not satisfy the join condition. These rows, the exceptions to the rule represented by the join condition, can provide useful information. An exception report includes these rows.

In NonStop SQL Release 1, users could perform a join operation. In NonStop SQL Release 2, a join operation is also called an inner join to distinguish it from an outer join. An inner join can combine rows from multiple tables, but it cannot at the same time identify rows that failed to satisfy the join condition. To generate an exception report, users had to invoke a sequence of queries together with specialized logic in embedded SQL programs.

In NonStop SQL Release 2, users can perform outer join operations by using the left join operator. An outer join combines rows from multiple tables; at the same time, it preserves rows that failed to qualify for the join. Thus, with a single outer join query, users can generate a complex exception report. Applications that use the outer join to generate exception reports realize an increase in programmer productivity because they:

- Benefit from a simpler design.
- Need less procedural logic and are therefore easier to code and maintain.
- Are easier to prototype through ad hoc SQL queries using the NonStop SQL Conversational Interface (SQLCI).

This article begins by defining the basic concepts related to the outer join. It compares the functions of inner join and outer join operations, describes the roles played by the various clauses in an outer join query, and shows how users can combine inner join and outer join operations.

The article assumes that readers are familiar with the Structured Query Language (SQL) and the concepts and terminology of relational databases. Readers should also be familiar with NonStop SQL and its Explain facility (*NonStop SQL Programming Reference Manual*, 1989a, 1989b, and 1989c).

Basic Concepts and Definitions

The outer join operation can be defined informally by explaining a few basic concepts. These concepts clarify the differences between an inner join and an outer join. They also define the three types of outer joins: full outer join, left outer join, and right outer join.

The definition of each operation is accompanied by an example. The examples use the tables shown in Figure 1.

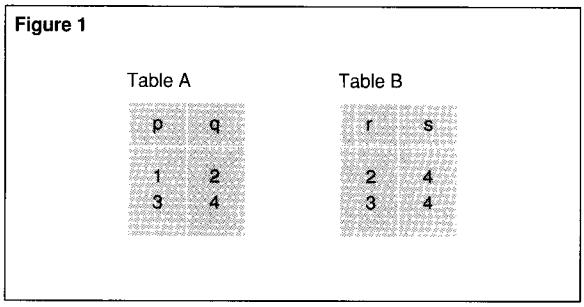


Figure 1.
Sample tables A and B.

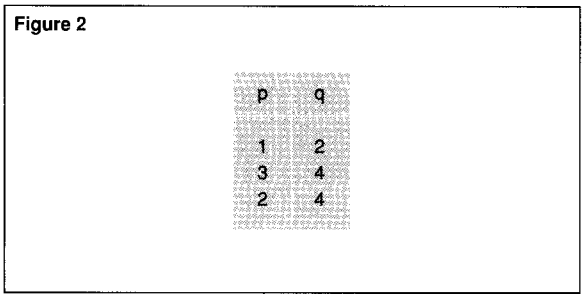


Figure 2.
Result of the union operation A U *B.*

The Null Value

In a relational database, a null value for a column represents missing information or indicates that a desired value is unknown. In this article, a null value is represented by a question mark (?).

The Union Operation

Two tables, A and B, are *union compatible* when they have the same number of columns and their columns in corresponding positions have compatible data types. A union of tables A and B is the set consisting of all rows from table A together with all rows from table B, when tables A and B are union compatible. The operation is represented by A *U* B. Its result is the table shown in Figure 2.

Figure 3.
Result of the Cartesian product A X B.

Figure 3

p	q	r	s
1	2	2	4
1	2	3	4
3	4	2	4
3	4	3	4

Figure 4.
Result of A join *B on A.q=B.r.*

Figure 4

p	q	r	s
1	2	2	4

Figure 5.
A-component and B-component of A join *B.*

Figure 5

p	q
1	2

r	s
2	4

The Join Operation

A Cartesian product of tables A and B is the set of rows formed by concatenating each row of table A with each row of table B. The operation is represented by A *X* B. Its result is the table shown in Figure 3.

A join condition is a predicate, or a combination of predicates, that evaluates to true, false, or null. The predicates may be of different types. A join predicate contains columns from two or more tables being joined. For example, A.q = B.r is a join predicate. A single-table predicate contains one or more columns from the same table. For example, A.p < 3 is a single-table predicate. A predicate that does not contain any columns is called an orphan predicate. For example, 6 > 4 is an orphan predicate.

The join of tables A and B, given a join condition C, is the set formed by selecting the rows from the Cartesian product of tables A and B that satisfy condition C. The operation is represented by A *join* B. The table shown in Figure 4 is the result of applying the join condition A.q = B.r to A *X* B.

Each row in the result of the join of tables A and B contains two components, a row derived from table A and a row derived from table B. (The two rows are concatenated, forming a single row.) All columns of a joined row that are derived from table A are together called the A-projection of A *join* B. In Figure 4, the columns p, q form the A-projection of A *join* B; the columns r, s form the B-projection of A *join* B.

The set formed by including only the component of the rows of A *join* B that contains the A-projection is called the A-component of A *join* B. Similarly, there is the B-component of A *join* B. (See Figure 5.)

The Difference Set

The difference of tables A and B, when A and B are union compatible, is the set of those rows from table A that do not occur in table B. A-B represents the difference of tables A and B. It produces the result shown in Figure 6.

The Join Complement

The join complement of table A with respect to A *join* B is a set of rows constructed in the following manner. First, take each row belonging to the difference of table A and the A-component of A *join* B. (See Figure 7.) Second, augment (extend) the row with as many null values as the number of columns in the B-projection of A *join* B. Third, add the augmented row to the join complement. Figure 8 shows the join complement of A *join* B.

Each row in the resulting set contains two components, a row derived from table A and its extension with null values. The null extension corresponds to the B-projection of A join B and indicates that there is no row in table B that can be joined with the row derived from table A. Thus, the join complement preserves information from table A when corresponding information is missing in table B. The notion of preservation applies only to those rows in table A that do not contribute to the result of A *join* B.

The rows in a join complement contain the same number of columns as the rows in the result of A *join* B. Also, because a row in a join complement is derived from the same parent tables as a row in A *join* B, columns in corresponding positions have the same data types. (The null extension introduces appropriate null values into these columns.) Therefore, given two tables to be joined, the join and the join complement produce sets of rows that are union compatible.

Figure 6

p	q
1	2

Figure 6.
Result of the difference of tables A and B (A - B).

Figure 7

p	q
3	4

Figure 7.
Result of A - (A-component of A join *B).*

Figure 8

p	q		
3	4	?	?

Figure 8.
The join complement of A join *B.*

The Full Outer Join Operation

The full outer join of tables A and B, given a join condition C, is represented by A *full join* B. It is the union of the following sets of rows:

- The join of tables A and B, represented by A *join* B.
- The join complement of table A, with respect to A *join* B.
- The join complement of table B, with respect to A *join* B.

The full outer join produces a result that contains all the rows that satisfy the join condition C. It also contains the preserved rows from table A and the preserved rows from table B. This operation is called a full outer join because it preserves rows from each of the participant tables.

In addition to the full outer join, there are two other outer join operations: the left outer join and right outer join. These joins produce results that are subsets of the result produced by a full outer join; they preserve rows from one of the tables, but not both. The operator determines which table will have its rows preserved.

The Left Outer Join Operation

The left outer join of tables A and B, given a join condition C, is represented by A *left join* B. It is the union of the following sets of rows:

- The join of tables A and B, represented by A *join* B.
- The join complement of table A with respect to A *join* B.

The left outer join preserves rows from table A. The operation is called a left outer join because the table from which information is to be preserved appears on the left side of the operator. Note that, unlike A *join* B, A *left join* B is not equal to B *left join* A. In other words, the left outer join operation is not commutative (the joining sequence of the tables cannot be changed without affecting the result). It is also not associative. (Associativity is another property that influences the sequence in which tables are joined.) Therefore, (A *left join* B) *left join* C is not equal to A *left join* (B *left join* C).

The Right Outer Join Operation

The right outer join of tables A and B, given a join condition C, is represented by A *right join* B. It is the union of the following sets of rows:

- The join of tables A and B, represented by A *join* B.
- The join complement of table B with respect to A *join* B.

The right outer join preserves rows from table B. It is called a right outer join because the table to be preserved appears on the right side of the operator. Like the left outer join, it is neither a commutative nor an associative operation.

Note that A *full join* B is equal to the union of A *left join* B and A *right join* B. Also, A *left join* B is equal to B *right join* A. More information about all three outer join operations appears in Date, 1986.

NonStop SQL Syntax

In NonStop SQL Release 2, users can formulate an outer join using new keywords in the FROM clause as shown in Figure 9. The syntax rules are described in detail in *NonStop SQL Programming Reference Manual*, 1989a, 1989b, and 1989c.

This syntax is identical to the one prescribed in the ANSI SQL2 draft standard in all respects but one (ISO-ANSI, 1989). In Release 2, NonStop SQL does not support parenthesized join operations. This syntax is implemented for the inner join and left join operations. Note that many left join operations can be performed in the FROM clause. NonStop SQL Release 2 provides a new ON clause to associate predicates explicitly with a specific left join operation. Predicates in the ON clause are used to realize the outer join.

Figure 9

```
FROM <table reference> [ { , <table reference> } ]

<table reference> ::=
    <table name> [ [AS] <correlation name> ]
    | <joined table>

<joined table> ::=
    <table reference> [ <join type> ] JOIN
    <table name> ON <join condition>

<join type> ::=
    INNER | LEFT

<join condition>      specifies a condition or a combination
                      of conditions that evaluate to
                      true, false, or unknown.

<correlation name> is an identifier.

<table name>          is an identifier.
```

Figure 9.
Join syntax in NonStop SQL Release 2.

Developing Applications Using the Outer Join

The examples in the remainder of this article show how to use the outer join in queries that can produce exception reports and other complex results. They illustrate the different results produced by inner join and outer join operators when they are placed in similar queries. It is important to demonstrate how these results are obtained because the results are not intuitively obvious. The examples differentiate between the functions of the WHERE and ON clauses. Finally, they show how inner and outer joins are combined and how the results are to be interpreted.

The examples in this article refer to the tables shown in Figure 10. The SALESEMP table contains data covering employees who are salespersons. It includes the employee number, employee name, region number, and manager's employee number. The REGION table contains the region number, region name, and a number indicating the location of the head office for each region. The ORDERS table contains, for each order, the order number, customer number, and the employee number of the salesperson who booked the order.

Figure 10

The SALESEMP table

EMP_NUM	EMP_NAME	REG_NUM	MGR_NUM
2703	Morrison, J.	7600	2705
2705	Hennessy, A.	7600	6554
2906	Nakagawa, E.	6400	6554
3598	Chu, F.	6470	2906
4096	Chow, J.	6420	3598

The REGION table

REG_NUM	REG_NAME	REG_HQLOC
6400	Japan	900
6420	Hong Kong	920
6470	Taiwan	970
7600	USA	100

The ORDERS table

ORD_NUM	CUST_NUM	BOOKED_BY
12	729	3598
33	912	11022
57	283	2705
77	1064	2906

Figure 10.
The sample database.

Figure 11.
Information about salespersons who have booked an order.

Figure 11

EMP_NUM	EMP_NAME	ORD_NUM
3598	Chu, F	12
2705	Hennessy, A.	57
2906	Nakagawa, E.	77

Figure 12.
Information about salespersons who have not booked an order.

Figure 12

EMP_NUM	EMP_NAME
2703	Morrison, J.
4096	Chow, J.

Examples of Inner Join Queries

The inner join query in the following example uses the SQL syntax available in NonStop SQL Release 1. (NonStop SQL Release 2 continues to support this syntax.) The query lists the employee number, name, and order-related information pertaining to salespersons who have booked an order.

```
SELECT S.EMP_NUM, S.EMP_NAME,
       O.ORD_NUM
  FROM SALESEMP S, ORDERS O
WHERE S.EMP_NUM = O.BOOKED_BY
```

The following query, expressed using NonStop SQL Release 2 syntax, is equivalent to the preceding query, even though the SQL syntax for the two queries looks different. NonStop SQL translates the latter form into the former one. Thus, both queries produce the same result and have the same performance characteristics.

```
SELECT S.EMP_NUM, S.EMP_NAME,
       O.ORD_NUM
  FROM SALESEMP S
       INNER JOIN ORDERS O
    ON S.EMP_NUM = O.BOOKED_BY
```

Both queries produce the 3-row table shown in Figure 11. The result contains only those rows that satisfy the join condition given in the WHERE clause in the first query and the ON clause in the second query. Order 33 does not appear in the result because it was booked by salesperson 11022, who is not an employee of the company.

Information about salespersons who have not booked any orders is missing from the result of this query. Users can obtain this information from the database by running an additional query. Consider the following query, which lists the employee number and the name of the salespersons who have not booked orders.

```
SELECT EMP_NUM, EMP_NAME
  FROM SALESEMP S
WHERE EMP_NUM NOT IN
      (SELECT BOOKED_BY
         FROM ORDERS)
```

This query computes the difference of the values occurring in the columns SALESEMP.EMP_NUM and ORDERS.BOOKED_BY. It produces the 2-row table shown in Figure 12. The subquery lists the employee numbers of those salespersons who have booked an order. The main SELECT statement lists the names of the salespersons who do not appear in the list returned by the subquery.

Since NonStop SQL Release 2 also supports the SQL UNION operator, users can simulate full, left, and right join operations. The following example simulates the left join operation by using the queries that gave the results shown in Figures 11 and 12.

```
SELECT S.EMP_NUM, S.EMP_NAME,
       O.ORD_NUM
  FROM SALESEMP S, ORDERS O
WHERE S.EMP_NUM = O.BOOKED_BY
UNION
SELECT EMP_NUM, EMP_NAME, -1
  FROM SALESEMP S
WHERE EMP_NUM NOT IN
      (SELECT BOOKED_BY
         FROM ORDERS)
```

The second SELECT statement is the same one that gave the result shown in Figure 12. Its select list is augmented with a -1 to ensure that the results of the two SELECT statements are union-compatible. The -1 corresponds to the column O.ORD_NUM in the select list of the first SELECT statement. Assuming that an order number is always greater than 0, the -1 represents an invalid order number in the union result. Thus, this augmentation of rows produces the join complement of the SALESEMP and ORDERS tables. By definition, the join of the SALESEMP and ORDERS tables in a union with their join complement is the left join of the two tables. Figure 13 shows the result of the preceding query, which is the left join of the SALESEMP and ORDERS tables. Note that this simulation of the left join operation is valid only because the tables SALESEMP and ORDERS do not contain any duplicated rows.

In Figure 13, the first and last rows have the value -1 in the column ORD_NUM. These rows represent salespersons who have not booked an order.

An Example of an Outer Join Query

A single outer join query can retrieve the information produced by the queries discussed earlier. The left outer join query in the following example preserves the employee numbers and names of the salespersons who have not booked orders. Each question mark in the result represents a null value.

```
SELECT  S.EMP_NUM, S.EMP_NAME,
        O.ORD_NUM
  FROM  SALESEMP S
        LEFT JOIN ORDERS O
    ON  S.EMP_NUM = O.BOOKED_BY
```

This outer join query produces the 5-row table shown in Figure 14. Rows 2 through 4 are the same as those shown in Figure 11. These rows represent the join of the SALESEMP and ORDERS tables. The rows having the values 2703 and 4096 in the EMP_NUM column belong to the difference set. They are preserved in Figure 14 by augmenting the order information with null values. The results shown in Figures 13 and 14 are similar; the only difference is that a null value in Figure 14 replaces the -1 in Figure 13.

Figure 13

EMP_NUM	EMP_NAME	ORD_NUM
2703	Morrison, J.	-1
2705	Hennessy, A.	57
2906	Nakagawa, E.	77
3598	Chu, F.	12
4096	Chow, J.	-1

Figure 13.
The result of a left join expressed using a union of SELECT statements.

Figure 14

EMP_NUM	EMP_NAME	ORD_NUM
2703	Morrison, J.	?
2705	Hennessy, A.	57
2906	Nakagawa, E.	77
3598	Chu, F.	12
4096	Chow, J.	?

Figure 14.
The result of an outer join query.

The Role of the ON Clause

A given join condition can be a combination of join, single-table, or orphan predicates.The ON clause allows a join condition to be associated unambiguously with a particular outer join operation in the FROM clause. Consider the following join:

```
  FROM  SALESEMP S, ORDERS O
 WHERE  S.EMP_NUM = O.BOOKED_BY
   AND  S.EMP_NUM < 2800
```

This example illustrates a join of the SALESEMP and ORDERS tables. The join condition relates rows in which a salesperson's employee number matches the employee number of the salesperson who has booked an order and the employee number is less than 2800. The join is performed only when the entire join condition is satisfied. Both predicates must evaluate to true before a row from the SALESEMP table is joined with a row from the ORDERS table.

Figure 15.
Influence of the ON-clause join condition on preserved rows.

Figure 15

EMP_NUM	EMP_NAME	ORD_NUM
2703	Morrison, J.	?
2705	Hennessy, A.	57
2906	Nakagawa, E.	?
3598	Chu, F.	?
4096	Chow, J.	?

In the following example, an outer join is performed using the same join condition as the inner join in the preceding example. In the outer join, the FROM clause appears as follows:

```
FROM  SALESEMP S
        LEFT JOIN ORDERS O
  ON  S.EMP_NUM = O.BOOKED_BY
 AND  S.EMP_NUM < 2800
```

By using the NonStop SQL Release 2 syntax, a query can combine inner and outer join operations in the same FROM clause. Therefore, it is necessary to distinguish between join conditions for the different join operations performed for the same query. This is achieved by specifying join conditions for an outer join in the ON clause and those for an inner join or between tables participating in different outer joins in the WHERE clause. The following example shows an inner join of the REGION table with the outer join of the SALESEMP and ORDERS tables.

```
FROM  REGION R, SALESEMP S
        LEFT JOIN ORDERS O
  ON  S.EMP_NUM = O.BOOKED_BY
 AND  S.EMP_NUM < 2800
WHERE  S.REG_NUM = R.REG_NUM
 AND  S.REG_NUM IN (6400, 7600)
```

The ON clause demarcates the join condition for the outer join from the join condition for the inner join (specified in the WHERE clause). If the ON clause were not available in NonStop SQL, all outer and inner join predicates would appear in the WHERE clause, as shown in the following example:

```
FROM  REGION R, SALESEMP S
        LEFT JOIN ORDERS O
WHERE  S.EMP_NUM = O.BOOKED_BY
  AND  S.EMP_NUM < 2800
  AND  S.REG_NUM = R.REG_NUM
  AND  S.REG_NUM IN (6400, 7600)
```

In this example, the relationship between join predicates and the corresponding join operation is clear. However, the association of single-table predicates becomes ambiguous. It is no longer clear whether the predicates S.EMP_NUM < 2800 and S.REG_NUM IN (...) should be associated with the join condition for the outer join or for the inner join.

The ON clause also realizes preservation of tables in outer join queries. It behaves like the WHERE clause when each row from the tables being outer-joined satisfies the join condition. These rows are added to the result, just as in the case of an inner join. However, if either row being outer-joined fails to satisfy any predicate appearing in the join condition, the row from the table to be preserved is augmented with an appropriate number of null values; then the augmented row is added to the result. In an outer join, the ON clause forms not only the join, but also the join complement.

The query in the following example emphasizes the influence that the join condition of the ON clause exercises on the rows that are preserved. Figure 15 shows the result of the following query.

```
SELECT  S.EMP_NUM, S.EMP_NAME,
          O.ORD_NUM
  FROM  SALESEMP S
          LEFT JOIN ORDERS O
    ON  S.EMP_NUM = O.BOOKED_BY
   AND  S.EMP_NUM < 2800
```

The row having the value 2703 in the EMP_NUM column is for one of the salespersons who has not booked an order. Thus, the row does not satisfy the first predicate in the join condition, S.EMP_NUM = O.BOOKED_BY. The row is preserved in Figure 15 by augmenting its order information with null values. The next row is for the salesperson who has an EMP_NUM value of 2705, has booked an order, and has an employee number less than 2800. The row contains original information from both the SALESEMP and ORDERS tables. The next two rows, having the values 2906 and 3598 in the EMP_NUM column, have employee numbers larger than 2800. Although they satisfy the first predicate, they fail to satisfy the second, S.EMP_NUM < 2800. Hence, their rows are also preserved by augmenting their order information with null values. The last row satisfies neither predicate in the join condition and is preserved in the same fashion.

The keywords LEFT JOIN and ON occur in a pair. For each LEFT JOIN that occurs in the FROM clause, there is a corresponding ON. Each pair causes the performance of an outer join operation. The table preserved in each outer join can, in turn, be inner-joined or outer-joined with another table. The query in the following example joins the SALESEMP table, which is preserved in the outer join of the SALESEMP and ORDERS tables, with the REGION table. The query lists the employee numbers and names of salespersons, along with order information and region name, for individuals in region 6400 or 7600. Figure 16 shows the result of this query.

```
SELECT S.EMP_NUM, S.EMP_NAME,
       O.ORD_NUM, R.REG_NAME
  FROM REGION R, SALESEMP S
       LEFT JOIN ORDERS O
    ON S.EMP_NUM = O.BOOKED_BY
   AND S.EMP_NUM < 2800
WHERE S.REG_NUM = R.REG_NUM
   AND S.REG_NUM IN (6400, 7600)
```

If the syntax of NonStop SQL Release 2 is used for an inner join query, the ON clause is transformed to become the WHERE clause. Therefore, the ON clause and the WHERE clause show identical behaviors for inner join queries.

Figure 16

EMP_NUM	EMP_NAME	ORD_NUM	REG_NAME
2703	Morrison, J.	?	USA
2705	Hennessy, A.	57	USA
2906	Nakagawa, E.	?	Japan

Figure 16.
The result of a join of the preserved SALESEMP table with the REGION table.

Figure 17.

EMP_NUM	EMP_NAME	ORD_NUM
2703	Morrison, J.	?
2705	Hennessy, A.	57

Figure 17.
Order status for salespersons in region 7600.

The Role of the WHERE Clause

Each ON clause for an outer join contains the predicates that are necessary to preserve the appropriate table. The WHERE clause contains the predicates that are not necessary for preservation but are used to perform inner joins or to relate columns belonging to the results of two or more different outer join operations. The WHERE clause is also used to retrieve specific information from the result of an outer join. In an outer join query, predicates from the WHERE clause that apply to a table participating in a particular outer join operation are evaluated only after that outer join operation is completed.

Figure 16 contains an example of a WHERE clause that relates a table in the FROM clause with the preserved table of an outer join operation. The following query lists the the employee numbers and names of the salespersons together with order related information for region 7600.

```
SELECT S.EMP_NUM, S.EMP_NAME,
       O.ORD_NUM
  FROM SALESEMP S
       LEFT JOIN ORDERS O
    ON S.EMP_NUM = O.BOOKED_BY
WHERE S.REG_NUM = 7600
```

Figure 17 shows the result of this query. It is a 2-row table that includes the salesperson having EMP_NUM 2703, who did not book an order, and the salesperson having EMP_NUM 2705, who did.

Figure 18

```
explain
    "SELECT S.EMP_NUM, S.EMP_NAME, O.ORD_NUM "
&"FROM SALESEMP S LEFT JOIN ORDERS O    "
&" ON S.EMP_NUM = O.BOOKED_BY    "
&"WHERE O.ORD_NUM IS NULL    "

QUERY PLAN                  : 1
STATEMENT TYPE              : SELECT

OPERATION 1.1  :  SCAN
    TABLE NAME              : \SQL.$SQL.TSRDB.SALESEMP
                              with correlation name S
    VIEW NAME               : NONE
    ACCESS TYPE             : RECORD for STABLE ACCESS
    LOCK MODE               : DEFAULT
    COLUMNS USED            : 2 out of 5 columns will be retrieved.
    TABLE SELECTIVITY       : 100% of rows will be selected.

        ACCESS PATH1        : PRIMARY
        SBB                 : VIRTUAL
        PRED. EVALUATED     : NONE
        PREDICATES APPLIED TO THE RESULT OF THE JOIN  :
                                O.ORD_NUM  IS NULL
        INDEX SELECTIVITY   : 100% of primary index will be selected.
        BEGIN KEY PRED.     : NONE
        END KEY PRED.       : NONE
    OPERATION COST          : 2

OPERATION 1.2  :  NESTED JOIN
        JOIN TYPE           : OUTER
        TABLE NAME          : \SQL.$SQL.TSRDB.ORDERS
                              with correlation name O
        VIEW NAME           : NONE
        ACCESS TYPE         : RECORD for STABLE ACCESS
        LOCK MODE           : DEFAULT
        COLUMNS USED        : 1 out of 4 columns will be retrieved.
        TABLE SELECTIVITY   : 33.33% of rows will be selected.

            ACCESS PATH1    : PRIMARY
            SBB             : VIRTUAL
            PRED. EVALUATED : by DISK PROCESS
                              S.EMP_NUM = O.BOOKED_BY
            INDEX SELECTIVITY : 100% of primary index will be selected.
            BEGIN KEY PRED. : NONE
            END KEY PRED.   : NONE
    OPERATION COST          : 2
TOTAL COST   : 4
```

Figure 18.
Explain report detailing evaluation of the WHERE clause for a LEFT JOIN query.

Figure 18 shows the NonStop SQL optimizer's Explain output for an outer join query. The Explain report lists the steps that NonStop SQL will follow to evaluate the query. It is useful for checking that a query has correct semantics, especially for the outer join operation, which is inherently complex. The plans that appear in Figures 18 and 19 demonstrate the differences in the evaluation of the predicates in the ON and WHERE clauses.

The left join query that generates the Explain report shown in Figure 18 is reformulated as an inner join query in Figure 19. The evaluation of the predicate O.ORD_NUM IS NULL differs in the two cases. In Figure 18, the Explain report states that the predicate is applied to the result of the join. This means that the predicate is evaluated after the preservation has been achieved through augmentation with null values.

By contrast, in Figure 19 the Explain report states that the same predicate is to be evaluated by the disk process. This means that the predicate is evaluated before the join phase, while NonStop SQL is retrieving rows from the ORDERS table. This is a more efficient approach, but an outer join query cannot use it. The outer join is a different type of operation than an inner join. In an outer join, predicates from the WHERE clause must be evaluated only after the operation is completed.

The Role of the GROUP BY and HAVING Clauses

NonStop SQL performs both the GROUP BY and HAVING operations after the join phase is completed. Hence they are independent of the different types of joins. There is no special treatment of the GROUP BY and HAVING clauses in outer joins.

Implications for Other SQL DML Commands

A FETCH statement on a cursor employing an outer join query must be coded to handle null values in the result. Otherwise the application might get an error at run time. Assume, for example, that the column ORD_NUM is declared to be NOT NULL in the definition of the table ORDERS. Now consider the following cursor, which is declared using an outer join query.

```
DECLARE CURSOR ORDBOOK AS
  SELECT S.EMP_NUM, O.ORD_NUM
    FROM SALESEMP S
         LEFT JOIN ORDERS O
      ON S.EMP_NUM = O.BOOKED_BY
```

Because the ORDERS table appears on the right side of the left join operator, the result table can contain null values in the column position corresponding to ORD_NUM. When preservation of SALESEMP occurs by augmenting its rows with null values, the left join operator disregards the NOT NULL attribute of a column. Hence, when a FETCH is performed on the cursor ORDBOOK, it is advisable to associate a null indicator variable with the host variable allocated to retrieve data from ORD_NUM. More information about null indicator values appears in *NonStop SQL Programming Reference Manual*, 1989a, 1989b, and 1989c. In the following example, INDVAR is an indicator variable for ORD_NUM.

```
FETCH ORDBOOK INTO :EMPNUM,
:ORDNUM :INDVAR
```

Whenever NonStop SQL returns a null value for ORDNUM, it sets INDVAR. The program must check INDVAR before attempting to use the value returned in ORDNUM. The same considerations apply for a SELECT ... INTO statement.

An INSERT INTO ... SELECT FROM statement utilizing an outer join query may also get an error at run time if columns in the table in which rows are to be inserted are declared as NOT NULL when they correspond to columns derived from the right table of an outer join.

Figure 19

```
explain
    "SELECT S.EMP_NUM, S.EMP_NAME, O.ORD_NUM "
&"FROM SALESEMP S INNER JOIN ORDERS O     "
&" ON S.EMP_NUM = O.BOOKED_BY     "
&"WHERE O.ORD_NUM IS NULL      "

QUERY PLAN : 1
STATEMENT TYPE  :  SELECT

OPERATION 1.1  :  SCAN
    TABLE NAME              : \SQL.$SQL.TSRDB.ORDERS
    VIEW NAME               : NONE
    ACCESS TYPE             : RECORD for STABLE ACCESS
    LOCK MODE               : DEFAULT
    COLUMNS USED            : 1 out of 4 columns will be retrieved.
    TABLE SELECTIVITY       : 33.33% of rows will be selected.

        ACCESS PATH1        : PRIMARY
        SBB                 : VIRTUAL
        PRED. EVALUATED     : by DISK PROCESS
                                O.ORD_NUM  IS NULL
        INDEX SELECTIVITY   : 100% of primary index will be selected.
        BEGIN KEY PRED.     : NONE
        END KEY PRED.       : NONE
    OPERATION COST          : 2

OPERATION 1.2  :  NESTED JOIN
        JOIN TYPE           : INNER
        TABLE NAME          : \SQL.$SQL.TSRDB.SALESEMP
        VIEW NAME           : NONE
        ACCESS TYPE         : RECORD for STABLE ACCESS
        LOCK MODE           : DEFAULT
        COLUMNS USED        : 2 out of 5 columns will be retrieved.
        TABLE SELECTIVITY   : 33.33% of rows will be selected.

            ACCESS PATH1    : PRIMARY
            SBB             : VIRTUAL
            PRED. EVALUATED : by DISK PROCESS
                              S.EMP_NUM = O.BOOKED_BY
            INDEX SELECTIVITY : 100% of primary index will be selected.
            BEGIN KEY PRED. : NONE
            END KEY PRED.   : NONE
    OPERATION COST          : 2
TOTAL COST   : 3
```

Figure 19.
Explain report detailing evaluation of the WHERE clause for an INNER JOIN query.

Figure 20.
The result of the SALESEMP LEFT JOIN ORDERS LEFT JOIN REGION query.

Figure 20

EMP_NUM	EMP_NAME	ORD_NUM	REG_NAME
2703	Morrison, J.	?	?
2705	Hennessy, A.	57	?
2906	Nakagawa, E.	77	Japan
3598	Chu, F	12	Taiwan
4096	Chow, J.	?	Hong Kong

Implications for Shorthand Views

A shorthand view based on an outer join query might return null values in those columns that are derived from the right table. Such a view can only participate in an outer join operation if it occurs on the left side of the keyword LEFT JOIN. Actually, no shorthand view involving either an inner join or an outer join can occur on the right side of the keyword LEFT JOIN. This limitation exists because NonStop SQL Release 2 does not support parenthesized outer joins.

Commutative and Associative Properties of Outer Joins

An inner join operation is commutative; the same result is obtained regardless of the sequence in which the tables are joined. For example, the two queries that follow yield the same result:

```
SELECT * FROM SALESEMP, ORDERS
  WHERE EMP_NUM = BOOKED_BY

SELECT * FROM ORDERS, SALESEMP
  WHERE EMP_NUM = BOOKED_BY
```

An outer join, on the other hand, is not commutative. The result depends on the sequence in which the tables appear in the FROM clause. To demonstrate this property, the following query lists the employee numbers, employee names, order status, and region names for salespersons belonging to all regions numbered between 6000 and 7000. The query uses two outer join operations to join three tables. The ORDER BY clause is added only for clarity.

```
SELECT  S.EMP_NUM, S.EMP_NAME,
        O.ORD_NUM, R.REG_NAME
  FROM  SALESEMP S
        LEFT JOIN ORDERS O
    ON  S.EMP_NUM = O.BOOKED_BY
        LEFT JOIN REGION R
    ON  S.REG_NUM = R.REG_NUM
   AND  R.REG_NUM BETWEEN
        6000 AND 7000
ORDER  BY S.EMP_NUM
```

According to the FROM clause of this query, NonStop SQL performs the outer join of the SALESEMP and ORDERS tables first. (See the 5-row table in Figure 20.) A null value appearing in the ORD_NUM column in any row indicates that the row does not satisfy the ON clause of the first outer join. Therefore, the rows with EMP_NUM 2703 and 4096 have a null value in the ORD_NUM column because these employees have not booked an order.

Next, NonStop SQL performs a second outer join on the result of the first outer join and the REGION table. A null value appearing in the REG_NAME column in any row indicates that the row does not satisfy the ON clause of the second outer join. The rows with EMP_NUM 2703 and 2705 have null values in the REG_NAME column because these employees both belong to region 7600, which lies outside the range 6000 to 7000 specified in the last ON clause.

The following query is the same as the preceding query except that it reverses the sequence in which the outer joins are performed. First, it forms an outer join of the REGION and SALESEMP tables. Next, it outer-joins the result of the first outer join with the ORDERS table.

```
SELECT S.EMP_NUM, S.EMP_NAME,
       O.ORD_NUM, R.REG_NAME
  FROM REGION R
       LEFT JOIN SALESEMP S
    ON S.REG_NUM = R.REG_NUM
   AND R.REG_NUM BETWEEN
       6000 AND 7000
       LEFT JOIN ORDERS O
    ON S.EMP_NUM = O.BOOKED_BY
ORDER BY S.EMP_NUM
```

Figure 21 shows the result of this query. The result does not contain any rows for the salespersons having EMP_NUMs 2703 and 2705. The first three rows are identical to the last three rows of the result in Figure 20. However, there is an additional row containing null values in the EMP_NUM, EMP_NAME, and ORD_NUM columns. In the query whose result is shown in Figure 20, the SALESEMP table was preserved because it was the leftmost table in the outer join sequence. In the previous query whose result is shown in Figure 21, the REGION table is preserved instead. Clearly, the semantics of the two queries are quite different.

Actually, because the join sequence does not affect the result of an inner join, the NonStop SQL optimizer decides the join sequence for inner joins. However, the optimizer upholds the outer join sequence prescribed in the FROM clause of an outer join query. Thus, for the last query above, NonStop SQL would begin by performing the outer join of the REGION and SALESEMP tables. Next, NonStop SQL would execute the second LEFT JOIN statement, which joins the result of the first outer join with the ORDERS table.

Note that even though the optimizer upholds the join sequence prescribed for an outer join, it still chooses the best access path for retrieving rows from each table. It also chooses the join technique in the same fashion as is done for inner joins.

Figure 21

EMP_NUM	EMP_NAME	ORD_NUM	REG_NAME
2906	Nakagawa, E.	77	Japan
3598	Chu, F.	12	Taiwan
4096	Chow, J.	?	Hong Kong
?	?	?	USA

Figure 21.
The result of the REGION LEFT JOIN SALESEMP LEFT JOIN ORDERS query.

Figure 22

EMP_NUM	EMP_NAME	MGR_NUM	EMP_NAME
2703	Morrison, J.	2705	Hennessy, A.
4096	Chow, J.	3598	Chu, F.

Figure 22.
The result of combining an inner join and an outer join.

In NonStop SQL Release 2, parenthesized outer joins are not permitted in the FROM clause. The outer join operation is not associative. Thus, (SALESEMP LEFT JOIN ORDERS) LEFT JOIN REGION will not produce the same result as SALESEMP LEFT JOIN (ORDERS LEFT JOIN REGION).

Combinations of Inner Joins and Outer Joins

Users can combine inner and outer join operations in the same query using the NonStop SQL Release 2 syntax. The following query lists the employee numbers and names of salespersons, together with the employee numbers and names of their managers, for individuals who failed to book an order. (See Figure 22.) Users can retrieve this information by means of a single SQL SELECT statement.

```
SELECT S.EMP_NUM, S.EMP_NAME,
       S.MGR_NUM, X.EMP_NAME
  FROM SALESEMP S
       LEFT JOIN ORDERS O
    ON S.EMP_NUM = O.BOOKED_BY
       ,SALESEMP X
WHERE O.ORD_NUM IS NULL
  AND S.MGR_NUM = X.EMP_NUM;
```

Figure 23.
The result of a left outer join.

Figure 23

EMP_NUM	EMP_NAME	ORD_NUM
3598	Chu, F	12
?	?	33
2705	Hennessy, A.	57
2906	Nakagawa, E.	77

Figure 24.
The result of a full outer join expressed by using left outer joins.

Figure 24

EMP_NUM	EMP_NAME	ORD_NUM
2703	Morrison, J.	?
2705	Hennessy, A.	57
2906	Nakagawa, E.	77
3598	Chu, F	12
4096	Chow, J.	?
?	?	33

In NonStop SQL Release 2, parenthesized join operations are not permitted in the FROM clause. Therefore, an expression such as (SALESEMP INNER JOIN ORDERS) LEFT JOIN REGION or SALESEMP LEFT JOIN (ORDERS INNER JOIN REGION) is considered illegal.

The Left Join: Compliance with ANSI SQL

The ANSI SQL2 draft standard recommends support for three types of outer join operations: left, right, and full (ISO-ANSI, 1989). Release 2 of NonStop SQL supports the left outer join. Although the right and full outer join operations are not yet supported, users can express certain right and full outer join queries in terms of the left outer join. Note that this is valid only if the tables do not contain any duplicate rows.

Simulating a Right Outer Join

The following query performs a left outer join of the ORDERS and SALESEMP tables, which preserves the ORDERS table. The query lists employee numbers, employee names, and order status to correlate the orders booked by salespersons. (See Figure 23.)

```
SELECT  S.EMP_NUM, S.EMP_NAME,
        O.ORD_NUM
  FROM  ORDERS O
        LEFT JOIN SALESEMP S
    ON  S.EMP_NUM = O.BOOKED_BY
```

This query accomplishes the same result as would be achieved by a right outer join of the SALESEMP and ORDERS tables. The statement FROM ORDERS O LEFT JOIN SALESEMP S in this query performs the same function as a FROM SALESEMP S RIGHT JOIN ORDERS O statement (if the right join operation were supported).

NonStop SQL does not perform this transformation because it can give rise to parenthesized join operations, which are not supported in NonStop SQL Release 2. The transformation of an operation like SALESEMP LEFT JOIN ORDERS LEFT JOIN REGION would yield (ORDERS LEFT JOIN REGION) RIGHT JOIN SALESEMP.

The question marks representing a null values indicate that order 33, booked by salesperson 11022, does not have matching information in the SALESEMP table. Apparently salesperson 11022 is not an employee of the company. (The salesperson might be a commission agent or an outside contractor.)

Simulating a Full Outer Join

A full outer join is the union of a left outer join and a right outer join. Users can specify a full outer join in terms of a union of two left outer joins, as the following query shows. This query creates a 6-row table and produces the same result as that produced by a full outer join if the full join operation were supported. (See Figure 24.) This transformation is valid only in the absence of duplicate rows in each of the participating tables.

```
SELECT S.EMP_NUM, S.EMP_NAME,
       O.ORD_NUM
  FROM SALESEMP S
       LEFT JOIN ORDERS O
    ON S.EMP_NUM = O.BOOKED_BY
UNION
SELECT S.EMP_NUM, S.EMP_NAME,
       O.ORD_NUM
  FROM ORDERS O
       LEFT JOIN SALESEMP S
    ON S.EMP_NUM = O.BOOKED_BY
```

Conclusion

The left outer join operation, introduced with NonStop SQL Release 2, provides an answer to users' needs and is a step toward greater ANSI SQL compliance. With this powerful new feature, a single SQL query can generate complex reports that require preservation of information belonging to certain tables.

References

Date, C. 1986. *Relational Databases—Selected Writings by C. J. Date*. Addison-Wesley.

ISO-ANSI. July 1989. *Database Language SQL2 and SQL3 X3H2-89-252* (working draft).

NonStop SQL Programming Reference Manual for C (release version C30L). 1989a. Tandem Computers Incorporated. Part no. 22967.

NonStop SQL Programming Reference Manual for COBOL85 (release version C30L). 1989b. Tandem Computers Incorporated. Part no. 24467.

NonStop SQL Programming Reference Manual for Pascal (release version C30L). 1989c. Tandem Computers Incorporated. Part no. 26942.

Acknowledgments

Mike Pong contributed substantially to the design of the outer join. Anoop Sharma designed and implemented the run-time support for the outer join. Haleh Mahbod implemented the SQL syntax.

Jay Vaishnav is a member of the NonStop SQL Compiler group and is working on the NonStop SQL optimizer. He joined Tandem in 1987.

Gateways to NonStop SQL

NonStop™ SQL is a high-performance relational database management system implemented on Tandem™ computer systems. In order to make NonStop SQL available to PC and workstation users, Tandem is providing gateways that connect popular SQL applications to NonStop SQL.

The SQL language is in widespread use and has become an ISO-ANSI standard (ANSI, 1989; ISO-ANSI, 1989a). SQL is supported by almost all database vendors, and several vendors are using SQL to develop a wide range of easy-to-use tools. Database vendors such as Oracle, Ingres, and Microsoft/Sybase provide widely used application programming interfaces for SQL on PCs and workstations. Demand for tools developed on these interfaces is high because they provide off-the-shelf solutions for decision support and ad hoc queries on a variety of platforms.

Although a standard for SQL exists, each vendor's implementation of SQL differs in significant ways. An application written on one interface will generally not run on another vendor's interface. Tandem, together with the other database vendors, is building SQL gateways to NonStop SQL so that Oracle, INGRES, and Microsoft/Sybase users can have access to NonStop SQL without having to change their applications. (Tandem is developing the Microsoft/Sybase SQL Server. The other gateways are being developed by their respective vendors with support from Tandem.)

The gateways to NonStop SQL provide two benefits. First, they make popular tools available to Tandem system users. Second, they allow users familiar with these tools on other platforms to take advantage of Tandem system fundamentals such as high availability, high performance, distribution, and fault tolerance.

This article discusses general design issues for SQL gateways and describes particular solutions for the gateways to NonStop SQL. (For an overview of NonStop SQL, see Cohen, 1988; for details about NonStop SQL features, see *NonStop SQL Programming Reference Manual*, 1989.) This article also discusses the standardization efforts that affect SQL gateways.

SQL Gateway Technology

A gateway connects two or more (usually) different systems by translating the information being passed between them. Ideally, the gateway can hide the system differences so that each system performs as though it were connected to a similar system. A user of the first system is said to obtain transparent access to the second system. Transparent access has the advantage that the user does not need to know how to use the second system. An application written for the first system can run unchanged and access the second system.

The gateways to NonStop SQL are based on a client-server architecture, which is used by most database vendors. In this case, an application requesting information is the client and the database system providing the information is the server. (A different form of client-server application, the Pathway transaction processing system, has existed on Tandem systems for several years.)

To retrieve information, a client application process makes a call to a server database process, often running on another machine. An SQL statement is passed in the call. The server executes the SQL statement and returns result data or status information to the client. (See Figure 1).

Several clients can communicate with a server concurrently, and a single user can be connected to several servers concurrently. The calls to the server are made by the client application through a library provided by the database vendor. Most vendors provide a precompiler to transform embedded SQL statements into library calls, while others allow applications to make the library calls directly. The message formats between clients and servers, as well as the servers themselves, are proprietary and are different for each vendor.

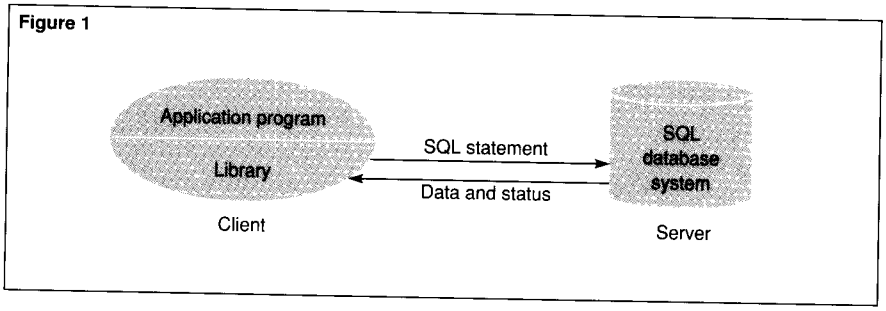


Figure 1.
Client-server architecture.

With vendors that employ a client-server architecture, a natural place for the gateway is at the client-server interface. The gateway takes the place of the database server. The client application performs as if it were communicating with its own server. Actually, it communicates with the gateway program, which is running on a different vendor's SQL system. Because the client-server interface of each database vendor is different, a separate gateway is needed to connect to each vendor's clients.

Let the term *client database system* refer to the SQL database system the client believes it is connected to. Let the term *server database system* refer to the actual SQL database system being accessed by the gateway. The terms are introduced here to distinguish the two database systems being discussed. For example, a user would place *client SQL* statements in an application, and the gateway would execute corresponding *server SQL* statements.

Figure 2.
Gateways to NonStop SQL.

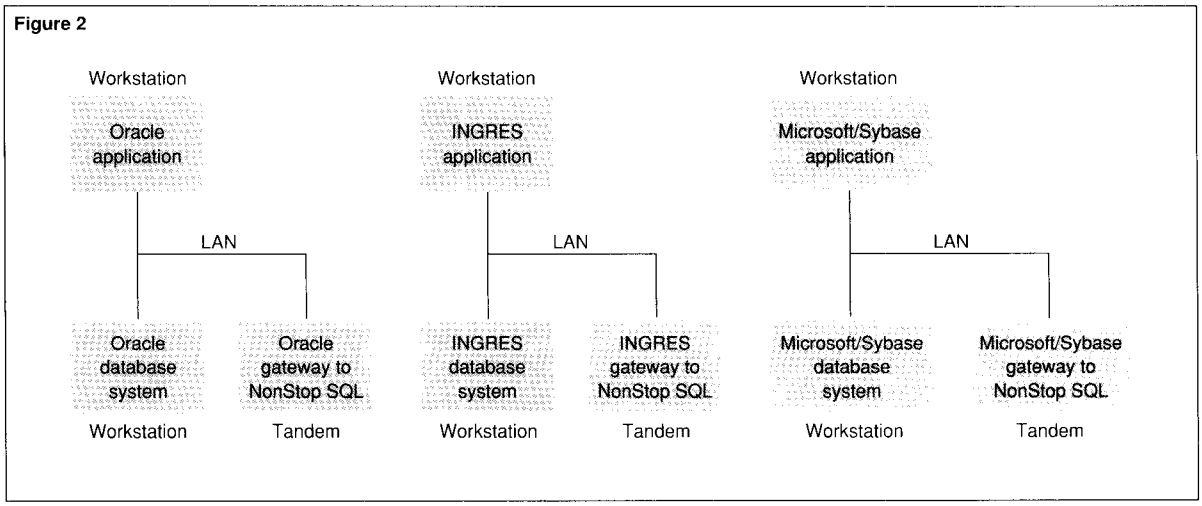


Figure 2 shows examples of gateways to Nonstop SQL. An Oracle application can access an Oracle database directly or a NonStop SQL database through a gateway. In this case, Oracle is the client system and NonStop SQL is the server system. Similarly, an INGRES or Microsoft/Sybase SQL application can make the same calls to its own SQL database or to a NonStop SQL gateway. In this case, INGRES or Microsoft/Sybase is the client system and NonStop SQL is the server system.

Figure 3 shows the operation of a typical gateway to NonStop SQL. A message containing an SQL statement arrives from the client. The gateway decodes the message and translates the SQL statement syntax from the client's SQL to NonStop SQL. Next, the gateway executes the statement and translates the resulting data or status information into the client's format. Finally, the gateway encodes the result in a message and returns it to the client.

The gateway contains code to communicate with the client application (derived from the client system's database server), code to access NonStop SQL, and new code that performs the translations. In Figure 3, the code shown to access NonStop SQL is similar in function to SQLCI2, the back-end server for the NonStop SQL conversational interface (SQLCI).

Gateway Issues

An SQL gateway must resolve several issues in addition to the differences between the client's and server's SQL syntax. These issues include variations in data types, SQL object names, SQL catalog structures, and error numbers. A constraint affecting all these issues is that the client application software is often an off-the-shelf package and cannot be altered to accommodate a gateway.

Connectivity

Most vendors use proprietary message formats and protocols for client-server communication. Formats and protocols define the message contents and valid sequences of messages between the client and server. Common transport mechanisms, such as TCP/IP or NetBios, provide basic interconnectivity. The vendor protocols are implemented above the transport level; they allow a client to identify and connect to a particular server as well as send SQL statements.

Gateways must support the proprietary vendor protocols, including features such as out-of-band cancel messages. An out-of-band message is delivered to the receiver as soon as it arrives. It supersedes the normal messages that have arrived earlier but have not been read from the incoming message queue.

A gateway must connect a client with a suitable server process. Some vendors provide a single multithreaded server, in which case all clients connect to a known address or name. Others provide a separate server for each client; the client first connects to a known server and then makes a second connection to a personal server. The gateway has to mimic this behavior.

Security

Because of their complexity, security issues can be difficult for gateways to resolve. Many vendors provide their own security. The database server owns the files containing the database objects. A client attempting to connect to a server must be authenticated, usually by providing a user ID and password. The client logs on either to the server or to a database, but not necessarily to the system on which the server is running. The server then controls access to SQL objects in the database.

The situation is different for users of Tandem systems. A user logs on to the system, and the Tandem Guardian™ 90 operating system or the Tandem Safeguard™ security system controls NonStop SQL file accessing.

Language Translation

Language translation involves mapping syntactic differences between different versions of SQL. Many vendors closely follow the ISO-ANSI SQL standard but also introduce differences, particularly in data definition statements such as CREATE TABLE or CREATE INDEX. In addition, vendors provide their own extensions to the SQL language such as specifying locking options. If possible, a gateway should support client system extensions by translating them into extensions supported by the server system.

Although it is a violation of transparency, sophisticated users wishing to develop a custom application can benefit from a pass-through feature, which allows the client to have direct access to the server system's version of SQL, including its extensions. In pass-through mode, the client uses server system syntax, and the gateway passes the statement to the server untouched.

For example, pass-through mode to NonStop SQL can be used to create SQL tables partitioned across the network or to set SQL compiler directives for parallel execution. Subsequently, existing client applications, using client system SQL, can take advantage of these features.

Data Type Translation

Although all SQL implementations support common data types like integers and character strings, the precise representations of these data types are diverse. Each vendor supports a different maximum length character string, and numeric types often differ in precision and scale limits. Date and time data types are also common, but they often differ in format.

Figure 3

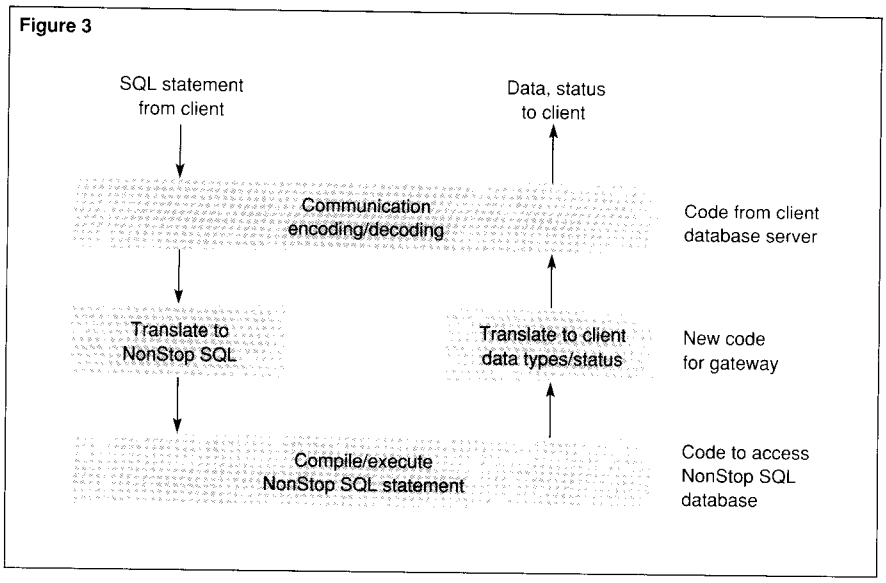


Figure 3.
Inside the gateway.

These differences can affect applications. For example, the following query retrieves part information for all parts weighing more than 48.57226.

```
SELECT *
  FROM PARTS
  WHERE WEIGHT > 48.57226
```

This query could return different results, depending on the number of digits of precision of the WEIGHT column. For example, assume that system A has four-digit precision and system B has nine-digit precision. The query running on system A would return all the parts returned by system B plus the parts weighing between 48.57 and 48.57226. Some vendors also support specialized data types (such as money) that are hard to support fully with common data types.

Object Naming

An SQL system deals with many kinds of objects, including databases, tables, views, indexes, columns, constraints, users, and programs. Each object is named and the names appear in SQL statements and SQL catalogs. Most vendors follow ISO-ANSI naming conventions, but discrepancies exist in conventions such as the lengths of name components and the special characters allowed in names. Tandem does not currently support the ISO-ANSI naming conventions for some types of objects.

A gateway must resolve these naming differences. A variety of techniques can be used, depending on the extent of the naming discrepancies. A typical gateway will use different techniques for the different types of names.

If the differences affect only a few marginal cases, the gateway can resolve them by restricting the names available to the client. For example, suppose the client system supports SQL table names of up to 32 characters, consisting of a-z, A-Z, 0-9 and $. Suppose, also, that the server system supports table names of only 31 characters, consisting of a-z, A-Z, and 0-9. It may be sufficient in such a case to restrict the client to using only server-valid names, because names using $ or names 32 characters long are rare. However, if the server system can only support 18-character names, the restriction would be too great, because many names can be longer than 18 characters.

If the sets of legal names in the client and server systems are significantly different, the gateway can map names by using tables. The gateway scans incoming SQL statements for client names, looks them up in the tables, and replaces them in the statement with corresponding names that are legal in the server system. This technique, called *name-mapping*, avoids restrictions on the clients' use of names but involves the extra work of maintaining the mapping tables.

A gateway must resolve differences in SQL naming conventions.

SQL Catalogs

An SQL catalog is a set of SQL tables that describes objects in the system such as tables, columns, views, and indexes. Catalogs are maintained by the system and are available for users to read. Some programs such as application generators use SQL catalogs extensively.

Almost all vendors' SQL catalogs differ significantly. The names of the catalog tables differ, as do the column order and the meaning and data type of each column. Because users read from the catalogs, the gateway must supply catalogs on the server system that look like the catalogs on the client system. Often, the gateway can accomplish this with SQL views over the server system catalogs. The view is given the name of the client catalog and provides the columns the client expects to see.

In some cases, a view cannot mimic a client catalog. For example, suppose the catalog on the server system uses a separate catalog table row to represent each column in an index. If the client expects to see all the columns of an index listed within one catalog row, an SQL view cannot be used.

When a view cannot be used, the gateway can support a separate SQL table that mimics the catalog table the client expects but contains data extracted from the catalog on the server system. This option is less desirable than using views because a separate table requires extra work to install and maintain. It may be difficult to keep the information in the table up to date if it describes server system objects that can be altered by other server system users (who don't use the gateway).

Transactions

Database vendors support transactions in various ways, just as they do SQL catalogs. A transaction represents a set of operations that are executed as a unit. The system guarantees that either all the operations are completed or none are completed. In some systems, transactions are optional. In others, a transaction must always be in progress. Similarly, some systems perform each data definition operation such as CREATE TABLE in a separate transaction, whereas other systems allow multiple data definition operations in a single transaction. In NonStop SQL, certain data definition operations on nonaudited tables must be performed outside of a transaction.

The gateway can resolve these differences by adopting the most restrictive of the client system and server system transaction models. These restrictions are not likely to cause problems because production applications usually query and modify data and only occasionally deal with data definitions. In most implementations, transactions that modify and query data behave similarly.

Error Handling

Vendor support for error codes and error message text differs widely. This can be a problem because applications often contain substantial logic that deals with error conditions. The gateway must map the error code values for common errors to values understood by the client application. Examples of common errors are "object not found" and "unique constraint violation." The gateway may map other error code values to special values representing gateway or server system errors. The gateway should also map object names in the error message text from server system names back to client system names.

Note that the gateway maps error codes and object names in error text in the opposite direction that it maps object names in SQL statements. In SQL name mapping, the gateway translates a client system name into a name the server system can understand. In error code mapping, the gateway translates an error code generated in the server system into an error code the client system can understand.

Administration

Administration includes maintaining SQL data definitions, monitoring system users and usage, and running utilities to perform tasks such as backing up and restoring data. Except for maintaining SQL data definitions, these administrative tasks differ greatly in each system and gateways do not try to make them transparent. A gateway-supplied pass-through feature can help client users perform these administrative tasks. Otherwise, the server system administrator can perform them.

Compatibility Across Releases

Because new versions of client and server systems are usually released at different times, it is difficult to provide compatibility across releases. The gateway vendor must provide a mode in each new version of the gateway that allows applications written for previous versions to run. The gateway program may have to contain code to run on multiple versions of the server system in order to take advantage of new features as they become available. Ideally, a new version of the gateway program should be released whenever a new client system version is released so that client applications using the new feature will run on the gateway. In general, the gateway program may have to support multiple versions of both the client and server systems.

Solutions for NonStop SQL Gateways

Gateways to NonStop SQL on Tandem systems are being developed to allow popular SQL applications running on other systems to use NonStop SQL. All the major gateway issues have been considered in designing gateways to NonStop SQL.

In practice, a gateway does not support totally transparent access. (It does not provide all features of the client database system.) However, to be successful, a gateway to be used for a particular set of SQL applications must support at least the subsets of SQL syntax, data types, and catalogs used by those applications. The SQL subsets used by popular applications center on data manipulation (SELECT, UPDATE, DELETE, INSERT) and simple data definition (CREATE/DROP TABLE, INDEX, VIEW). These subsets have provided the design focus for the gateways to NonStop SQL. To promote easy connectivity of future applications, Tandem is active in standardization efforts in such areas as SQL, transaction management, and remote data access.

Security

The Guardian 90 operating system provides security for the NonStop SQL gateways. Clients that connect to a gateway must present a Guardian 90 user ID and password. With some gateways, clients do this directly, while with others the gateway does it on their behalf. Access to NonStop SQL objects is based on the Guardian 90 user ID presented at connect time.

Most other database vendors support some form of the ISO-ANSI GRANT/REVOKE security model. The GRANT and REVOKE statements are not simulated by the gateways to NonStop SQL. This is not generally a problem for client applications because most of them do not use GRANT and REVOKE.

Figure 4

Client name	NonStop SQL name	Type
ADMIN.DEPARTMENTS	\N.$V1.ADMIN.DEPARTME	TABLE
ADMIN.EMPLOYEES	\N.$V2.ADMIN.E345	TABLE
ADMIN.EMPD10	\N.$V1.ADMIN.EMPD10	VIEW

Figure 4.
A name-mapping table.

Name Mapping

In the NonStop SQL gateways, object naming requires special attention. Most SQL vendors support the ISO-ANSI naming convention (SCHEMA.OBJECTNAME) for tables, views, and indexes. Each component of the name is a character string, although the name lengths vary among vendors. Currently, NonStop SQL only supports Guardian 90 file names of the form \NODE.$VOL.SUBVOL.FILENAME for these objects.

Applications cannot run transparently on a gateway if the object names are different. Consequently, the gateways to NonStop SQL support name mapping for objects with Guardian 90 names. The name-mapping tables are implemented as SQL tables, and the gateways use NonStop SQL to access and maintain them.

Figure 4 illustrates a name-mapping table. The table contains the client name, the corresponding NonStop SQL (Guardian 90) name, and the type of object.

The gateways use the name-mapping tables to replace client SQL names with NonStop SQL names in SQL statements. To recognize and replace names in statements, the gateways must usually parse the statement.The parsing is needed because recognizing name types can be difficult. Consider, for example, the SELECT list of the subquery in the following query:

```
SELECT COL1
   FROM TAB X
   WHERE EXISTS
         (SELECT X.COL2,Y.COL2
          FROM Y
          WHERE COL1=COL3)
```

Y in Y.COL2 should be mapped because it is a table name, but X in X.COL2 should not because it is a correlation name for table TAB. (NonStop SQL follows ISO-ANSI conventions for correlation names, so they do not have to be mapped.)

Creating and Placing Name-Mapping Tables. The way in which the gateways create and place a name-mapping table depends on the architecture of the SQL database. Commonly, an SQL database is a set of objects described in a single SQL catalog. Databases are in a one-to-one correspondence with SQL catalogs. In many SQL implementations, users must explicitly connect to a database or include the database name in an object name in order to access objects in the database.

Because of its unique distributed system, Tandem is able to extend this common database model. A Tandem network can contain an arbitrary number of NonStop SQL catalogs, and users can directly access any SQL object in the network by specifying its full Guardian 90 name. Thus, the network is a single distributed database instead of a distributed network of databases.

For compatibility with other vendors' notions of a database, the gateways to NonStop SQL usually define a "database" as the set of objects registered in a single NonStop SQL catalog. One set of name-mapping tables is associated with a single NonStop SQL catalog and can only refer to objects in that catalog. One gateway has relaxed this restriction to allow objects registered in other catalogs to be added to the mapping tables.

A NonStop SQL catalog with name-mapping tables is visible as a database to gateway clients. This means the clients "see" the "database" and can access its objects. The gateway provides utilities to make NonStop SQL catalogs visible (create the name-mapping tables) or invisible (erase them). In some cases, a client-issued CREATE DATABASE command can create a NonStop SQL catalog and then execute the utility to make the catalog visible.

Creating Client and Server Object Names. To make an existing NonStop SQL catalog visible, the gateway must generate the client-system object names and place them in the name-mapping tables. In general, a client name can be any valid name, but it is desirable to make the client name suggestive of the Guardian 90 name from which it is derived. Similarly, if the client creates an SQL table, T, the gateway must choose a Guardian 90 name for the new table. Again, it is desirable to choose a name that can be related to T.

One technique is for the gateway to try to equate the client name with the last component of the Guardian 90 name. An exact match is not always possible because the client name can exceed eight characters or the matching name may already be in use. In these cases, other techniques such as truncation or random selection are used to generate names.

An example is shown in Figure 4. The client name DEPARTMENTS is truncated to the Guardian 90 name DEPARTME. The name EMPD10 is used in both the client system and the NonStop SQL system. Finally, the client name EMPLOYEES is mapped to the Guardian 90 name E345 because the Guardian 90 name ADMIN.EMPLOYEE is already in use.

Security for Name-Mapping Tables. Security for gateway name-mapping tables is a difficult issue. On the one hand, name-mapping tables should be tightly secured because they represent the gateway view of NonStop SQL. On the other hand, name-mapping tables should be loosely secured because the gateway processes usually run under common Guardian 90 user IDs and must modify the tables for data definition statements. The latter approach is followed in the gateways to NonStop SQL. Note that this approach does not impair the integrity of NonStop SQL objects.

The gateways provide special utilities to check and correct the name-mapping tables as needed. These utilities are required because the name-mapping tables are not active. For example, if a NonStop SQL user drops a NonStop SQL table, the appropriate name-mapping table entries are not automatically deleted.

Pass-Through Mode

NonStop SQL provides several extensions to SQL, primarily for use in high-performance, production applications. For example, users can specify file options with the CREATE TABLE statement, specify SQL compiler options that allow a query to execute in parallel, and control locking granularity and duration.

To provide client access to these extensions, most gateways to NonStop SQL provide a pass-through mode. The client can issue any NonStop SQL statement and the gateway executes it directly. No names are mapped and no translation is performed. The pass-through mode is indicated either by a special dynamic SQL verb or by an escape sequence within a client SQL statement.

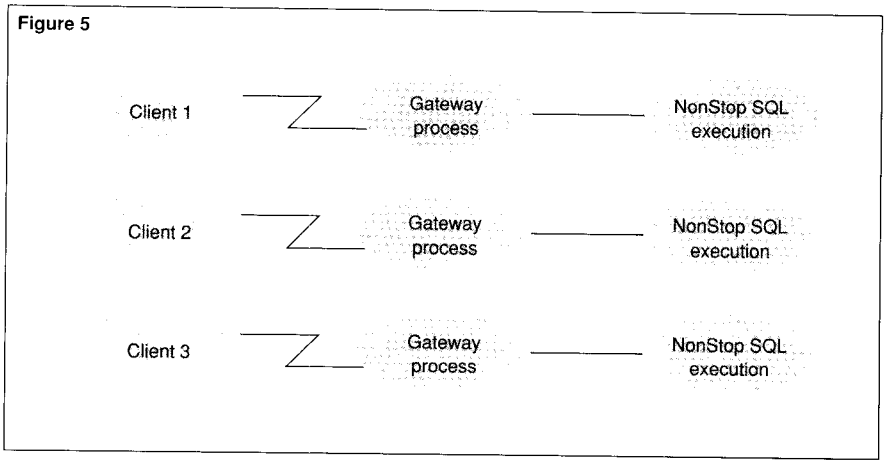


Figure 5.
Gateway architecture showing the gateway processes for three clients.

Tandem Gateway Architecture

In designing and implementing a gateway, one must consider the various issues associated with the translation tasks the gateway performs. One must also consider a second set of issues associated with the architecture of the gateway and its environment because the gateway to NonStop SQL is an application program running on Tandem systems.

The Tandem gateways use standard transport mechanisms such as TCP/IP and NetBios for connectivity. Clients, or servers on their behalf, must provide a Guardian 90 logon (user ID and password). It can be difficult for a client application package to do this. (The application code cannot be altered in any way.) In some vendor systems, a client connects first to a database server and then to one of the databases the server owns. In others, a client connects to a logical database name. Library code bound into the client application consults a local table or a name server to determine where the database or server resides, what its real name is, and how to connect to it.

One way to provide connectivity is to run a one-time utility on the client system. The utility performs the Guardian 90 logon, creates a gateway process on the Tandem system, and causes the gateway process to wait for input. The utility also registers the gateway server process in the local table or name server. When the client application makes a connection, it becomes connected to the existing gateway process. If the client system fails and is restarted, the utility must be rerun.

The process architecture of the gateway raises another important issue. Many database vendors provide a single multithreaded server for a set of databases. In contrast, Tandem provides a separate gateway process for each client. This architecture provides isolation between clients to benefit security and performance, and it allows clients to run easily under different Guardian 90 user IDs. However, it also limits the number of concurrent clients to fewer than a typical Pathway application can support.

Figure 5 shows the gateway processes for three concurrent clients. Each gateway server actually consists of two processes, the gateway process and a NonStop SQL execution process. The gateway process performs all gateway tasks except executing NonStop SQL statements. It passes the NonStop SQL statements to a second process for execution.

This two-process architecture allows the gateway to support out-of-band cancel requests that must take effect immediately, even if an SQL statement is currently being executed. NonStop SQL does not provide a NOWAIT interface. Instead, the gateway simulates a NOWAIT interface by using the NOWAIT option for the messages it sends to the NonStop SQL execution process. This frees the gateway process to monitor the communication line for a client-issued cancel command. If a cancel command arrives, the gateway process stops the NonStop SQL execution process (effectively interrupting the SQL statement) and starts a new one. This two-process architecture is the same as the one the NonStop SQL conversational interface (SQLCI) uses to support the BREAK key function.

Standards

Most of the issues for gateway design discussed above arise because of differences in SQL implementations. The standardization of SQL and other interfaces, together with vendor compliance, can greatly reduce the issue list. In addition to ISO-ANSI SQL, several other standardization efforts currently in progress affect SQL gateways. It is important to note that the intense efforts of the standardization committees and the participation of virtually all vendors reflect the importance of interoperability among the various implementations of SQL.

The Remote Data Access (RDA) committees of ISO and ANSI are working to standardize remote access to data, including SQL data (ISO-ANSI, 1989b). They are developing standardized verbs for connecting to a remote server and for sending and executing statements. Other ANSI and ISO committees are working on transaction support. The RDA committee will incorporate their results into the RDA standard.

A consortium of vendors and users called SQL Access is working to extend the standardization efforts so that they can apply to actual implementations. SQL Access addresses practical details not covered by the RDA and ISO-ANSI SQL standards. For example, it proposes choosing a simple subset of ISO-ANSI SQL syntax and data types acceptable to all database vendors and users. It also addresses network naming, standardized SQL catalogs, standardized error numbers, and precise but efficient message formats and protocols. To show feasibility, several members of SQL Access are building a demonstration system consisting of clients and servers on a number of different platforms. SQL Access actively shares its results with ISO-ANSI committees and other organizations, including Open Systems Foundation (OSF), X/OPEN, and National Institute for Standards and Technology (NIST).

Tandem played a key role in forming SQL Access and participates actively in the consortium. Tandem also participates in the ISO-ANSI SQL, RDA, and other standardization efforts. Progress in this direction will reduce the number and complexity of gateways. Successful standardization would provide clients with a single interface to access all database systems and would allow each database vendor to build a single gateway to achieve connectivity to all client systems.

Conclusion

Tandem is building a gateway that connects Microsoft/Sybase clients to NonStop SQL and is participating in the development of two other gateways that connect Oracle and INGRES clients to NonStop SQL. With these gateways, Tandem database users can benefit from popular application tools on PCs and workstations.

At the same time, Tandem is actively contributing to standardization efforts relating to SQL interconnectivity. These standardization efforts will help stem the proliferation of gateways.

Database vendors face an interesting challenge if they intend to adhere to standards and also differentiate their products. Tandem is ideally suited to this challenge because standard interfaces still allow outside clients to take advantage of the inherent strengths of Tandem systems: high availability, high performance, distribution, and fault tolerance.

References

ANSI. 1989. Standard for Database Language (SQL). ANSI X3.135-1989.

Cohen, H. 1988. Overview of NonStop SQL. *Tandem Systems Review*. Vol. 4, No. 2. Tandem Computers Incorporated. Part no. 13693.

ISO-ANSI. 1989a. Database Language SQL2 and SQL3 (working draft). ANSI X3H2-89-252 or ISO DBL FIR-3.

ISO-ANSI. 1989b. Generic Remote Data Access Service and Protocol. ANSI X3H2.1-89-113 or ISO/JTC 1/SC 21 N 3606.

NonStop SQL Programming Reference Manual. 1989. Tandem Computers Incorporated. Part no. 84258.

Acknowledgments

I would like to thank the members of the NonStop SQL and connectivity groups and the reviewers of this article for their many suggestions and corrections.

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

Batch Processing in Online Enterprise Computing

Until now, most large businesses have satisfied their various computing needs by maintaining more than one database. Typically, one or more large databases support batch processing, and other databases, often running on separate hardware and using different database structures, handle online transaction processing (OLTP) and queries. The disadvantages of managing separate databases are many. Because information must be transferred from one database to another, one of them is always out of date. Also, operating different database designs is expensive.

These multiple databases came into being as a way to work around the limitations of traditional computing architectures. For example, hierarchical databases were not suitable for all types of data accesses, or systems did not have high enough availability or expandability to meet the demands of all users. Therefore, multiple databases began to proliferate.

Tandem offers a solution to the multiple-database dilemma. Enhancements in the performance and operability of Tandem™ software now allow users to execute batch jobs and submit ad hoc queries as well as support OLTP on a single, enterprise-wide database. OLTP, batch processing, and query processing can occur simultaneously, without impairing the high performance and continuous availability associated with Tandem systems. These achievements are possible on Tandem systems because of the unlimited expandability and high availability inherent in the Tandem architecture. Furthermore, users can meet all these computing needs using a relational database management system, the Tandem NonStop™ SQL system, without sacrificing performance.

The challenge for Tandem has been to integrate batch processing with an online environment. Batch jobs should execute on a large database while multiple users continue to have immediate OLTP and query access to current information that may be distributed across a worldwide network. The same information must be available at the same time for multiple uses. Many Tandem products contribute to achieving this goal, including Release C30 of the Guardian™ 90 operating system, NonStop SQL Release 2, and the NonStop Cyclone™ mainframe computer system.

This article describes the enhancements in batch processing introduced with these Tandem products. It discusses the advantages of integrating batch processing with OLTP. Next, it describes the important software requirements for online enterprise computing, including I/O optimization, record locking, transaction protection, and new utilities such as online file reorganization. The article also describes performance improvements provided by the NonStop Cyclone mainframe computer. Finally, it describes parallel batch processing, a key performance enhancement for an integrated online enterprise.

Why Batch Processing Is Needed

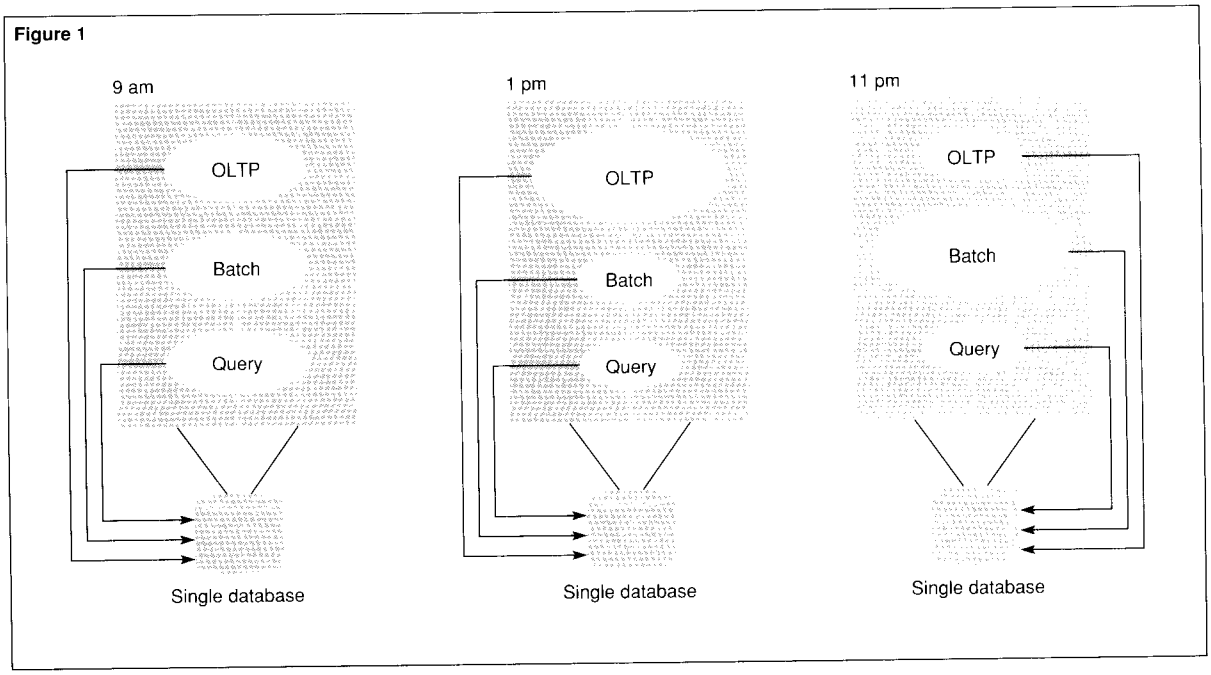
Although a growing number of business applications use OLTP, batch processing will continue to play an important role in an integrated online enterprise. Because batch processing is highly efficient, it can accomplish certain large, repetitive tasks at a great savings in cost. For example, in the banking industry, interest must be applied to groups of accounts once a month. The application software must use the same formula (the new interest) to update every record in the customer table or file. A batch application can save time and system resources by reading and changing the account information in a single, sequential process.

The objective of batch processing is to process as many records as possible over a set time period. Typically, data access is sequential (records are read and updated in order), a more efficient method than the random data access usually required by OLTP. Also, users can perform batch processing during off-peak hours to make even more efficient use of system resources.

Finally, many businesses will continue to keep batch applications for historical reasons. Converting a large batch application to an OLTP application can be complicated. The cost of developing the new application, training personnel, and managing the conversion can be too high to justify.

An online enterprise should include both batch processing and OLTP. When deciding whether to design a particular business application for OLTP or batch processing, users should weigh the benefits of OLTP (improved customer service and savings in time) against those of batch processing (efficient execution of large computing tasks).

Figure 1.
Online enterprise computing accesses a database with varying levels of OLTP, query processing, and batch processing at different times of day. A typical example is shown in this figure.



The Benefits of Using a Single Database

An online enterprise that uses a single database for OLTP, batch processing, and ad hoc queries offers many advantages to its users. By having immediate access to information from both centralized and distributed business operations, executives can improve the accuracy and speed of their decisions. Batch jobs that access the online database can also use the most current information. If older batch programs or activities such as sorting require database extracts, users can create the extracts from the same up-to-date data. Operating a single database can be less expensive than operating two databases because it requires only one hardware and software environment.

A single database eliminates the cost of transferring data from one database to another. Also, in traditional two-database installations (a database for batch and another for OLTP), batch jobs often execute during off-peak hours (usually overnight) and finish by the time customers resume using OLTP applications (usually the next morning). With Tandem's single, online database, batch jobs can execute concurrently with OLTP operations. Release C30 of the Guardian 90 operating system and the Tandem NetBatch™-Plus job scheduling software help to keep batch processing from interfering with OLTP response times. Though most batch processing will continue to take place during off-peak hours, the single database reduces the pressure to complete batch jobs within a fixed time period. (See Figure 1.)

The information itself is much easier to manage in a single database than in two databases. Users do not have to maintain data relationships between the OLTP and batch databases. This also saves processing time and eliminates the software needed to communicate between two databases.

Achieving Online Batch Processing

When a single database supports all types of data access (OLTP, query processing, and batch processing), the distinctions between those types of access are no longer absolute. Both OLTP applications and batch jobs can be thought of as processing transactions, though the two types of transactions may be of different sizes. Online transactions are relatively small and occur at random. Usually, batch jobs process larger transactions or large amounts of work that comprise what may be considered a virtual transaction. Batch jobs are scheduled, whereas queries can be thought of as batch jobs that arrive at random.[1]

Because all three types of access can occur at the same time, the online enterprise must manage them so that they do not interfere with one another. Therefore, batch jobs that update the same database as OLTP (while OLTP is active) are fundamentally different from traditional batch jobs. For example, batch jobs in an online enterprise require sophisticated record locking because the system cannot allow them to hold exclusive locks on entire data files.

Furthermore, because batch jobs may be updating the online database system, they cannot recover from errors merely by restoring files and rerunning. In some instances, transaction protection appropriate for batch processing is also required.

Tandem calls this new style of batch processing *online batch processing* (OLBP). Batch processing becomes online when a batch job updates information and a split second later an OLTP transaction can use that new information. Conversely, an online batch job can access information updated a split second before by an OLTP transaction. To support online batch processing, a single computer environment must provide the required high performance and functionality for all types of data access.

[1]Typically, ad hoc queries do not alter data; batch jobs may or may not alter data.

Tandem supports OLBP with the development of four major elements:

- Software performance and functionality.
- Flexible batch scheduling.
- Hardware that supports OLTP, batch processing, and query processing.
- Parallel job processing that provides a linear improvement in performance with each processor added to the system.

Software Performance and Functionality

The software features that achieve OLBP performance and functionality in a Tandem environment include low-level integration of database queries, automatic I/O optimization, effective record locking, transaction protection appropriate for OLBP, protection of OLTP response times from batch processing, and batch utility performance. These features are also discussed in more detail in the September 1989 issue of the *Tandem Systems Review* (Keefauver, 1989).

Low-Level Integration

A key feature of OLBP is its efficient use of computer resources. The low-level integration of NonStop SQL into the Guardian 90 operating system enhances processing performance (Borr, 1988). For example, NonStop SQL can send a single request to the disk process (a part of the Guardian 90 operating system) to update multiple data records. Because almost no data is passed up through higher layers of software, large savings in time and system resources result.

Automatic Optimization of I/O

To improve programmer productivity and to ensure optimal batch processing performance, the selection of optimal disk I/O used in sequential processing has been automated. For example, NonStop SQL Release 2 automatically selects optimal blocking and buffering of application I/O. It can transfer as little as 512 bytes or as much as 56 kilobytes of data at a time. Programmers no longer need to program and test various I/O transfer methods in an attempt to find the optimal method.

Aspects of the traditional record-at-a-time interface used by the Tandem Enscribe record management system have also been enhanced. For example, COBOL85 applications that use entry-sequenced files created by Enscribe can take advantage of fast I/O routines. These routines transparently block and buffer disk I/O, providing up to ten times faster throughput than unblocked access methods.

Record Locking

In an online enterprise, batch jobs may often run at the same time and have access to the same data as online transactions. To allow OLBP and OLTP to coexist, record lock management must become more sophisticated. For example, batch jobs operating on a retail inventory database should lock the fewest records for the shortest time needed to accomplish an operation. Database designers have applied the rule of minimal required locking to OLTP operations; they must apply a similar rule to OLBP jobs in an online environment.

When a batch job obtains access to a data file, it must lock only the records it needs at any one time so that OLTP can continue to operate on the other records in the file. For example, if retail clerks process purchases that change the inventory database online, a batch job that changes or updates inventory levels cannot lock the entire file. If the batch job is written to share the database with OLTP, the job can run whenever it is requested and provide completely up-to-date information. Therefore, to permit concurrent batch processing and OLTP, each batch job must efficiently and quickly apply (and remove) record locks.

With the Guardian 90 operating system, the application designer can specify a lock on a single record (the default), an entire file, or a range of records. A *generic lock*, which locks a range of records having a common key prefix, can reduce the overhead of applying and removing locks. It also reduces demands on main memory. A single lock takes about 52 bytes of main memory. Using a generic lock, the retail inventory batch job can lock a category of items (such as slacks or dress shirts) comprising 100 records in about the same time it would take to lock a single record. The job not only uses less main memory, but also runs faster and uses less total CPU time. Because generic locks are easy to use, the application designer can accurately apply the minimal number of locks needed at any one time.

Transaction Protection for OLBP

When a batch job obtains access to a database where OLTP transactions are also executing, Tandem Transaction Monitoring Facility (TMF™) software may be needed to provide transaction protection for the records being accessed, just as it does for OLTP transactions. TMF includes many optimizations, including some that increase batch throughput.

Tandem continues to enhance TMF to meet the needs of concurrent OLTP, OLBP, and query processing. More information on using TMF to optimize batch applications appears in Keefauver (1989).

Simultaneous OLTP, Query Processing, and Batch Processing

By taking advantage of lulls in the demand for OLTP, OLBP can take place concurrently with OLTP. Thus, OLBP can accomplish more batch work than traditional batch processing, which often occurs during fixed, off-peak time periods (usually overnight).

Enhancements have been made to the Guardian 90 operating system to make sure that disk I/Os of low-priority jobs, such as batch jobs, have little or no impact on higher priority jobs, such as OLTP. The disk process (a part of Guardian 90 Release C30) executes I/O requests in priority order, matching the Guardian 90 priority of the requesting jobs.

These enhancements also keep low-priority jobs from monopolizing the disk process after it starts executing them. After the disk process begins executing a request, it avoids being monopolized by a large job by periodically checking its request queue. If a request of higher priority than the one it is currently executing appears, it suspends the current request and immediately executes the higher priority request. Thus, an online transaction will always interrupt and supersede a long-running batch (or query) request. During a lull in online transactions, the disk process continues executing the low-priority batch request, starting where it left off. Low-priority batch processing can now run at any time during the day by utilizing spare system resources that are not otherwise committed.

OLBP can accomplish more batch work than traditional batch processing.

OLBP Utilities

Tandem has enhanced the performance of OLBP utility programs. For example, with Release C30 of Guardian 90, enhancements to disk controller microcode (in the Tandem 3120 and 3125 models) allow Tandem systems to write data to disks more quickly. Because of this performance enhancement, several utilities can write files up to 35 percent faster than they could in previous releases. This enhancement benefits such bulk I/O disk write utilities as TMF transaction audit trail writes, database loads, File Utility Program (FUP) file duplicating, the Tandem FASTSORT batch sorting utility, and RESTORE data recovery.

Parallel processing can also improve the performance of Tandem utilities. For example, the FASTSORT program takes advantage of parallelism. By specifying the parallel processing option using two processors, FASTSORT can sort about twice as quickly as it can using one processor (Gray, 1986).

The online reorganization utility, introduced with Release C30 of Guardian 90, can reorganize audited Enscribe files or NonStop SQL tables online. By reorganizing a file in place (operating on one data or index block at a time), the utility has little or no impact on OLTP performance. Online transactions and batch jobs can continue to access and update the file during a reorganization. Reorganizing files helps to keep batch processing and OLTP running at peak performance (Smith, 1990).

Hardware for OLTP and Batch Processing

The online enterprise needs hardware flexible and powerful enough to support all the demands placed on it. With the hardware and software enhancements of recent years, Tandem has improved batch processing performance about tenfold (Keefauver, 1989; Oleinick and Shah, 1986). The Tandem NonStop Cyclone system improves batch processing throughput another threefold. It allows data transfers to be driven at much higher rates than before. The NonStop Cyclone can consistently run more than one batch job concurrently and provide increased throughput.

The NonStop Cyclone system features direct memory access (DMA), which transfers data from the I/O channel to main memory with very little processor intervention. By improving the speed and efficiency of data transfers to and from main memory, the NonStop Cyclone can easily handle the great number of data transfers required in batch processing.

Users can configure the NonStop Cyclone with four I/O channels (twice as many as other Tandem systems). Because batch processing usually places a heavier load per second on I/O channels than OLTP does, the additional channel capacity is especially valuable for batch jobs. Enhancements to the I/O channel itself make it possible for the NonStop Cyclone to transfer data in large bursts, another feature that improves batch job performance.

Parallel Batch Processing

Tandem has long provided parallel processing to boost OLTP performance. Now, Tandem provides at least four ways to apply the benefits of parallel performance to batch processing.

Figure 2

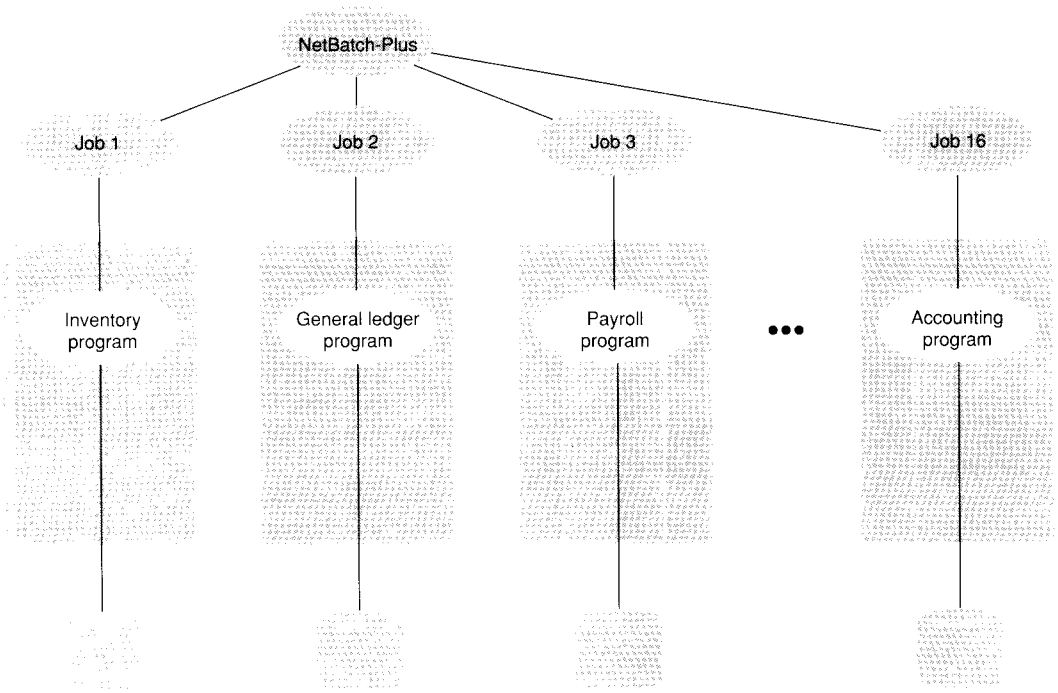


Figure 2.
Inter-job parallelism. Sixteen different batch jobs run in 16 processors scheduled by NetBatch-Plus.

On a Tandem multiprocessor system, users can execute multiple batch jobs, each using a different type of parallel processing, while OLTP applications are also using the same files. Moreover, by using the Tandem NetBatch-Plus batch scheduling software available with Guardian 90 Release C20, users can schedule and control all batch jobs, including those that employ parallelism, across all nodes in a Tandem network (Earle and Wakashige, 1990). Four approaches to parallel batch processing on Tandem systems are:

- Inter-job parallelism.
- Automatic intra-job parallelism.
- Designed-in intra-job parallelism.
- Job step pipelining.

Inter-Job Parallelism

This first approach to parallel batch processing utilizes multiple processors to run multiple batch jobs. For example, in a Tandem system containing 16 processors, users can schedule 16 batch jobs to run simultaneously, one job in each processor. (See Figure 2.) This feature is called *inter-job parallelism* and is available on most vendor systems.

Intra-Job Parallelism

With NonStop SQL Release 2, users can request that the system automatically perform a single batch job in parallel. For example, in a 16-processor system in which database tables are partitioned over 16 (or more) disk volumes, NonStop SQL can distribute a single job to all 16 processors and update the disk partitions simultaneously.

Figure 3

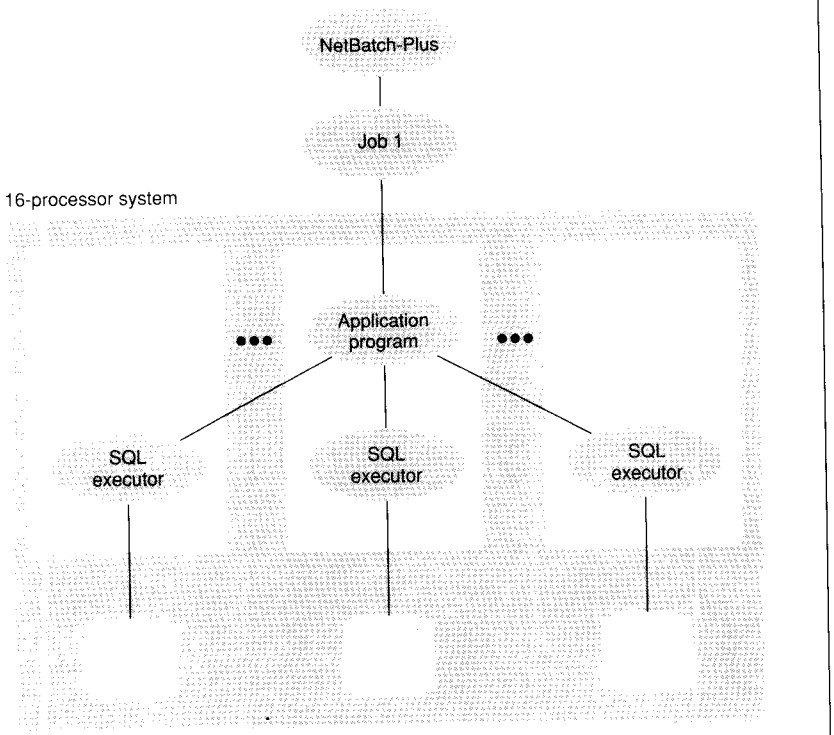


Figure 4

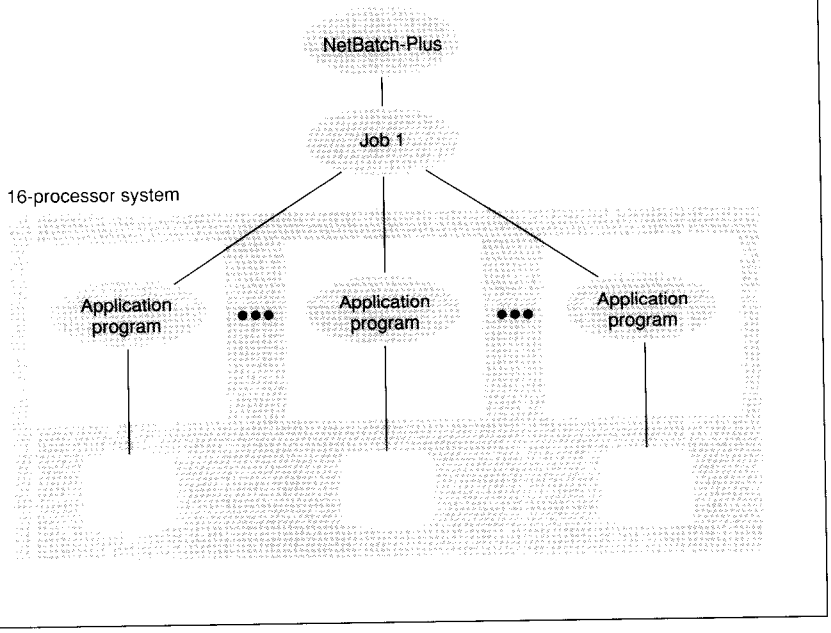


Figure 3.
Automatic intra-job parallelism. NonStop SQL Release 2 controls the execution of 16 subprocesses to simultaneously update 16 partitions of a NonStop SQL table. NetBatch-Plus can control the application program that contains the NonStop SQL statements responsible for the parallelism.

Figure 4.
Designed-in parallelism. Programmers can design a batch program to work against 16 different disk file partitions at once. If desired, NetBatch-Plus can pass startup parameters and control the 16 copies of the batch program.

This feature, called *automatic intra-job parallelism*, is especially useful for large batch jobs requiring repeated operations. For example, a batch application might need to update all bank customer records, adding seven percent interest to those records with money market accounts. The first processor and disk volume processes customer records A through C, the second processes records D through F, and so on. (See Figure 3.)

If the job runs in parallel across 16 database partitions, it will execute in approximately one-sixteenth the time it would take a single processor to execute. Intra-job parallelism is described in detail elsewhere in this issue of the *Tandem Systems Review* (Moore and Sodhi, 1990).

Designed-in Parallelism

While automatic intra-job parallelism operates at the system level, users can also design specific applications to process in parallel. *Designed-in intra-job parallelism* can benefit complex batch applications by distributing a complicated task among several processors.

For example, users can design a batch program so that a separate copy of the program executes in each of 16 processors and uses the disk volumes associated with that processor. The parallel design allows the application to process 16 portions of the batch job simultaneously. (Users do not have to design parallelism into a batch application comprising a simple series of NonStop SQL queries, because automatic intra-job parallelism can process each NonStop SQL query in parallel.)

Figure 4 shows NetBatch-Plus controlling a batch application designed to execute in parallel. More information about designed-in intra-job parallelism appears in Keefauver (1989) and de Torok (1989).

Job Step Pipelining

When a batch job contains discrete program steps, users can design the job to use multiple processors in parallel. This feature, called *job step pipelining*, is especially effective when each successive step uses the output of a previous one. Each job step executes in a different processor and passes its output directly to the next process, bypassing disk I/O.

Because the message-based Guardian 90 architecture can pass data from one processor to another so efficiently, this method can greatly reduce the run time of a job. For example, a job with job steps distributed across 16 processors can use less than one-sixteenth the run time it would use in a single processor. (See Figure 5.) Job step pipelining also reduces the need for temporary file disk space, which batch processing often requires.

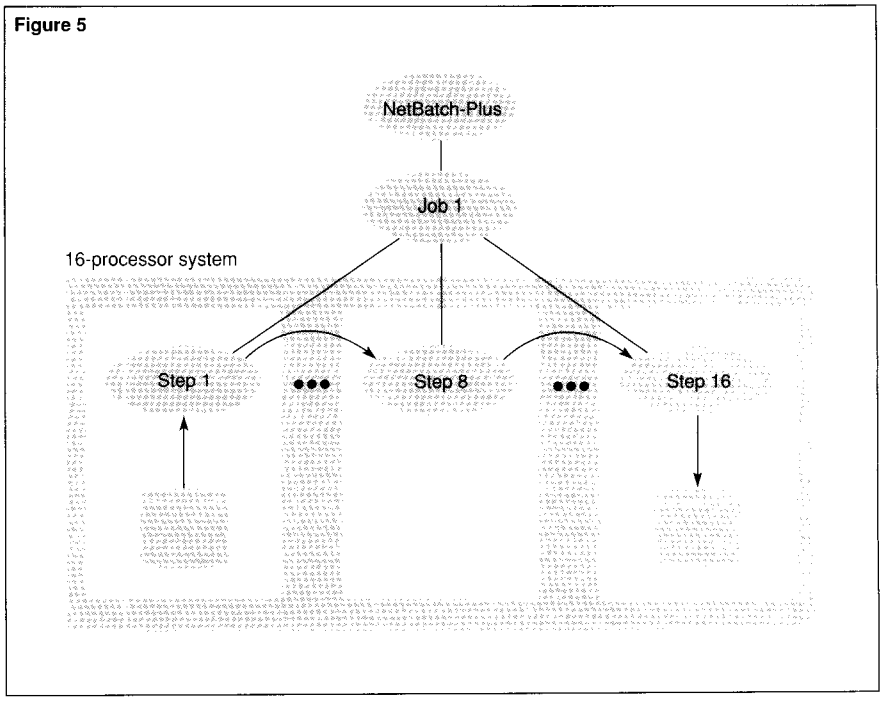
The Benefits of Parallelism

By improving the performance of OLTP, OLBP, and query processing, parallelism benefits the online enterprise in two important ways. First, users who start with a Tandem system that provides satisfactory performance can maintain that performance as their database grows simply by adding resources to their system. The ability to perform tasks on a large database as quickly as on a small one is called *scaleup*.

Second, users who want to improve the performance of their enterprise can simply add resources to their system. For many applications, doubling resources can cut processing time in half. The ability to improve the performance of a stable database in this way is called *speedup*. Scaleup and speedup are discussed elsewhere in this issue of the *Tandem Systems Review* (Englert and Gray, 1990).

To demonstrate the capabilities of parallelism provided by NonStop SQL, Tandem ran many benchmark tests on different Tandem systems.[2] The scaleup tests showed that the elapsed time for database queries remained constant when the number of resources grew in linear proportion to the size of the database. For example, a complex query that took one hour to execute on one processor continued to execute in one hour as the database and the number of processors both grew eightfold.

Figure 5



The speedup tests showed that the elapsed time for database queries decreased in linear proportion to the number of resources used. A complex query that executed in eight minutes on one processor executed in one minute when eight parallel processors were used. The benchmark tests demonstrating scaleup and speedup are described elsewhere in this issue of the *Tandem Systems Review* (Englert et al., 1990).

Figure 5.
Job step pipelining. One job with 16 steps (each consisting of an executing program) processes the data without using disk scratch file space. By the time the first step reads the last data record, only a few seconds remain before the last step finishes executing. If desired, NetBatch-Plus can also control this form of processing.

[2]Results were audited by Codd and Date Consulting Group, San Jose, California.

Users can easily expand most Tandem systems from 2 to 16 processors. With a fiber-optic link, users can expand a system to as many as 224 processors. Using the Tandem Expand™ network software, users can extend their system into a geographically distributed network of up to 4080 processors, where similar speedup and scaleup factors are possible.

Conclusion

The traditional view of mixed workloads is that OLTP, batch processing, and query processing cannot coexist cost-effectively on one system because they make such different demands on the system. This belief has led most large enterprises to implement a multiple-database strategy to satisfy their information processing needs.

Tandem's approach to online enterprise computing combines the cost-effective elements of batch processing with an online environment. By designing and implementing parallel processing and database technology for online enterprise computing, Tandem offers a flexible, cost-effective alternative to the multiple-database strategy. Tandem NonStop systems provide a hardware and software architecture that can support a single online database exploiting the benefits of concurrent OLTP, online batch processing, and query processing.

References

Borr, A. 1988. Technical Paper: High-Performance SQL Through Low-Level System Integration. *Tandem Systems Review*. Vol. 4, No. 2. Tandem Computers Incorporated. Part no. 13693.

de Torok, D. 1989. The International Tandem Users' Group. *Tandem Users' Journal*. Vol. 10, No. 3. May/June 1989, Chicago, Ill.

Earle, G. and Wakashige, D. 1990. NetBatch-Plus: Structuring the Batch Environment. *Tandem Systems Review*. Vol. 6, No. 1. Tandem Computers Incorporated. Part no. 32986.

Englert, S. and Gray, J. 1990. Performance Benefits of Parallel Query Execution and Mixed Workload Support in NonStop SQL Release 2. *Tandem Systems Review*. Vol. 6, No. 2. Tandem Computers Incorporated. Part no. 46987.

Englert, S. et al. 1990. NonStop SQL Release 2 Benchmark. *Tandem Systems Review*. Vol. 6, No. 2. Tandem Computers Incorporated. Part no. 46987.

Gray, J. 1986. FASTSORT: An External Sort Using Parallel Processing. *Tandem Systems Review*. Tandem Computers Incorporated. Vol. 2, No. 3. Part no. 83938.

Keefauver, T. 1989. Optimizing Batch Processing. *Tandem Systems Review*. Vol. 5, No. 2. Tandem Computers Incorporated. Part no. 28152.

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

NetBatch-Plus Data Sheet. 1989. Tandem Computers Incorporated. Part no. 101030.

Oleinick, P. and Shah, P. 1986. A Performance Retrospective. *Tandem Systems Review*. Vol. 2, No. 3. Tandem Computers Incorporated. Part no. 83938.

Smith, G. 1990. Online Reorganization of Key-Sequenced Tables and Files. *Tandem Systems Review*. Vol. 6, No. 2. Tandem Computers Incorporated. Part no. 46987.

Tandem Performance Group. 1988. Tandem's NonStop SQL Benchmark. *Tandem Systems Review*. Vol. 4, No. 1. Tandem Computers Incorporated. Part no. 11078.

Timothy Keefauver is product manager for batch processing, Disk Process 2, and optical storage. Before joining Tandem, he worked in systems management, design and management consulting, econometric research, and marketing research using various vendor systems. Tim holds a B.A. in economics from Monmouth College and an MBA in business policy from the University of Chicago.

TANDEM SYSTEMS REVIEW ORDER FORM

Use this form to subscribe, change a subscription, and order back copies. Tandem customers must complete Part A of the subscription order form and the questionnaire on the reverse side of this page. Submit the completed form to your local Tandem representative for approval. The Tandem representative will complete Part B of this form and submit it for processing.

Part A and reverse side of this page. ***To be completed by the Tandem customer.***

Subscription information:

_____
COMPANY

_____
NAME

_____
JOB TITLE

_____
DIVISION

_____
ADDRESS

_____

_____

_____
COUNTRY

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

Please turn page to complete form.

☐ New subscription

☐ Address change
Subscription number: _____
(Your subscription number is in the upper right corner of the mailing label.)

Back order requests:

☐ Vol. 1, No. 1, February 1985
☐ Vol. 1, No. 2, June 1985
☐ Vol. 2, No. 1, February 1986
☐ Vol. 2, No. 2, June 1986
☐ Vol. 2, No. 3, December 1986
☐ Vol. 3, No. 1, March 1987
☐ Vol. 3, No. 2, August 1987
☐ Vol. 4, No. 1, February 1988
☐ Vol. 4, No. 2, July 1988
☐ Vol. 4, No. 3, October 1988
☐ Vol. 5, No. 1, April 1989
☐ Vol. 5, No. 2, September 1989
☐ Vol. 6, No. 1, March 1990

Your Tandem representative can order these for you through COURIER (see 2 below).

Part B. ***To be completed by the Tandem representative.***

Please complete this portion of the form to approve the above request. Incomplete requests will be returned for resubmittal.

_____
NAME DEPARTMENT NUMBER

_____
TITLE

_____
LOC TELEPHONE NUMBER

_____
CUSTOMER NUMBER SYSTEM NUMBER

_____
SIGNATURE

1. For subscription processing only, send form to:

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

2. For requesting back orders for customers, use COURIER. The menu sequence is: Marketing Information, Literature Orders, Tandem Systems Review. The COURIER form allows the literature order to be sent directly to your customer's address.

   ☐ Date of COURIER order submittal: _____

This page must be completed by the Tandem customer.

1. Which of the following best describes your title? (check only one)

1 ☐ President/CEO	2 ☐ Director/VP info services	3 ☐ MIS/DP manager
4 ☐ Software develop manager	5 ☐ Programmer/Analyst	6 ☐ System operator
7 ☐ End-user	8 ☐ Other_____	

2. What is your (or your company's) association with Tandem? (check all that apply)

9 ☐ Tandem customer	10 ☐ Tandem employee	11 ☐ Third-party vendor
12 ☐ Consultant	13 ☐ Other_____	

3. What is company's primary operation? (check only one)

14 ☐ Education	15 ☐ Financial/Bank/Insurance	16 ☐ Government/Military
17 ☐ Healthcare/Medical	18 ☐ Manufacturing	19 ☐ Retail/Wholesale
20 ☐ Research/Library	21 ☐ Telecommunications	22 ☐ Transportation
23 ☐ Utilities	24 ☐ Reseller/Distributor	25 ☐ Consultant
26 ☐ Other_____		

4. How many employees does your company employ?

27 ☐ 24 or less	28 ☐ 25-99	29 ☐ 100-499
30 ☐ 500-999	31 ☐ 1000-4999	32 ☐ 5000 or more

5. Which applications run on your systems? (check all that apply)

33 ☐ Database management	34 ☐ Accounting	35 ☐ Electronic mail
36 ☐ PC/mainframe access	37 ☐ Word processing/Publish	38 ☐ Software development
39 ☐ Scientific/Engineering	40 ☐ LANs/Networking	
41 ☐ Other_____		

6. Approximately how many dollars of Tandem-based products and services do you anticipate your company will purchase in 1991?

42 ☐ $50,000 or less	43 ☐ $50,000-150,000	44 ☐ $150,000-500,000
45 ☐ $500,000-1M	46 ☐ $1M-5M	47 ☐ More than $5M

7. Are you planning on purchasing Tandem third-party products in the coming year?

48 ☐ Yes	49 ☐ No

8. In what areas would you like to see more articles? (check all that apply)

50 ☐ Guardian/Systems	51 ☐ Database management	52 ☐ Data comm/Networking
53 ☐ Tools/Utilities	54 ☐ Languages	55 ☐ UNIX
56 ☐ Industry applications	57 ☐ Third-party products	
58 ☐ Other_____		

TANDEM SYSTEMS REVIEW CUSTOMER SURVEY

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

1. How useful is each article in this issue?

An Overview of NonStop SQL Release 2
01 ☐ Indispensible 02 ☐ Very 03 ☐ Somewhat 04 ☐ Not at all

Performance Benefits of Parallel Query Execution and Mixed Workload Support in NonStop SQL Release 2
05 ☐ Indispensible 06 ☐ Very 07 ☐ Somewhat 08 ☐ Not at all

The NonStop SQL Release 2 Benchmark
09 ☐ Indispensible 10 ☐ Very 11 ☐ Somewhat 12 ☐ Not at all

Parallelism in NonStop SQL Release 2
13 ☐ Indispensible 14 ☐ Very 15 ☐ Somewhat 16 ☐ Not at all

Online Reorganization of Key-Sequenced Tables and Files
17 ☐ Indispensible 18 ☐ Very 19 ☐ Somewhat 20 ☐ Not at all

The Outer Join in NonStop SQL
21 ☐ Indispensible 22 ☐ Very 23 ☐ Somewhat 24 ☐ Not at all

Gateways to NonStop SQL
25 ☐ Indispensible 26 ☐ Very 27 ☐ Somewhat 28 ☐ Not at all

Batch Processing in Online Enterprise Computing
29 ☐ Indispensible 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 ☐ Other_____

3. Your title or position:

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

4. Your association with Tandem:

46 ☐ Tandem customer 47 ☐ Tandem employee 48 ☐ Third-party vendor 49 ☐ Consultant
50 ☐ Other_____

5. Comments

_____

_____

_____
NAME

_____
COMPANY NAME

_____
ADDRESS

▶ FOLD

▶ FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

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

POSTAGE WILL BE PAID BY ADDRESSEE

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

▶ FOLD

▶ FOLD

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

MARC BRANDIFINO
LOC NUM 50-00
NEW YORK NY DISTRICT

Part No. 46987

400109 10/90 Printed in USA