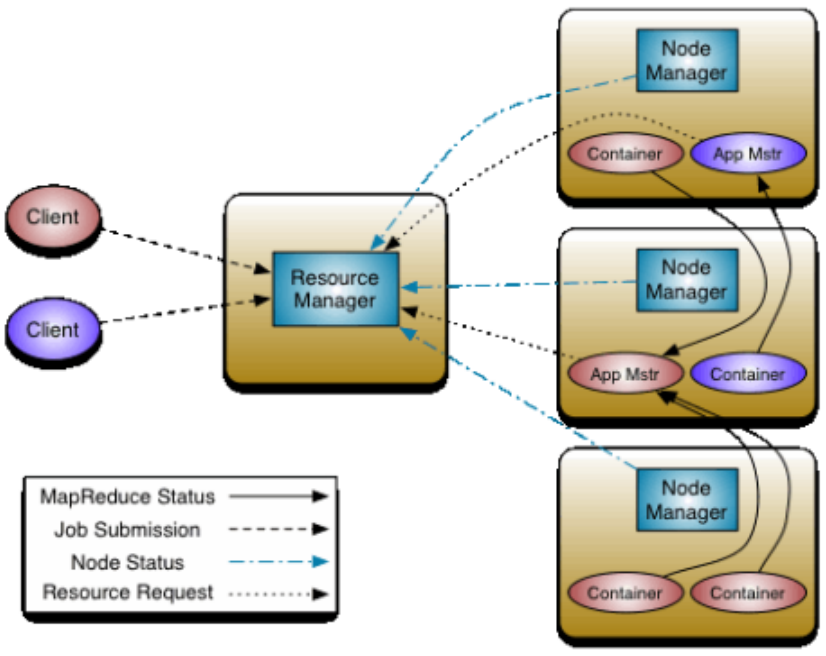
July 24, 2015 | BY Hao Zhu (/blog/author/hao-zhu/)

In this blog post, I will discuss best practices for YARN resource management. The fundamental idea of MRv2(YARN) is to split up the two major functionalities—resource management and job scheduling/monitoring, into separate daemons. The idea is to have a global ResourceManager (RM) and per-application ApplicationMaster (AM).

The ResourceManager(RM) and per-node slave, the NodeManager (NM), form the data-computation framework. The ResourceManager is the ultimate authority that arbitrates resources among all the applications in the system.

Please read the Hadoop Documentation (http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html) under YARN concept and architecture first before reading this article.



This blog post covers the following topics regarding YARN resource management, and also provides best practices for each topic:

1. How does warden calculate and allocate resources to YARN?
2. Minimum and maximum allocation unit in YARN
3. Virtual/physical memory checker
4. Mapper, Reducer and AM's resource request
5. Bottleneck resource

1. How does warden calculate and allocate resources to YARN?

n a MapR Hadoop cluster, warden sets the default resource allocation for the operating system, MapR-FS, MapR Hadoop services, and MapReduce v1 and YARN applications. Details are described in MapR documentation:
Resource Allocation for Jobs and Applications (http://doc.mapr.com/display/MapR/Resource+Allocation+for+Jobs+and+Applications?_ga=1.231934426.430765848.1434152907)
.

YARN can manage 3 system resources— memory, CPU and disks. Once warden finishes calculations, it will set environment variable YARN_NODEMANAGER_OPTS for starting NM.

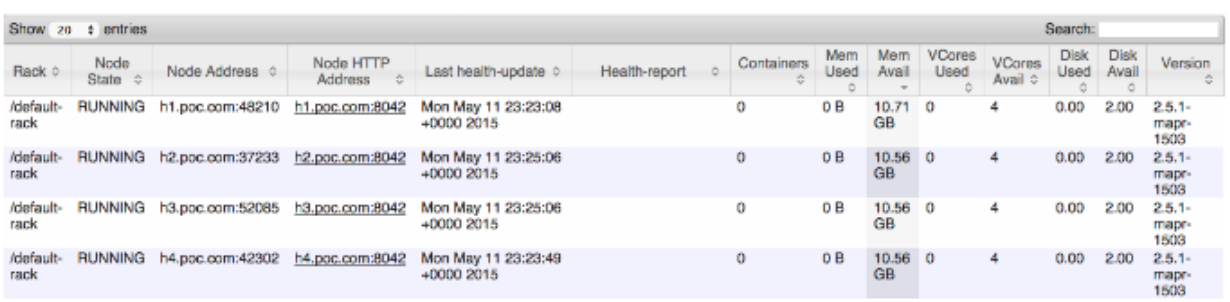
For example, if you "vi /proc//environ" you can find the settings below:

```
YARN_NODEMANAGER_OPTS= -Dnodemanager.resource.memory-mb=10817
-Dnodemanager.resource.cpu-vcores=4
-Dnodemanager.resource.io-spindles=2.0
```

They can be overridden by setting the three configurations below in yarn-site.xml on NM nodes and restarting NM.

- yarn.nodemanager.resource.memory-mb
- yarn.nodemanager.resource.cpu-vcores
- yarn.nodemanager.resource.io-spindles

To view the available resources from each node, you can go to RM UI(http://:8088/cluster/nodes), and find out the "Mem Avail", "Vcores Avail" and "Disk Avail" from each node.


Show 20 entries Search:

Rack	Node State	Node Address	Node HTTP Address	Last health-update	Health-report	Containers	Mem Used	Mem Avail	VCores Used	VCores Avail	Disk Used	Disk Avail	Version
/default-rack	RUNNING	h1.poc.com:48210	h1.poc.com:8042	Mon May 11 23:23:08 +0000 2015		0	0 B	10.71 GB	0	4	0.00	2.00	2.5.1-mapr-1503
/default-rack	RUNNING	h2.poc.com:37233	h2.poc.com:8042	Mon May 11 23:25:06 +0000 2015		0	0 B	10.56 GB	0	4	0.00	2.00	2.5.1-mapr-1503
/default-rack	RUNNING	h3.poc.com:52085	h3.poc.com:8042	Mon May 11 23:25:06 +0000 2015		0	0 B	10.56 GB	0	4	0.00	2.00	2.5.1-mapr-1503
/default-rack	RUNNING	h4.poc.com:42302	h4.poc.com:8042	Mon May 11 23:23:49 +0000 2015		0	0 B	10.56 GB	0	4	0.00	2.00	2.5.1-mapr-1503

In this step, make sure warden fully considers all services for resource allocation because some services do not have dedicated parameters in warden.conf, e.g., Drill and Impala. If you plan to allocate 10% of total memory for Drill and 5% for Impala on this node, please carve out those 15% memory to parameters: service.command.os.heapsize.percent/max/min.

If memory are over allocated to YARN, huge swap may be used and kernel OOM killer may be triggered to kill the container process.

Below error is a sign of OS OOM and probably memory is over allocated to YARN.

```
os::commit_memory(0x0000000000000000, xxxxxxxxx, 0) failed;
error='Cannot allocate memory' (errno=12)
```

If we see that, just double check if warden takes into account all memory consumed services on that node, and reduce the memory allocated by warden if needed.

2. Minimum and maximum allocation unit in YARN

Two resources—memory and CPU, as of in Hadoop 2.5.1, have minimum and maximum allocation unit in YARN, as set by the configurations below in yarn-site.xml.

Configuration in yarn-site.xml	Default value
yarn.scheduler.minimum-allocation-mb	1024
yarn.scheduler.maximum-allocation-mb	8192
yarn.scheduler.minimum-allocation-vcores	1
yarn.scheduler.maximum-allocation-vcores	32

Basically, it means RM can only allocate memory to containers in increments of "yarn.scheduler.minimum-allocation-mb" and not exceed "yarn.scheduler.maximum-allocation-mb";

And it can only allocate CPU vcores to containers in increments of "yarn.scheduler.minimum-allocation-vcores" and not exceed "yarn.scheduler.maximum-allocation-vcores".

If changes required, set above configurations in yarn-site.xml on RM nodes, and restart RM.

For example, if one job is asking for 1025 MB memory per map container(set mapreduce.map.memory.mb=1025), RM will give it one 2048 MB(2*yarn.scheduler.minimum-allocation-mb) container.

If you have a huge MR job which asks for a 9999 MB map container, the job will be killed with the error message below in the AM log:

```
MAP capability required is more than the supported max container capability in the cluster.
Killing the Job. mapResourceRequest: 9999 maxContainerCapability:8192
```

If a Spark on YARN job asks for a huge container with size larger than "yarn.scheduler.maximum-allocation-mb", the error below will show up:

```
Exception in thread "main" java.lang.IllegalArgumentException:
Required executor memory (99999+6886 MB) is above the max threshold (8192 MB) of this cluster!
```

In the above two cases, you can increase "yarn.scheduler.maximum-allocation-mb" in yarn-site.xml and restart RM.

So in this step, you need to be familiar with the lower and upper bound of resource requirements for each mapper and reducer of the jobs and set the minimum and maximum allocation unit according to that.

3. Virtual/physical memory checker

NodeManager can monitor the memory usage(virtual and physical) of the container. If its virtual memory exceeds "yarn.nodemanager.vmem-pmem-ratio" times the "mapreduce.reduce.memory.mb" or "mapreduce.map.memory.mb", then the container will be killed if "yarn.nodemanager.vmem-check-enabled" is true;

If its physical memory exceeds "mapreduce.reduce.memory.mb" or "mapreduce.map.memory.mb", the container will be killed if "yarn.nodemanager.pmem-check-enabled" is true.

The parameters below can be set in yarn-site.xml on each NM nodes to override the default behavior.

Configuration in yarn-site.xml	Default value
yarn.nodemanager.vmem-check-enabled	false
yarn.nodemanager.pmem-check-enabled	true
yarn.nodemanager.vmem-pmem-ratio	2.1

This is a sample error for a container killed by virtual memory checker:

```
Current usage: 347.3 MB of 1 GB physical memory used;
<font color="red">2.2 GB of 2.1 GB virtual memory used</font>. Killing container.
```

And this is a sample error for physical memory checker:

```
Current usage: <font color="red">2.1gb of 2.0gb physical memory used</font>;
1.1gb of 3.15gb virtual memory used. Killing container.
```

As in Hadoop 2.5.1 of MapR 4.1.0, virtual memory checker is disabled while physical memory checker is enabled by default.

Since on Centos/RHEL 6 there are aggressive allocation of virtual memory due to
OS behavior
(https://www.ibm.com/developerworks/community/blogs/kevgrig/entry/linux_glibc_2_10_rhel_6_malloc_may_show_excessive_virtual_memory_usage?lang=en)
, you should disable virtual memory checker or increase yarn.nodemanager.vmem-pmem-ratio to a relatively larger value.

f the above errors occur, it is also possible that the MapReduce job has memory leaking or the memory for each container is just not enough. Try to check the application logic and also tune the container memory request—"mapreduce.reduce.memory.mb" or "mapreduce.map.memory.mb".

4. Mapper,Reducer and AM's resource request

MapReduce v2 job has 3 different container types—Mapper, Reducer and AM.

Mapper and Reducer can ask for resources—memory, CPU and disk, while AM can only ask for memory and CPU.

Below are a summary of the configurations of resource requests for the three container types.

The default values are from Hadoop 2.5.1 of MapR 4.1, and they can be overridden in mapred-site.xml on the client node or set in applications like MapReduce java code, Pig and Hive Cli,etc.

- Mapper:

Configuration in mapred-site.xml	Default value
mapreduce.map.memory.mb	1024
mapreduce.map.java.opts	-Xmx900m
mapreduce.map.cpu.vcores	1
mapreduce.map.disk	0.5

- Reducer:

Configuration in mapred-site.xml	Default value
mapreduce.reduce.memory.mb	3072
mapreduce.reduce.java.opts	-Xmx2560m
mapreduce.reduce.cpu.vcores	1
mapreduce.reduce.disk	1.33

- AM:

Configuration in mapred-site.xml	Default value
yarn.app.mapreduce.am.resource.mb	1536
yarn.app.mapreduce.am.command-opts	-Xmx1024m
yarn.app.mapreduce.am.resource.cpu-vcores	1

Each container is actually a JVM process, and above "-Xmx" of java-opts should fit in the allocated memory size. One best practice is to set it to 0.8 * (container memory allocation). For example, if the requested mapper container has mapreduce.map.memory.mb=4096, we can set mapreduce.map.java.opts=-Xmx3277m.

There are many factors which can affect the memory requirement for each container. Such factors include the number of Mappers/Reducers, the file type(plain text file , parquet, ORC), data compression algorithm, type of operations(sort, group-by, aggregation, join), data skew, etc. You should be familiar with the nature of this MapReduce job and figure out the minimum requirement for Mapper,Reducer and AM. Any type of the container can run out of memory and be killed by physical/virtual memory checker, if it doesn't meet the minimum memory requirement. If so, you need to check the AM log and the failed container log to find out the cause.

For example, if the MapReduce job sorts parquet files, Mapper needs to cache the whole Parquet row group in memory. I have done tests to prove that the larger the row group size of parquet files is, the larger Mapper memory is needed. In this case, make sure the Mapper memory is large enough without triggering OOM.

Another example is AM running out of memory. Normally, AM's 1G java heap size is enough for many jobs. However, if the job is to write lots of parquet files, during commit phase of the job, AM will call ParquetOutputCommitter.commitJob(). It will first read footers of all output parquet files, and write the metadata file named "_metadata" in output directory.

This step may cause AM being out of memory with below stacktrace in AM log:

```
Caused by: <font color="red">java.lang.OutOfMemoryError</font>: GC overhead limit exceeded
        at java.lang.StringCoding$StringEncoder.encode(StringCoding.java:300)
        at java.lang.StringCoding.encode(StringCoding.java:344)
        at java.lang.String.getBytes(String.java:916)
        at parquet.org.apache.thrift.protocol.TCompactProtocol.writeString(TCompactProtocol.java:298)
        at parquet.format.ColumnChunk.write(ColumnChunk.java:512)
        at parquet.format.RowGroup.write(RowGroup.java:521)
        at parquet.format.FileMetaData.write(FileMetaData.java:923)
        at parquet.format.Util.write(Util.java:56)
        at parquet.format.Util.writeFileMetaData(Util.java:30)
        at parquet.hadoop.ParquetFileWriter.serializeFooter(ParquetFileWriter.java:322)
        at parquet.hadoop.<font color="red">ParquetFileWriter.writeMetadataFile</font>(ParquetFileWriter.java:342)
        at parquet.hadoop.<font color="red">ParquetOutputCommitter.commitJob</font>(ParquetOutputCommitter.java:51)
        ... 10 more
```

The solution is to increase the memory requirement for AM and disable this parquet feature by “set parquet.enable.summary-metadata false”.

Besides figuring out the minimum memory requirement for each container, sometimes we need to balance the job performance and resource capacity. For example, jobs doing sorting may need a relatively larger “mapreduce.task.io.sort.mb” to avoid or reduce the number of spilling files. If the whole system has enough memory capacity, we can increase both “mapreduce.task.io.sort.mb” and container memory to get better job performance.

In this step, we need to make sure each type of container meets proper resource requirements. If OOM happens, always check AM logs first to figure out which container and what is the cause per stack trace.

5. Bottleneck resource

Since there are three types of resources, different containers from different jobs may ask for different amount of resources. This can result in one of the resources becoming the bottleneck. Suppose we have a cluster with capacity (1000G RAM,16 Cores,16 disks) and each Mapper container needs (10G RAM,1 Core, 0.5 disks): at most, 16 Mappers can run in parallel because CPU cores become the bottleneck here.

As a result, (840G RAM, 8 disks) resources are not used by anyone. If you meet this situation, just check the RM UI(http://:8088/cluster/nodes) to figure out which resource is the bottleneck. You can probably allocate the leftover resources to jobs which can improve performance with such resource. For example, you can allocate more memory to sorting jobs which used to spill to disk.

Key takeaways:

1. Make sure warden considers all services when allocating system resources.
2. Be familiar with lower and upper bound of resource requirements for mapper and reducer.
3. Be aware of the virtual and physical memory checker.
4. Set -Xmx of java-opts of each container to 0.8 * (container memory allocation).
5. Make sure each type of container meets proper resource requirement.
6. Fully utilize bottleneck resource.

In this blog post, you’ve learned best practices for YARN resource management. If you have any further questions, please ask them in the comments section below.