Paper SAS2140-2016

Best Practices for Resource Management in Hadoop

James Kochuba, SAS Institute Inc., Cary, NC

ABSTRACT

SAS® solutions that run in Hadoop provide you with the best tools to transform data in Hadoop. They also provide insights to help you make the right decisions for your business. It is possible to incorporate SAS products and solutions into your shared Hadoop cluster in a cooperative fashion with YARN to manage the resources. Best practices and customer examples are provided to show how to build and manage a shared cluster with SAS applications and products.

INTRODUCTION

Hadoop can provide a platform for shared computing and storage. A Hadoop platform allows for many technologies to run against the same data. This creates an environment to access data with different tools and store the data with different formats. This also creates an environment to interact with the same data for computing capabilities like analytics. The Hadoop environment allows the user to choose different technologies to do all the work on the same hardware for optimal performance. The Hadoop environment is now the ultimate shared environment, allowing the computing to happen on the same hardware as the data storage.

SAS technologies can take advantage of this shared Hadoop environment, creating the ultimate analytic environment. SAS technologies can be grouped into four major areas that integrate with Hadoop:

1) SAS/ACCESS® Interfaces
2) SAS® In-Database Analytics
3) SAS® In-Memory Analytics
4) SAS® Grid Computing

The shared Hadoop environment can run SAS technologies as well as open-source tools and other products. When there is an abundant amount of resources, sharing the environment works great and people never have resource problems. Only when resources become constrained do people start to complain that their tools do not work and other tools are bad and should be restricted from the shared environment. The way to solve this is to use a resource orchestrator to determine when and how much each of the tools can use of the environment.

The audience for this paper is the users and administrators of the shared Hadoop environment. This paper helps to set the expectations and tuning choices when setting up and using a shared Hadoop environment. The paper will break down each SAS technology's integration with a shared Hadoop environment. Each section will discuss what this technology can do (from a high level), how it integrates with Hadoop, and some tuning advice.

GENERAL HADOOP RESOURCE UNDERSTANDING

Yet Another Resource Negotiator (YARN) was added to Hadoop to help make the environment a better shared environment for multiple tools. YARN addresses the need for a resource orchestrator. However, before taking a closer look at YARN, it is important to understand the two main methods of orchestrating resources.

The ideal resource orchestrator is a resource manager that can quickly dictate exactly who gets what resources to meet service level agreements and prioritize the requests for resources. Resource managers monitor and restrict real resource usage allowing the resources to be properly focused on

priorities. Resource managers do understand random very important requests and will move them to the top, pushing all other requests to the side, understanding true priority without permanently reserving resources.

The other option for resource orchestrator is a resource negotiator. A resource negotiator schedules when requests can use resources to meet service level agreements and priorities over time. Resource negotiators will use a picture of available resources to schedule the work. All of this sounds like a nicer way for a shared environment resource to be managed. But if users are expecting things to happen at a specific instance in time, a negotiator can have issues because it does not know if resources are available and it will have to negotiate when it can allow the new request to run.

YARN is a resource negotiator, not a resource manager.

YARN will schedule resources in a best effort but does not monitor the true resource usage to determine how things are really doing in the environment. YARN hopes the requestor will stay in the boundaries of the scheduled request. YARN does not care when a resource outside of the control of YARN, like Hadoop Distributed File System (HDFS) data node processes, take resources on the machine allocated to YARN, so this can cause resources to overcommit and create system issues. YARN's best effort for scheduling helps for the general usage, but has issues with sporadic very important requests. YARN treats each request as an independent application. This means that YARN does not understand any relationships between each application. This design can cause potential issues with applications that have dependences on other applications to run, and can cause major delays, deadlocks, or even failures. YARN does its best to negotiate resources in a shared Hadoop environment, but expectations need to be in check when resources become tight, because YARN is not a resource manager.

More details can be read at the Apache Hadoop web site, and Figure 1 is a picture of the design.

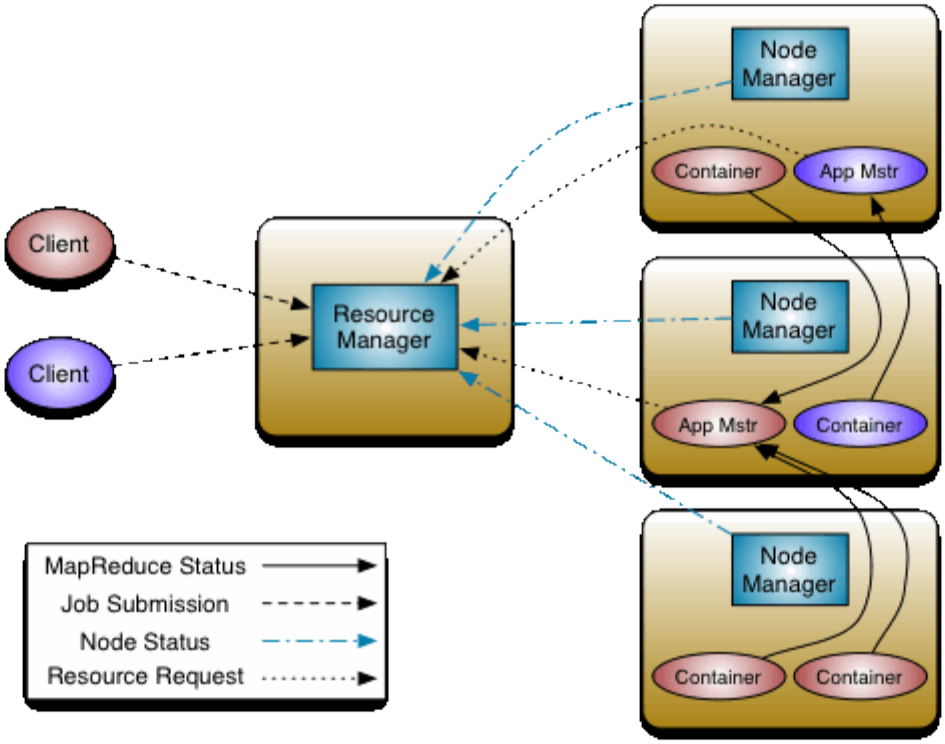

Figure 1. YARN Design, Duplicated from "Apache Hadoop YARN" at https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html

Overall, YARN provides two capabilities to all applications that want to participate with YARN on a shared Hadoop cluster:

1) Scheduling of resources
2) Containers to attempt resource restriction

Note that not all processes in a Hadoop environment participate with YARN. Example processes that don't participate: HDFS NameNode, HDFS DataNode, Hive Metastore, HiveServer2, and so on.

SAS technologies can participate with YARN, providing a better integration and experience with a shared Hadoop environment. This does mean YARN needs to be properly tuned for many different types of applications to run, not focused on a specific application profile. The following are specific YARN settings that impact running an application with YARN on the same hardware:

`yarn.nodemanager.resource.memory-mb`	Maximum amount of physical memory, in MB, that can be allocated for containers on each of the worker nodes in the Hadoop cluster. The default on some Hadoop clusters can be 20% or less of total RAM. This is problematic, for some high-performance procedures and in-memory processes can use more than that. When you set this value too high (>80% of available RAM) there might not be enough resources for background processing.
`yarn.scheduler.minimum-allocation-mb`	Minimum amount of physical memory, in MB, that can be allocated per container.
`yarn.scheduler.maximum-allocation-mb`	Maximum amount of physical memory, in MB, that can be allocated per container.
`yarn.nodemanager.resource.cpu-vcores`	Number of virtual CPU cores that can be allocated for containers. This value covers all applications and their containers running on this node or physical system. MapReduce and TKGrid containers are competing for limited reservation spots. For example: When set to 16, after 16 containers are allocated on this node, other jobs do not launch until the running jobs finish returning resources back to the node manager.
`yarn.scheduler.minimum-allocation-vcores`	The smallest number of virtual CPU cores that can be requested per container.
`yarn.scheduler.maximum-allocation-vcores`	The largest number of virtual CPU cores that can be requested per container.
`yarn.resourcemanager.scheduler.class`	The class used for resource manager. (Hortonworks and Cloudera used different defaults and currently prompt writing custom classes.)

Table 1. Hadoop YARN Settings

TUNING ADVICE

People need to understand that negotiations take time, and YARN negotiates the resources. There are processes that don't run in YARN that do run on the shared Hadoop environment. When a tool integrates with YARN, it can act differently when resources become constrained. So be sure to tune YARN for general application usage.

The default of most Hadoop installations is to tune YARN for equal containers as the minimum container size, focused more on MapReduce workloads. For example, a node with 8 cores and 256 GB RAM would have a YARN minimum container of 1 vcore and 32 GB RAM. Many MapReduce applications are small applications only requiring 2 to 4 GB RAM to run, therefore wasting the other 28 to 30 GB RAM reserved in YARN. This will cause other applications, like large in-memory applications, to queue up, waiting longer for resources to be negotiated even if resources are idle. For example, a node with 8 cores and 256 GB RAM with a YARN minimum container of 1 core and 4 GB RAM can still properly run

the MapReduce applications and reduce the negotiation time to get a large in-memory application, which can be requesting 1 core and 64 GB RAM per container.

Be aware that the nodemanager memory and vcore settings are global to all nodemanager with Cloudera Manager and Ambari UI. So with a heterogeneous hardware environment, this will require a more complex setup, with multiple separate node groups or removal of central management capabilities, to present the full available resources to YARN in the environment.

SAS/ACCESS INTERFACES

With Hadoop, there can be multiple data source interfaces to access the same data, such as Hive and Impala. Each data source interface has pros and cons based on what is expected and what is actually configured on the Hadoop platform. SAS/ACCESS software gives the SAS server the capability to interface with a specific data source on Hadoop, providing the best interaction with the data.

SAS/ACCESS Interface to Hadoop interfaces with HDFS, Hive, and Hive Server2. SAS/ACCESS Interface to Impala interfaces with Impala server. Both of these SAS/ACCESS interfaces can push SQL procedure calls and other SAS procedures (e.g., SUMMARY, MEANS, etc.), which convert to SQL-like code, to these data source servers to run in parallel on Hadoop. Then the data source servers will do the SQL-like work for SAS either inside the server process or scheduling with YARN for more resources.

The SAS server uses client XML files to interact with the data source server, Hive or Impala. The client XML settings can be tuned specifically for the SAS server and leave the Hadoop platform XML files for generic usage.

TUNING ADVICE

Since SAS/ACCESS Interface is just using Hive or Impala data source server to do the work, the tuning should be done at the data source server. The tuning advice is to have a separate YARN queue for SAS data source work requiring YARN resources, and remember this is determined by data source server. The specific queue will help separate the SAS YARN resources from all others using the data source server. Then the YARN administrators will know which applications are causing concerns and can tune YARN queues to do a better job negotiating resources when things get constrained over time.

Note that each YARN scheduler will have a custom way to create and tune YARN queues, so please review the details about the specific scheduler when creating the queue.

Note that the SAS libraries will not pass the all properties set inside the client XML files in the connection to the data source. The PROPERTIES= option can be added to the LIBNAME statement to add lost properties, like mapreduce.job.queuename, to the library connection. (See the documentation for the PROPERTIES= LIBNAME option in *SAS/ACCESS for Relational Databases: Reference*.)

Be aware that configuring the Hive back end to use Apache Tez or Spark can improve the performance of Hive YARN applications, but these back ends can hurt the performance of other applications on the shared Hadoop platform. Tez and Spark provide great performance improvements over MapReduce because they don't give back the resources to YARN to re-negotiate. Tez and Spark will keep the resources to complete the current task, even if the task is a low-priority item and high-priority items will wait for these resources. The tuning advice is to create a separate queue for all Hive applications and properly tune the YARN queue to not use up all resources in the cluster unless this application is the more important application that will ever run on the cluster. Because when resources are in contention, the Tez or Spark backed applications will not give up resources to run faster, holding resources from other applications in a shared Hadoop environment.

SAS IN-DATABASE

The SAS In-Database process is the SAS Embedded Process. This technology can be used in multiple functions:

- SAS® Scoring Accelerator
- SAS® Data Quality Accelerator
- SAS® In-Database Code Accelerator
- Parallel lift

The Scoring Accelerator runs SAS scoring code in Hadoop. The Data Quality Accelerator runs SAS data quality standardizations in Hadoop and is packaged in SAS® Data Loader for Hadoop. The Code Accelerator runs SAS DS2 code in Hadoop and is packaged in the SAS Data Loader for Hadoop product. The SAS accelerators are pushing the SAS code down to the Hadoop cluster to run, and they do not translate SAS code to another language like SQL. The parallel lift functionality is giving the SAS In-Memory technology a faster function to load data, in parallel, from Hadoop and is packaged in SAS/ACCESS to Hadoop.

The Embedded Process processing integrates as a MapReduce application that will be scheduled and controlled by YARN. Embedded Process does the majority of the processing, math computations, using C code that runs in the native Java process of a YARN MapReduce container. This means the total process size, which is what node managers monitor to determine if a container is compliant or not, is shared between the Java heap, only Java objects, and the native heap, lower-level process objects and C code usage. If a container is not compliant, the node manager will kill the container.

The following are MapReduce settings that impact running Embedded Process with YARN on the same hardware:

`mapreduce.map.memory.mb`	Memory size for the map task container.
`mapredure.map.java.opts`	Heap size for child JVMs of maps.
`mapreduce.reduce.memory.mb`	Memory size for the reduce task container.
`mapreduce.reduce.java.opts`	Heap size for child JVMs of reduce.
`mapreduce.job.reduce.slowstart.completedmaps`	Fraction of the number of maps in the job that should be complete before reduces are scheduled for the job. (Default behavior is to launch reduces well before all mappers complete.)
`mapreduce.job.queuename`	Specifies the YARN queue for all MapReduce applications used by this client.

Table 2. Hadoop MapReduce Settings

When Embedded Process is used for Scoring Accelerator, Data Quality Accelerator, or parallel lift, the MapReduce application is only a map task. The map task will do all the work for this functionality. When Embedded Process is used for Code Accelerator work, the MapReduce application can be a map task or a map and reduce task or a reduce task or multiple MapReduce applications. Code Accelerator work can create the multiple dependent applications to YARN, and YARN will treat each of these applications as independent work.

TUNING ADVICE

The majority of the time, the MapReduce default YARN settings are large enough when using the Scoring Accelerator, Data Quality Accelerator, or parallel lift function of the Embedded Process. When using the Embedded Process function in Code Accelerator, the memory settings for the MapReduce framework most likely will need to be increased for these applications to complete successfully. The MapReduce

process is a Java process that has two memory areas: Java heap and native heap. The SAS math capabilities run in the native heap area of the MapReduce process. The MapReduce settings can allow the overall container to increase in size (`mapreduce.map.memory.mb` or `mapreduce.reduce.memory.mb`) or increase container size giving the extra memory only to the Java heap (`mapredure.map.java.opts` or `mapreduce.reduce.java.opts`). When tuning the MapReduce YARN settings for Code Accelerator, only increase the overall container size.

Another tuning advice when using the Code Accelerator functionality is to place these application requests on an isolated queue. The isolated queue manages the scenarios when SAS DS2 code requires multiple MapReduce applications, creating an application dependency, to complete the SAS program. The isolated queue helps address YARN's limitation with dependent applications. Note that this does not mean dependent applications on the isolated queue will start right after each other with YARN. These applications will still need to negotiate with the overall YARN environment.

Be aware that the `mapreduce.job.reduce.slowstart.completedmaps` settings can be too aggressive giving away YARN resources to reducer task, which will be waiting for the application's map task to complete. The concern is the reducers can be given resources before all the map tasks were given resources, wasting resources on processes that are waiting for work and potentially creating a deadlock when resources are constrained in a shared environment.

SAS IN-MEMORY ANALYTICS

By default SAS In-Memory technology is not integrated with YARN. To enable the YARN integration, the TKGrid installation requires modifications to the resource.settings file. Once this file is modified properly (please refer to the "SAS Analytics on Your Hadoop Cluster Managed by YARN" paper), SAS In-Memory technology will schedule resources with YARN before using them.

When working with YARN, SAS In-Memory processes do schedule the work with YARN. SAS In-Memory processes do not run in the YARN containers, but these processes do honor the memory resources scheduled. With the limitations of YARN today, not running in the YARN containers allows SAS In-Memory technology to provide:

- Better resource management capabilities
- Flexible licensing

SAS In-Memory technology is better able to manage resources outside YARN, allowing SAS In-Memory capabilities to provide better error messages and better outcomes when resources are constrained. When running inside a YARN container, YARN only knows to kill a process when that process uses more resources than originally scheduled. If the process is killed, then SAS code can only provide a generic failed message because it can't determine if the failure was because of a resource issue (YARN killing it) or if the process actually had a problem and crashed. Because SAS® LASR™ Analytic Server is a long-running server, SAS LASR can load/unload new data any time for multiple users to access. If SAS LASR runs inside a yarn container and is killed, all users would lose the whole SAS LASR instance, requiring them to restart a new SAS LASR instance and remember to reload the previous data that fit inside the restrictions.

When SAS In-Memory technology runs outside the container, SAS In-Memory can see the resources are going over the requested limit and provide a more informative user message. This allows SAS In-Memory technology to gracefully shut down. Also, this allows SAS LASR to stop the new data from loading that would exceed the limit, while leaving the previously loaded data available. This provides a nicer environment for SAS programs while integrating with YARN.

SAS In-Memory products have core-based licenses. Based on the expected usage of SAS In-Memory technology, a sizing calculation can be created recommending the number of cores to achieve the business user's expectations. This sizing calculation might be less than the number of cores in the shared Hadoop cluster. SAS In-Memory products can be restricted to a specified number of cores on the cluster using cgroup, ceiling settings, allowing fewer cores to be purchased than the full Hadoop cluster.

Today, YARN does not fully understand this concept of an application type completely restricted to a limited number of cores on specific nodes. Therefore, SAS In-Memory processes running outside the YARN container allows the application instance to run only against a specific number of cores per the license while still using YARN scheduling.

TUNING ADVICE

SAS In-Memory technology allows the administrator to tune based on YARN and operating system settings. The usage scenario will determine which setting is the best option to restrict usage in a shared environment. The following table lists scenarios for which to use TKGrid settings to lock down the memory usage to the YARN requests:

	OS Ulimit (TKMPI_ULIMIT property)	In-Memory Memory Setting (TKMPI_MEMSIZE property)
HPA (hdat)	RECOMMENDED	REQUIRED
HPA (hdfs csv)	OPTIONAL	REQUIRED
HPA (Hive)	OPTIONAL — If used, then note that a Jproxy process is created on namenode to make Hadoop request for data, and this will have a virtual memory of ~15 GB loading all Hadoop JAR files.	REQUIRED
HPA (sas7bdat)	OPTIONAL	REQUIRED
LASR (create action)	REQUIRED	REQUIRED
LASR (add action, hdat)	NOT USEFUL — Since data is passed through, this process does not grow.	OPTIONAL — Data is passed through this process, so process does not grow.
LASR (add action, hdfs csv)	NOT USEFUL — Since data is passed through, this process does not grow.	OPTIONAL — Data is passed through this process, so process does not grow.
LASR (add action, Hive)	NOT USEFUL — Since data is passed through, this process does not grow.	OPTIONAL — Data is passed through this process, so process does not grow.
LASR (add action, sas7bdat)	OPTIONAL	REQUIRED

Table 3. TKGrid Memory Settings Recommendations for YARN

When using SAS In-Memory capabilities, many scenarios create a strong dependency relationship between the SAS In-Memory processes and data feeders, that is, Embedded Process. This means that in YARN, there will be two applications that have dependencies that YARN does not understand. These scenarios are where queues can help make sure the dependency is less likely to deadlock the system. Make sure the SAS In-Memory queue maximum resource usage is not 100% (e.g., 90% or 80%), and place the data feeder work in another queue to allow for data feeders to still run if needed.

Deep details about the resource.settings file customization be found in “SAS Analytics on Your Hadoop Cluster Managed by YARN” paper in “Appendix 3: Creating the resource.settings File” section.

SAS GRID COMPUTING

A SAS server will run a SAS program that includes many different actions for a user. The SAS server will determine if specific actions can take advantage of the above technologies or if the action will have to run

locally. The SAS server is a single server that potentially needs to handle a large amount of work, and hence SAS created SAS Grid. SAS Grid allows the SAS server to run on multiple machines to create a grid work environment. More details about SAS Grid can be found at "Introduction to SAS Grid Computing" on the SAS support site. With SAS Grid on Hadoop, the SAS Grid can run all of SAS on the shared Hadoop environment integrated with YARN.

SAS Grid on Hadoop will submit each single SAS program to Hadoop as a unique YARN application. The SAS YARN application will have an ApplicationMaster, managing the YARN application, and the application, where SAS program runs. The amount of resources needed by the application can be tuned based on the application type in SAS® Grid Manager for Hadoop configuration.

When a SAS program is submitted to SAS Grid Manager for Hadoop to run, a SAS YARN application on the Hadoop cluster will be started. If the SAS program has a specific action that can take advantage of a SAS/ACCESS interface, SAS In-Database function, or SAS In-Memory, then the SAS YARN application will submit another YARN application for that specific action to use the other technologies. This means a single SAS program can create multiple YARN applications with major dependencies to be successful. YARN will see and treat each application as independent unique applications.

TUNING ADVICE

SAS Grid jobs create two containers: one ApplicationMaster and one application. The ApplicationMaster is the Grid Manager and the application is a SAS workspace server. These SAS Grid jobs commonly run for a long time (much longer than Hadoop MapReduce tasks). So when a lot of SAS Grid applications are submitted, these can slowly take all resources from the shared environment, not allowing other applications to run. This is why creating a queue for SAS Grid applications and making sure this queue has maximum resources set to less than 100%, will make sure these SAS jobs don't take over the Hadoop environment.

Since the SAS Grid jobs are one ApplicationMaster and one application, multiple concurrent requests to do work can cause a deadlock. YARN deadlocks by giving all resources to only ApplicationMasters who will all be waiting for resources to create their application to run the SAS program. This is where configuring the queue's maximum running applications will make sure concurrent SAS Grid jobs requests to a queue don't allocate all resources with only ApplicationMasters.

Here is a simple SAS program example:

```
libname gridlib hadoop
        server=hadoophive.sas.com
        user=sasdemo
        database=default
        subprotocol=hive2;


proc hplogistic data=gridlib.testfile;
  class c1 c2 c3 c4 c5 c6 c7 c8 c9 c10;
  model y=joinvalue x1-x25;
  performance nodes=all details;
run;
```

The simple SAS program creates a library reference to a Hive data source and then performs a high-performance logistic regression on Hive data. The simple SAS program uses all the SAS technology by using SAS Grid Manager for Hadoop to run the SAS program, SAS/ACCESS to Hadoop to connect to Hive, SAS In-Memory to run high-performance logistic regression, and Embedded Process to parallel lift the Hive data into the SAS In-Memory instance.

A common scenario is a shared Hadoop cluster that has all the SAS technologies in Hadoop and YARN set up with two queues: a "default" queue for applications other than SAS, and an "A" queue for SAS user applications. When the user runs the simple SAS program, the execution becomes a complex YARN interaction with this scenario. The interaction steps are the following:

1. The SAS user submits the simple SAS program for SAS Grid Manager for Hadoop, which submits a SAS Grid job to YARN queue “A”. YARN sees the SAS Grid job and schedules to create the application manager first. Then the application manager will request YARN to schedule resources for the SAS workspace application, which runs the simple SAS program.
2. The simple SAS program running in the SAS workspace application executes the first line of SAS code, the LIBNAME statement, where the SAS workspace application will make a direct connection to the Hive server. This action is not managed by YARN and happens quickly.
3. The SAS program in the SAS workspace application moves to the next lines in the SAS code to call SAS In-Memory technology to run the high-performance logistic regression. SAS In-Memory technology, configured to use YARN, creates a SAS In-Memory instance, which submits a new application request to YARN queue “A”. YARN sees the SAS In-Memory request and schedules to create the application manager. The SAS In-Memory application manager requests YARN to create an application on all the nodes that make up the SAS In-Memory instance. YARN negotiates when all SAS In-Memory applications are able to run. Once YARN completes the scheduling of all the containers for SAS In-Memory applications, the SAS In-Memory instance can start to run the simple SAS program request.
4. The SAS In-Memory instance sees that the SAS program is requesting to run against data in Hive. The SAS In-Memory instance submits a new application request to YARN queue “A” to run the Embedded Process parallel lift functionality. YARN negotiates the new application to run when space is available. When YARN is able to schedule one of the Embedded Process applications to run, SAS In-Memory will start to receive the pieces of the Hive data from each Embedded Process application. Once all the Embedded Process applications run, the SAS In-Memory instance will have all the Hive data. Then the Embedded Process application is completed with YARN.
5. The SAS In-Memory instance runs the high-performance logistic regression against all the Hive data, and when it completes then the YARN SAS In-Memory application is completed.
6. The SAS Grid application sees the SAS program is completed because it ran all lines of SAS code and then SAS Grid application will notify the SAS client with results and complete the SAS Grid job in YARN.

This scenario shows how a simple SAS program creates multiple independent YARN applications that have strong dependencies making up the single SAS program. If any of these YARN applications fail to negotiate resources with YARN, the whole SAS program fails. If YARN kills any of these applications to help provide better prioritization to other applications, then the whole SAS program fails. If YARN only allows two applications to run on queue “A”, then YARN will deadlock this simple SAS program causing the end user to time out and fail the SAS program. Overall, understanding the common usage scenarios for a shared Hadoop cluster helps properly tune YARN to reduce the potential user interaction failures.

If the majority of the SAS programs submitted use SAS In-Database and SAS In-Memory technologies, then the maximum resource usage for SAS Grid Manager for Hadoop queue should be extremely low. The container size should be small and queue resource maximized. Tuning YARN this way will allow more resources to be given to SAS In-Database and SAS In-Memory technologies, allowing them to start up and run faster, which will allow all SAS programs to complete faster. The SAS programs will queue up waiting to start, but the design will allow for the overall system to perform better, completing more work.

CONCLUSION

YARN is the Hadoop answer to resource orchestration. SAS provides many technologies to run a SAS program that provide the best environment for the specific action. This allows the SAS program to run faster against larger data. All of these technologies can run on a shared Hadoop environment and integrate with YARN as a central resource orchestrator. YARN is not perfect, and negotiation for resources during resource constrained times can be slow to change. Because of this, there is no silver bullet or simple knob to turn that will make all things in a shared environment work perfectly.

The best tuning in the shared Hadoop environment for the YARN settings and OS will need to balance the software usage requirement and the business users' expectations. The best practice is to first set up a simple resource orchestration that meets license requirements. Then slowly tune the settings to remove less-important items from taking resources, or reserve more resources for high-priority items. Leave the middle-priority items as is, because these will be extremely hard to modify with a resource negotiator. Remember, tools act differently when resources become constrained with a resource negotiator. With this advice, the end result will be a shared Hadoop environment that will do the best it can, based on the priority of work to achieve good service levels and user experience.

REFERENCES

"Apache Hadoop Yarn." The Apache Software Foundation. June 29, 2015. Available at https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html.

"SAS Analytics on Your Hadoop Cluster Managed by YARN." July 14, 2015. Available at http://support.sas.com/rnd/scalability/grid/hadoop/SASAnalyticsOnYARN.pdf.

Troung, Kim. 2014. "How Apache Hadoop Helps SAS: SAS HPA and SAS LASR Analytic Server Integrate with Apache Hadoop YARN." Available at http://hortonworks.com/blog/apache-hadoop-helps-sas/.

SAS Institute Inc. 2015. "PROPERTIES= LIBNAME Option." *SAS/ACCESS 9.4 for Relational Databases: Reference*. Cary, NC: SAS Institute Inc. Available at http://support.sas.com/documentation/cdl/en/acreldb/68028/HTML/default/viewer.htm#p0ly2onqqpbys8n1j9lra8qa6q20.htm.

SAS Institute Inc. "Introduction to SAS Grid Computing." Available at http://support.sas.com/rnd/scalability/grid/. Accessed February 2016.

ACKNOWLEDGMENTS

I would like to thank Cheryl Doninger, Senior Directory of Platform Research and Development, and Tom Keefer, Principal Solution Architect, for technically reviewing this paper and adding more details.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

James Kochuba
Global Enterprise Architecture Practice
james.kochuba@sas.com