

# Born To Be Parallel

**Why Parallel Origins Give Teradata  
Database an Enduring Performance Edge**

By:

Carrie Ballinger

Senior Technical  
Consultant

Teradata Corporation

**TERADATA**  
*Raising Intelligence*

# Born To Be Parallel

## Table of Contents

<i>Executive Summary</i>	2
<i>Introduction</i>	3
<i>Query Parallelism</i>	3
<i>Data Placement for Parallel Performance</i>	7
<i>Intelligent Use of the Interconnect</i>	10
<i>Parallel-Aware Optimizer</i>	11
<i>Synchronization of Parallel Activity</i>	13
<i>Conclusion</i>	13

## **Executive Summary**

This paper reveals some of the techniques architected into the Teradata® Database that have allowed parallelism in its fullest form to blossom. Being born for parallelism is one of the most significant factors in establishing Teradata Corporation's dominance and success in the data warehouse market today, and we'd like you to understand why.

# Born To Be Parallel

## Introduction

Parallel processing has become necessary as data warehouse demand continues to expand to higher volumes, greater numbers of users, and more applications. Parallelism is everywhere. It has been added underneath, layered on top, and re-engineered into almost all existing database management systems, and in the process, has left some in the user community confused about what it means to be a parallel database. Teradata Database was the first, the original, product consciously designed from the base up to be a parallel system for decision support.

## Query Parallelism

### What is Query Parallelism?

Executing a single SQL statement in parallel means breaking the request into smaller components and having all components worked on at the same time, with a single answer delivered. Parallel execution can incorporate all or part of the operations within a query and can significantly reduce the response time of an SQL statement, particularly if the query reads and analyzes a large amount of data.

With a “Just say no!” attitude toward single-threaded operations, Teradata Database designers parallelized everything, from the entry of SQL statements to the smallest detail of their execution. The Teradata Database’s entire foundation

was constructed around the idea of giving each component in the system many sister-like counterparts. Not knowing where the future bottlenecks might spring up, developers weeded out all possible single points of control, and effectively eliminated the conditions that breed gridlock in a system before they become a problem.

Limitless interconnect pathways, optimizers, host channel connections, gateways, and units of parallelism can be defined in Teradata Database. This increases flexibility and control over performance that is crucial to today’s large-scale decision support.

### Teradata Database’s Unit of Parallelism

Teradata Database’s basic unit of parallelism is the AMP. From system configuration time forward, all queries, data loads, backups, index builds, in fact everything that happens in a Teradata system, is shared across the pre-defined number of AMPs. The parallelism is total, predictable, and stable.

In Teradata V1, these AMPs were physical uniprocessors. With Teradata V2 they became virtual AMPs with many co-existing on a single SMP node as a collection of UNIX® processes and many SMP nodes working together to form a single massively parallel processing (MPP) system.

### Teradata Database’s Dimensions of Query Parallelism

While the AMP is the fundamental unit of apportionment and delivers basic query parallelism to all work in the system, there are two additional parallel dimensions woven into the Teradata Database specifically for query performance. These are referred to here as Within-a-Step parallelism and Multi-Step parallelism. A description of all three dimensions of parallelism that Teradata Database applies to a query follows:

#### Query Parallelism

Query parallelism is enabled in Teradata Database by hash-partitioning the data across all the AMPs defined in the system. Hash partitioning is discussed in the next section. An AMP provides all the database services on its allocation of data blocks. Before the database is loaded, the system is configured to support a given number of these AMPs, anywhere from two to 20 per node, depending on the power of the underlying hardware. All relational operations, such as table scans, index scans, projections, selections, joins, aggregations, and sorts, execute in parallel across all the AMPs simultaneously and unconditionally. Each operation is performed on an AMP’s data independent of the data associated with the other AMPs.

# Born To Be Parallel

## *Within-a-Step Parallelism*

A second dimension of parallelism that will naturally unfold during query execution is an overlapping of selected database operations referred to here as within-a-step parallelism. The optimizer splits an SQL query into a small number of high-level database operations called steps, and dispatches these distinct steps to the AMPs in the system for execution.

A step is often a large chunk of work. It can be simple, such as scan a table and return the result, or complex, such as scan two tables with row qualifications, join the tables, redistribute the join result on specified columns, sort the redistributed rows and place the result in an intermediate table. Within each of these potentially large chunks of work known as steps, multiple relational operations are processed in parallel by pipelining. While a table scan is taking place, rows selected can be pipelined into a join process. (See Figure 1.) Pipelining is the ability to begin a task before its predecessor task has completed. Pipelining will take place whenever possible within each distinct step.

This dynamic execution technique, in which a second operation jumps off of a first one to perform portions of the step in parallel, is a key to increasing the basic query parallelism. The relational-operator mix of a step is carefully chosen by Teradata Database to avoid stalls within the pipeline.

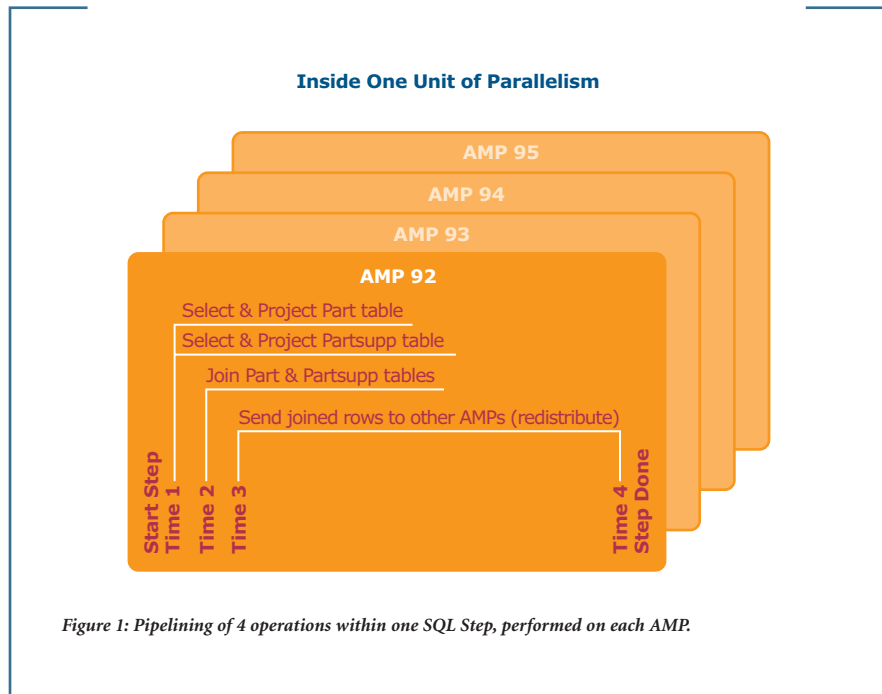


Figure 1: Pipelining of 4 operations within one SQL Step, performed on each AMP.

## *Multi-Step Parallelism*

Multi-step parallelism, sometimes referred to as bushy parallelism, is an added level of parallel activity unique to Teradata Database. Multi-step parallelism is enabled by executing multiple steps of a query simultaneously across all units of parallelism in the system. One or more processes are invoked for each step on each AMP to perform the actual database operation. Multiple steps for the same query can execute at the same time to the extent that they're not dependent on results of previous steps.

Figure 2 shows a system configured with four AMPs, and a query that has been optimized into seven steps. Step 2.2

demonstrates within-a-step parallelism (as does step 1.2), where two different tables (Lineitem and Order) are scanned and joined together (three operations), all three pipelined within one step. Steps 1.1 and 1.2, as well as 2.1 and 2.2, demonstrate multi-step parallelism, as two distinct steps are made to execute at the same time, within each AMP.

## **Even More Parallel Possibilities**

In addition to the three dimensions of parallelism shown in Figure 2, Teradata Database offers an SQL extension called a Multi-Statement Request that allows several distinct SQL statements to be

# Born To Be Parallel

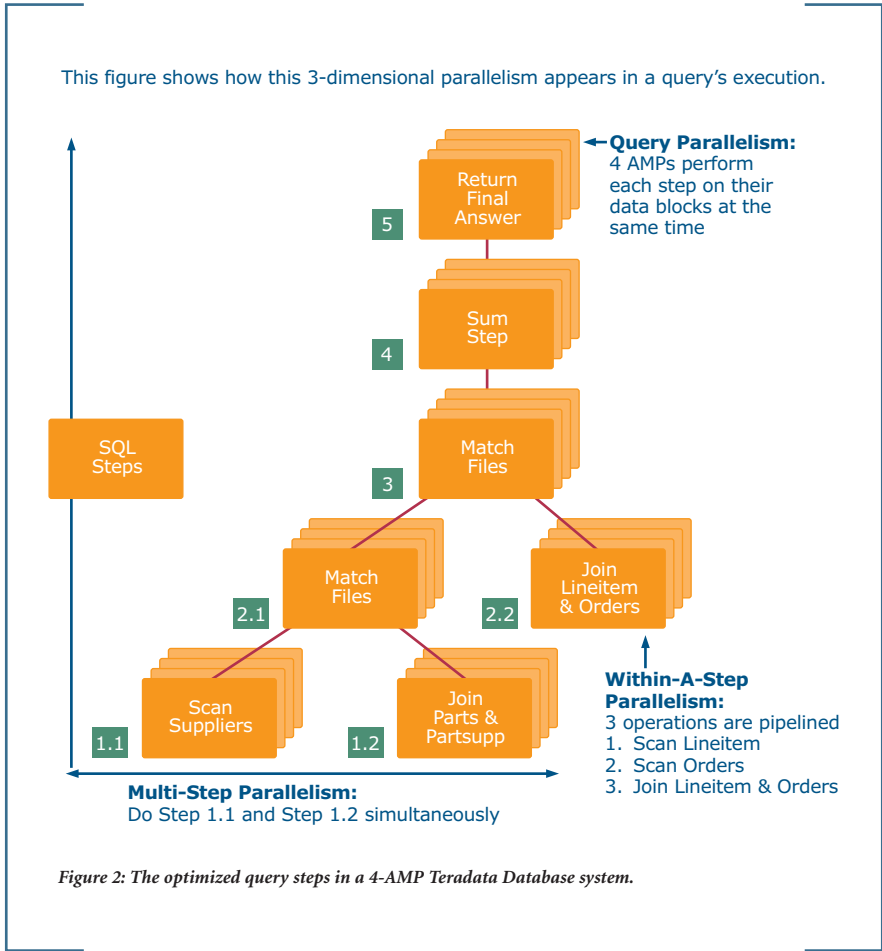


Figure 2: The optimized query steps in a 4-AMP Teradata Database system.

bundled together and sent to the optimizer as if they were one. Teradata Database will attempt to execute these SQL statements in parallel. When this feature is used, any sub-expressions that the different SQL statements have in common will be executed once, and the results shared among them. Known as common sub-expression elimination,

this means that if six select statements were bundled together, and all contained the same sub-query, that sub-query would only be executed once. Even though these SQL statements are executed in an inter-dependent, overlapping fashion, each query in a multi-statement request will return its own distinct answer set.

This multifaceted parallelism isn't easy to choreograph unless it's planned for in the early stages of product evolution. Most other database products fall far short of this depth and thoroughness.

## I'll Have My Parallelism Straight Up

Teradata Database has built-in self-awareness techniques. For example, knowledge of the number of parallel units and the processing power behind each of them, as well as being able to predict how each will behave, gives Teradata Database many advantages in performance as well as operational simplicity. Teradata Database operates as a single, integrated unit, with all parallel units intrinsically aware of the entire system. While other databases that offer parallelism as an option may incur the overhead of having a special coordinator task pull the parallel activities together, Teradata Database is free from such requirements.

Awareness of being part of a team means that fewer internal questions need asking or answering. Teradata Database's persistent parallelism eliminates conversations that other DBMSs must have with themselves and copies of them to determine how to divide the work for parallelization. And even when such databases have divided the work up for the first step in a query plan, more conversations must take place to determine how to re-divide the work for potentially different degrees of parallelism available for executing the next step.

# Born To Be Parallel

Imagine a relay race in which the runners didn't know ahead of time who or how many runners would make up each team. Hand signals, shoulder shrugging, and yelling might be needed to accomplish smooth and equally spaced hand-offs during the race, taking energy that could and should be directed at performance. Database systems communicate internally with messages, and the more unnatural and erratic the work being attempted, the more time and energy the system must spend generating, sending, and reading these messages.

## Parallel Systems Don't Have to Degrade as Concurrency Grows

Because parallel systems have the potential to give a single user a high percentage of total system resources, they also have proven to be very vulnerable in controlling allocation of the same resources when they must be shared.

Teradata Database is an exception to this because developers were given the opportunity initially to plant techniques deep in the product's base that maximize throughput while multiple dimensions of parallel activity are made available for each user on the system. Some other products did not have that luxury because they have approached parallelization after product maturity.

Operating near the resource limits without exhausting any of them and without causing the system to thrash requires effective work flow management. Teradata Database provides work flow management by monitoring the utilization of critical resources, such as CPU, memory, and interconnect. If any of these reach a threshold value, it triggers the throttling of message delivery, allowing work already underway to complete. Teradata Database performs this throttling at the lowest possible level, i.e., AMP, and only to the AMPs where it's needed. This internal watchfulness is automatic to the product and does not depend on a large staff of DBAs. It does not prevent new queries from executing or reduce their parallelism.

## What is Thrashing?

In a virtual memory system, physical memory is filled with pages of active processes. If too many active processes need access to too many pages for the available physical memory, paging between physical memory and disks takes place. This overhead, commonly referred to as thrashing, is non-productive work and can reduce the overall system throughput.

Since messages are the unit of work delivery, Teradata Database has the ability to throttle messages and control the amount and type of work performed in the system if a congested state is building up. Informa-

tion held in the message headers that come from the optimizer contain the user priority of the originating request, the amount of work that this message represents in a rough sense. The message subsystem, in conjunction with the interconnect software, allows the throttling information to be passed upstream. This controls message generation at the origin and cools the demand immediately.

Scheduling resource usage combined with a system of task priorities is also used by Teradata Database to control the flow of parallel activity when multiple users are on the system. A single AMP may support hundreds of queued and up to 80 active tasks at any one time, from either the same request or many different requests. It is Teradata Database Priority Scheduler that recognizes order, importance, decides who's up next, and passes control of resources around appropriately.

Priority Scheduler was engineered to ensure the efficient completion of all work in the system, no matter how long or short an individual query may run. The same priority scheduler, designed into the base Teradata Database product and used internally by the software, can be used by the DBA to manage how and to what degree platform resources are used by groups of Teradata users.

# Born To Be Parallel

## Data Placement for Parallel Performance

When conceptualizing how Teradata Database's parallel units could optimally process data within an MPP system, it became clear to the product's designers that physical data partitioning could either help or hinder that parallel advantage. From an MPP perspective, many of the available partitioning choices were no longer useful. One of the most common problems encountered with these traditional data placement schemes was that while the data might be balanced across disks or nodes, this did not guarantee that the actual database work would be balanced. Equal processing effort across parallel units became a key criterion as Teradata Database evolved.

Teradata Database solved the balance-of-work issue by permanently assigning data rows to AMPs and using an advanced hash algorithm to allocate data. This decision to marry data to the parallel units led to the selection of a single partitioning scheme that supports an even distribution of data no matter what the patterns of growth or access, while formalizing an equal sharing of the workload – hash partitioning.

## Teradata Database's Hash Partitioning

Data entering a Teradata Database are processed through a sophisticated hashing algorithm and automatically distributed across all AMPs in the system. In addition to being a distribution technique, this hash approach serves as an indexing strategy. This significantly reduces the amount of DBA work normally required to set up direct access. To define a Teradata Database, the DBA simply chooses a column or set of columns as the primary index for each table. The value contained in these indexed columns is used to determine the AMP, which owns the data, as well as a logical storage location within the AMP's associated disk space, all without performing a separate CREATE INDEX operation.

To retrieve a row, the primary index value is again passed to the hash algorithm, which generates the two hash values, AMP and Hash-ID. These values are used to immediately determine which AMP owns the row and where the data are stored. There are several distinct advantages in decision support applications that flow from this use of hash partitioning.

### *No Key Sequence*

One dramatic side-effect of using the hashing algorithm as an indexing mechanism is the absence of a user-defined

order. Often, database systems use some variant of balanced trees (B-Trees) for indexing, which are constructed based on an alphabetic sequence specified by the user. As new data enters the system and random entries are added to the B-Tree indices, these structures will gradually become unordered and require overflow techniques. Either the index quickly becomes more expensive to use or the entire structure must be made unavailable for reorganization. Hash algorithms do not care about user-defined alphabetic order, they don't use secondary structures that require reorganization, and there is never a need to sort the data before loading or inserting.

### *Ease of Joining*

Hash partitioning of primary index values allows rows from different tables with high affinities to be placed on the same node. Columns that constitute the join constraint between two tables as the primary index of both tables will be designated so associated rows-to-be-joined will reside on the same AMP on the same node. Since two rows can only be joined if they reside in the memory of one of the nodes, this co-location reduces the inter-connect traffic that cross-node joins necessitate, thus improving query times, and freeing the interconnect for other work. (See Figure 3.)

# Born To Be Parallel

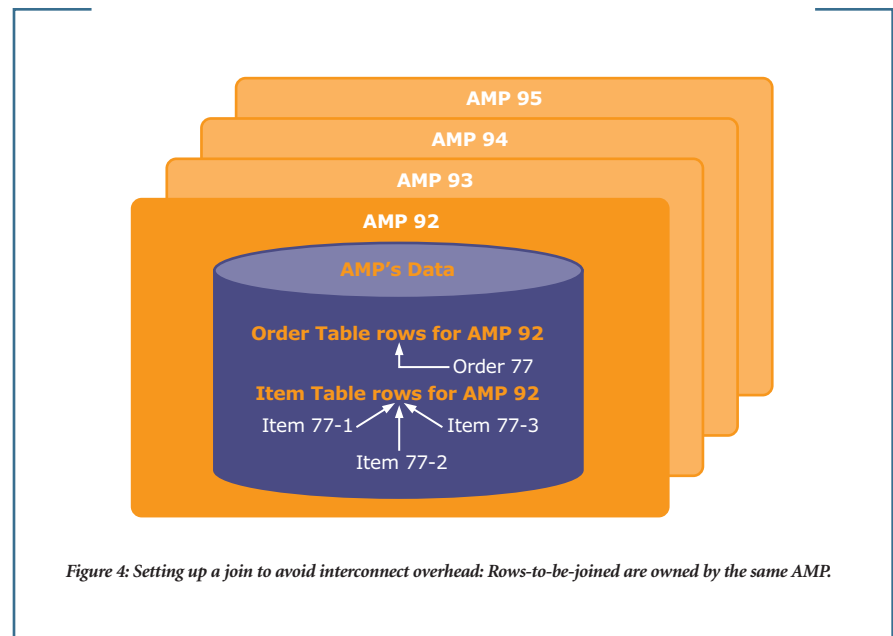
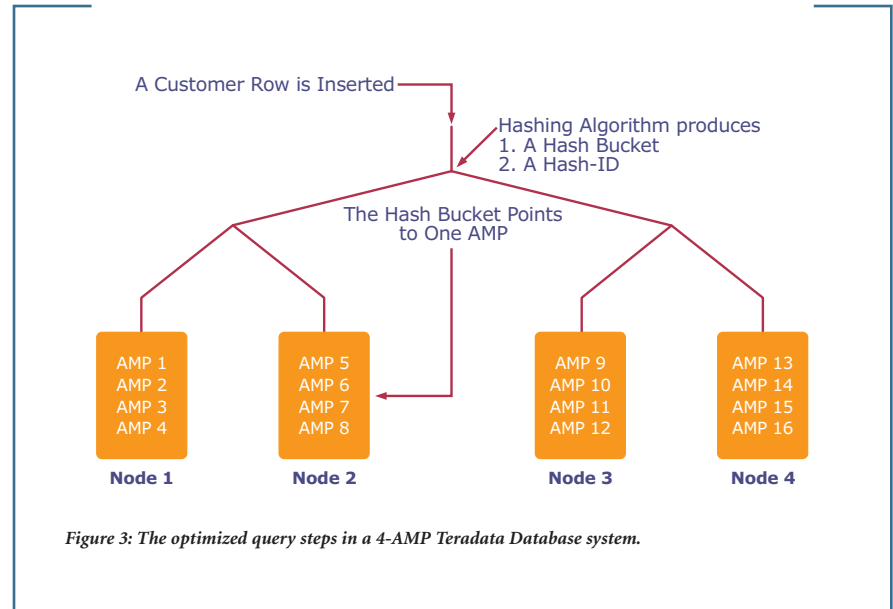
## Simplicity of Setup

The only effort hashed data placement requires is the selection of the columns that are in the primary index of the table. From that point on, the process of creating and maintaining the physical placement of data is completely automated. No files need to be allocated, sized, or named. No unload-reload activity is ever required.

Teradata Database's hash partitioning is the only approach that offers balanced processing and balanced data placement, while minimizing interconnect traffic and freeing the DBA from time-consuming definitions and reorganizations. The hashing algorithm has been deepened and improved over time, making for even smoother distribution of data. (See Figure 4.)

## Teradata's Partitioned Primary Index Extends Hashed Data Placement

All of the benefits of hashed data placement and simplicity of setup apply to how Teradata has incorporated partitioning into its product offering. Teradata's partitioned primary index (PPI) feature allows tables to be partitioned on columns of interest, while retaining the traditional use of the primary index for even data distribution, AMP-local joins, and efficient access when the primary index values are specified in the query.



# Born To Be Parallel

- **Step 1:** Rows are automatically hashed to an AMP based on Primary Index Row Hash value.
- **Step 2:** The row's partitioning key determines the partition on that AMP where the row will be placed.
- **Step 3:** Within the partition, rows are logically stored in Row Hash sequence.

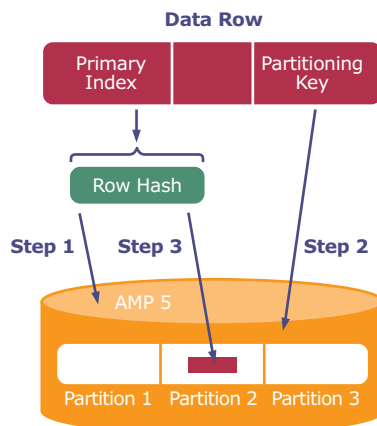


Figure 5: PPI approach to sharing AMPs.

In Teradata, the primary index value of a row determines the AMP that will own the row, whether the primary index is partitioned or not. But if the table has had its primary index defined as partitioned, then before the row is physically stored, it will first be placed within the correct partition on that AMP. Within that partition the row will be stored in hash sequence.

Because the PPI approach spreads each partition across all AMPs, the work of processing a single popular partition, such as this week's sales data, will be shared by all AMPs in the system. (See Figure 5.)

## Why Traditional Data Placement Schemes Conflict With MPP Goals

The traditional data placement choices listed below fall short of parallel processing requirements because they're unable to provide a balanced workload, they have inherent data accessibility weaknesses, and they tend to flood the interconnect.

### *Data Set Partitioning Leads to Hot Spots*

The DBA, who must batch up data into data sets as it arrives into the system, has total control over where the data physically reside when using this manual approach to data placement. It is possible to accomplish a balanced data load with data set partitioning; however, it's impossible to establish a balanced processing load.

Because the DBA is manually assigning the data one load file at a time, a small subset of the files will contain the most recent data.

Because a significant amount of processing in a data warehouse is comparing the most recent data to some earlier time period, the vast majority of users will be attempting to access the newer data sets at the same time.

In MPP or clustered environments, the DBMS has no way of telling if rows from two tables, which have been partitioned by data set, that are to be joined are located on the same processing node. Any join activity using this approach is likely to pass enormous amounts of data across the interconnect, reducing overall system throughput.

### *Range Partitioning Contributes to Imbalance*

Traditional range partitioning is differentiated from Teradata Database's partitioned primary index capability because range partitioning places rows on disk solely on the basis of the partitioning key value. Range partitioning is attractive for some databases when there is repetitive access based on the same constraint, because the DBA can partition the table into multiple collections of rows based on those particular values. But range partitioning offers several difficult challenges for a parallel database and its DBA.

# Born To Be Parallel

First, business data does not arrive politely balanced, so a thorough analysis of the data distribution must be undertaken to start setting up the partitions across nodes. Second, if it's desirable to locate commonly joined rows from different tables onto the same nodes to reduce interconnect traffic, that co-location will be more difficult. This is not trivial when tables are partitioned on different columns. Third, a method of balancing the data for each table across multiple nodes must be achieved. Fourth, since data demographics change over time, the partitioning scheme must be regularly revisited, recalculated, and data re-juggled.

Even assuming this data balancing effort is successful, processing may remain unbalanced. Most realistic range partitioning strategies are in some way a function of time. Even the seemingly random selection of item color for a retailer will vary greatly by season. Some of the processing nodes will always contain significantly more current data than others. As with data set partitioning, this will tend towards unbalanced processing.

## *Schema Partitioning Can Overwork the Interconnect*

Schema partitioning is the assignment of data to specific physical processors or nodes and has proven useful when there is a need to restrict portions of the hardware to handling certain groups of tables, or

schemas. While this is generally viewed as a means of increasing performance for specific tables, and can be applied in useful ways on an SMP-only database, it has not proven hugely successful in the MPP world. In most cases, to join data from two schema-partitioned tables, all rows must be redistributed across the interconnect for each query. This data movement may significantly impact performance for the affected queries, and the increased interconnect traffic may also impact overall system throughput. Further, node imbalances are aggravated if this option is used on anything but very small tables.

## **Logical Addressing of Data Offers Additional Advantages**

While Teradata Database uses hashing to determine where a data row will be placed, that data row is never given a permanent physical address in the database. Taking an unusual approach to how data are organized, Teradata Database relies on logical addressing. Data in a Teradata system are stored in flexibly-sized data blocks that are loosely anchored together via logical addressing and are able to be floated from one physical disk location to another as needed.

Most databases use fixed-size pages to store rows. The physical address of such rows can only be changed by removing and re-inserting the rows, such as happens during a database reorg. In

these fixed page systems, space is usually allocated before it is required, and when a fixed page is full, rows that need to be inserted at that location must use overflow pages.

Overflow pages, which are unknown to Teradata Database, cause performance degradation as a table matures over time because they add additional I/O to accessing and inserting rows.

Because Teradata Database rows use logical addressing and are not tied to a physical location, they can be moved or re-blocked without impacting on-going work in the database. This flexibility to consolidate or expand data blocks anytime allows Teradata Database to do many space-related housekeeping tasks in the background and avoid table unloads and reloads common to fixed page data-bases. This advantage increases database availability and translates to less DBA support.

## **AMP Ownership of Data**

While balanced processing was a central theme during Teradata Database's evolution, another goal emerged — ownership of or responsibility for the individual data rows. Without dividing up or parallelizing data integrity responsibility, activities such as locking must be handled in one central place or in an unnatural hand-off

# Born To Be Parallel

fashion, which could consume large amounts of CPU and interconnect resources and create a bottleneck. Parallel responsibility for data, alongside parallel processing, emerged together as simple twin births that followed the hash partitioning decision.

With some other database systems, any process can take temporary control of an object but some central process must control locking access to ensure data integrity. But the problem with the central locking mechanism is two-fold. First, the locking strategy itself may become a bottleneck unless locking can be completely suspended. Second, a task that wishes to use a row must retrieve a lock. When it is done, it must pass control, usually in the form of the row itself, to the next process wishing access. While this is quite feasible in an SMP environment where shared memory pointers are available, it will quickly saturate any interconnect on the market.

An extreme example of this is one reason a shared disk approach will never be able to offer high-end scalability. Systems that rely on shared-disk will always use some dynamic algorithm to determine which instance of the database accesses which data rows. The accessing instance may or may not be the same as the instance which owns, and therefore controls the locks, for the row. Bottlenecks will develop on all but lightly used systems. With Teradata Database, each row is owned by exactly

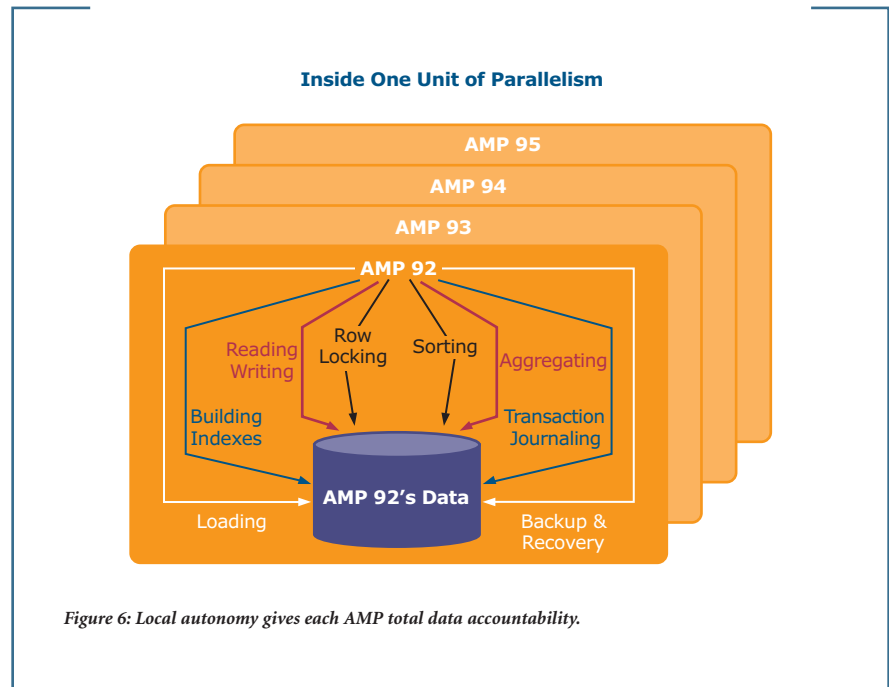


Figure 6: Local autonomy gives each AMP total data accountability.

one AMP, and this AMP is the only one that can create, read, update, or lock that data. (See Figure 6.) All transaction logging is under the local control of the AMP. A local lock manager functions independently in each AMP, and maintains its own set of locks. Because of this, logging and locking are performed locally across all AMPs in the configuration in parallel, control of data is consistent, and interconnect traffic can be reduced.

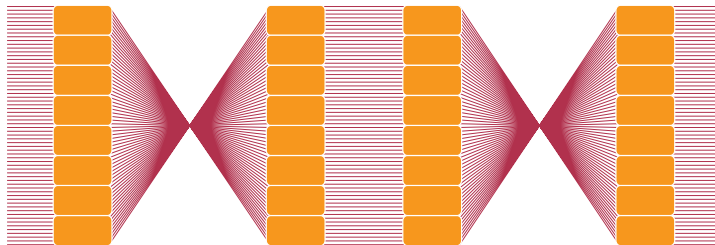
## Intelligent Use of the Interconnect

Because of the number of tasks a parallel operation can generate, poor use of any resource will quickly fan out, leading to

scalability issues for a parallel database. If taken for granted, no resource can become more tenuous in an MPP system than the communication link between nodes, the interconnect. Teradata Database architects had the advantage of knowing they were designing an MPP-database system, and therefore, were able to develop the database software with the issues of inter-nodal communication in the front of their minds.

The BYNET™ used by Teradata Database is a fully scalable, folded Banyan switching network. (See Figure 7.) Teradata Database attempts to use as much BYNET functionality as possible. When combined

# Born To Be Parallel



One of the two folded banyan BYNET networks for a 64-Node Teradata configuration. Each node has many BYNET paths to each other node in the system.

*Figure 7: The optimized query steps in a 4-AMP Teradata system*

with Teradata Database, the BYNET delivers messages, moves data, collects results, and coordinates work among AMPs in the system.

Just as telecommuting frees up traffic lanes by keeping employees off congested roads, Teradata Database minimizes interconnect traffic by encouraging stay-at-home, same-node activity where possible. AMP-based object ownership keeps activities such as locking local to the node. Hash partitioning that supports co-location of to-be-joined rows reduces data transporting for a join. All aggregations are ground down to the smallest possible set of sub-totals at the local (AMP) level first. And even when BYNET activity is required, use of dynamic BYNET groups keeps the number of AMPs that must exchange messages down to the

bare minimum. BYNET-driven semaphores, described in the final section, provide a shortcut method for coordinating parallel query processing across the interconnect. Teradata Database's query optimizer knows and takes into account the cost of sending a table's rows across the BYNET, and factors this expense into the final join plan that it constructs.

While the most important bandwidth preservation method used by Teradata Database is minimizing data movement, there are some other useful aspects of the interconnect that aid performance.

### Point-to-Point Communication

Teradata Database will always use broadcasts in cases where all AMPs in the system require the same information, such as the duplication of the rows of a

table, or sending a dispatcher message for an all-AMP operation, such as applying a table-level lock. But the cheaper point-to-point messaging is considered when a subset of AMPs is required. The BYNET point-to-point communication is similar to a standard phone call over the public telephone network. These monocast circuits connect one sender node to one receiver node. Generally known as a non-collision architecture, this approach minimizes the total volume of data the interconnect has to handle.

### General Purpose Transports

As discussed above, many parallel DBMSs are built using available components for SMP systems. Some manufacture parallelism by creating separate copies of the DBMS. Then communications using available protocols, such as TCP/IP, become necessary. Other products use TCP/IP for all communications between instances.

The problem with this approach is that general purpose transports, such as TCP/IP, service many needs and have never been fine-tuned for parallel query processing. The messages are encased in packets, with headers, messages, and footers. If a message is too large for the protocol, it will be broken up into multiple packets. One of the reasons LANs never carry close to their theoretical limits is that large amounts of interconnect bandwidth available for messages is consumed by packet overhead.

# Born To Be Parallel

## Bandwidth Conservation

Teradata Database sees the interconnect as more than a postman delivering mail. It also understands and uses this network as a group coordinator. For example, Teradata Database uses a BYNET back channel to perform what amounts to a two-phase commit on each message transmitted.

A packet-switched network would be required to perform a minimum of three separate serial transmissions to accomplish the same goal. Because the BYNET was designed specifically for parallel query activity, it is more lightweight and efficient than the packet networks.

Because the interconnect is often an afterthought when MPP architecture evolves from an SMP base, it has become an easily abused resource. Parallel DBMSs can quickly over-extend this unfamiliar resource. It is generally not a single shortcoming or product bug that floods an interconnect, but a poor data partitioning scheme, a multitude of excessive messages, and unnecessary data shipping.

## Parallel-Aware Optimizer

The optimizer is the intricate piece of database software that functions similar to a travel agent, booking the most efficient travel path through the many joins, selections, and aggregations that a single query may require. If a candidate for public office wants to make speeches

in ten cities across the U.S. and has only three days to cover all the campaign stops, the travel agent must be highly sensitive to time, sequence of stops, and distance traveled when creating the itinerary. The same sensitivity is required by a query optimizer.

Two factors determine how well an optimizer will do its job. First, the quality and depth of environmental knowledge that is available as input. Second, the sophistication and accuracy of the costing routines that allows an efficient plan to be produced. Optimizers created for databases where parallelism is an option are ill-equipped if asked to do double-duty in a parallelized version of the same database. Extensive code changes or optimizer rewrites will be required because the cost of most operations and choices that are available with parallelism will be different. Imagine on a road trip that you must choose between a four-lane highway that is ten miles long or a one-lane road that is six miles long. If you don't see and understand the highway's parallel abilities, you might choose the route that is fewer miles but which will take longer to drive.

DBAs that spend significant time in complex MPP data placement tuning exercises, in the hopes of ensuring future query performance, may only see a benefit for very simple queries. It doesn't matter how the data have been cut up, classified, and ordered, if the optimizer hasn't been

trained to look for and interpret data partitioning changes, then none of this work can influence the final query plan.

In evaluating the quickest way to perform a six-table join, an optimizer must know about how long each operation that makes up the query will take so it can decide which two tables to join first, at what point to build the aggregation, and whether to apply selection criteria before, after, or during a join. An inefficient optimizer can cost high-volume decision support queries hours, not just seconds, of time.

## Teradata Database's Parallel Knowledge

Teradata Database's optimizer is grounded in the knowledge that a query may be executing on a massively parallel processing system. The optimizer is aware of the number of AMPs per node configured in the system, and understands that the amount of parallelism is the same for all operations. It has row count information and knows how many of a table's rows each parallel unit will be managing so I/O can be costed more effectively. It also considers the impact of the type and number of CPUs on the node. The optimizer calculates an AMP to CPU ratio that compares the number of parallel units on a node against the available CPU power. It uses this information to build the most efficient query plan. (See Figure 8.)

# Born To Be Parallel

*An optimizer must consider an increasing number of join sequences as the number of tables in the query grows*

For a 4-table join – **24**

For a 6-table join – **720**

For a 8-table join – **40,320**

For a 10-table join – **3,628,800**

Figure 8.

## Using the AMP-to-CPU Ratio

One result of this awareness is that CPU-intensive query operations, when there is a choice, tend to be chosen less frequently when the optimizer senses that proportionally less CPU power is available compared to number of parallel units. Consider a product join, which is a many-to-many activity in which the DBMS tries to join every row of one table to every row of a second table. This type of activity, which tends to be high on comparisons, usually requires a larger proportion of CPU resources than other join choices. Teradata Database's optimizer will choose a product join less frequently in a configuration that has less powerful or fewer CPUs feeding into the AMP-to-CPU ratio.

In another example, the optimizer uses its knowledge of the number of AMPs to decide whether or not to duplicate one table's rows to all other AMPs as one alternative in preparing for a join. Table

duplication will happen less frequently in a system with hundreds of AMPs compared to a system with tens of AMPs, because duplication is a broadcast activity from all nodes in the system to all other nodes in the system. This type of broadcast, while it does engage the BYNET, is frequently CPU-intensive, as all AMPs in the configuration are involved in a more extensive amount of sending and receiving effort compared to other join choices. The optimizer weighs the cost in CPU resources for the duplication activity, and knows that this cost will be relatively more expensive as the number of AMPs increases.

And finally, Teradata Database's hashed partitioning dovetails smoothly with the predictable number of parallel units. The even spread of data keeps the optimizer's estimates in tune with reality.

In simple queries at low volume, a born-to-be-parallel optimizer may not greatly out-distance the less intelligent models. Like a multiple choice test with only one choice listed for each question, when few decisions are required, the level of skill behind making the decision is less important. Today's powerful hardware can easily hide a multitude of software sins at the low end, but no amount of hardware can hide software failures as data volume and environmental complexity increase. With a changing, dynamic MPP application running against a large and growing volume of data, the intelligence and thoroughness of the optimizer will make or break the application.

## Synchronization of Parallel Activity

When the master plan includes a clear, well-defined vision, the master builder can blueprint efficient, tightly fit connections among all the components that evolve into the superstructure. Whether it's the chapters in a Pulitzer-prize winning novel or the instrumental voices of a classical symphony, creating harmony and counter-point among contributing parts is a formula for success.

Teradata Database didn't have a life before parallelism came along, so it has not inherited any biases or habits from some previous single server version of itself. Because of that, its file system, messaging sub-system, locking mechanisms, and optimizer all work together, fit together, and act together for the same goal – parallel query performance. Some of the architectural directions and techniques that promote and take advantage of this tight integration are discussed here.

## Coordination during Query Execution

A key winner from and contributor to this singular synchronization is query processing itself. As explained in the first section, Teradata Database breaks down database requests into independent steps when replying to SQL requests. The steps are dispatched across the BYNET from one of the Parsing Engines, one step (or several steps in the case of multi-step parallelism) at a time, to AMPs in the system. Only after the step is completed by all participating

# Born To Be Parallel

AMPs will be the next step (or steps) be dispatched. This ensures all parallel units in the database are working on the same piece of work at the same time. (See Figure 9.)

## Building on the Cooperation

Building on top of this natural teamwork are some additional features, such as synchronized scans, to increase overall system throughput. Traditionally, database systems have tried many ways to maximize the value of one piece of work. One example is the caching of disk data blocks to avoid re-reading them. Teradata Database improves the effectiveness of this caching for large table scans through the use of a feature called synchronized scans. Synchronized scans permit new table scans to begin at the current point of an already running scan of the same table. This results in the avoidance of I/O for the subsequent scans. Since these scans may not progress at the same rate, any task is free to initiate the next data read. The benefit of this feature increases as active users are added to the system.

Additionally, Teradata Database maximizes the value of each piece of work by reusing intermediate answer sets within the same query. The answer sets, known as spool files, may be the result of complex join processing that required hours to build. Reusing these intermediate results avoids redoing expensive work, such as

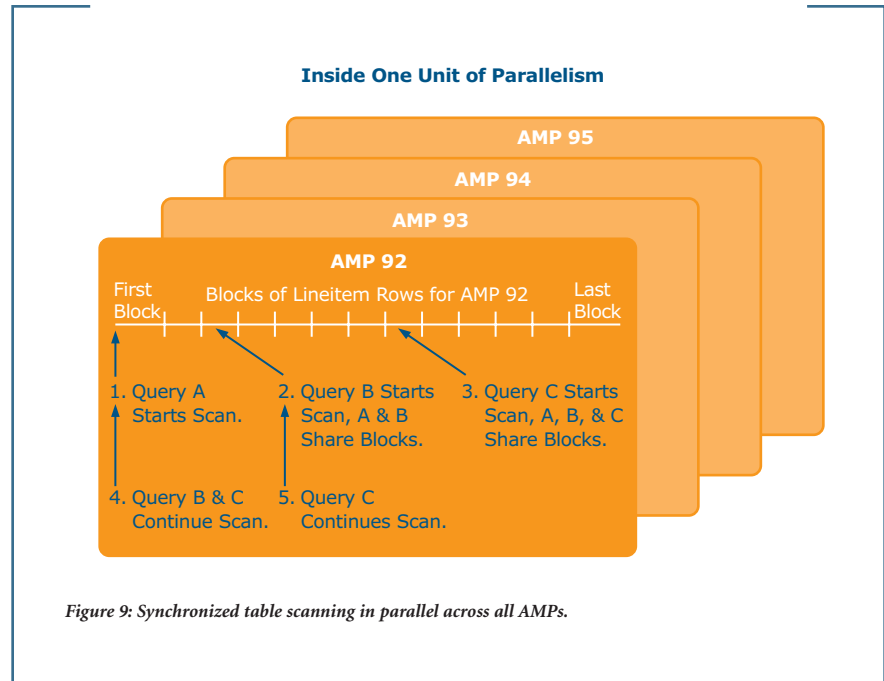


Figure 9: Synchronized table scanning in parallel across all AMPs.

scans, row re-distributions, and joins. Self-joins and correlated sub-queries are just two of the examples of where reusable spool files provide a performance boost.

## Examples of Global Synchronization

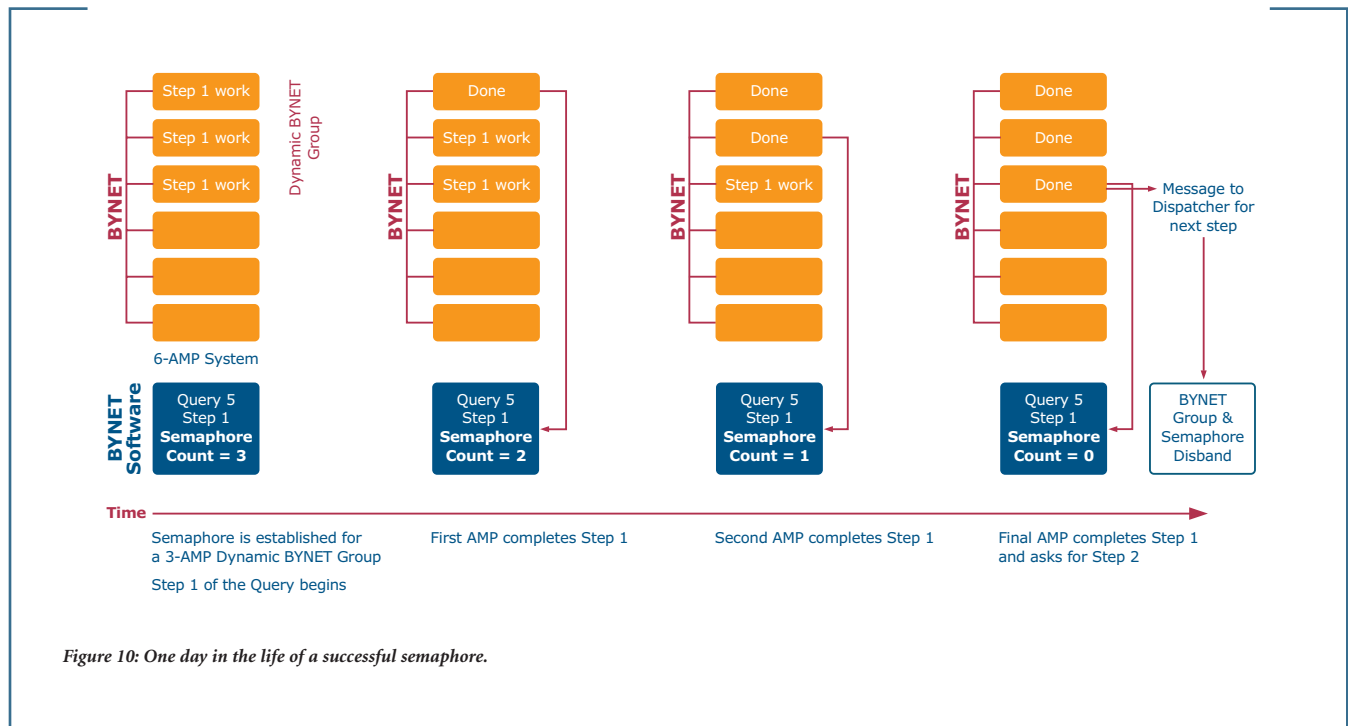
Two techniques provide a good example of how this integration and cooperation work: Dynamic BYNET groups and global semaphores minimize message flow within the database software by offering simpler alternatives to intense message-passing when parallel units require coordination. And even more significantly, these features of few words both show a healthy reliance

on the BYNET, demonstrating a melding of database and interconnect intelligence that has proven to be greater than the sum of its parts.

## Dynamic BYNET Groups

The Dynamic BYNET Group is simply an on-the-spot association of the AMPs that will be working on one specific query step. It is possible for many of these BYNET groups to exist at any point in time. When a query is optimized and the first step is ready to be dispatched, a message will be automatically sent across the BYNET, but directed only to the AMPs that are actually needed in doing that step's work. This may

# Born To Be Parallel



be all the AMPs in the system, or it may be a subset, or just one. Receipt of this step message causes the AMP to be automatically enrolled in the BYNET group without the database software having to initiate a separate communication.

Group AMP functionality is an optimizer opportunity that takes advantage of dynamic BYNET groups, to better service tactical queries. This feature eliminates some all-AMP optimizer steps and replaces them with a step that engages just a subset of the AMPs, if the subset is all that the query requires. This reduces the resources required for such a query, and frees up unneeded AMPs to service other work.

## Semaphores

Associated with the dynamic BYNET group are success and failure semaphores, which have monitoring and signaling functions. Each success semaphore contains a count that reflects the status of that BYNET group's activity. Semaphores are parallel infrastructure objects that are globally available because they live within the BYNET software. If you think of the BYNET as a continuously moving train, BYNET software is the easy-to-get-to train station that sits between the interconnect and the more distant database software. The semaphores' jobs are to signal when the first AMP in the group completes the optimizer step being worked on, and when the last AMP in the group completes the same step. The

success semaphore's count is reduced by one when an AMP reports in and will be reduced to zero when the last AMP completes.

When the final AMP completes its work, it sees that the semaphore is registering zero and it knows it is the last one done. Because it is the last participant to complete this step, this AMP sends a message to the dispatcher to send out the next optimized step for that query. This is the only actual message sent to the dispatcher concerning this step, whether the dynamic BYNET group is composed of three or 300 AMPs. (See Figure 10.)

# Born To Be Parallel

## MPP Abort Processing

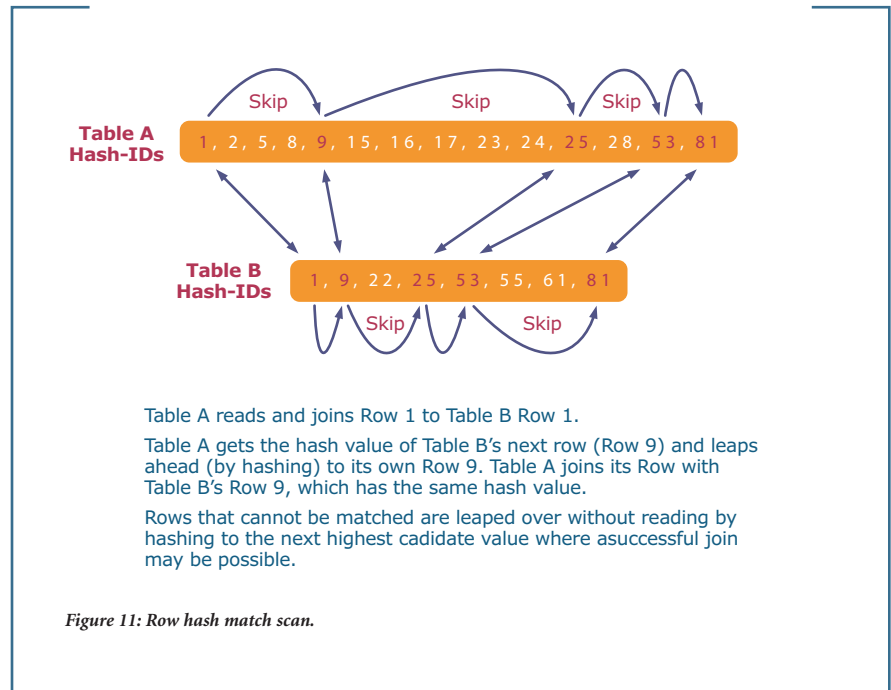
Global abort processing is particularly challenging in MPP systems. In Teradata Database this activity is simplified by using semaphores. Failure semaphores signal to related tasks within the same dynamic BYNET group to abort a given step should one AMP in the group experience a serious error condition, thus freeing up system-wide resources immediately.

In a decision support workload, it is possible for hours of time to be spent on a single step of a query. If one AMP experiences a fatal error and the other parallel units are deaf to it and continue their work, the throughput of the entire system will be compromised.

With global synchronization built into the infrastructure of the product, with the ability to recognize and orchestrate the parallel activity that is going on, and with no single points of control, it becomes increasingly straightforward and natural to streamline performance and handle growing volumes of users on Teradata Database.

## Expanded Use of Teradata Database's Hashing Functionality

A key benefit of architecting a product with a single focus in mind is that highly-efficient routines can not only be synchronized, but can be re-used in the base with great effect. Teradata Database's hashing functionality, in addition to being the foundation of physical data placement, provides a major boost when it comes to parallelizing and sharing



other work. This section will discuss how hashing builds performance into joins, aggregations, and system reconfigurations.

## Hashing Helps Balance Join Processing

Any time rows of data need to be brought together from different AMPs and nodes in the system, hashing is the tool relied upon to divide and balance the work. For certain types of joins, the merge join for example, the join column values are run through the hashing algorithm and the value of the hash bucket that comes out of that process tells the system which AMP will manage the actual join of those particular rows. That way all the AMPs in the system help perform these joins, each doing a subset of the work.

## Row Hash Match Scan

A Row Hash Match Scan may take place during merge join processing, adding significant performance to this common join. Rows to be joined from both tables are sorted in row-hash sequence. While scanning the two tables in parallel at the same time it is joining the rows, Teradata Database will use the hashing technique to allow the scan of one table to leap ahead in its scan to the hash-ID where the second table is already positioned. The example in Figure 10 illustrates the scan-leap-scan ability that can significantly shorten the time to implement the join. (See Figure 11.)

# Born To Be Parallel

A Large Aggregation is performed locally, then globally, across all AMPs

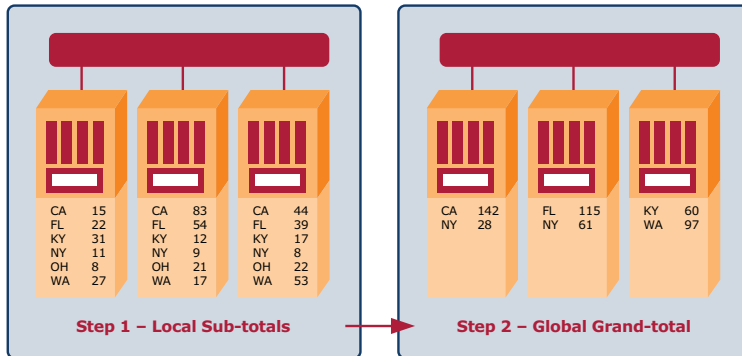


Figure 12: A count of the number of delinquent customers by state.

## Aggregate Processing is Shared across AMPs Using Hashing

When a large sum or aggregation is performed, the work is also shared across all AMPs, and the sharing is administered by means of hashing. In Figure 12, which shows an aggregation, each AMP performs a local aggregation on its own data as a first step. Then the values of the GROUP BY columns ('CA', 'FL', etc.) are hashed, and the hash bucket that results for each distinct value will point to the AMP in charge of building up the global aggregate for that one piece of the aggregate. For simplicity of illustration, each physical node shown below has only one AMP. In a real system, there may be anywhere from two to 20 AMPs per node, each performing its own part of the global grand total.

Reconfig: 1/3 of the rows from each node are relocated to the new node's AMPs

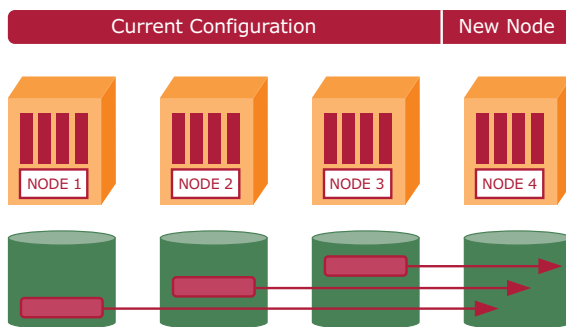


Figure 13: Reconfiguration utility.

## Hashing is the Foundation of the Reconfiguration Utility

A final important use of the hashing algorithm is during the expansion of the database by the addition of hardware. This is usually required when the data in the database have increased and additional nodes and AMPs are introduced into the system. (See Figure 13.) Teradata Database will not begin any work on the expanded system until the data have been reappor-tioned evenly across the old and the new AMPs together. Rather than unload the entire database, Teradata Database uses a reconfiguration utility to move a small percent of the rows from the current

# Born To Be Parallel

AMPs into the new AMPs. The percent of movement from each current AMP will depend on the percent of new AMPs being introduced to the system.

The reconfiguration utility automates its work by simply changing the system hash maps. These hash maps are the entities that reside on each node and tell the system which hash buckets are assigned to which AMPs. Some hash buckets are changed to point to the new, empty AMPs. Each AMP then reads each of its rows, but moves only those that now hash to one of the new AMPs. Even for very large systems, this is a time-effective, non-intrusive process that leaves most of the data undisturbed.

## Conclusion

Quality is accompanied by a sense of effortlessness. This paper has illuminated some of the processes and underpinnings contributing to Teradata Database's effortless nature. These include automated data placement, multi-dimensional query parallelism, database/interconnect synergy, and an optimizer sensitive to both the environment and the cost of operations. When a product is born to be parallel, then most wasted energy and unnatural efforts can be eliminated. The result is a product that is focused, integrated, and enduring.

BYNET is a trademark of Teradata Corporation. UNIX is a registered trademark of the Open Group. Teradata continually improves products as new technologies and components become available. Teradata, therefore, reserves the right to change specifications without prior notice. All features, functions, and operations described herein may not be marketed in all parts of the world. Consult your Teradata representative or Teradata.com for more information.

Copyright © 2004-2007 by Teradata Corporation All Rights Reserved. Produced in U.S.A.