SAS® Analytics on Your Hadoop Cluster Managed by YARN

SAS, 2015-07-14

Contents

1. Introduction

Many people are trying to consolidate their big data processing onto a shared Hadoop cluster. Multiple analytic workloads co-located on Hadoop require workload and resource management to prevent a single application from consuming all of the resources. YARN (Yet Another Resource Negotiator) provides that resource management in a Hadoop environment. SAS is assisting this consolidation by adding YARN support to a number of its products. See Appendix 1 for a list of SAS 9.4M3 products which work with Hadoop and can be managed by YARN.

SAS Grid Manager for Hadoop (SAS Grid), SAS Visual Analytics (VA) and SAS High-Performance Analytics (HPA) are all distributed technologies that run on a cluster of computers. Although these products are different in many ways, SAS has integrated each of these technologies with YARN to enable coordination of their resource usage when running on a shared Hadoop environment. This allows you to install these products in a shared Hadoop cluster and to use YARN as their overarching resource manager. SAS Grid, VA, and HPA workloads will coordinate with other YARN-managed workloads. When integrated with YARN, these products can be managed and monitored like other YARN applications.

This paper explores ways to run these three products together on a single shared Hadoop cluster managed by YARN. This paper will share the results of running these SAS products using YARN for resource management along with some lessons learned. A Hive workload will also be added to the cluster to determine the impact of an additional workload on resource consumption.

2. Summary of Results

1. YARN is a viable resource manager for SAS 9.4 M3 workloads which include SAS Grid Manager for Hadoop (SAS Grid), SAS Visual Analytics (VA) and SAS High-Performance Analytics (HPA) .

2. Parameters useful for managing SAS workloads under YARN on Linux include:
   YARN: memory, vcores, queue, and YARN scheduler.
   Linux: nice value, virtual memory ulimit, CPU cgroups.

3. It is important, but not always easy, to have a meaningful estimate of memory usage for different workloads, because memory is a primary resource managed by YARN. It is challenging to fully utilize memory because overcommitting memory can lead to errors and crashes.

4. Understanding the TKGrid resource.settings script is important for configuring VA and HPA integration with YARN. SAS Grid uses sasgrid-policy.xml to configure its YARN integration.

5. When using YARN to manage SAS workloads, it is convenient to use Linux nice values to balance resource usage. However, CPU cgroups managed by YARN and managed outside of YARN can help with more difficult resource management challenges.

6. When using YARN to manage these SAS workloads, it is helpful to define a separate YARN queue for each workload type. It may take some experimentation to get queue definitions that balance diverse workloads the way you want.

3. Resource Management under YARN

YARN is the resource manager for Hadoop clusters. A detailed explanation of YARN is beyond the scope of this paper, however we will provide a brief overview of the YARN components and their interactions. In response to a resource request by an application master, YARN (specifically, the Resource Manager) returns an allocation for a certain number of containers using a certain amount of memory (MB) and CPU (virtual cores or vcores). The application master submits this approved allocation along with the programs it wants to run to the assigned node managers which then run the programs.

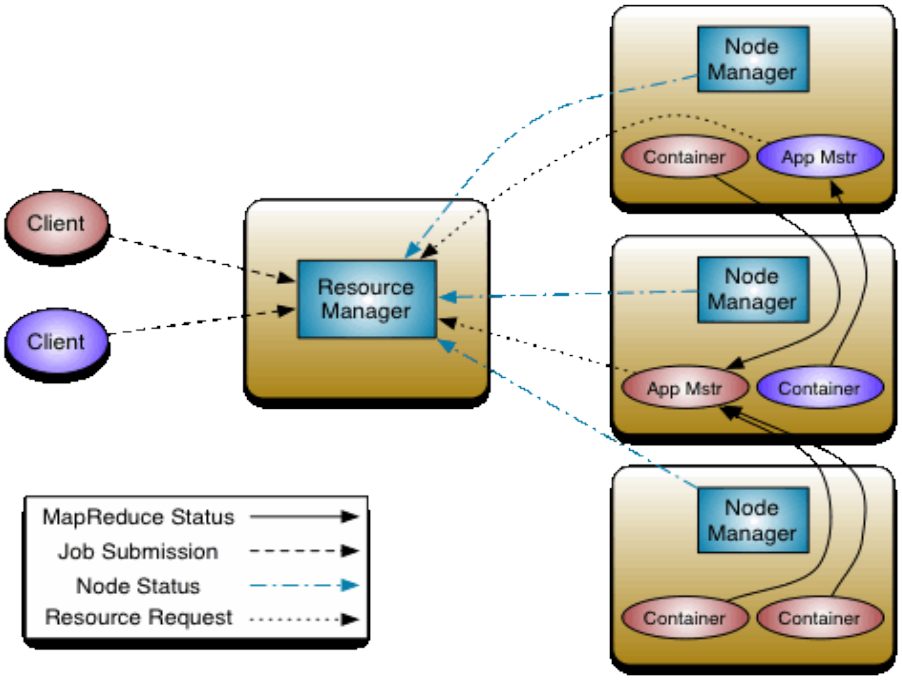


Diagram from http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html

As applications (or jobs) on a cluster compete for resources over time, YARN tries to fairly distribute those resources to meet the varying demands. YARN evaluates resource requests in light of its assigned resources, which resources are in use, its scheduler logic, and defined application queues (or pools). As one application completes and frees resources, YARN will seek to assign those to the next waiting application.

Although YARN is allocating resources, it is not necessarily constraining resources. That is, a container assigned a certain amount of memory and CPU is not necessarily limited to those assignments unless YARN is also configured to enforce those limitations. (We discuss some options for constraining resources in YARN in the rest of this paper.) Also, applications often use less than their assigned memory and CPU since resource assignments are theoretical maximums. YARN launches subsequent jobs as if applications are using all their assigned resources.

To make the best use of YARN, it is helpful to have an understanding of the resource needs of your applications. Such predictions can be difficult to make. However, the more you use YARN with your applications, the more you will be able to refine your understanding of resource requirements and the interactions of concurrent applications.

4. YARN Interacting with SAS Grid, VA and HPA

The workloads of SAS Grid, VA and HPA are very different and therefore the ways in which these technologies interact with YARN differ.

SAS Grid submits one SAS job at a time to be run on one node in the Hadoop cluster. SAS Grid includes a SAS Grid application master so that it can run SAS jobs through the YARN node managers. These SAS jobs have their resources managed like other YARN containers. SAS Grid allows the user to define different application types where each application type has a certain number of vcores, amount of memory, and YARN queue assignment. An application started by SAS Grid can be assigned to any of the defined application types.

VA and HPA interact with YARN in a different fashion from SAS Grid. VA and HPA both make use of the SAS High-Performance Analytics Environment (TKGrid) infrastructure. VA and HPA continue to execute their work on the cluster through TKGrid but interact with YARN though a proxy application master. Therefore, VA and HPA processes on the cluster coordinate with YARN but are not directly managed by YARN. Different VA actions and the various HPA procs can run with distinct configurations for their YARN resource requirements. Those configurations are fixed before start up. It is possible to specify a memory allocation, vcore count, and YARN queue as well as other parameters. Although VA and HPA resource usage is not constrained by the usual YARN constraints, the TKGrid infrastructure limits resource usage through its own mechanisms. We will examine how to configure TKGrid resource usage below.

VA is different from the HPA procs because VA is typically started as a long running server with which many users will interact. An HPA procedure runs and returns an answer to a single user. When we run VA with its YARN integration, VA will appear as a running YARN app for as long as the VA LASR server is running. VA runs on top of this LASR server. The LASR server is a TKGrid distributed application. When LASR is running on the Hadoop nodes, it can be coordinated with YARN as a YARN application. VA/LASR resources (containers, memory, and vcores) might be committed for long periods of time. If VA/LASR is running with fixed resources for long periods on a cluster, the cluster administrator may decide to remove those resources needed by VA/LASR from the pool of resources know to YARN and let YARN handle management of the rest of the system resources. Or, if the resources used by VA/LASR will change over time, then VA/LASR can be run under YARN along with everything else.

5. SAS Embedded Process

The SAS Embedded Process (EP) enables user-created or application-generated DS2 code to run in the Hadoop cluster. It is a light-weight, multi-threaded distributed component used for: table manipulation, scoring, data quality and delivering data to TKGrid or back into the Hadoop cluster after processing. VA

and HPA can directly access data stored in HDFS in the SASHDAT format. Such access occurs in a high-performance distributed fashion. The EP allows VA and HPA to access data in Hadoop stored in data formats besides SASHDAT and to access that data in a distributed fashion. The EP uses Map-Reduce for interacting with HDFS, and therefore can be managed by YARN as a MR application.

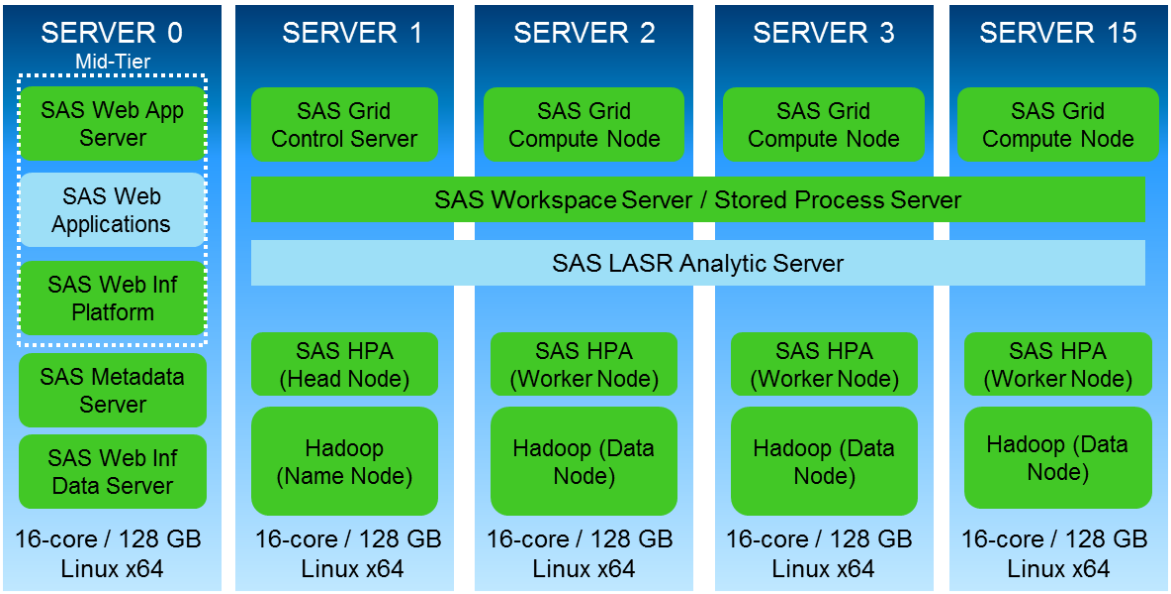
As of the SAS 9.4 M3 release, the EP runs fully inside an MR container and is fully managed by YARN. Because the EP is fully integrated with Map-Reduce, the MR container size may need to be increased compared to the previous version of the EP. Please see Appendix 1 for an example of an error thrown when the EP runs out of memory in its MR containers.

In our testing, we mostly ran with data in SASHDAT format. However, we did repeat some tests using the EP and data in SPDE format to confirm that functionality. Because the EP makes use of a number of MR containers for data transfer, we did observe a modest increase in resource usage.

6. SAS/ACCESS Interface to Hadoop

SAS/ACCESS to Hadoop is another SAS product that runs as a YARN application. This product makes use of Hive, so it runs as a Map-Reduce job on YARN. SAS/ACCESS to Hadoop is tuned as a normal Hive and MR application. We did not make use of this product in our testing, but we mention it as another example of how SAS is providing various ways to work with the Hadoop ecosystem including YARN.

7. Our Test Environment



Our test results reflect our particular test environment, workloads, and configuration. Your environment may show different results. We used a 16-nodecluster of commodity Linux boxes with Hadoop 2.5. Because SAS Grid requires shared storage, each node was attached to SAN storage which was shared using GFS2. We had 14 Hadoop data nodes. We ran with SAS Grid, VA and HPA workloads.

The SAS Grid workload consisted of common SAS procs such as SUMMARY, FORMAT, and DATASETS as well as DATA steps to do such things as create indexes and do file I/O. Each workload created a single SAS session on a single data node. We varied the number of concurrent SAS sessions running, but for most of our final tests we would submit a request for 48 SAS sessions to YARN. YARN would then spread those across the 14 data nodes. For the 2 hour test period, when one session completed, we would submit a request to start a new session so YARN was always handling 48 sessions. Because each SAS Grid job runs only on a single data node, our SAS Grid jobs are light users of resources. One job is usually using only 5%-10% of a single data node's available CPU resource.

The VA workload simulated a number of users of SAS Visual Analytics Explorer using the web interface to visualize data. Each user creates a variety of reports including correlation reports, line charts, bar charts, and some forecasting. The size of the data set used was 112GB distributed across the 14 data nodes. For our tests, we were always running 54 concurrent test users. This workload is a heavy CPU user, using over 80% of available CPU on the cluster most of the time.

The HPA workload loops through a cycle of HPA procs including HPNEURAL, HPLOGISTIC, HPSAMPLE, HPMDB, HPREDUCE, and HPTREE. Each procedure executes across all 14 data nodes. We varied the number of concurrent instances of this workload. For most of our final tests, we were submitting 8 instances of this workload concurrently, but YARN would decide how many instances (where each instance is a YARN application) it could run at any particular time. The HPA jobs are generally not heavy CPU users. One instance is typically using less than 10% of the cluster CPU though an instance can peak at around 50%.

8. YARN Resource Management Tuning

YARN is configured to manage a subset of the hardware resources available including memory, vcores, and queue assignment. The balance of the resources are managed by the OS alone. Each Hadoop vendor has their recommendations on how best to configure YARN. That will be the starting point from which you will work to adjust YARN to the particular needs of your environment.

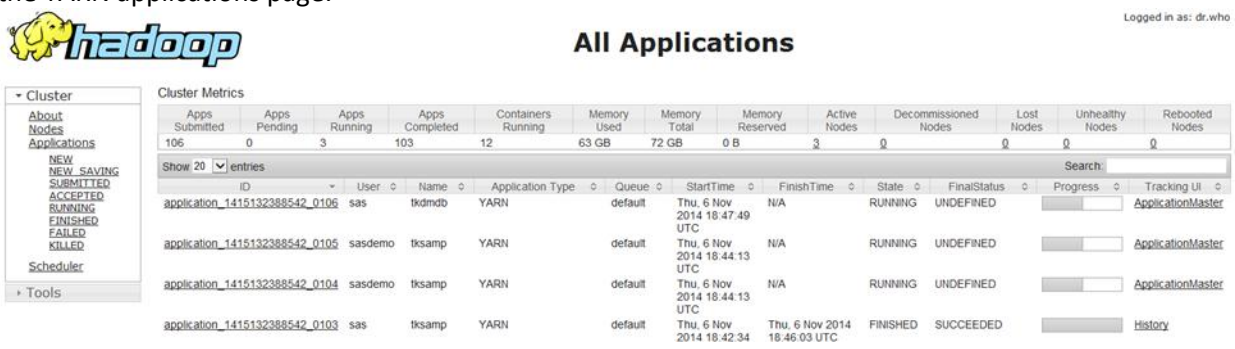
YARN will launch as many jobs as it has resources to support. When YARN evaluates whether or not to grant an application master request for resources, it weighs several factors:

1. Resources assigned to YARN for management.

yarn.nodemanager.resource.memory-mb: Amount of physical memory on each node, in MB, that can be allocated for containers. If you set this value too low, you may be leaving memory unused on your data node. If you set this value too high (over 80% of RAM), you may not have enough memory for other necessary processing not managed by YARN.
yarn.nodemanager.resource.cpu-vcores: Number of virtual CPU cores (vcores) on each node that can be allocated for containers. A container must have at least one available vcore in order to run. Therefore, the number of vcores limits the total number of containers that can run. The number of vcores can be set much higher or lower than the number of physical cores, but start with your vendor's recommendation. With YARN, a multi-threaded container can use unlimited CPU even if assigned to only one vcore. For more, see below in "Estimating Memory and Vcores".

2. Resources currently committed.

You can monitor resources used in YARN at http://myhadoop:8088/cluster/apps. This is an example of the YARN applications page:



YARN does not track resources actually used by an application when calculating resources available. It only tracks resources it has granted to an application. By default, YARN will allow running jobs to complete. As some of a job's containers complete, YARN may give a smaller fraction of the system resources to subsequent containers for that job if new applications are waiting for resources. YARN can be configured to preempt some containers by killing them in order to free up resources for waiting applications.

3. Queues.

You can monitor queues at http:// myhadoop:8088/cluster/scheduler. Fair Scheduler and Capacity Scheduler queues have different attributes, but their functionality is similar. For most of our tests we used the Fair Scheduler. In the Fair Scheduler, when applications are assigned to two different queues, and those applications are competing for resources, applications in a queue with weight 2 should receive approximately twice as many resources as applications in a queue with the default weight of 1. If there is no competition for resources, an application is free to use all the resources it requests.

Other important YARN configuration settings:

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.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.

Note: If (yarn.nodemanager.resource.memory-mb / yarn.nodemanager.resource.cpu-vcores) is greater than yarn.scheduler.maximum-allocation-mb, then YARN will not be able to assign all the memory it is managing.

9. Considering Map-Reduce (MR) Configuration

SAS Grid, VA and HPA all interact with the YARN configuration, and in many cases, are also affected by the Map-Reduce configuration. For example, SAS Grid uses Oozie for workflow scheduling which uses the MR framework. HPA and VA often use the EP to load data which also uses the MR framework. You need to consider the MR configuration used by other applications running on the same Hadoop cluster since other MR applications will be sharing resources with SAS Grid, VA and HPA. Also, if you are making use of SAS Embedded Process (EP) with VA or HPA, then the EP will be affected by the MR configuration.

You should tune your YARN configuration to optimize for all applications on the cluster including SAS Grid, VA and HPA. Then use your MR configuration for fine-tuning how MR jobs run. For instance, you should not set yarn.scheduler.minimum-allocation-mb too high just to insure you have larger MR containers when you can control MR container size with MR configuration.

Map-Reduce Configuration to consider:

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 which should be complete before reduces are scheduled for the job.

Note: In general, we recommend setting mapreduce.job.reduce.slowstart.completedmaps to 1 in order to avoid a deadlock condition where reducers are started which deprive mappers of necessary resources while those reducers are waiting for those same mappers.

SAS Data Loader for Hadoop, specifically the Code Accelerator and Data Quality Accelerator features, and SAS Scoring Accelerator products use the SAS Embedded Process to run SAS DS2 code in the cluster using the Map-Reduce framework. As mentioned above, in SAS 9.4 M3, the Embedded Process runs inside the Map-Reduce Java processes which are fully managed by YARN. This allows our Accelerator capabilities to be fully YARN-compliant and managed. This also means you need to properly tune the Map-Reduce properties to size these to allow for the processing power and memory needed to complete successfully. These settings can be tuned in the mapred-site.xml on the client side (where SAS server runs) so they are not Hadoop cluster wide settings.

10. SAS Grid Resource Management Tuning Parameters

SAS Grid uses the sasgrid-policy.xml file (policy file) to describe the resource usage by its different application types. For an application type, it defines a number of parameters including: memory, vcores, queue, and nice value. Because it runs as a YARN container, its cgroup assignment is controlled by YARN. When you launch a job under SAS Grid, you are able to assign the application type defined in the policy file. See Appendix 3 for a simple sasgrid-policy.xml file that we used in testing. It defines a single application type. The sasgrid-policy.xml file can appear in several possible default locations. We stored it in HDFS under /tmp/SASGrid.

SAS Grid also makes use of Oozie for workflow scheduling. Oozie uses the Map-Reduce framework. If you are using Oozie, you should consider how your MR configuration will affect Oozie's behavior.

11. Tuning for VA and HPA on YARN

In order for VA and HPA to make use of YARN, the sas.grid.provider.yarn.jar file must be in the <HADOOP_HOME>/lib directory and a resource.settings file must be defined in the TKGrid root directory. VA and HPA both use TKGrid to execute their work. VA and HPA resource configuration for YARN is done in the TKGrid resource.settings script. This script allows us to distinguish VA/LASR actions and HPA procs by application name (called TKMPI_APPNAME in resource.settings) and give each procedure different resource settings. Appendix 2 provides additional details for creating a resource.settings file. A copy of this file must be deployed across all the TKGrid nodes. The sample file provided in the TKGrid distribution is a simple example and is commented out by default. When the file does not exist or is commented out, YARN is not used by TKGrid.

When interacting with YARN, TKGrid requests a reservation for a container on each node and does not execute until all containers obtain a reservation. The resource.settings file describes the YARN containers used by each TKGrid application running on a particular YARN cluster including VA/LASR actions and HPA procedures.

The TKGrid resource.settings file contains a number of configuration parameters that control VA and HPA resources: memory (TKMPI_MEMSIZE), vcores (TKMPI_YARN_CORES), and queue name (TKMPI_YARN_QUEUE). We can also set a Linux nice value (TKMPI_NICE), "ulimit -v" (TKMPI_ULIMIT) to control maximum virtual memory used, and a Linux cgroup name (TKMPI_CGROUP). In our testing, we mainly used settings for memory, vcores, queue, and nice value. We also tested using "ulimit -v" and cgroups, but these settings were not necessary to achieve a good balance between our particular concurrent workloads. The various settings of the resource.settings file are assembled into a command line named TKMPI_RESOURCEMANAGER which issues the request to the YARN resource manager.

In general, VA and HPA will limit their memory usage to the setting specified in TKMPI_MEMSIZE and will return an error to the SAS log if their memory usage exceeds that limit. However, when VA makes use of SASHDAT type files, it will memory map these files which will not be counted against the TKMPI_MEMSIZE. The user is able to limit VA memory usage including memory mapping in several ways including using TKMPI_ULIMIT in resource.settings or using the VA Administrator UI and going under the "LASR" tab to set "Tables Limit (MB)". If the VA/LASR server is created with TKMPI_ULIMIT set to the same as the TKMPI_MEMSIZE, then when a LASR table is loaded that exceeds that limit, the load will fail. If you use "Tables Limit (MB)" in the VA Administrator UI to limit table size, a table load that begins under the limit will continue even if it exceeds the limit, but subsequent table loads will fail. You should always set TKMPI_MEMSIZE for an HPA procedure or LASR create action you want coordinated by YARN because YARN will use this setting for the container size. You do not need to set TKMPI_ULIMIT for an HPA procedure, but you should set it for a LASR create action if you want the LASR memory limit to be enforced.

When to use TKMPI_ULIMIT or "ulimit -v" also depends on the storage type for your data. The table below summarizes a suggested approach for various storage types.

	ulimit –v	TKMPI_MEMSIZE
HPA (hdat)	Optional	REQUIRED
HPA (hdfs csv)	Optional	REQUIRED
HPA (HIVE)	Concern	REQUIRED
HPA (sas7bat)	Optional	REQUIRED
LASR create	REQUIRED	REQUIRED
LASR add (hdat)	Not useful	Not useful
LASR add (hdfs csv)	Not useful	Not useful
LASR add (HIVE)	Concern	Not useful
LASR add (sas7bat)	Optional	REQUIRED

Optional: may give additional memory restriction, but its use is not necessary.
Not Useful: not needed because this process streams data to another process and should not be coordinated with YARN.
Concern: Not recommend because Hive access uses a jproxy process on the namenode requiring a large amount (>15GB) of virtual memory.

12. Estimating Memory and Vcores

It can be a challenge to estimate memory usage for an application. To begin with, it is not always clear what "memory usage" means. A program can have dedicated real memory, but it can also link in shared libraries. These shared libraries may be referenced by other running programs without taking up additional memory. A running program may have some of its memory paged out. When a program does file I/O, it may cause some data to be cached into memory. A program can memory map a file so that data is paged into memory as referenced. A long running program may have a brief spike of high memory usage.

Memory is often the most inflexible of the resources used in computing. It is often possible to throttle CPU and I/O available to a program without impacting the logic of the program. However, how a program uses memory is tightly tied to its logic and data structures. Cut a program's CPU in half, and it will probably take twice as long to execute. Cut its memory in half, and it may crash or halt before completion. If your memory estimates are much bigger than they need to be, then you may have underutilized resources. If your estimates are smaller than they need to be, your programs may not complete.

Sometimes it takes some trial and error to determine what container sizes will successfully execute most programs without unnecessarily tying up large amounts of memory. Watching running programs and looking at logs of past programs can help your estimates. The TKGridMon utility is very helpful in monitoring actual memory usage by every application running under TKGrid including VA and HPA procedures. YARN tracks memory based on the requested maximum size for a particular container. If so configured, YARN can kill a container that goes beyond a physical memory and/or virtual memory limit (in Apache Hadoop 2.5, pmem-check-enabled is true by default which causes YARN to kill a container that goes beyond the max memory size.).

Estimating vcores needed by an application has its own challenges. By default, Apache Hadoop 2.5 does not limit the amount of CPU used by a vcore. If cgroup control is enabled in YARN, this also does not put a strict limit on CPU used by a vcore. Apache Hadoop 2.6 does provide a way to strictly control CPU, but in practice, this is often not needed. If a program running in a container is multi-threaded, it will use what CPU it needs until it is competing for CPU with other programs. Then the OS will balance the CPU usage. A reasonable starting point is to roughly scale the number of vcores requested to the amount of memory requested. If most programs are running with 4GB containers with 1 vcore but a second program needs 8GB containers, you might consider assigning it 2 vcores per container. If in doubt, start by requesting 1 vcore per container.

For our tests, we tried several memory allocations and settled on the following:

Memory per data node	128GB
Memory managed by YARN	80GB
Memory for VA per container	40GB
vcores for VA per container	12
Memory for HPA per container	8GB
vcores for HPA per container	1
Memory for SAS Grid per container	4GB
vcores for SAS Grid per container	1

We gave YARN only 80GB out of 128GB of system memory in order to create more competition for the memory resource. Because VA and HPA aim to keep most data in-memory, an estimate of their memory usage can begin with the size of the data they will be loading to memory. Also, VA and HPA both request one container per data node. In the case of HPA, we settled on 8GB containers because the most demanding procedure needed that much memory. Even though we knew the test data would fit in the memory allocated for VA, we still set "ulimit -v" to 40GB for VA and confirmed that loading data above this limit failed with a clear error message.

13. Memory Recommendations for HPA and VA/LASR

Tables 2 and 3 highlight the procedures used in TKGrid along with a configuration recommendation for the resource.settings file and if they should even be managed at all.

TKMPI_APP NAME / Procedure	Description	SAS Code	Application Duration	Recommended RAM	Define in resource.settings file
tkhpatest	In-memory file test	proc hpatest	Short running	1.25GB container	Maybe
lasrhoo	Grid monitor job, SAS/ACCESS engine for SASIOLA	LIBNAME lasrlib SASIOLA	Long running	1GB container	No
lasracc	SAS/ACCESS engine to Hadoop	LIBNAME hivelib HADOOP	Long running	1GB container	No
lasr	In-Memory table space	proc lasr	Long running	1GB container	No
tkds2gsrv	SAS LASR Analytic Server In-memory table space.HPDS2 (data movement)	proc hpds2	Long running High-performance PROC	1.25 size of files 1GB container	Yes Maybe
tksamp	HP sample	proc hpsample	High-performance PROC	2.25 size of files	Yes

Table 2. Common TKGrid Procedures

Table 3 outlines high-performance PROCs and suggested RAM sizes for each type in the resource.settings file.

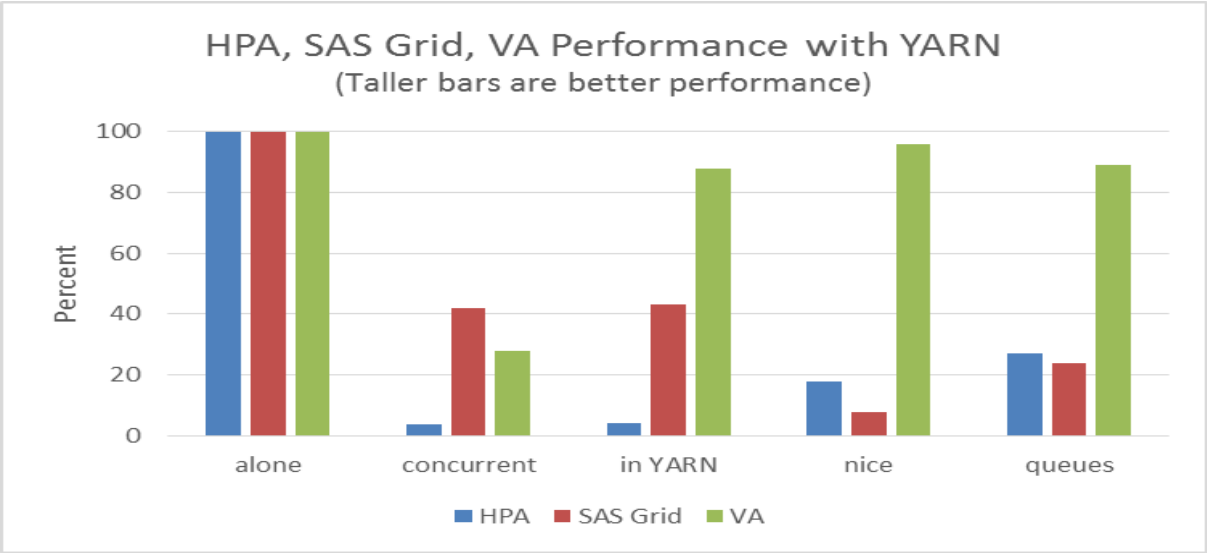
TKMPI_APPNAME / (The high-performance PROC Hadoop YARN uses.)	**Container RAM Sized Needed for Executing.**
HP procs validated for this paper: tklogis, tknueral, tktmine, tkreduce, tkreg, tkhp3, tkhpsumm	1 to 1.25 times file size
Other HP procs which were tested for this paper	1 to 1.25 times file size

Table3. High-performance PROCs and suggested RAM sizes.

14. Test Results of Resource Management with YARN

The goal of our testing is to achieve the following balance: to preserve the interactive response time of VA while still giving HPA predictable run times for its procs. We assume SAS Grid jobs are mostly batch (as compared to VA and HPA), so they are the lowest priority, but we still want SAS Grid to be able to execute a reasonable amount of work on the cluster.

To measure performance, for SAS Grid and HPA, we measure the number or procs executed during the test time, and we measure how long each procedure runs. For VA, we measure only the response times of user transactions, and we take the normalized reciprocal of this time. In all cases, better performance (more transactions and smaller response time) will show up as a taller bar and less performance will be a shorter bar. For SAS Grid and HPA, a score of 50 means it completed half as many procs compared to a score of 100. For VA, a score of 50 means the response time was twice as long compared to a score of 100. Our goal visually is to keep the VA bar nearly as tall as when VA runs alone and to get the HPA and SAS Grid bars as tall as we can without shortening the VA bar too much.



We began our testing by running SAS Grid, VA and HPA each alone to see their performance with no competition. In the chart above, the first cluster of columns is the performance of each package running alone on the grid with no competition. This performance then becomes the normalized measurement of 100.

In the second cluster, we ran all three technologies concurrently with VA and HPA outside of YARN and with SAS Grid able to use all YARN resources. In this scenario, the 3 technologies are competing for resources only under the control of the OS. In this case, VA response times are not preserved but are nearly 3 times (score is about 33) what they were without the other two packages running. This does not preserve VA response times and HPA is not able to do a meaningful amount of work.

In the third cluster of columns, we run all the packages under YARN but without any nice values and without defining separate YARN queues. VA response times are restored, but HPA is still not able to do much work.

In the fourth cluster of columns, we assign nice value of 15 to SAS Grid and of 5 to HPA. These results are acceptable according to our goals since VA response time is preserved, HPA is able to do some work, and SAS Grid is also doing a little work.

In the fifth cluster of columns, we continue assigning nice values as in the previous test, but now we also assign each package to a separate YARN queue. After a number of tests, we settled on queues defined like this:

	HPA	SAS Grid	VA
queue weight	3	1	1
max running apps	4	32	n/a
policy	DRF	DRF	DRF

In this case, VA response times are still acceptable being only a little slower than when VA runs alone. SAS Grid and HPA are both doing reasonable amounts of work (about 25% of the workload they could accomplish when each was running alone), in particular, more work than when only nice values are used.

With the Fair Scheduler, "queue weight" and "max running apps" can be changed in real time. The new definitions will then affect subsequent job dispatching. Thus, if an administrator found VA response times running slowly, it would be possible to tune down "max running apps" for SAS Grid and HPA which would likely soon free up resources for VA.

We observed that the Fair Scheduler seemed to favor SAS Grid applications which were requesting a single container each time as opposed to the larger HPA request for 14 containers. (Note that a running HPA application has 14 containers, 1 on each data node, while a SAS Grid application is a single container on a single data node.) Giving the HPA queue more weight seemed to give us the balance we were looking for. Since VA was started before the test and running throughout the test, it was not affected by YARN decisions. However, if it were necessary to restart VA, then it might be advisable to turn down the "max running apps" for other queues to make sure YARN had the resources necessary to start VA.

15. The Capacity Scheduler

The next test was to re-run the final scenario from above (SAS Grid, HPA and VA running concurrently with nice settings and weighted queues) but replacing the Fair Scheduler with the Capacity Scheduler. After some experimentation, we found that the Capacity Scheduler could be configured to give the behavior we desired. Once again, it was necessary to favor HPA to bring it into more of a balance with SAS Grid. Our final settings were: HPA = 40%, SAS Grid = 9%, VA = 50%.

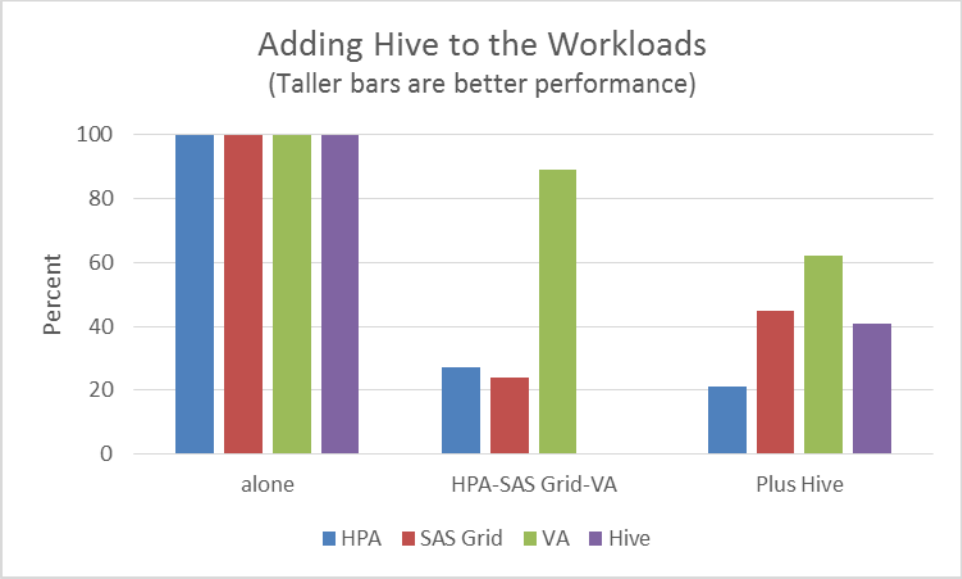
With either scheduler, if you are running very similar workloads, then it will be easier to predict how a particular scheduler configuration will cause the workloads to behave. However, with workloads that have very different behavior such as our three workloads, the simple ratio of queue weights or queue capacities will not create an equivalent behavior in the workloads. For instance, we used configurations where the Fair Scheduler weight is 3 for HPA and 1 for SAS Grid and the Capacity Scheduler percentages are 40% for HPA and 9% for SAS Grid. With very heterogeneous workloads, either scheduler may take some experimentation.

In the Capacity Scheduler, to specify the maximum number of running applications on a queue, you can set the attribute yarn.scheduler.capacity.<queue_path>.maximum-am-resource-percent. This specifies the percent of the queue's resources which can be used to launch app masters (where a setting of "0.1" is 10%, which is the default). Knowing the size of your app master containers and how much memory is allocated to your queue, then you can calculate how many app masters, and hence the maximum concurrent applications, are permitted on that queue. Because SAS Grid launches an app master for each one-container application, we changed this setting to 0.3 (30%) so that SAS Grid was able to launch enough applications to use the queue's resources.

16. Adding a Hive Workload

Our next test was to see how well the three SAS products would run with the addition of a heavy Map-Reduce workload. For the map reduce workload we used Hive executing a subset of the TPC-DS benchmark against the 3TB size TPC-DS data set. Once again, we tested the TPC-DS benchmark on the grid alone to get a baseline which we set to the scale of 100. In this case, we were measuring how long each query took to execute.

Each SAS Grid and HPA job that we were using took up a modest portion of the resources on the cluster. The VA workload, however, is typically using over 80% of available CPU on the cluster. The Hive workload is also able to use all available resources for long periods. Therefore, this heavy Hive workload has a bigger impact on the VA response times than SAS Grid or HPA. Whether this impact to response times would be within requirements depends on the each user's scenario. Our small test cluster was heavily stressed by the two big workloads. To improve VA response times, we could have reduced the number of concurrent VA test users, or we could have increased the average Hive query time, or we could have added servers to the cluster. Using YARN as a resource manager, these choices are in the hands of the cluster administrator.



For the HPA, SAS Grid, VA, Hive test, we once again used the Fair Scheduler with some different settings. As before, it took several attempts to get a configuration that was close to our goals of protecting VA response time while also getting significant work done by the other workloads. We reduced the VA container to 35GB and 8 vcores. Hive containers were 4GB with 1 vcore. Hive used as many containers as it could get from YARN. These were our final queue settings:

	HPA	SAS Grid	VA	Hive
queue weight	3	1	1	1
max running apps	2	24	n/a	n/a
policy	DRF	DRF	DRF	DRF

VA response time fell by 40% which means it came close to doubling. Query time for Hive was a little more than doubled. Considering that the resources were almost completely used before the Hive load was added, this seems a pretty reasonable result.

17. The Possibility of Using cgroups

Although we did not run performance tests with cgroups, we did test the configuration to see how the different workloads made use of cgroups. Without cgroups. YARN is managing resources mostly based on the dispatch of workloads given its understanding of memory and CPU resources available and requested. The Node Manager can be configured to kill jobs that exceed their memory limit, but this is a bit of a blunt instrument to control resources. Cgroups give YARN the possibility of actually constraining resources in an elegant fashion. Of course, they also add some complexity. If you can balance your workloads without using cgroups that can simplify your management. If you are struggling to get your workloads to run well together, cgroups can probably help.

Currently, YARN only manages CPU through cgroups applied to containers. When cgroups are enabled for containers in YARN, then YARN creates a cgroup for each container, giving that container the default share of the CPU resource. By default, a container gets a CPU share equal to its number of vcores times

1024. This makes assignment of vcores a useful way to control CPU shares. A container with 2 vcores will get twice the CPU shares of a container with 1 vcore. When there is competition for CPU, then the first container will get an average of about twice the CPU as the second container.

Because VA and HPA do not run under the control of a YARN container, they support the ability to assign their jobs to cgroups in their static configuration. Each VA/LASR action or different HPA job can be assigned to a different static cgroup. To make the VA/HPA cgroups coordinate with YARN cgroups, it is necessary to scale their cgroups shares the same way that YARN is scaling its cgroups. The number of vcores assigned to YARN becomes the fraction by which to scale CPU. If YARN is told to manage the CPU resource as 16 vcores, and VA is configured to request 8 vcores from YARN, then you should create a static cgroup for VA which has a CPU share equal to 8 times 1024. This cgroup is then assigned to VA in its resource.settings file as TKMPI_CGROUP. Then VA running with this share will coordinate properly with other YARN jobs running under their dynamic cgroups.

Appendix 1: SAS Products that can be managed by YARN

SAS/ACCESS to Hadoop
SAS/ACCESS to Impala
The Embedded Process which enables
    Scoring Accelerator for Hadoop
    Code Accelerator for Hadoop
    Data Quality Accelerator for Hadoop
SAS Grid Manager for Hadoop
SAS High Performance Analytic Server
SAS LASR Server which enables
    SAS Visual Analytics
    SAS Visual Statistics
    SAS IMSTAT

Appendix 2: Example of Embedded Process Memory Error

Below is an example of errors returned when the Embedded Process runs out of memory in any of its MR containers.

SAS Code Log:

NOTE: Attempting to run DATA Step in Hadoop.
ERROR: Map/Reduce job failed. Could not run Hadoop job.
FATAL: Unrecoverable I/O error detected in the execution of the DATA step program. Aborted during the COMPILATION phase.
    Hadoop Job (HDP_JOB_ID), job_1433352103056_0027, SAS Map/Reduce Job, http://cdhd00.eos.sas.com:8088/proxy/application_1433352103056_0027/
…
NOTE: The SAS System stopped processing this step because of errors.
NOTE: SAS set option OBS=0 and will continue to check statements. This might cause NOTE: No observations in data set.

YARN Application Master logs:

2015-06-08 09:09:17,796 INFO [AsyncDispatcher event handler] org.apache.hadoop.mapreduce.v2.app.job.impl.TaskAttemptImpl: attempt_1433352103056_0027_m_000000_0 TaskAttempt Transitioned from RUNNING to FAIL_CONTAINER_CLEANUP
2015-06-08 09:09:17,797 INFO [AsyncDispatcher event handler] org.apache.hadoop.mapreduce.v2.app.job.impl.TaskAttemptImpl: Diagnostics report from attempt_1433352103056_0027_m_000000_0: Container [pid=16773,containerID=container_1433352103056_0027_01_000003] is running beyond physical memory limits. Current usage: 1.1 GB of 1 GB physical memory used; 2.6 GB of 2.1 GB virtual memory used. Killing container.
…
Container killed on request. Exit code is 143
Container exited with a non-zero exit code 143

Appendix 3: Creating the resource.settings File

TKGrid is the framework which executes all LASR actions and all HPA procedures on the grid. (A LASR server is required by VA.) If the TKGrid resource.settings file is configured correctly, TKGrid will use its contents to send a resource request to YARN. The resource.settings file is actually a shell script which is run by TKGrid each time it executes a job for LASR or HPA. You may have specific resource settings in this file for any particular LASR action or HPA procedure by using a conditional statement. The condition could depend on the user executing the job (USER). Or the condition could depend on the procedure being executed (TKMPI_APPNAME).

Each HPA procedure has a unique TKMPI_APPNAME which allows you to specify different resource settings for each procedure. LASR, however, has one TKMPI_APPNAME for several actions such as create, add, and term. Typically, you will want to control the memory size of the LASR server when it is created. Most other LASR actions use a modest amount of memory and are short-lived so they do not need to be managed by YARN. The exception is that if you do a LASR add using a table with sas7bdat data format, then this add process will read the data into local memory. For this reason, if you are adding a sas7bdat data format table to LASR, you may want to have this add action controlled by YARN and limited by TKMPI_ULIMIT.

TKGrid provides a parameter (TKMPI_INFO) which can be set in your SAS code which calls the LASR create action. To use this parameter, you will need to start your LASR server from SAS code. You can otherwise manage your LASR server, such as loading tables and stopping the server, using the VA Administrator UI. You can use TKMPI_INFO to pass any information you please to the resource.settings script. In this example, we pass the LASR action name we are calling and the total memory size we want for the LASR server. This total memory (in MB) will be split across each of the data nodes as one container per node:

```
options set=TKMPI_INFO = "create:120000";
proc lasr create PORT=10099
            path="/tmp"
            signer="http://<authserver_hostname>:7980/SASLASRAuthorization";
            performance host="<tkgrid_hostname>"
            install="/opt/TKGrid_REP"
            nodes=all;
run;
```

TKMPI_INFO then becomes available inside the resource.settings script as we will illustrate below in the complex example. Using TKMPI_INFO allows us to pass a memory size (or other resource parameter) at the time an HPA or LASR procedure is called. Otherwise, the resource.settings file parameters are defined when the file is created.

Because resource.settings is a script that is executed when each TKGrid job is created, it can be debugged by adding code to it like:

```
date >> /tmp/tkgridyarn.log
echo "TKMPI_INFO=$TKMPI_INFO" >> /tmp/tkgridyarn.log
echo $TKMPI_RESOURCEMANAGER >> /tmp/tkgridyarn.log
echo "TKMPI_APPNAME=$TKMPI_APPNAME >> /tmp/tkgridyarn.log
```

To determine the TKMPI_APPNAME for a procedure, you can observe the running procedure in the TKGridMon utility. This will also show you how much memory is being used from moment to moment.

1. Simple resource.settings example

Note: this forces all SAS TKGrid actions to be managed by YARN equally. This is great for seeing what TKGrid actions are called by the client.

```
# The number of cores to allocate to each host's container. (REQUIRED)
#
export TKMPI_YARN_CORES=1

# The amount of memory in megabytes to allocate to each host's container (REQUIRED)
# In this example, 5 GB of RAM is reserved for ALL TKGrid Container requests, on EVERY
TKGrid Node
#
export TKMPI_MEMSIZE=5000

# Length of time TKGrid should wait for the resource allocation in seconds (OPTIONAL)
# 3600 = 1 hour (which might be too long to wait for most folks, shorter times might
make more sense
# Jobs waiting to executing show as "pending" in the YARN Application monitor
#
export TKMPI_YARN_TIMEOUT=3600

# The queue to submit the job to (OPTIONAL)
# This setting is typically only useful when YARN queues are used
#
export TKMPI_YARN_QUEUE=default

# SPECIAL NOTE:  this line is all one line (no carriage returns)
# in the resource.settings file, it is split here for readability
#
# This environment variable is used to make the request to the YARN Resource Manager
# TKMPI_YARN_HOSTS and TKMPI_APPNAME variables are automatically set by TKGrid
#
export TKMPI_RESOURCEMANAGER="java -Xmx256m -Xms256m -cp \"`$HADOOP_HOME/bin/hadoop
classpath`\" com.sas.grid.provider.yarn.tkgrid.JobLauncher --masterMem 2000 --javaMem
500 --hostlist \$TKMPI_YARN_HOSTS --cores \$TKMPI_YARN_CORES --memory \$TKMPI_MEMSIZE
--priority \$TKMPI_YARN_PRIORITY --timeout \$TKMPI_YARN_TIMEOUT -
-jobname $TKMPI_APPNAME --queue $TKMPI_YARN_QUEUE"
```

2. Example of a resource.settings File Used In Testing

```
#In this script, only the LASR create action and particular HPA procs will
#call YARN for a resource allocation.

#proc LASR create called in SAS code will set TKMPI_INFO to "create".
#For example:
#  options set=TKMPI_INFO = "create";
if [ "$TKMPI_INFO" = "create" ]; then
#Limit LASR server to one 40GB container on each data node.
export TKMPI_MEMSIZE=40000
export TKMPI_ULIMIT="-v 40000000"
export TKMPI_YARN_PRIORITY=2
export TKMPI_YARN_TIMEOUT=3600
```

```
export TKMPI_YARN_CORES=12
export TKMPI_YARN_QUEUE=va
export TKMPI_RESOURCEMANAGER="java -Xmx256m -Xms256m -cp
\"`$HADOOP_HOME/bin/hadoop classpath`\"
com.sas.grid.provider.yarn.tkgrid.JobLauncher --masterMem 2000 --javaMem 500
--hostlist \$TKMPI_YARN_HOSTS --cores \$TKMPI_YARN_CORES --memory
\$TKMPI_MEMSIZE --priority \$TKMPI_YARN_PRIORITY --timeout
\$TKMPI_YARN_TIMEOUT --jobname $TKMPI_APPNAME --queue $TKMPI_YARN_QUEUE"
fi

#Only the listed HPA procs will be coordinated with YARN.
if [ "$TKMPI_APPNAME" = "tkdmdb" ] || [ "$TKMPI_APPNAME" = "tklogis" ] || [
"$TKMPI_APPNAME" = "tkneural" ] || [ "$TKMPI_APPNAME" = "tkreduce" ] || [
"$TKMPI_APPNAME" = "tksamp" ] || [ "$TKMPI_APPNAME" = "tkhp3" ]; then
#This cpu cgroup was created to work with YARN cgroups as described
#in "The Possibility of Using cgroups" above.
export TKMPI_CGROUP="cgexec -g cpu:cgHPA"
# Let all listed HPA procs use 8GB containers and 1 vcore.
export TKMPI_MEMSIZE=8000
export TKMPI_YARN_PRIORITY=2
export TKMPI_YARN_TIMEOUT=3600
export TKMPI_YARN_CORES=1
export TKMPI_YARN_QUEUE=hpa
export TKMPI_RESOURCEMANAGER="java -Xmx256m -Xms256m -cp
\"`$HADOOP_HOME/bin/hadoop classpath`\"
com.sas.grid.provider.yarn.tkgrid.JobLauncher --masterMem 2000 --javaMem 500
--hostlist \$TKMPI_YARN_HOSTS --cores \$TKMPI_YARN_CORES --memory
\$TKMPI_MEMSIZE --priority \$TKMPI_YARN_PRIORITY --timeout
\$TKMPI_YARN_TIMEOUT --jobname $TKMPI_APPNAME --queue $TKMPI_YARN_QUEUE"
fi
```

3. Complex resource.settings Example

This groups the different TKGrid actions into one of the four scenarios: No YARN, minimum YARN container, lift only one times the file size, lift 2.5 times the file size (Appears to be the maximum amount used for a high-performance PROC). And this allows the SAS code to provide hints to the resource.settings file to properly schedule and restrict TKGrid resource usage.

Note: The “No YARN” scenario has to be protected because this can provide a hole for rogue SAS code.

This example:

- Allows the SAS coder to provide hints to change the default file size lifting to memory and to provide details on LASR actions so the shared cluster usage is tuning to SAS code requirements.
    - SAS code hints require the TKMPI_INFO environment variable. You can use any delimiter such as “:”if you want to provide multiple hints in the same string. Example:

      option set=TKMPI_INFO=”<memorysize>:<lasraction>”;
- Allows the administrator to note the default file size lifting to memory and number of TKGrid nodes in Hadoop cluster. This helps calculate the container size for the different TKGrid library scenarios.
- This includes LASR procedure controls so the shared cluster can run both HPA actions and LASR
- Sets the CPU restrictions based on the user using TKGrid.
- Set the memory restrictions based on TKGrid action scenarios.

YARN makes resource requests for SAS LASR Analytic Server actions like lasrhoo and lasracc as they are lightweight or managed by YARN through MapReduce. You might group these with the minimum YARN container group. This is done when you want these groups managed by YARN.

Note: This is not a file to start your deployment with unless you know what you are doing.

Note: This example provides IT protection from rogue sas code by:

- Forcing all memory hints to be large then 0
- Forcing all LASR actions to be YARN monitored even with small container

Note: this example we would only change the first two settings:

- LARGEMEMORY – maximum memory size you want to allow sas code to use when there are no hints
- NODENUMBER – number of nodemanagers that have TKGrid installed on them

```
#Default total memory size or largest file size in MB
LARGEMEMORY=16384
#Number of nodemanagers
NODENUMBER=4

##################################################
#Default flag for LASR actions
LASRACTION="VAVS"

# Parse TKMPI_INFO which can contain a number for total memory size in MB
# and/or the LASR action name "create" separated by colon.
# If it is only a memory size, that will be used for HPA procs instead of the default.
arraylist=(${TKMPI_INFO//:/ })
for i in "${arraylist[@]}"
do
  if [ $i == "" ]
  then
    continue
  elif (( $i > 0 )) 2>/dev/null
  then
    LARGEMEMORY=$i
    continue
  else
    case ${i^^} in
      "create")
        LASRACTION="create"
       ;;
    esac
  fi
done

# 1 GB per container
scenario1=1024
# Divide memory by nodenumber since data will be split into one container per node,
# Then add 10% extra to each container size.
scenario2=$(( LARGEMEMORY / NODENUMBER * 11 / 10))
# Lift 2.5 times the size of file
scenario3=$(( scenario2 * 25 / 10 ))

#Example of how various users may have different resource allocations.
if [ "$USER" == "sasdemo2" ]; then
       # The number of cores to allocate to each host's container.
```

```
       export TKMPI_YARN_CORES=3
elif [ "$USER" == "sasdemo" ]; then
       # The number of cores to allocate to each host's container.
       export TKMPI_YARN_CORES=1
else
       export TKMPI_YARN_CORES=2
fi

# The priority of the application if scheduler uses it.
export TKMPI_YARN_PRIORITY=2
# Length of time TKGrid should wait for the resource reservation in seconds.
export TKMPI_YARN_TIMEOUT=3600
# The YARN queue to submit the job to.
export TKMPI_YARN_QUEUE=default


case "$TKMPI_APPNAME" in
  "lasrhoo"|"lasracc")
       echo "lasrhoo/lasracc - do not use YARN"
  ;;
 "tkhpsumm"|"tklogis"|"tkcorr"|"tkreg"|"tknueral"|"tkhp3"|"tkdmdb"|"tkreduce"|"tkimp")
       # The amount of memory in megabytes to reserve per container per server.
       export TKMPI_MEMSIZE=$scenario2
       export TKMPI_RESOURCEMANAGER="java -Xmx256m -Xms256m -cp
\"`$HADOOP_HOME/bin/hadoop classpath`\" com.sas.grid.provider.yarn.tkgrid.JobLauncher
--masterMem 2000 --javaMem 500 --hostlist \$TKMPI_YARN_HOSTS --cores
\$TKMPI_YARN_CORES --memory \$TKMPI_MEMSIZE --priority \$TKMPI_YARN_PRIORITY --timeout
\$TKMPI_YARN_TIMEOUT --jobname $TKMPI_APPNAME --queue $TKMPI_YARN_QUEUE"
  ;;
  "tksamp")
       export TKMPI_MEMSIZE=$scenario3
       export TKMPI_RESOURCEMANAGER="java -Xmx256m -Xms256m -cp
\"`$HADOOP_HOME/bin/hadoop classpath`\" com.sas.grid.provider.yarn.tkgrid.JobLauncher
--masterMem 2000 --javaMem 500 --hostlist \$TKMPI_YARN_HOSTS --cores
\$TKMPI_YARN_CORES --memory \$TKMPI_MEMSIZE --priority \$TKMPI_YARN_PRIORITY --timeout
\$TKMPI_YARN_TIMEOUT --jobname $TKMPI_APPNAME --queue $TKMPI_YARN_QUEUE"
  ;;
  "lasr")
      case $LASRACTION in
        "create")
          export TKMPI_MEMSIZE=$scenario2
          export TKMPI_RESOURCEMANAGER="java -Xmx256m -Xms256m -cp
\"`$HADOOP_HOME/bin/hadoop classpath`\" com.sas.grid.provider.yarn.tkgrid.JobLauncher
--masterMem 2000 --javaMem 500 --hostlist \$TKMPI_YARN_HOSTS --cores
\$TKMPI_YARN_CORES --memory \$TKMPI_MEMSIZE --priority \$TKMPI_YARN_PRIORITY --timeout
\$TKMPI_YARN_TIMEOUT --jobname ${TKMPI_APPNAME}_SERVER --queue $TKMPI_YARN_QUEUE"
         ;;
        *)
          export TKMPI_MEMSIZE=1024
          export TKMPI_RESOURCEMANAGER="java -Xmx256m -Xms256m -cp
\"`$HADOOP_HOME/bin/hadoop classpath`\" com.sas.grid.provider.yarn.tkgrid.JobLauncher
--masterMem 2000 --javaMem 500 --hostlist \$TKMPI_YARN_HOSTS --cores
\$TKMPI_YARN_CORES --memory \$TKMPI_MEMSIZE --priority \$TKMPI_YARN_PRIORITY --timeout
\$TKMPI_YARN_TIMEOUT --jobname ${TKMPI_APPNAME}_ACTION --queue $TKMPI_YARN_QUEUE"
         ;;
       esac
  ;;
  *)
       export TKMPI_MEMSIZE=$scenario1
       export TKMPI_RESOURCEMANAGER="java -Xmx256m -Xms256m -cp
\"`$HADOOP_HOME/bin/hadoop classpath`\" com.sas.grid.provider.yarn.tkgrid.JobLauncher
--masterMem 2000 --javaMem 500 --hostlist \$TKMPI_YARN_HOSTS --cores
\$TKMPI_YARN_CORES --memory \$TKMPI_MEMSIZE --priority \$TKMPI_YARN_PRIORITY --timeout
```

```
\$TKMPI_YARN_TIMEOUT --jobname $TKMPI_APPNAME --queue $TKMPI_YARN_QUEUE"
  ;;

esac
```

4. Common Errors in resource.settings

It is common to have issues with the resource.setting file related to hidden characters. This causes errors like "Failure to enumerate the grid". Refer to a working/known resource.settings file after getting an error to make sure recent changes did not cause the error.

Listed below are some other common errors that we discovered during testing:
Example 1: File was too big to fit into reserved container space. This is the error displayed when this occurs

```
30         proc hpcorr data=dhdat.carsbig;
31           performance nodes=all details;
32         run;

NOTE: The HPCORR procedure is executing in the distributed computing environment with
3 worker nodes.
ERROR: There are no observations in the input data set.
ERROR: Failure loading file from Hadoop.
NOTE: The SAS System stopped processing this step because of errors.
```

Example 2: This error was because there was not enough RAM in the system to honor the `resource.setting` memsize request for the SAS LASR Analytic Server. This was fixed this by dropping the RAM request size in *resource.settings*

```
235        proc lasr
236          create port=&freelasrport
237          path="/tmp";
238          performance host=&mytkgridroot
239          nodes=all;
240        run;

NOTE: The LASR procedure is executing in the distributed computing environment with 3
worker nodes.
ERROR: Communication failure between LASR Analytic Server and client application.
ERROR: Communication failure between LASR Analytic Server and client application.
ERROR: The server-side process of the LASR Analytic Server could not be started. The
requested port (10099) might be in use by
       another application or it is not a valid port on the host.
NOTE: The SAS System stopped processing this step because of errors.
```

Appendix 4: Example of a sasgrid-policy.xml file

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<GridPolicy  defaultAppType="normal">
<!-- "defaultAppType" points to the application type to use if none is specified for job -->
   <GridApplicationType name="normal">
      <jobname>SASGridNormalJob</jobname>
      <priority>2</priority>
      <nice>15</nice>
      <memory>4000</memory>
      <vcores>1</vcores>
      <queue>grid</queue>
   </GridApplicationType>
</GridPolicy>
```

References and Resources

SAS High-Performance Analytics Infrastructure 2.94: Installation and Configuration Guide

The Apache Software Foundation “NextGen MapReduce (YARN)” (Accessed December 2014). http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html

Red Hat (2014). “Resource Management Guide” Chapter 1: Introduction to Control Groups (Cgroups). https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Resource_Management_Guide/ch01.html

SAS Institute Inc. 2014. “SAS/ACCESS® 9.4 for Relational Databases: Reference, Sixth Edition”. Cary, NC. http://support.sas.com/documentation/cdl/en/acreldb/67589/PDF/default/acreldb.pdf

SAS Institute Inc. 2014. “SAS/ACCESS® 9.4 for Relational Databases: Reference, Sixth Edition” SAS/ACCESS Interface to Impala section. Cary, NC. http://support.sas.com/documentation/cdl/en/acreldb/67589/HTML/default/viewer.htm#p0rkug1n9ub7b0n132xjxknz1qvv.htm

Murthy, Arun. "SAS high-performance capabilities with Hadoop YARN" SAS Institute, The Data Roundtable, SAS high-performance capabilities with Hadoop YARN. (August 20, 2014). http://blogs.sas.com/content/datamanagement/2014/08/20/sas-high-performance-capabilities-with-hadoop-yarn

Truong, Kim “How Apache Hadoop Helps SAS” Hortonworks, (August 20, 2014) http://hortonworks.com/blog/apache-hadoop-helps-sas/

SAS Institute Inc. 2014 SAS LASR Analytic Server 2.5 Reference Guide, Managing Resources with YARN (Experimental). http://support.sas.com/documentation/cdl/en/inmsref/67629/HTML/default/viewer.htm#n1xubdvcg8gt05n1uuffmiacw16g.htm