In this blog post, you'll learn how memory is sliced and diced from the time you start warden, how memory is allocated to each MapR service installed on the node, and what memory is made available for MR1/MR2 for running jobs.

Usually in a MapR cluster, you would normally have MR1 services (JT and TT) or YARN services running (RM or NM). However, there are use cases in multi-tenant environments where users from one group would like to run MR1 jobs, while the other group would like to run jobs on MR2 on the same cluster; this use case is supported on a MapR cluster. The big advantage of this use case is that it is easy to transition users from MR1 to MR2, since both services are already running on the same cluster, and there is no need for special deployment or copying of data to new clusters.

It is very important to understand how memory is managed and divided among the services installed on the node, and how to predict the outcome of tweaking config parameters to obtain the desired results:

Environment Details : My test setup (All default config)

MapR version = 4.1.0.31175.GA

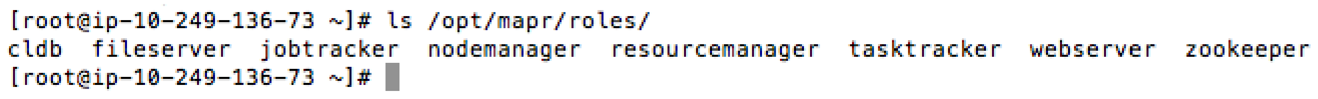
Number of CPU = 32

Memory = 60 Gb

Number of Disks=6

Cluster (MR1 and MR2)

You can check all the MapR services installed on the node by listing the services in roles directory.



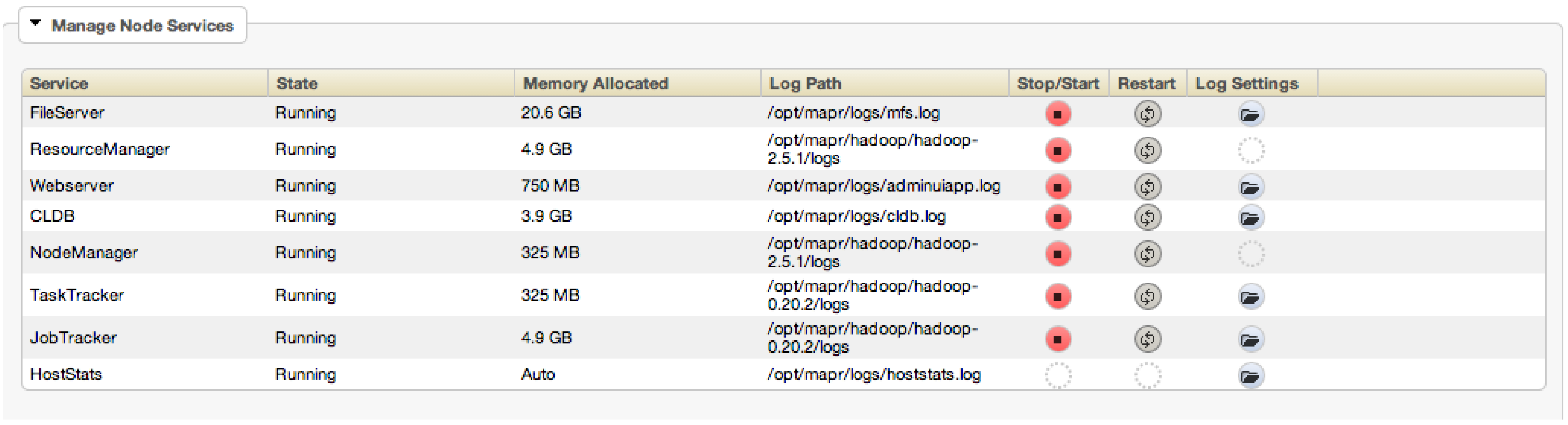
```
[root@ip-10-249-136-73 ~]# ls /opt/mapr/roles/
cldb  fileserver  jobtracker  nodemanager  resourcemanager  tasktracker  webserver  zookeeper
[root@ip-10-249-136-73 ~]#
```

Memory is reserved and calculated for each service via memory parameters in warden.conf. Values for each service in warden.conf will determine how much memory will be committed to the configured services, as well as how much is reserved for general OS/administration. The leftover memory on the node will be assigned to MR1 and MR2 collectively for concurrently running Jobs.

On our test node, we have ZooKeeper, CLDB, MFS, JobTracker, TaskTracker, webserver, Node Manager and Resource Manager configured. Granted, these are a lot of services for one node considering the memory on my test setup, but it's fine for demonstrating how memory gets assigned to each service. Note that this is not recommended for production setup. Each service requires memory, so warden will be allocating some percentage of the memory to each service (an explanation of how warden calculates memory for each service is discussed later in this article), and leftover memory is assigned to MR1 and MR2 jobs on the node.

The figure below is a snapshot from the MCS screen, which shows memory utilization by each service:

Dashboard => Node Properties => Manager Node services (Section)



Service	State	Memory Allocated	Log Path	Stop/Start	Restart	Log Settings
FileServer	Running	20.6 GB	/opt/mapr/logs/mfs.log	■	↻	📂
ResourceManager	Running	4.9 GB	/opt/mapr/hadoop/hadoop-2.5.1/logs	■	↻	
Webserver	Running	750 MB	/opt/mapr/logs/adminuiapp.log	■	↻	📂
CLDB	Running	3.9 GB	/opt/mapr/logs/cldb.log	■	↻	📂
NodeManager	Running	325 MB	/opt/mapr/hadoop/hadoop-2.5.1/logs	■	↻	
TaskTracker	Running	325 MB	/opt/mapr/hadoop/hadoop-0.20.2/logs	■	↻	📂
JobTracker	Running	4.9 GB	/opt/mapr/hadoop/hadoop-0.20.2/logs	■	↻	📂
HostStats	Running	Auto	/opt/mapr/logs/hoststats.log			📂

Based on the screen snapshot, here is the memory assignment for each service:

OS : 3.9 GB MFS : 20.6 GB RM : 4.9 GB Webserver : 750 MB CLDB : 3.9 GB TT : 325 MB

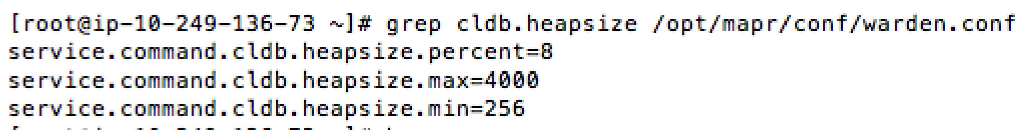
NM : 325 MB JT : 4.9 GB ZK : 614 MB Warden : 614 MB

Total = ~ 40.7 GB (Calculated value for memory consumed by all services)

Understanding the Memory Calculations For Each Service

Memory settings for the operating system, Fileserver, CLDB, Webserver and Hadoop services such as TaskTracker and JobTracker are configured in the warden.conf file. Warden.conf file is located in /opt/mapr/conf. Other services such as NodeManager and ResourceManager have their own warden..conf file within /opt/mapr/conf/conf.d

Why did warden decide to give memory to each service? For our demonstration, let's look at CLDB: why was it assigned 3.9 GB? What was the calculation/formula behind this?



```
[root@ip-10-249-136-73 ~]# grep cldb.heapsize /opt/mapr/conf/warden.conf
service.command.cldb.heapsize.percent=8
service.command.cldb.heapsize.max=4000
service.command.cldb.heapsize.min=256
```

Warden uses the formulae below to internally to calculate actual heap size, depending on parameters set in warden.conf. If any changes are made to warden.conf, then warden would need to restart for the new setting to apply.

Actual Heap size for service =max(heapsize.min, min(heapsize.max, total-memory * heapsize.percent / 100))

```
= Max (256 , min( 4000 ,61440 * 8/100 ))

= Max (256 , Min ( 4000, 4915))

=4000 MB = ~3.9 GB
```

Similarly, memory would be calculated this way for all individual services.

Cross Verify Fileserver Memory Calculation

From Warden.log : We can verify that our calculation is correct, since Fileserver memory from MCS matches warden.log. The line below verifies memory allocated to Fileserver = 21097 MB = ~ 20.6 GB

2015-05-07 21:34:32,442 INFO com.mapr.warden.service.KVStoreService [main]: Before adjustment, memory for KVStore = 21097.0

Total Memory Consumed by Configured Services

The lines below in warden.log verify that the total memory consumed by all MapR configured services = 41703 MB = ~40.7 GB, which matches our earlier total memory consumed (40.7 GB) by the services calculations.

2015-05-07 21:34:32,454 INFO com.mapr.warden.WardenManager [main]: Total node memory: 61722920

2015-05-07 21:34:32,454 INFO com.mapr.warden.WardenManager [main]: Total consumed memory: 41703

Total Available Free Memory

Available free memory (totalGlobalAvailableMemory) for MR1 and MR2 jobs is 18573 MB, as indicated in the log line below. This value is calculated by the difference between "total node memory" and "total consumed memory": 61722920 - 41703 = 18573 MB

2015-05-07 21:34:32,454 INFO com.mapr.warden.WardenManager [main]: Still available free memory: 18573

Now, there is one formula which is executed if the node is a 'zknode' or 'cldb' node

(being conservative to reserve memory):

totalFreeMemoryInMB=round(max(totalGlobalAvailableMemory * 0.75, totalGlobalAvailableMemory - 1500))

```
= round ( max (18573*0.75 , 18573 -1500 ))

= round ( max ( 13929.75 , 17073) )

= 17073 MB
```

Memory for MR1 and MR2

The log line below in warden.log validates that total memory for running MR1 and MR2 is set to "**memoryForTasks = 17073"**

2015-05-07 21:35:05,032 INFO com.mapr.warden.WardenMapRed [tasktracker_monitor]: Before adjustment, **memoryForTasks = 17073,** memory for service = 8536.5

Resources on the Node

Now that we have calulated memory needed for running MR1 and MR2 jobs while warden starts up, notice that warden also looks for CPU and disks available on the node which can be used while running jobs. This is done to take all resources into account while doing slot calculations, to make sure there are no bottlenecks when jobs are run in parallel.

2015-05-07 21:34:32,454 INFO com.mapr.warden.WardenManager [main]: Total node cpu cores : 32

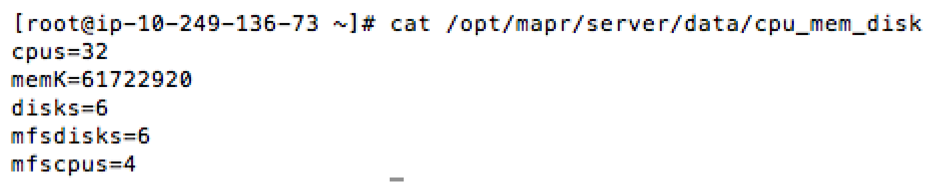
2015-05-07 21:34:32,454 INFO com.mapr.warden.WardenManager [main]: Total node mfs cpu cores : 4

2015-05-07 21:34:32,454 INFO com.mapr.warden.WardenManager [main]: Total available cpu cores : 28

2015-05-07 21:34:32,454 INFO com.mapr.warden.WardenManager [main]: Total node disks : 7

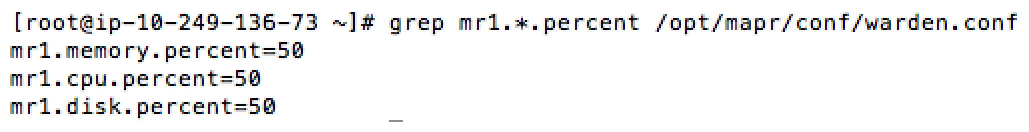
2015-05-07 21:34:32,454 INFO com.mapr.warden.WardenManager [main]: Total node mfs disks: 6

This value is also written to file /opt/mapr/server/data/cpu_mem_disk



```
[root@ip-10-249-136-73 ~]# cat /opt/mapr/server/data/cpu_mem_disk
cpus=32
memK=61722920
disks=6
mfsdisks=6
mfscpus=4
```

The next task for warden is to determine what resources should be available for MR1 jobs and MR2 jobs respectively. This is done based on the config in warden.conf below:



```
[root@ip-10-249-136-73 ~]# grep mr1.*.percent /opt/mapr/conf/warden.conf
mr1.memory.percent=50
mr1.cpu.percent=50
mr1.disk.percent=50
```

By default, property mr1.memory.percent,mr1.cpu.percent,mr1.disk.percent is set to 50%, which means MR1 and MR2 should divide the resources evenly. These values only apply when TaskTracker and NodeManager roles are installed on a node.

Note: If only the TaskTracker role is installed, these values will be ignored, and 100% of the resources will be available to process MR1 jobs regardless of the warden.conf settings. I will publish another blog post soon, which will detail how memory calculation works when none of the MR2 services are installed.

Since we have defaulted 50% resources to both MR1 and MR2, we can see this accurately displayed in warden.log. It says "memory for service = 8536.5" (i.e., 50% of memory is reserved for MR2).

2015-05-07 21:35:05,032 INFO com.mapr.warden.WardenMapRed [tasktracker_monitor]: Before adjustment, memoryForTasks = 17073, memory for service = 8536.5

Warden.log also displays "Before adjustment" and "After adjustment" as shown below. It's important to understand what "after adjustment" means and how MR1 memory is calculated, since clearly memory for MR1 tasks "memory for MR1 tasks = 8192 " is not 50% of 17073MB.

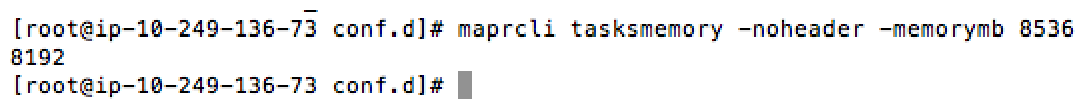
2015-05-07 21:35:08,490 INFO com.mapr.warden.WardenMapRed [nodemanager_monitor]: Before adjustment, memoryForTasks = 17073, memory for service = 8536.5

2015-05-07 21:35:10,453 INFO com.mapr.warden.WardenMapRed [nodemanager_monitor]: After adjustment, memory for MR1 tasks = 8192, memory for service = 8881.0

Final MR1 memory calculations

Below, you can see that the command is invoked in the background by warden.

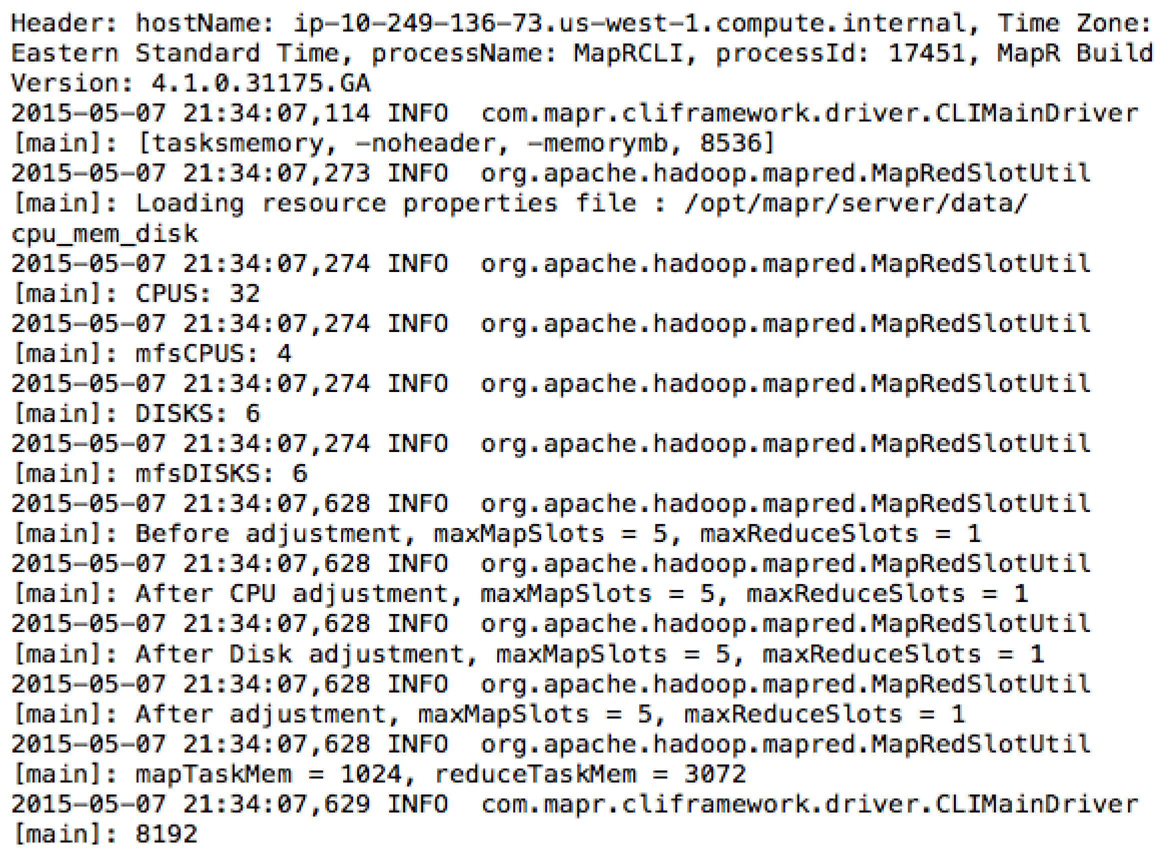
maprcli tasksmemory -noheader -memorymb



```
[root@ip-10-249-136-73 conf.d]# maprcli tasksmemory -noheader -memorymb 8536
8192
[root@ip-10-249-136-73 conf.d]#
```

The output of the above command is the memory MR1 tasks will use without causing any bottlenecks on the system resources when attempts are running in parallel. This is the value which is logged in warden.log as "After adjustment, memory for MR1 tasks = **8192**" which was shown in the earlier log snippet.

The steps involved when the above command runs is logged in /opt/mapr/logs/maprcli-mapr*.log



```
Header: hostName: ip-10-249-136-73.us-west-1.compute.internal, Time Zone:
Eastern Standard Time, processName: MapRCLI, processId: 17451, MapR Build
Version: 4.1.0.31175.GA
2015-05-07 21:34:07,114 INFO  com.mapr.cliframework.driver.CLIMainDriver
[main]: [tasksmemory, -noheader, -memorymb, 8536]
2015-05-07 21:34:07,273 INFO  org.apache.hadoop.mapred.MapRedSlotUtil
[main]: Loading resource properties file : /opt/mapr/server/data/
cpu_mem_disk
2015-05-07 21:34:07,274 INFO  org.apache.hadoop.mapred.MapRedSlotUtil
[main]: CPUS: 32
2015-05-07 21:34:07,274 INFO  org.apache.hadoop.mapred.MapRedSlotUtil
[main]: mfsCPUS: 4
2015-05-07 21:34:07,274 INFO  org.apache.hadoop.mapred.MapRedSlotUtil
[main]: DISKS: 6
2015-05-07 21:34:07,274 INFO  org.apache.hadoop.mapred.MapRedSlotUtil
[main]: mfsDISKS: 6
2015-05-07 21:34:07,628 INFO  org.apache.hadoop.mapred.MapRedSlotUtil
[main]: Before adjustment, maxMapSlots = 5, maxReduceSlots = 1
2015-05-07 21:34:07,628 INFO  org.apache.hadoop.mapred.MapRedSlotUtil
[main]: After CPU adjustment, maxMapSlots = 5, maxReduceSlots = 1
2015-05-07 21:34:07,628 INFO  org.apache.hadoop.mapred.MapRedSlotUtil
[main]: After Disk adjustment, maxMapSlots = 5, maxReduceSlots = 1
2015-05-07 21:34:07,628 INFO  org.apache.hadoop.mapred.MapRedSlotUtil
[main]: After adjustment, maxMapSlots = 5, maxReduceSlots = 1
2015-05-07 21:34:07,628 INFO  org.apache.hadoop.mapred.MapRedSlotUtil
[main]: mapTaskMem = 1024, reduceTaskMem = 3072
2015-05-07 21:34:07,629 INFO  com.mapr.cliframework.driver.CLIMainDriver
[main]: 8192
```

Note: If needed, I can write a separate post explaining the steps involved in calculating how this command determines memory for MR1
.

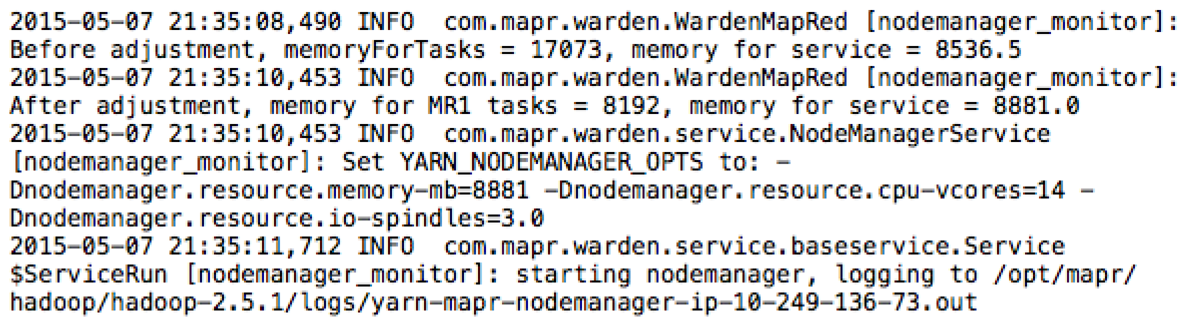
Final MR2 Memory Calculations

Since MR1 is already allocated 8192MB, the rest of the memory from total memory for tasks is given back to the Nodemanager.

Memory for Nodemanager (MR2) = Total memory for tasks - Memory for MR1

```
                    = 17073 -  8192
```

= **8881**



```
2015-05-07 21:35:08,490 INFO  com.mapr.warden.WardenMapRed [nodemanager_monitor]:
Before adjustment, memoryForTasks = 17073, memory for service = 8536.5
2015-05-07 21:35:10,453 INFO  com.mapr.warden.WardenMapRed [nodemanager_monitor]:
After adjustment, memory for MR1 tasks = 8192, memory for service = 8881.0
2015-05-07 21:35:10,453 INFO  com.mapr.warden.service.NodeManagerService
[nodemanager_monitor]: Set YARN_NODEMANAGER_OPTS to: -
Dnodemanager.resource.memory-mb=8881 -Dnodemanager.resource.cpu-vcores=14 -
Dnodemanager.resource.io-spindles=3.0
2015-05-07 21:35:11,712 INFO  com.mapr.warden.service.baseservice.Service
$ServiceRun [nodemanager_monitor]: starting nodemanager, logging to /opt/mapr/
hadoop/hadoop-2.5.1/logs/yarn-mapr-nodemanager-ip-10-249-136-73.out
```

Hence 8881MB gets assigned to NodeManager; this memory is used by NodeManager for processing YARN containers on the node in MB.

Key Takeaways

1. The memory for each service is calculated from warden min, max and percent settings
   1. Warden will allocate percentages of the memory to the OS and each service chipping away memory and leftover memory for MR1 and MR2 jobs on the node.
   2. If node is a CLDB or ZK node, some memory is reserved based on the formula below:

totalFreeMemoryInMB=round(max(totalGlobalAvailableMemory * 0.75, totalGlobalAvailableMemory - 1500))

4. Warden will determine what resources should be available for MapReduce v1 jobs and MR2 jobs respectively, based on the config (mr1.memory.percent, mr1.cpu.percent, mr1.disk.percent) in the warden.conf file, and if TT and NM are configured on the node.
   1. MR1 memory is calculated based on the formula below, which ensures that there is no resource bottleneck.

maprcli tasksmemory -noheader -memorymb

6. The memory given to MR2 is the total memory for tasks minus the memory reserved for MR1.

Memory for Nodemanager (MR2) = Total memory for tasks - Memory for MR1

In this tutorial, you learned how memory is managed from the time you start warden, how memory is allocated to each MapR service installed on the node, and how memory is made available on MR1/MR2 for running jobs. In a future blog post, I'll describe how memory calculation works when none of the MR2 services are installed. Please let me know if you have any feedback on the tutorial.