WHITE PAPER

Born to Be Parallel, and Beyond

Original work by Carrie Ballinger – Cloud Performance Architect
Revised and updated by Douglas Ebel – Director, Technical Product Marketing

09.22 / TERADATA VANTAGE

teradata.

Table of Contents

Teradata's Enduring Performance Advantage

Teradata's ability to solve the most complex analytics problems of our day is unmatched. Available today in an array of deployment options, Teradata's history has evolved from an on-prem analytics appliance where limited resources drove deep expertise in workload management and query optimization. In the cloud, this same expertise translates into superior execution performance without the risk of costs escalating out of control.

Teradata customers from every industry and around the world depend on Teradata for predictable price performance and mission-critical reliability to run their business. The performance offered by Vantage empowers them to run analytics that would be too expensive or time-consuming otherwise. For example, allowing them to continuously run predictive models on every household in their markets rather than only simple models on a subset of customers.

To ensure that we continue to deliver on the promise of Teradata, we have continuously invested in the architecture of our core analytics engine. As we have done so, we continue to follow key tenets that arose from being born at a time of scarcity because we believe that although resources in the cloud may be infinite, budgets are not.

The enduring performance advantage of the Advanced Analytics Engine is a direct result of early, somewhat unconventional design decisions made by a handful of imaginative architects. Intended for a more technical audience, this paper describes and illustrates some of these key fundamental components of the Advanced Analytics Engine that are as critical to performance now as they were then, and upon which today's features and capabilities rest.

Discussions of these specific areas are included in this paper:

- Multidimensional parallel capabilities
- A parallel-aware query optimizer
- The BYNET's considerable contribution
- A flexible and fast way to find and store data
- Internal self-regulation of the flow of work
- Managing the flow of work externally with Workload Management

It is important to note that the scope of this whitepaper is limited to important, foundational components of database performance. It is not a comprehensive discussion of all the aspects of the Teradata AdvanceAnalytics Engine or the platform.

Multidimensional Parallel Capabilities

Everything in Vantage is parallelized—from the entry of SQL statements to the smallest detail of their execution—to weed out any possible single point of control and to effectively eliminate the conditions that can breed gridlock in a system. It is this foundational architecture, dating back to our humble beginnings, that continues to deliver unmatched performance and the best price per query in the market today.

The Teradata basic unit of parallelism is the AMP (Access Module Processor), a virtual processing unit that manages all database operations for its portion of a table's data. Many AMPs are typically configured on a given node. (20 to 40 or more are common.) Everything that happens in a Teradata system is distributed across a pre-defined number of AMPs with each AMP acting like a microcosm of the database, supporting such things as data loading, reading, writing, journaling, and recovery for all the data that it owns. (see Figure 1). Importantly, parallel units work cooperatively together behind the scenes—an unusual strength that drives higher performance with minimal overhead.

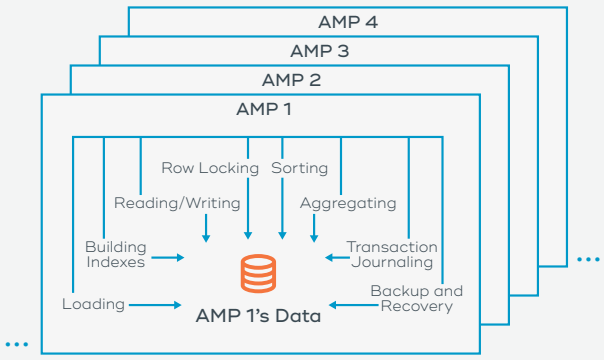

Figure 1. Inside a unit of parallelism.

Types of Query Parallelism

While the AMP is the fundamental unit of parallelism, there are two additional parallel dimensions woven into the Advanced Analytics Engine, specifically for query performance. These are referred to here as "within-a-step" parallelism, and "multi-step" parallelism. The following sections describe these three dimensions of parallelism:

Parallel Execution Across AMPs

Parallel execution across AMPs involves breaking the request into subdivisions, and working on each subdivision at the same time, with one single answer delivered. Parallel execution can incorporate all or part of the operations within a query and can significantly reduce the response time of a request, particularly if the query or function reads and analyzes a large amount of data.

Parallel execution is usually enabled in Teradata by hash-partitioning the data across all the AMPs defined in the system. Once data is assigned to an AMP, the AMP provides all the database services on its allocation of data blocks. All relational operations such as table scans, index scans, projections, selections, joins, aggregations, and sorts execute in parallel across the AMPs simultaneously. Each operation is performed on an AMP's data independently of the data associated with the other AMPs.

.

teradata.

Within-a-Step Parallelism

Within-a-step parallelism is when the optimizer carefully splits a request into a small number of high-level database operations and dispatches these distinct operations for execution in a process called pipelining. Here each operation can continue on without waiting for the completion of the full results from the first operation. The relational-operator mix of a step is carefully chosen by the Teradata optimizer to avoid stalls within the pipeline. (see Figure 2)

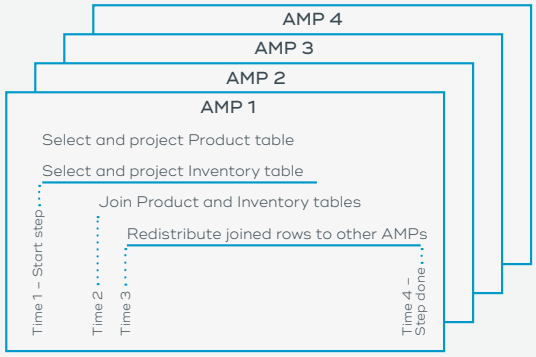

Figure 2. Pipelining of 4 operations within one query step.

Multi-Step Parallelism

Multi-step parallelism is enabled by executing multiple "steps" of a query simultaneously, across all the participating units of parallelism. One or more tasks are invoked for each step on each AMP to perform the actual database operation. Multiple steps for the same query can be executing at the same time to the extent that they are not dependent on results of previous steps.

This automated multifaceted parallelism is not easy to choreograph unless it is planned for in the early stages of product evolution. In addition to these three dimensions of parallelism for each query, such as described here, we will see additional elements below that ensure that Teradata customers get maximum value from every system. It is important to note, the Advanced Analytics Engine applies these multiple dimensions of parallelism automatically, without user intervention, hints or special setup.

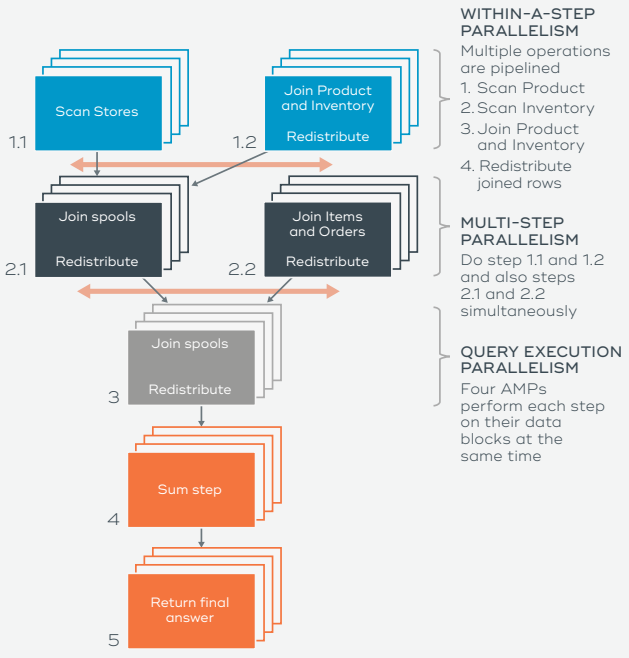

Figure 3. Multiple types of parallelism combined.

The figure shows four AMPs supporting a single query's execution, and the query has been optimized into 7 steps. Step 1.2 and Step 2.2 each demonstrate within-a-step parallelism, where two different tables are scanned and joined together (three different operations are performed). The result of those three operations is pipelined into a sort and then a redistribution, all in one step. Steps1.1 and 1.2 together (as well as 2.1 and 2.2 together) demonstrate multi-step parallelism, as two distinct steps are chosen to execute at the same time, within each AMP.

Multi-Statement Requests

In addition to the three dimensions of parallelism shown in Figure 3, Multi-Statement Requests allow several distinct SQL statements to be bundled together and sent to the optimizer as if they were one unit. These will be run in parallel as long as there are no dependencies among the statements. More importantly, any sub-expressions that the different statements have in common will be executed once, and the results shared among them (see Figure 4).

teradata.

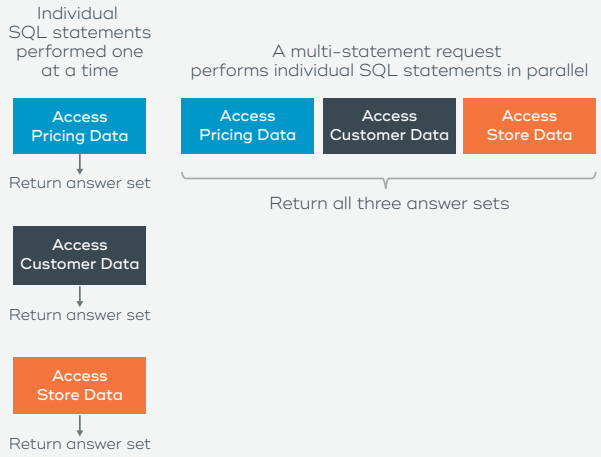

Figure 4. A multi-statement request.

Parallel Use of I/O

In addition, the Advanced Analytic Engine supports synchronized scanning of large tables. This permits a new full-table scan to begin at the current point of an ongoing scan of the same large table in another session, thus reducing the I/O load and supporting higher concurrency.

Parallel-Aware Optimizer

Having an array of parallel techniques can turn into a disadvantage if they are not carefully applied around the needs of each request. Orchestration of the different parallelization techniques is driven by our optimizer, which takes on a number of tasks to ensure optimal use of resources. The optimizer lives within a component called the "Parsing Engine" or PE. The default configuration uses two PEs per node with each capable of coordinating the query planning, coordination of execution, and returning results for 120 sessions each. A four node system would have 8 PEs supporting 960 sessions and a 24 node system would have 48 PEs supporting 5,760 sessions. This is both scalable and fault tolerant by eliminating single points of congestion or failure.

Join Planning

Joining tables in a linear fashion (join table1 to table2, then join their result to table3, and so on) can have a negative impact on query time.

Instead, the Teradata optimizer accesses and joins multiple tables simultaneously and also leverages different types of joins (e.g. indexed access, table scan) to build a more intelligent query plan.

The Teradata optimizer seeks out tables within the query that have logical relationships between them and also groups tables that can be accessed and joined independently from the other subsets of tables. Those are often candidates to execute within parallel steps. Figure 5 illustrates the differences when optimizing a six-table join between a plan that is restricted to linear joins, and one that has the option of performing some of the joins in parallel.

Sizing up the Environment

In addition to the parallelism methods described above, the optimizer takes into account numerous other factors including the profile of the data itself, the number of AMPs on each of the nodes and the processing power of the underlying hardware. Putting all this information together, the optimizer comes up with a price in terms of resources expected to be used for each of several candidate query plans, then picks the least costly candidate. Considering many factors including movement of data, the lowest cost plan is the plan which will take the least system resources to execute on the wide variety of platforms that we support.

Hiding Complexity

Unlike other solutions, Teradata's optimizer completely automates the complexity behind query planning. Users have complete freedom to submit everything from simple tactical queries to very complex ad hoc analytic queries and the optimizer will ensure that all requests are delivered in the most efficient manner. This allows customers to build complex data models with dozens of joins which provides a richer dataset for analytics.

Evolution

Although the fundamentals have remained the same, the Advanced Analytics Engine has continued to evolve over time to meet customer needs. This includes everything from the ability to support tables with no primary index to stage data for in-database transformation or push-down processing by client tools or new types of joins.

teradata.

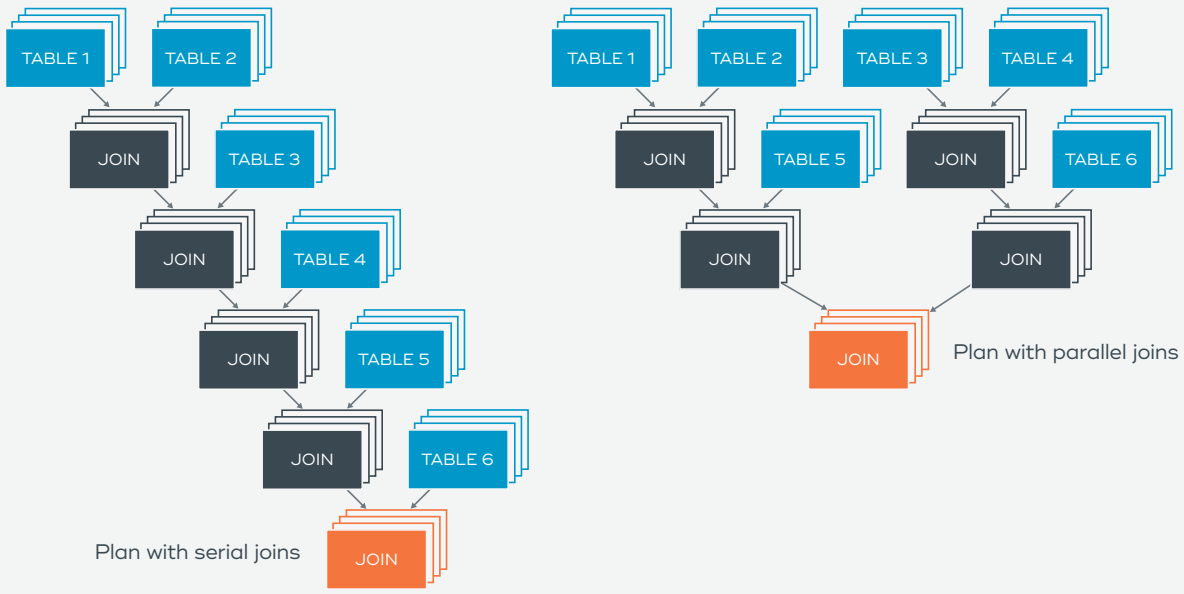

Figure 5. A bushy query plan vs. a serial plan.

There are now nearly 20 join strategies that are chosen automatically by the optimizer. It will incrementally plan and execute when there is uncertainty about the size of an intermediate result set, and it will re-write queries to eliminate redundant logic. The goal is always the same: ensuring that our customers enjoy the lowest cost per query in the industry.

Being Parallel in the Ecosystem

In today's environment, data may reside in other file systems or data management systems. Files in cloud storage may be defined as foreign tables. The optimizer will assign the task of reading and interpreting CSV, Parquet or JSON files to AMPs. As with everything else, the files making up a foreign table in cloud storage will be assigned across the AMPs to be read in parallel.

Teradata's Query Grid can be used to access data in other data management systems The Advance Analytics Engine's optimizer can decide on whether to select the raw data or push down some of the selection and aggregation processing to the other platform to reduce the size of data to be retrieved. Meanwhile, the optimizer may have the AMPs performing other parts of the query processing until the data is retrieved from the other DBMS.

BYNET's Considerable Contribution

Another important component of the Teradata architecture is referred to as the BYNet. This acts as the interconnection between all of the independent parallel components. (see Figure 6). Originally implemented within the hardware of our on-premises systems, this functionality is now implemented directly into the cloud network facilities. Beyond just passing messages, the BYNET is a bundle of intelligence and low-level functions that aid in efficient processing at practically each point in a query's life. It offers coordination as well as oversight and control to every optimized query step.

In short, the BYNet acts as flight coordinator ensuring that the entire system is working in concert and managing situations as they arise. This can include everything from ordering results from across parallel units, adjusting to hardware failures, or monitoring for points of congestion.

teradata.

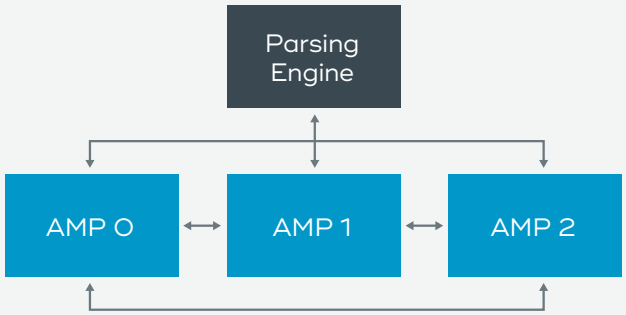

Figure 6. AMPs and PEs communicate using messages.

Messaging

A key role of the BYNET is to support communication between the PE and AMPs and from AMPs to other AMPs. These simple message-passing requirements are performed using a low-level messaging approach, bypassing more heavyweight protocols for communication.

- Sending a step from the PE to AMPs to initiate a query step
- Redistributing rows from one AMP to another to support different join geographies
- Sort/merging a final answer set from multiple AMPs

Even though message protocols are low-cost, the Advanced Analytics Engine goes further by minimizing interconnect traffic wherever possible. Same AMP, localized activity is encouraged wherever possible. AMP-based ownership of data keeps activities such as locking and some of the simple data processing local to the AMP. Hash partitioning that supports co-location of to-be-joined rows reduces data transporting prior to a join. All aggregations are ground down to the smallest possible set of sub-totals at the local (AMP) level first before being brought together globally via messaging.

It is notable that another side effect of this extremely efficient coordination of AMPs is our ability to offer exceptionally faster performance for tactical queries than other vendors.

BYNET Groups

Without the BYNET's ability to combine and consolidate information from across all units of parallelism, each AMP would have to independently talk to each other AMP in the system about each query step that is underway. As the configuration grew, such a distributed approach to coordinating query work would quickly become a bottleneck.

Instead, BYNET groups create a dynamic relationship between AMPs that are working on a specific step which keeps the number of AMPs that must exchange messages down to the bare minimum. As a step begins to execute, one or more channels are established that loosely associate all AMPs in the dynamic BYNET group that is executing the step. The channels use monitoring and signaling semaphores in order to communicate things like the completion or the success/failure of each participating AMP. If a tight coordination did not exist among AMPs in the same BYNET group, then the problem-free AMPs would continue to work on the doomed query step, eating up resources in unproductive ways (Figure 7). In general, the only message that is set back to the PE is the final completion message whether the dynamic BYNET group is composed of three or 3000 AMPs.

Final Answer Set Sort/Merge

Never needing to materialize a query's final answer set inside the database has long been a Teradata differentiator. The final sort/merge of a query takes place within the BYNET as the answer set rows are being funneled up to the client as needed. This happens at the AMP, Node and finally PE level with only the highest values being processed until the client needs more. The final answer set never has to be brought together saving considerable resources. A potential "big sort" penalty has been eliminated—or actually, never existed.

teradata.

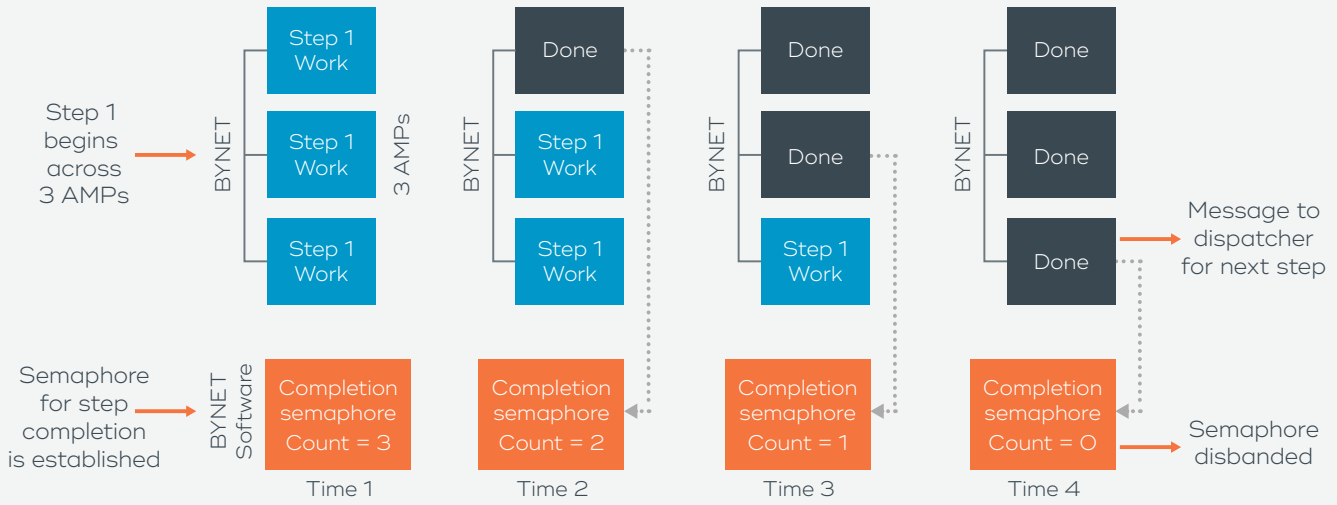

Figure 7. A completion semaphore.

A Flexible, Fast Way to Find and Store Data

Another very important factor behind the enduring Teradata performance is how space is managed which is done by a sub-system that is simply referred to as the "file system." The file system is responsible for the logical organization and management of the rows, along with their reliable storage and retrieval.

The file system in Teradata was architected to be extremely adaptable, simple on the outside but surprisingly inventive on the inside. It was designed from Day One to be fluid and open to change. The file system's built-in flexibility is achieved by means of:

- Logical addressing, which allows blocks of data to be dynamically shifted to different physical locations when needed, with minimal impact to active work.
- The ability for data blocks to expand and contract on demand, as a table matures.
- An array of unobtrusive background tasks that do continuous space adjustments and clean-up.

Teradata was architected in such a way that no space is allocated or set aside for a table until such time as it is needed. Rows are stored in variable length data blocks that are only as big as they need to be. These data blocks can dynamically change size and can be moved to different locations on the cylinder or even to a different cylinder, without manual intervention or end-user knowledge. With the development of Teradata Virtual Storage (TVS), the database will assess the frequency of access of data and can move it between different speed storage media to optimize response time for the end user.

This section takes a close look at how file system frees up the administrator from mundane data placement tasks, and at the same time provides an environment that is friendly to change.

How Data is Organized

For data stored inside the database, Teradata permanently assigns data rows to AMPs using a simple scheme that lends itself to an even distribution of data—hash partitioning. (Figure 8). In addition to being a distribution technique, this hash approach to data placement serves as an indexing strategy.

teradata.

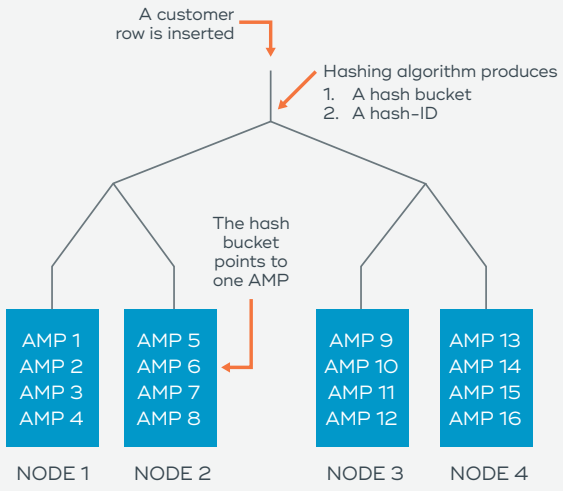

Figure 8. A row's primary index hash bucket points to the AMP that owns it.

To retrieve a row, the primary index data value is passed to the hashing algorithm, which generates the two hash outputs: 1) the hash bucket which points to the AMP; and 2) the hash-ID which helps to locate the row within the file system structure on that AMP. There is no space or processing overhead involved in either building a primary index or accessing a row through its primary index value, as no special index structure needs to be built.

Hashed data placement is very easy to use and requires no setup. The only effort a DBA makes is the selection of the columns that will comprise the primary index of the table such as customer number, order number or product key. From that point on, the process is completely automated. No files need to be allocated, sized, monitored, or named. No DDL needs to be created beyond specifying the primary index in the original CREATE TABLE statement. No unload-reload activity is ever required.

Once the owning AMP is identified by means of the hash bucket, the hash-ID is used to look up the physical location of the row on disk. Which virtual cylinder and sector holds the row is determined by means of a tree-like three-level indexing structure (as shown in Figure 9). It is enough to say here that the data is automatically and dynamically indexed down to the exact data block for exceptional retrieval speed.

This is incredibly important, especially for tactical queries that are often leveraged by business applications.

Easy Accommodation of Data Growth

The Advanced Analytics Engine is built using a logical addressing model as a low impact way to adjust to data growth. Data for each table in a Teradata system is stored in flexibly-sized data blocks that are assigned to logical cylinders. The block assignment of a row is based on its hash value. If a block grows beyond a DBA-specified maximum size, it is automatically split to make room for more rows and the cylinder index is updated. If a logical cylinder gets full, blocks can be moved to a different logical cylinder and the cylinder indexes are updated. On retrieving a row, the hash of the primary index identifies the AMP, the index of cylinders in the AMP point to the cylinder, and the cylinder index points to the block to be read. Figure 10 explains this behavior visually.

This adaptable behavior delivers numerous benefits. Random growth is accommodated at the time it happens. Rows can easily be moved from one location to another without affecting in-flight work or any other data objects that reference that row. There is never a need to stop activity and re-organize the physical data blocks or adjust pointers.

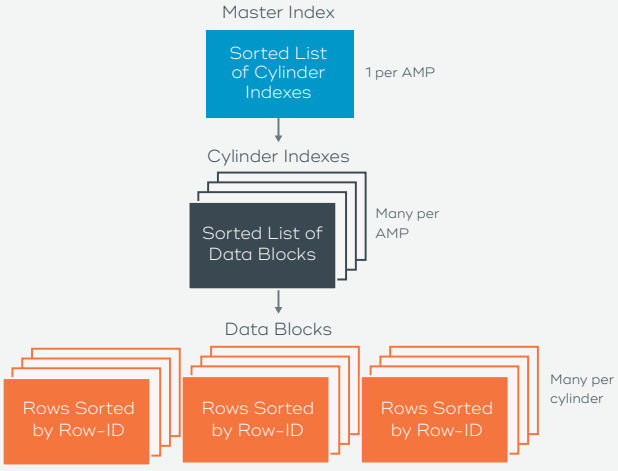

Figure 9. A three-level indexing structure identifies a row's location on an AMP.

teradata.

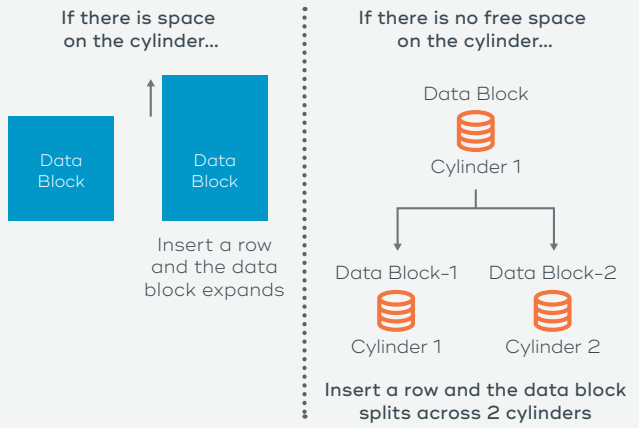

Figure 10. A new row is inserted into an existing data block.

This flexibility to consolidate or expand data blocks anytime allows the Advanced Analytics Engine to do many space-related housekeeping tasks in the background and avoid table unloads and reloads common to fixed-sized page databases. This advantage increases database availability and translates to less maintenance tasks for the DBA.

Multi-level Row Partitioning

Added to this storage architecture is the ability to partition the table by one or more columns to make it faster to access data without the need of full table scans or the costly maintenance of secondary indexes. For example, a transaction table might be partitioned on transaction date, week, or month. If a query constrains on a period of time for those transactions, the optimizer will figure out which partitions need to be read, whether the table was partitioned on day week, month or other time period ranges. You could also add additional partitioning columns like country, district, or brand. A query with a constraint on either partitioning column or both will reduce the amount of data to be read to satisfy a query. The hashed cylinder and row access is accomplished within the defined partitions.

Column Partitioning

Tables can also be stored with columns in separate partitions. This has the advantage of focusing I/O on just the columns of data needed in a query instead of the entire row. This also supports vertical compression techniques where a value is stored once for use in consecutive rows. Column partitioning can be combined with row partitioning to further reduce the amount of I/O needed to satisfy a query.

Indexes

The primary index for a table takes no space and by calculating the hash value of a constraint on that column, its row can usually be retrieved in a single I/O. Partitioning also requires no space and allows for a significant reduction in I/O and improvement in response time. The Advanced Analytics Engine also supports traditional secondary indexes. These are valuable with a frequently used, high cardinality column exists such as customer number on a table such as Orders where the logical primary index for the orders table is the Order_ID.

Also supported are Join Indexes which are transparent to the user or their BI tools but are leveraged by the optimizer to eliminate join and aggregation processing. As the base tables are maintained these join indexes are automatically maintained. If one join index is a more aggressive aggregation of another, after the base table is updated, the lower-level aggregation is re-calculated, then those values are aggregated to maintain the more aggressive aggregation. If analysis of usage in the query logging indicate that the join index is not being used, it can be dropped and there is no impact to the syntax of the user's queries.

Work Flow Self-Regulation

A shared-nothing parallel database has a special challenge when it comes to knowing how much new work it can accept, and how to identify congestion that is starting to build up inside one or more of the parallel units. With the optimizer attempting to apply multiple dimensions of parallelism to each query that it sees, it is easy to reach very high resource utilization within a Teradata system, even with just a handful of active queries.

teradata.

Designed for stress, the Advanced Analytics Engine is able to function with large numbers of users, a very diverse mix of work, and a fully-loaded system. Being able to keep on functioning full throttle under conditions of extreme stress relies on internal techniques that were built inside the database to automatically and transparently manage the flow of work, while the system stays up and productive.

Even though the data placement conventions in use with the Advanced Analytics Engine lend themselves to even placement of the data across AMPs, the data is not always accessed by queries in a perfectly even way. During the execution of a multi-step query, there will be occasions when some AMPs require more resources for certain steps than do other AMPs. For example, if a query from an airline company site is executing a join based on airport codes, you can expect whichever AMP is performing the join for rows with Atlanta (ATL) to need more resources than does the AMP that is joining rows with Anchorage (ANC). Some of this uneven processing demand has been reduced by the optimizer splitting the data into separate spool files and applying different join strategies for the busy airports and the less busy ones. However, some unevenness of processing demands will remain.

AMP-Level Control

The Advanced Analytics Engine manages the flow of work that enters the system in a highly-decentralized manner, in keeping with its shared-nothing architecture. There is no centralized coordinator to become a bottleneck. There is no message-passing between AMPs to determine if it's time to hold back new requests. Rather, each AMP evaluates its own ability to take on more work, and temporarily pushes back when it experiences a heavier load than it can efficiently process. And when an AMP does have to push back, it does that for the briefest moments of time, often measured in milliseconds.

This bottom-up control over the flow of work was fundamental to the original architecture of the database as designed. All-AMP step messages come down to the AMPs, and each AMP will decide whether to begin working on it, put it on hold, or ignore it. This AMP-level mindfulness is the cornerstone of the database's ability to accept impromptu swings of very high and very low demand, and gracefully and unobtrusively manage whatever comes its way.

AMP Worker Tasks

AWTs are the tasks inside of each AMP that get the database work done. This database work may be initiated by the internal database software routines, such as dead-lock detection or other background tasks. Or the work may originate from a user-submitted query. These pre-allocated AWTs are assigned to each AMP at startup and, like taxi cabs queued up for fares at the airport, they wait for work to arrive, do the work, and come back for more work.

Because of their stateless condition, AWTs respond quickly to a variety of database execution needs. There is a fixed number of AWTs on each AMP. For a task to start running it must acquire an available AWTs. Having an upper limit on the number of AWTs per AMP keeps the number of activities performing database work within each AMP at a reasonable level. AWTs play the role of both expeditor and governor.

As part of the optimization process, a query is broken into one or many AMP execution steps. An AMP step may be simple, such as read one row using a unique primary index or apply a table level lock. Or an AMP step may be a very large block of work, such as scanning a table, applying selection criteria on the rows read, redistributing the rows that are selected, and sorting the redistributed rows.

The Message Queue

When all AMP worker tasks on an AMP are busy servicing other query steps, arriving work messages are placed in a message queue that resides in the AMP's memory. This is a holding area until an AWT frees up and can service the message. This queue is sequenced first by message work type, which is a category indicating the importance of the work message. Within work type the queue is sequenced by the priority of the request the message is coming from.

Messages representing a new query step are broadcast to all participating AMPs by the PE. In such a case, some AMPs may provide an AWT immediately, while other AMPs may have to queue the message. Some AMPs may dequeue their message and start working on the step sooner than others. This is typical behavior on a busy system where each AMP is managing its own flow of work.

Once a message has either acquired an AWT or been accepted onto the message queue across each AMP in the dynamic BYNET group, then it is assumed that each AMP will eventually process it, even if some AMPs take longer than others. The sync point for the parallel processing of each step is at step completion when each AMP signals across the completion semaphore that it has completed its part. The BYNET channels set up for this purpose are discussed more fully in the BYNET section of this paper.

Turning Away New Messages

Each AMP has flow control gates that monitor and manage messages arriving from senders. There are separate flow control gates for each different message work type.7 New work messages will have their own flow control gates, as will spawned work messages. The flow control gates keep a count of the active AWTs of that work type as well as how many messages are queued up waiting for an AWT.

Once the queue of messages of a certain work type grows to a specified length, new messages of that type are no longer accepted and that AMP is said to be in a state of flow control, as shown in Figure 15. The flow control gate will temporarily close, pulling in the welcome mat, and arriving messages will be returned to the sender. The sender, often the PE, continues to retry the message, until that message can be received on that AMP's message queue.

Because the acceptance and rejection of work messages happens at the lowest level, in the AMP, there are no layers to go through when the AMP can get back to normal message delivery and processing. The impact of turning on and turning off the flow of messages is kept local—only the AMP hit by an over-abundance of messages at that point in time throttles back temporarily.

Riding the Wave of Full Usage

Teradata was designed as a throughput engine, able to exploit parallelism to maximize resource usage of each request when only a few queries are active, while at the same time able to continue churning out answer sets in high demand situations. To protect overall system health under extreme usage conditions, highly-decentralized internal controls were put into the foundation, as discussed in this section.

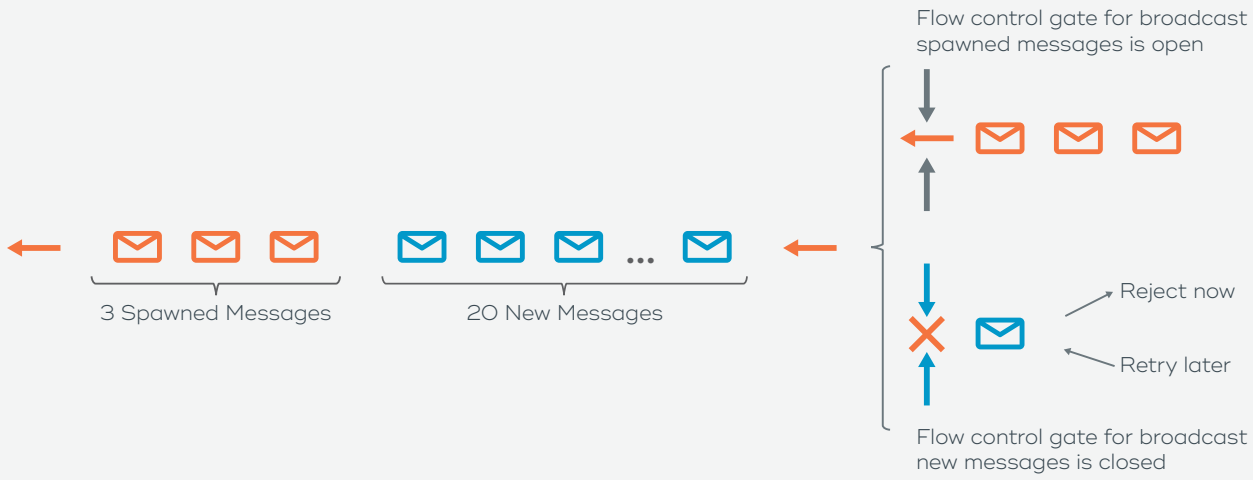

Figure 11. Flow control gates close when a threshold of messages is reached.

teradata.

The original architecture related to flow control and AMP worker tasks has needed very little improvement or even tweaking over the years. 80 AWTs per AMP is still the default setting for new Teradata systems. The number can be increased for more powerful platforms that aren't achieving full utilization or platforms with large number of active queries with diverse response time expectations. Message work types, the work message queue, and retry logic all work the same as they always did.

There have been a few extensions in regard to AMP worker tasks that have emerged over time, including:

- Setting up reserve pools of AWTs exclusively for use by tactical queries, protecting high priority work from being impacted when there is a shortage of AWTs.

- Automatic reserve pools of AWTs just for load utilities that become available when the number of AWTs per AMP is increased to a very high level, intended to reduce resource contention between queries and load jobs for enterprise platforms with especially high concurrency

Workload Management

The second section in this whitepaper called attention to the multifaceted parallelism available for queries on the Advanced Analytics Engine. The subsequent section discussed how the optimizer uses those parallel opportunities in smart ways to improve performance on a query-by-query basis. And the previous section illustrated internal AMP-level controls to keep high levels of user demand and an over-abundance of parallelism from bringing the system to its knees.

In addition to those automatic controls at the AMP level, Teradata has always had some type of system-level workload management, mainly priority differences, that are used by the internal database routines.

The Original Four Priorities

One of the challenges faced by the original architects of Teradata Database was how to support maximum levels of resource usage on the platform, and still get critical pieces of internal database code to run quickly when it needed to. For example, if there is a rollback taking place due to an aborted transaction, it benefits the entire system if the reversal of updates to clean up the failure can be executed quickly.

It was also important to ensure that background tasks running inside the database didn't lag too far behind. If city streets are so congested with automobile traffic that the weekly garbage truck can't get through and is delayed for weeks at a time, a health crisis could arise.

The solution the original architects found was a simple priority scheme that applied priorities to all tasks running on the system. This rudimentary approach offered four priority buckets, each with a greater weight than the one that came before: L for Low, M for Medium, H for High and R for rush. The default priority was medium, and indeed most work ran at medium, and was considered equally-important to other medium priority work that was active.

However, database routines and even small pieces of code could assign themselves one of the other three priorities, based on the importance of the work. Developers, for example, decided to give all END TRANSACTION activity the rush priority, because finishing almost-completed work at top speed frees up valuable resources sooner, and was seen as critical within the database. In addition, if the administrator wanted to give a favored user a higher priority, all that was involved was manually adding one of the priority identifiers into the user's account string.

Background tasks discussed in the section about space management were designed to use priorities as well. Some of these tasks, like the task that deletes transient journal rows that are no longer needed, were designed to start out at the low priority, but increase their priority over time if the system was so busy that they were not able to get their work accomplished. This approach kept such tasks in the background most of the time, except when their need to complete becomes critical.

Impact of Mixed Workloads

The simple approach to priorities was all the internal database tasks required. And early users of the database were satisfied running all their queries at the default medium priority. But requirements shifted over time as Teradata users began to supplement their traditional decision support queries with new types of more varied workloads.

In the late 1990's, a few Teradata sites began to issue direct look-up queries against entities like their Inventory tables or their customer databases, at the same time as their standard decision support queries were running. Call centers started using data in their Teradata Database to validate customer accounts and recent interactions. Tactical queries and online applications blossomed, at the same time as more sites turned to continuous loading to supplement their batch windows, giving their end users more timely access to recent activity. Service level goals reared their head. Stronger, more flexible workload management was required. Today it is typical for 90% of the queries to execute in < 1 second.

Evolution of Workload Management

While the internal management of the flow of work has changed little, the capabilities within system-level workload management have expanded dramatically over the years. As the first step beyond the original four priorities, Teradata engineering developed a more extensive priority scheduler composed of multiple resource partitions and performance groups, and the flexibility of assigning your own customized weighting values. These custom weightings and additional enhancements make it easier to match controls to business workloads and priorities than the original capabilities designed more for controlling internal system work.

Additional workload management features and options that have evolved over the years include:

- Ability to define workloads by username, client logon ID, profile, the application they are using, the database objects they are referencing or the optimizer's assessment of the query characteristics
- Concurrency control mechanisms, called throttles, that can be placed at multiple levels and tailored to specific types of queries or users.
- An improved and more effective priority scheduler to accompany the Linux SLES 11 operating system that can protect short, critical work more effectively from more resource-intensive lower-priority jobs.
- Rules to reject queries that are poorly written or that are inappropriate to run at certain times of the day.
- Ability to automatically change workload settings by time of day or system conditions.
- Ability to automatically reduce the priority of a running query which exceeds the threshold of resources consumed for its current priority.
- Ability to give a percentage of resources to a workload, either as a maximum percentage or an "at least" percentage.
- A user-friendly front-end GUI called Viewpoint Workload Designer that supports ease of setup and tuning.

Workload management in Teradata has proven to be rapidly expanding area, indispensable to customers that are running a wide variety of work on their Teradata platform. While internal background tasks and subsets of the database code continue to run at the four different priority levels initially defined for them, many Teradata sites have discovered that their end users' experiences are better and they can get more work through the system when taking advantage of the wider workload management choices today. And many do just that.

Managing Workload Management

To know whether the system is meeting required performance or is being impacted by new, unplanned, or poorly constructed workloads, it is critical to have logging of system activity. The query logging in 18 tables and 993 columns records everything about query execution including use of system resources, SQL, steps, objects, and a textual description of the query execution plan. The Resource Usage logging in 12 tables and 1878 columns records everything happening at the system level including node, AMP, AWT, and device.

The logging levels are optional and may be combined with the Performance Data Capture Routines (PDCR) for historical analysis and capacity planning. No other DBMS has the maturity of logging as the Vantage Advanced Analytic Engine.

Conclusion

Foundations are important. Teradata's ability to grow in new directions and continue to sustain its core competencies is a direct result of its strong, tried-and-true foundation. As our engine has matured the same fundamentals have been adapted to new technology advances. For example, in initial releases, the AMP was a physical computer which owned its own disk strive and directly managed how data was located on its disks. Today an AMP is a software virtual processor that co-exists with other such virtual processors on the same node all of whom share the node resources. Yet each AMP maintains its shared-nothing characteristics, same as in the first release.

The natural evolution towards the virtualization of key database functionality is significant because it broadens the usefulness of the Advanced Analytics Engine. For much of its history, Teradata database software has run on purpose-built hardware, where the underlying platform has been optimized to support high throughput, critical SLAs, and solid reliability. While those benefits remain well-suited for enterprise platforms, this virtualization opens the door for the Advanced Analytics Engine to participate in more portable, less demanding solutions. Public or private cloud architectures, as well as as-a-service offerings, can now enjoy the core Advanced Analytics Engine capabilities as described in this white paper.

This white paper attempts to familiarize you with a few of the features that make up important building blocks of the Advanced Analytics Engine, so you can see for yourself the elegance and the durability of the architecture. This paper points out recent enhancements that have grown out of this original foundation, building on it rather than replacing it.

These foundational components have such a widespread consequence that they simply cannot be tacked on as an afterthought. The database must be born with them.

About Teradata

Teradata is the connected multi-cloud data platform company. Our enterprise analytics solve business challenges from start to scale. Only Teradata gives you the flexibility to handle the massive and mixed data workloads of the future, today. Learn more at Teradata.com.

17095 Via Del Campo, San Diego, CA 92127 Teradata.com