

Road rules for throttles

Controlling query concurrency in the Teradata Database.



Carrie Ballinger
Senior database analyst,
Special Projects
Teradata Certified Master

Living in the greater Los Angeles area, I've become addicted to listening to traffic flows and freeway conditions on the radio, and have taken to mentally solving these traffic-flow problems. This isn't really so different from solving technical problems. I have found that writing this column has allowed me to channel some of my traffic-flow obsessions in a more productive direction. My focus this issue is on controlling query concurrency using throttles in the Teradata Database.

The ins and outs of throttles

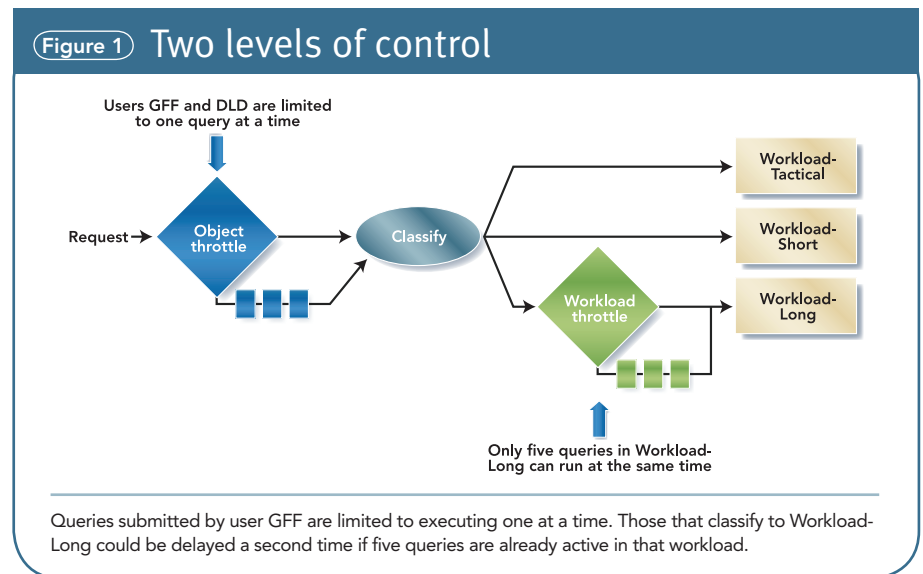
Imagine your database as a busy freeway and your queries as the vehicles. The number of cars and how fast they are traveling affects each driver's experience in the same way as the number of active queries in a database. Slower vehicles can impede the progress of faster cars just as large workloads can influence the performance of short queries.

To avoid slowdowns at rush hour, road rules and devices are created; for instance, metered lights at the on-ramps help streamline the ongoing traffic. Like the metered lights, throttles are tools that keep the work entering the database at an

even, predictable level. When a throttle is defined, simple rules are established that limit concurrency for a certain category of objects, such as users or specific workloads.

Once a throttle is active, a counter keeps track of the number of requests that are executing under the throttle's control. If a query is ready to begin execution but the throttle's counter has reached the limit prescribed by the rule, that query is placed in a delay queue.

A common technique is to set the object throttle rules to limit the number of active queries individual users are allowed at any point in time. The object throttle is applied when the query is first seen. In the diagram, the object throttle rules limit User GFF to one active query at a time. Any additional queries submitted by GFF will be held in the delay queue until the earlier ones are complete. For queries that classify to specific workloads, as shown in figure 1, an additional



When the counter falls below the limit, a query is released from the queue, based on first-in-first-out (FIFO) conventions.

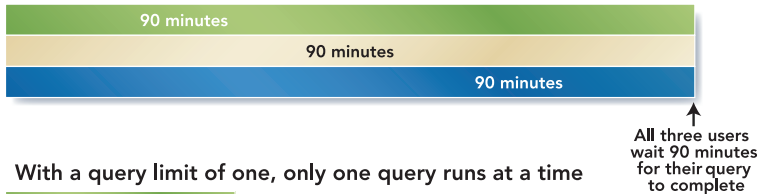
If Teradata Active System Management is enabled, an individual query may be delayed by both an object throttle and a workload throttle. (See figure 1.) A

throttle can be used for even greater concurrency control at the workload level.

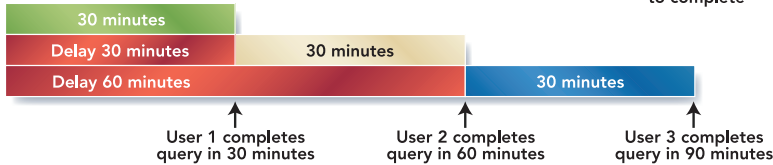
The workload throttle in the diagram is limited to five simultaneous Workload-Long queries. So, even after GFF's query is released from the delay queue for the object throttle, it could be delayed a

Figure 2 Throttle limits improve user experience
(A hypothetical example)

With no query limit, all three queries run concurrently



With a query limit of one, only one query runs at a time



When these queries are run concurrently, they must share resources. This extends the time for each query threefold. The average completion time improves when queries are run with a query limit.

second time by the workload throttle until one of the pending Workload-Long queries is completed. If the query were classified as Workload-Short, it would have run immediately after being released by the object throttle rule.

Throttles are popular choices for Teradata customers because they act as anti-congestion devices. I have seen many cases where reducing concurrency below the point of system saturation improves overall throughput. At the same time,

this often provides more consistency for shorter workloads. And while it may seem counterintuitive, throttles can improve the average user's experience, even though some queries will spend time in the delay queue. (See figure 2.)

At what point are throttle counters decremented?

Dear Carrie:

Some of my queries have long answer sets. Since I'm using a lot of throttles,

I am concerned that these queries are holding onto a query slot while the long answer set is being returned. When is the query considered complete, and at what point is the counter lowered so the next query can begin running?

—Tired of Waiting

Dear Waiting:

The query is considered finished when the final database step has completed and the last-completed AMP worker task is released. This is before the BYNET driver returns the answer set. Even if the answer set is large, your next query can begin execution without having to wait for the client to receive the entire answer set.

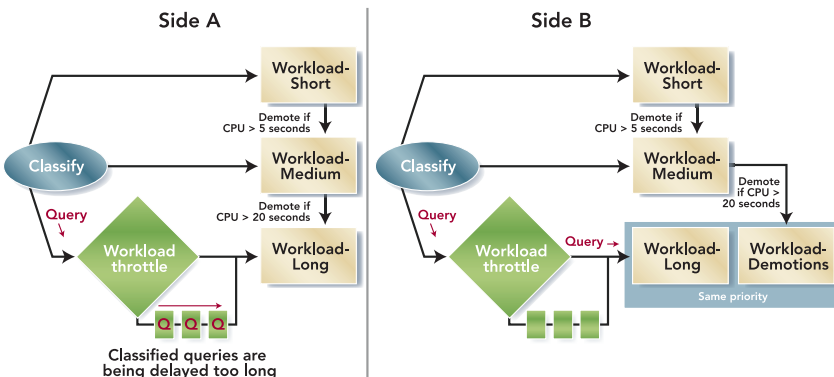
How much memory is used for delayed requests?

Dear Carrie:

How much memory is used by each request that is held in the delay queue? We frequently reach low free-memory levels (less than 50MB) on some nodes during the ETL [extract, transform and load] window, and we want to avoid more memory pressure from very long delay queues.

—Memory Keeper

Figure 3 Demotions dominate query slots



If a workload that is receiving demotions also has a throttle, and the demotions are taking all of the query slots (side A), one solution is to create a new workload just for the demotions (side B).

Dear Keeper:

The process that manages the query delays runs on a single node and starts off using a single 4MB segment of memory. If it later exhausts this memory, it tries to obtain additional 4MB segments, up to a total of 16MB. If memory shortages interfere with any of these memory segment acquisitions, a couple of requests will be released from the delay queue. This will free up some memory for the newly delayed requests while keeping the queue FIFO as much as possible. Counters and other objects used to manage throttles are also held in these memory segments.

The delay queue information that is kept for each request is not extensive; for

example, the delay queue does not need to keep the query text. Instead, all of the detail needed to run the query, such as the text and the plan, is held in the dispatcher. Each delay queue entry is 1024 bytes, making a limit of close to 16,000 queries that can be held in the delay queue at any one time. So, in essence, long, complex queries that are delayed do not necessarily add stress to memory.

How can the conflict between my demoted queries and my throttle counter be resolved?

Dear Carrie:

I'm using Teradata Active System Management and have two medium-priority workloads that can demote queries into a third, lower-priority workload called WD-Low. The problem is that WD-Low has a query limit of six, which is frequently exceeded because of the volume of demoted queries. As a result, queries that classify to WD-Low get stuck in the delay queue for what seems like forever, waiting for the counter to fall below the limit.

—*Beyond the Limit*

Dear Beyond:

WD-Low's query counter gets incremented once for each demotion into that workload. This ensures that a workload throttle counter always reflects the actual concurrency within the workload. Because demoted queries can't be delayed, queries that classify to WD-Low must wait longer in the delay queue when demotions are frequent.

My advice is to demote the exception queries into a new workload that does not have a workload throttle and keep the original WD-Low (along with its throttle) exclusively for queries that classify to WD-Low. Then, map both workloads to the same priority scheduler allocation group so work from both will run at the same low priority. This way the query limit counter on WD-Low won't

TABLE 1: DBQLogTbl EXCERPT FROM A V2R6.2 SYSTEM

StartTime	FirstStepTime	DelayTime	WDID	ExtraField2
6:47:01	6:47:10	8	72	3
6:46:53	6:47:00	6	72	3
16:01:51	16:02:59	68	72	3

TABLE 2: DBQLogTbl EXCERPT FROM A TERADATA DATABASE 13.0 SYSTEM

StartTime	FirstStepTime	DelayTime	WDDelay-Time	WDID	TDWMRuleID
15:11:56.22	15:19:12.90	436.68	?	?	1

increase with every query demoted. (See figure 3, page 2.)

Another, easier option is to tune classification criteria so queries classify more appropriately and fewer demotions are needed.

Which throttle delayed my query?

Dear Carrie:

We are on Teradata Database V2R6.2 and use both workload throttles and object throttles. When a query is delayed I can see the delayed time in Database Query Log [DBQL], but is there an easy way to find out which throttle delayed a query after the fact?

—*Getting to the Source*

Dear Source:

As of Teradata Database V2R6.2, if a query was delayed by an object throttle, DBQL reports a Rule ID in Extrafield2 of the DBQLogTbl. Views won't show you this column; instead, you have to access the base table to see it. By looking at the TDWM.RuleDefs table, you can match that Rule ID to the object throttle rule.

In Teradata Database 12.0, the DBQLogTbl adds a column TDWM-RuleID that serves this same purpose. This new column can be accessed by name in the base tables and views.

Just so you know, Rule IDs are unique within the rule set and are assigned at

the time the rule is created. However, a new row is added to the rules definition table each time the rule is modified, even though the Rule ID is not changed. To view the most current rule in RuleDefs, find the one with RemoveDate = 0.

It is possible for a delay time to be reported in the DBQL row while Extrafield2 remains blank. If you see that combination, you can assume that a workload throttle was responsible for the delay. That workload identifier can be found in the column WDID, which is also in the DBQLogTbl row.

Be aware that if the query was under the control of two different object throttles, both throttles could have contributed to the delay. However, only the Rule ID of the first throttle encountered is recorded in DBQL.

I want to share with you examples of output from DBQLogTbl taken from a V2R6.2 system and from a Teradata Database 13.0 system. These examples will illustrate how to identify which throttle caused a query delay.

Table 1 is a subset of columns and rows from DBQLogTbl taken from a production Teradata site that runs Teradata Database V2R6.2. These three rows represent queries that were all delayed by an object throttle rule before being classified to a workload.

The number of seconds reported in the DelayTime column represents the

difference between the first step time and the query's start time.

The WDID column provides the workload identifier (ID) where that query is classified—in this case, all of the queries are classified to workload ID 72. ExtraField2 shows that all of these queries were delayed by an object throttle with a rule ID of 3. Table 2 on page 3 shows an excerpt of a DBQL output from a Teradata Database 13.0 system in which an object throttle caused a query to be delayed. In this example, workloads are not being used.

Delay times caused by workload throttles and object throttles combined (DelayTime) and those caused only by workload throttles (WDDelayTime) appear in separate columns in Teradata Database 13.0. WDDelay-Time will not contain a value when an object throttle is solely responsible for the delay, as is the case here. In Teradata Database V2R6.2, ExtraField2 held the TDWMRuleID value. As of Teradata Database 12.0 a new column, specifically named TDWMRuleID, identifies which object throttle caused the delay.

When is a throttle rule limit changed?

Dear Carrie:

When a limit on a throttle is changed, will the delay queue be drained before that change is instituted? Or does the delay queue stay intact, while only the limit gets modified?

—*Change-Averse*

Dear Change:

When a throttle rule limit is changed explicitly, or automatically by the system, queries that are already running and query entries that are in the delay queue are rechecked and recounted against the new rule parameters. The delayed queries are not released for execution unless it is determined that the modified rule no longer controls them.

In the case of a simple throttle limit change, when the queries that were previously queued are re-evaluated, they will be recounted and re-queued, usually in the order in which they were submitted. If the throttle limit increased, some previously delayed queries will be released. However, while the system constantly tries to maintain the FIFO order of the queue, that

precise order cannot be guaranteed when throttle changes are made.

Must throttle rules change after a hardware reconfiguration?

Dear Carrie:

After a hardware reconfiguration, does an administrator need to do anything to the throttle rules, such as reloading rule sets? I've enabled both object throttle and workload categories, and my rules will be active at the time of the reconfig. Will I need to redefine them?

—*Rule Follower*

Dear Follower:

No, you won't need to redefine the rules. Teradata Dynamic Workload Manager is controlled by the TDWM.gdo record that resides on each node. That record will not be altered by the reconfig. If the configuration is drastically changed because of the reconfig, then you may want to adjust some of your rules. For example, you might want to allow a somewhat higher query concurrency on a platform that has more processing power. **T**