**The Db2 Adaptive Workload Manager Part 1: Stability out of the Box**

Since Version 9.5, Db2 has had a robust set of workload management capabilities. In version 11.5 Db2 introduced the adaptive workload manager which is a game changer for concurrency control. This is the first in a series of blogs that will do a deep dive into the adaptive workload manager. This first article will describe the motivation for workload management, the challenges posed by todays analytic workloads and the ways in which the adaptive workload manager rises to meet these challenges.

**Why Workload Management?**

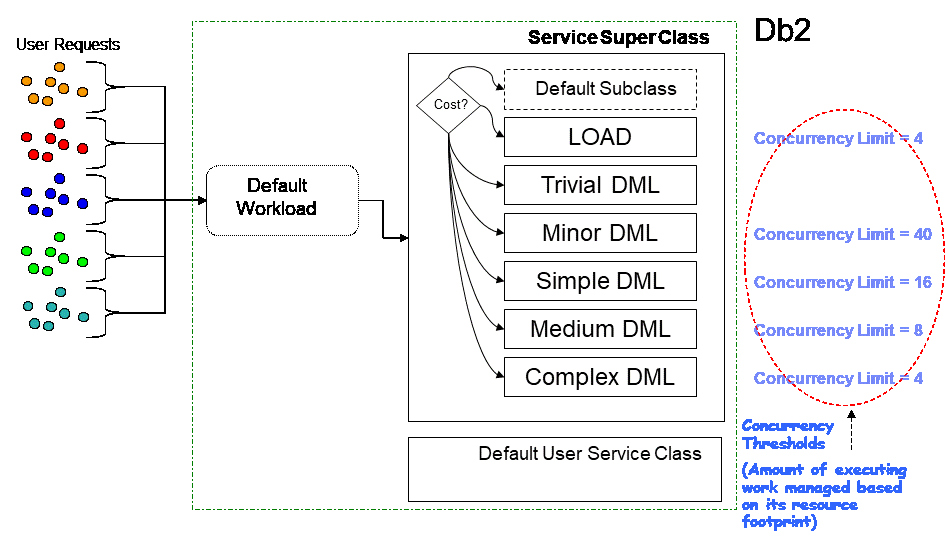
Let’s begin by considering the reasons why workload management capabilities are of interest within a database. Modern data warehousing systems working on a finite set of resources are expected to handle a wide variety of workloads while remaining responsive. Workload composition is typically highly variable and will include a mix of point queries, interactive reporting, heavy analytics and continuous data ingest. Supporting high concurrency is a key user demand making effective workload management critical in these types of systems.

Given this framework of finite resources, variable workloads and high concurrency we can state the requirements of a workload management as follows:

1. Ensure system stability and responsiveness. It is important to ensure system resources are not overcommitted so that queries remain responsive, but at the same time ensure the system is well utilized so that resources are not wasted.
2. Support for workload prioritization and isolation. Given we are dealing with a limited set of resources, it is important to be able to subdivide resources such more resources are given to higher priority work. Furthermore, it is desirable to be able to isolate different applications and users to ensure more consistent response times.
3. Support for workload governing and monitoring. When it comes to resource utilization there will always be “bad actors”, queries that are poorly written or resource intensive queries intended to run during off hours that are incorrectly executed at a peak time. It is necessary to have the capability to define rules to detect and abort such rogue queries. It is also necessary to have comprehensive monitoring capabilities so that the demands on the system resources can be understood.

**Traditional Workload Management and The Challenges of Modern Analytic Workloads**

In the past Db2 has relied on fixed concurrency limits (for example the CONCURRENDBCOORDACTIVITIES threshold) to manage workload concurrency of analytic workloads on the database. The older Db2 workload manager best practices document describes a method to achieve system stability by classifying work into different cost buckets/classes and then applying successively more restrictive concurrency limits as the relative cost of the work in the classes increases. The result is a configuration as shown:



This is a fairly common strategy among database vendors, where concurrency limits may be known as different names (e.g. slots, throttles, queues, etc).

Flaws become visible in this approach when we consider the shift towards in-memory column store technologies in modern analytic workloads. With columnar table-based queries fixed resources like memory become the limiting factor vs. CPU. When CPU is the dominant resource over-admission is less of a concern because work may run slower, but overall system stability can be maintained. Contrast this to memory where over admission leads to failures rather than slowdown when the memory is overcommitted. Overcommitment of memory can also lead to spilling memory objects to disk resulting in performance issues and overall increase in disk utilization.

Consider as well that the failures may not be limited to the queries that are using the most memory. For example, suppose a query is submitted that uses 90% of the database memory, followed by 50 queries requiring 2% each. In such a scenario if the large query gets the memory it requires, rather than a single failure for the large query we may observe 45 failures (in this example) as the large query and first 5 smaller queries get the memory they need, and then all other smaller queries fail due to insufficient memory.

For workloads with high variance in query memory requirements, configurations based on fixed concurrency limits are necessarily sub-optimal and difficult to tune. Returning to the prior example, if there are queries that require > 50% of the total database memory, to guarantee stability it would be necessary to use a fixed concurrency limit of 1 (i.e. ensuring we never run 2 large queries at the same time). Clearly this would be an unacceptable configuration, so larger concurrency limits are used where a risk of overcommitting resources is incurred.

**Adaptive Workload Management Technology**

The adaptive workload manager was created to address the inherent limitations in fixed concurrency limits; the key word here being “fixed”.

The adaptive workload manager performs concurrency control by admitting queries based on query resource requirements instead of fixed limits. In particular the adaptive workload manager considers the following query resources:

* Threads, where the effective query degree is used to estimate the threading resources required for a query and the total thread resources for the database is determined by the WLM\_AGENT\_LOAD\_TRGT (target thread load per cpu core) database configuration parameter and the number of CPU cores.
* Shared sort memory, where query sort memory requirements are determined by optimizer estimates or query history (if the query has run before) and the total shared sort memory for the database is determined by the SHEAPTHRES\_SHR database configuration parameter. Note that “sort memory” is a bit of a misnomer; this is really query working memory as it is used for numerous operations including sort, group by, hash join, flowing of columnar vectors etc. The sort terminology originates from legacy usage where sort operations were the primary consumer.

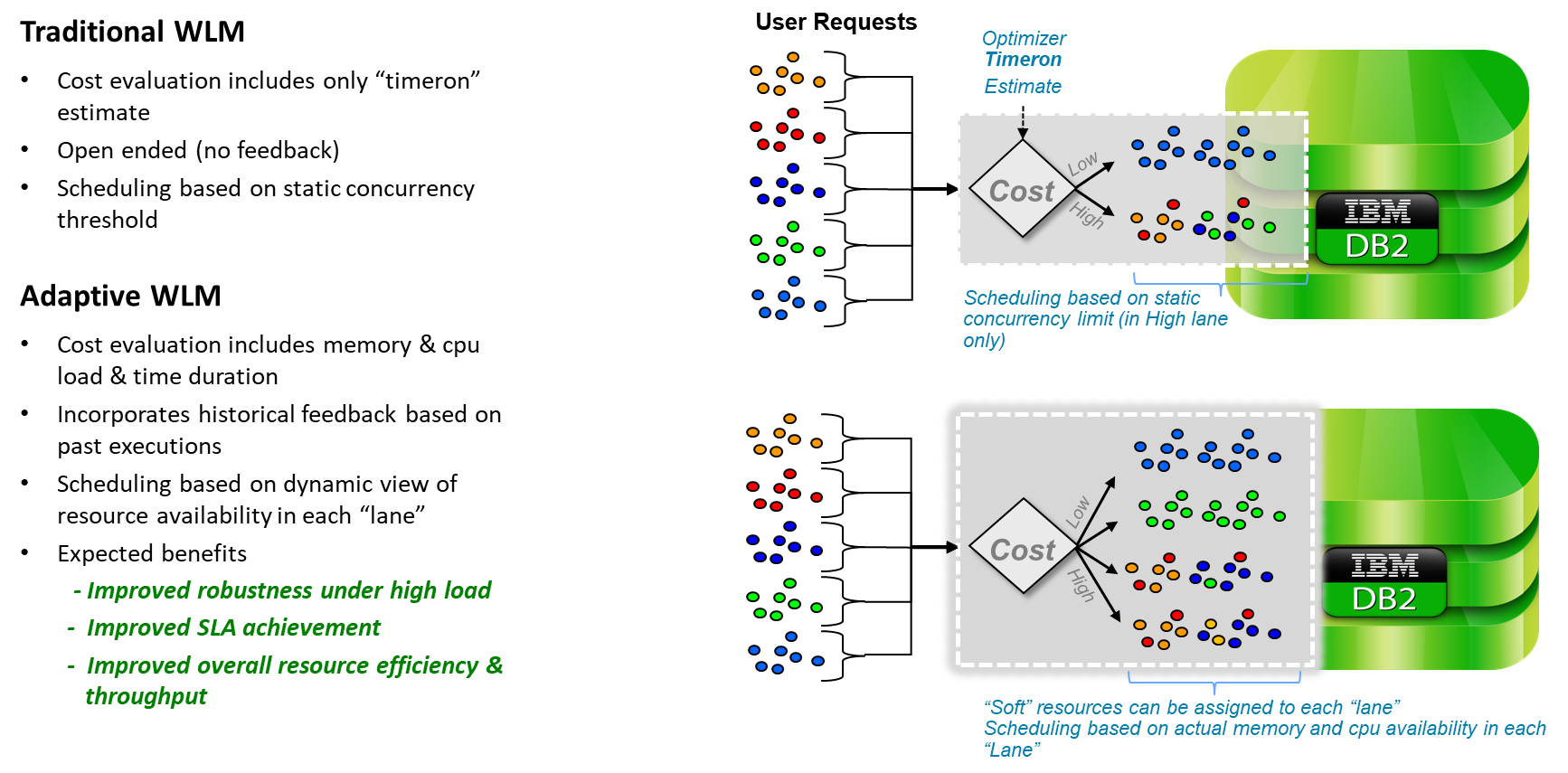
When a user submits a query for execution against the database, the adaptive workload manager estimates the resources required for the query and then performs a fit check against the resources currently available. If there are sufficient resources available then the query is allowed to execute, otherwise the query is queued until resources become available.

Resource based admission in effect becomes a flexible concurrency limit where concurrency is determined by total resources configured and the resource requirements of the currently executing queries. Let’s return to the example from the previous section where a query arrives that requires 90% of the memory followed by 50 queries requiring 2% each. It is not possible to configure a fixed concurrency limit in such a scenario that guarantees stability AND reasonable system resource utilization since stability requires a concurrency limit of 6 (5 queries using 2% and 1 using 90%), where best case utilization would be 100% but worst case would be 12% (if the large query completes execution ahead of the smaller ones). A higher fixed concurrency limit risks overcommitment of memory and failures. The same example with the adaptive workload manager would admit the query using 90% of the memory followed by 5 more queries using 2% each. If the 90% query completes, the remaining 45 queries using 2% will be admitted since there are now resources available, resulting in a concurrency of 50 queries and corresponding utilization of 100%.

In addition to resource based admission, there are a few other important features that are built into the adaptive workload manager:

* Short query bypass – Very short queries generally have a negligible impact on resource utilization and queuing these queries can have a detrimental effect on performance (i.e. if queries are short enough, the serialization required to enter/exit queue can exceed query lifetime). By default, queries with an estimated timeron cost of < 25000 or an estimated runtime of < 1 second will bypass the adaptive workload manager, effectively running uncontrolled. This remains consistent with the traditional Db2 workload manager best practices where it was recommended that trivial queries should enter the database without concurrency control. The one exception is that queries estimated to use > 2% of the total sort memory (sheapthres\_shr) will not bypass, since a relatively small number of such queries could overwhelm database resources.
* Intelligent job scheduling – Work is subdivided among different subclasses by estimated runtime where each subclass has a “soft entitlement” to a portion of the configured resources. This allows the adaptive workload manager to schedule work more intelligently by ensuring when there are concurrent queries of different sizes, some percentage of resources are always available for the shortest queries. A good analogy would be lanes on a highway where you want to make sure slow traffic stays in one lane and fast traffic in another so the fast traffic doesn’t get slowed down. By keeping the shortest queries in one “lane” and giving them a portion of resources, they do not get blocked by the larger work.
* Feedback mechanism – There are a number of feedback mechanisms informing the decisions made by the adaptive workload manager. Resource usage is constantly monitored on all database partitions and fed back into admission decisions such that if over-admission has occurred (e.g. if resource usage estimates were low) the admission subsystem can put on the brakes and queue work until resources free up. Conversely if we see a pattern of under admission, incoming resource estimates can be adjusted/reduced to allow for more admission and better overall utilization. In addition, the actual memory used by queries which have previously been executed is maintained in a history and used in place of optimizer estimates when queries are re-executed to improve the accuracy of admission decisions.

The following diagram provides a summary comparison of the traditional static, fixed limit concurrency control approach to the new adaptive workload manager:



The goals of the adaptive workload manager are:

* Deliver true automatic workload management out of the box with zero tuning. Concurrency adjusts the workload that is currently executing to provide stability and optimal resource utilization
* Removes need to configure and tune fixed concurrency limits
* Improved stability and performance
* Enables much simpler and more powerful user admission controls (to be covered in a future blog post)

If we revisit the requirements for a workload management system, we can see that just enabling the adaptive workload manager satisfies the first requirement of providing system stability and responsiveness. This replaces the older manual process of subdividing work by cost, defining fixed concurrency limits and monitoring/re-iterating the process as workloads change.

It is worth noting that the adaptive workload manager complements rather than replaces traditional Db2 workload manager capabilities. When the adaptive workload manager is used you can still make use of existing Db2 workload manager capabilities including workloads, service classes, thresholds and work action sets. The adaptive workload manager simply adds resource-based concurrency control into this mix.

**Enabling the Adaptive Workload Manager**

The adaptive workload manager is enabled by default on the following offerings:  
\* Db2 Warehouse on Cloud  
\* Db2 Warehouse  
\* Db2 On-Premise starting in v12.1.0 for newly created databases using either columnar tables or the database partitioning feature. Existing databases must “opt-in”.

To check if the adaptive workload manager is enabled, examine the WLM\_ADMISSION\_CTRL database configuration parameter; a value of YES means the adaptive workload manager is enabled.

If the adaptive workload manager is not enabled, a simple one-time migration process must be followed to create the default Db2 workload manager catalog objects that the adaptive workload manager depends upon. Migrate by invoking the WLM\_ENABLE\_ADMISSION\_CTRL stored procedure and then re-activating the database. After migration is completed admission control can be toggled on/off by updating the WLM\_ADMISSION\_CTRL database configuration parameter.

**Frequently Asked Questions**

1) I have an important user or application; can I configure the adaptive workload manager so that queries from this user/application bypass and never queue?   
  
No. The only way to maintain system stability is to ensure all non-trivial queries are managed. If an application or user were to unconditionally bypass the adaptive workload manager, large queries from this application or user could overwhelm system resources leading to instability. The notion of “never-queue” is inherently incompatible with stability. Furthermore, unrestricted resource utilization from the application or user could starve out other work. Future blogs will examine customization of the adaptive workload manager to specifically allocate resources to more important work to minimize queuing.   
  
2) Does the adaptive workload manager control runtime resource usage?   
  
No. The adaptive workload manager works by controlling the admission of queries (i.e. admit based on estimates of resources required and queue work if there are no resources available). Once a query is executing the adaptive workload manager has no direct control over the resources used.

3) Is the adaptive workload manager suitable for all workload types?

No. The adaptive workload manager is generally not suitable for pure OLTP type workloads, for example those commonly found on databases using the Db2 pureScale feature. For pure OLTP type workloads query-based concurrency control (either fixed limit or resource based) can lead to performance degradations if queuing occurs. For databases running only an OLTP type workload it is recommended to examine other workload management strategies including CPU controls and connection level pooling. For databases that are a mix of OLTP and Analytic workloads, the adaptive workload manager is still suitable, where the OLTP portion of the workload would bypass the adaptive workload manager.

**Conclusion**

In this blog we have examined the motivation for the adaptive workload manager and how this new technology is able to provide stability out of the box, achieving one of the key goals for database workload management. Upcoming blogs will discuss a range of topics including the resource prioritization and isolation using the simple user model enabled by the adaptive workload manager, monitoring resource usage and queuing, dealing with queries which have large resource requirements and handling queries with poor memory estimates.