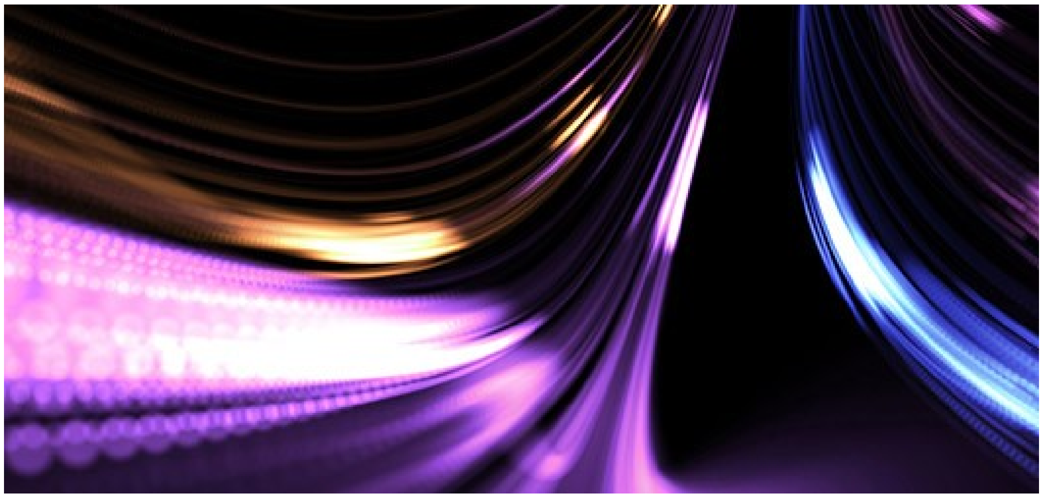
IBM Systems MEDIA

Metering and Capping on z/OS for Apache Spark

z/OS will continue to introduce new features that safeguard mission-critical workloads, satisfy all the other workloads, and fully and efficiently utilize system resources.



By Jessie Yu, Michael Gildein, Elpida Tzortzatos, Tom Rankin

09/11/2017

One of the strengths of z/OS is the ability to run multiple workloads simultaneously, within one z/OS image or across multiple images, while maintaining high system utilization. z/OS workloads have different, often competing, performance completion and resource requirements. The Workload Manager (WLM) component of z/OS is designed to balance these requirements by making the best use of system resources while maintaining the optimal throughput and system responsiveness.

Unlike many resource managers on distributed platforms, WLM allows the user to define performance goals and importance to each goal in business terms, rather than in hardware terms. WLM then decides how much resource, such as CPU and memory, to give to a workload to meet a goal. WLM constantly monitors the system and adapts processing to meet defined performance goals. However, WLM's primary focus is to ensure that performance goals are met, and it doesn’t always prevent a workload from overachieving and consuming a large amount of shared system resources.

For many new workloads on z/OS, having a means of controlling or capping resource consumption might be useful. IBM z/OS Platform for Apache Spark (z/OS Spark), for example, is known for its aggressive in-memory data caching to achieve performance gains. IBM Cloud Provisioning and Management for z/OS is another example that requires hard limits to support its multi-tenancy. The new metering and capping support for z/OS allows the system capacity planner more granular control over CPU and memory consumption for various workloads and enables the system to host new workloads more easily. For instance, by limiting the z/OS Spark service class, you can safely allow interactive data analysis to be performed on the same system as production transactions.

The new metering and capping support consists of two major features:

- Honor Priority by Service Class
- Memory Capping

Honor Priority by Service Class

Honor Priority by Service Class allows granular control over which work is eligible for z Systems Integrated Information Processor (zIIP) can be processed by the general-purpose central processors (GCPs). Before this enhancement, the IIPHONORPRIORITY option in the IEAOPTxx parmlib member controlled whether the GCPs could run zIIP-eligible work for all workloads on the system. This situation was not always ideal if you wanted to restrict certain workloads (such as z/OS Spark) to only zIIPs while allowing others (such as Db2) to obtain help from the GCPs.

As its name implies, the new feature adds an Honor Priority setting to the WLM service class definition. If you accept the default value for or specify the Honor Priority value to DEFAULT for a service class, then the IIPHONORPRIORITY setting is used to determine whether work within the service class can overflow to GCPs. If you specify the Honor Priority value to NO, then zIIP-eligible work within the service class is not allowed to overflow to GCPs, regardless of the IIPHONORPRIORITY setting. Since DEFAULT and NO are the only available options, you cannot enable work within a service class to overflow to GCPs if IIPHONORPRIORITY is set to NO. Note that this new feature applies to work eligible for zEnterprise Application Assist Processor as well.

Memory Capping

The other main feature, Memory Capping, puts an upper limit (in GBs) on real memory that can be consumed by address spaces that are associated with a WLM resource group. A new field, Memory Limit, is added to the WLM resource group definition. When a resource group is defined with a nonzero Memory Limit, the system creates a memory pool, which represents the accounting of physical frames used by address spaces that are associated with the resource group (these address spaces are also called pool members). The maximum size of a memory pool equals the Memory Limit value of the corresponding resource group. If a memory pool is approaching its limit, the z/OS system will take actions against its members, thus preventing negative impact to other workloads on the system.

As an example, you can define a WLM resource group with a Memory Limit of 10 GB and specify that all address spaces beginning with ‘SPARK1X’ exist in that memory pool. If the collection of physical frames used by SPARK1X* address spaces approaches the pool limit of 10 GB, the z/OS system will start paging out virtual pages that belong to those SPARK1X* address spaces to keep the pool below the memory limit.

The system can also slow down pool members in other ways if the pool is near or over its memory limit. Note that a memory pool can be over its limit because certain high priority system requests are always satisfied immediately. If the memory pool continues to stay over the limit, the pool members might even be terminated. To avoid performance degradation or pool member termination, you can dynamically increase the limit of a memory pool.

Note: You can use the IBM.Function.ApacheSpark fix category and the APSPARKF/K keyword in RETAIN to identify the PTFs that enable the new Meter and Capping support.

Using Metering and Capping Features for z/OS Spark

You can specify goals for z/OS Spark work the same way that you do for other z/OS workloads: by associating work with a service class. Apache Spark uses a master/worker architecture and consists of multiple processes. You can use the _BPX_JOBNAME environment variable to assign unique jobnames for these processes. Doing so allows the processes to be grouped easily into one or more WLM service classes by specifying a classification rule for the OMVS subsystem with a qualifier of jobname (transaction name).

As mentioned earlier, you can set the Honor Priority attribute to NO for your z/OS Spark service classes if you want to minimize the amount of z/OS Spark work processed by the GCPs. However, give careful consideration to setting Honor Priority attributes to NO on service classes, especially for highly utilized specialty processors.

For instance, a z/OS Spark service class that is restricted to zIIPs might consume much of the zIIP capacity and cause other zIIP-eligible workloads, such as Db2, to overflow to GCPs. In addition, z/OS Spark applications, by default, fall into the default OpenEdition MVS (OMVS) service class. If the default OMVS service class is restricted to specialty engines, other processes in the same service class, such as terminals or SSH sessions, might become unresponsive. z/OS Spark applications might also experience timeouts during driver and executor communication if they are waiting too long for CPU time.

Because z/OS Spark workloads are typically resource-intensive, you might consider defining resource groups to ensure that your z/OS Spark workload doesn’t impact other workloads that, in business terms, are more important. In addition, z/OS Spark executors are good candidates for having physical memory usage capped, since they tend to consume considerable memory.

You might, however, experience performance degradation or even termination of your z/OS Spark applications if they reach their memory limit. It’s a good practice, therefore, not to place the z/OS Spark master and worker daemons in a memory-limited resource group so that the z/OS Spark cluster remains alive even if the system terminates executors. Also, adjust your z/OS Spark settings, such as SPARK_WORKER_CORES and SPARK_WORKER_MEMORY, to match the WLM resource group specifications. Doing so would help z/OS Spark to dispatch work within its limits and reduce unintended consequences.

New Opportunities

New types of workloads are continually being introduced to z/OS. Accordingly, z/OS will continue to introduce new features that safeguard mission-critical workloads, satisfy all the other workloads, and fully and efficiently utilize system resources.

Bob Pfeiffer contributed to the technical and editorial review of this article.

About the author
Jessie Yu has 12 years of experience on the z/OS platform, doing design, development and test.

Michael Gildein is a z/OS Test Architect specializing in analytics as well as chief product owner of the Continuous Regression and Testware squads.

Elpida Tzortzatos is a Distinguished Engineer and IBM Z Architect working on IBM z/OS Core Design.

Tom Rankin is a tester in the z/OS Memory Management area.