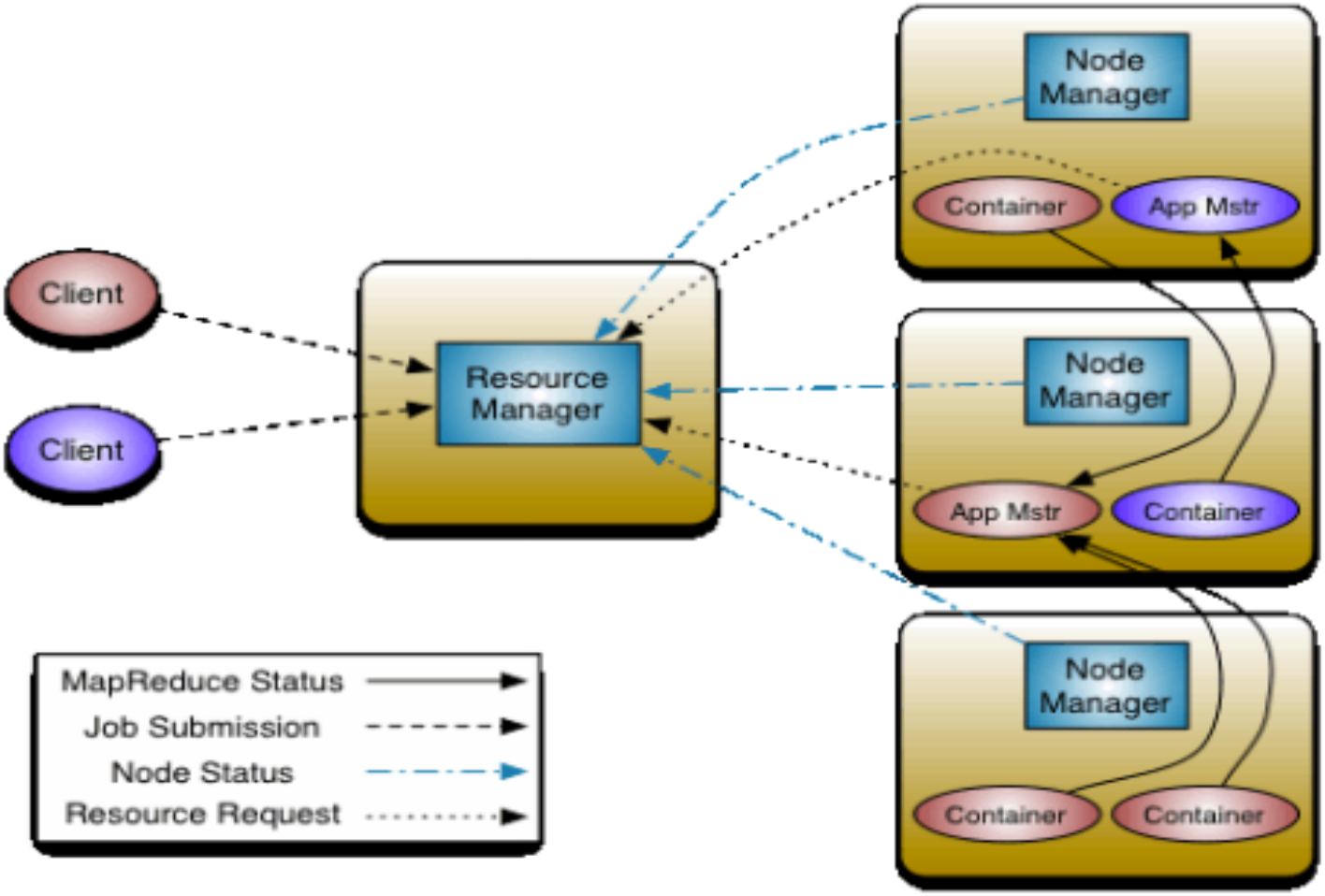
YARN

Yet Another Resource Negotiator

Please do not Yawn - grab a tea, coffee !!

YARN Architecture



Important Points

- YARN is to split up the functionalities of resource management and job scheduling/monitoring into separate daemons.
- The **ResourceManager** is the ultimate authority that arbitrates resources among all the applications in the system.
- **NodeManager** is the per-machine framework agent who is responsible for containers, monitoring their resource usage (cpu, memory, disk, network) and reporting the same to the ResourceManager/Scheduler.
- The per-application **ApplicationMaster** is framework specific library and is tasked with negotiating resources from the ResourceManager and working with the NodeManager(s) to execute and monitor the tasks.
- The Scheduler is pure scheduler in the sense that it performs no monitoring or tracking of status for the application - no guarantees about restarting failed tasks either due to application failure or hardware failures

What is a container?

- A container is a YARN JVM process.
- In MapReduce the application master service, mapper and reducer tasks are all containers that execute inside the YARN framework.

Let’s understand

- YARN Resource Manager (RM) allocates resources to the application through logical queues which include memory, CPU, and disks resources.
- By default, the RM will allow up to:

    - **yarn.scheduler.maximum-allocation-mb** - 8192MB
    - **yarn.scheduler.minimum-allocation-mb** – 1024MB

    - AM can only request resources from the RM that are in increments of ("**yarn.scheduler.minimum-allocation-mb**") and do not exceed ("**yarn.scheduler.maximum-allocation-mb**")
    - The AM is responsible for rounding off ("**mapreduce.map.memory.mb**") and ("**mapreduce.reduce.memory.mb**") to a value divisible by the ("**yarn.scheduler.minimum-allocation-mb**")

Let's understand

- **YARN**
    - yarn.scheduler.minimum-allocation-mb
    - yarn.scheduler.maximum-allocation-mb
    - yarn.nodemanager.vmem-pmem-ratio
    - yarn.nodemanager.resource.memory.mb
- **MapReduce**

Map Memory
    mapreduce.map.java.opts
    mapreduce.map.memory.mb
Reduce Memory
    mapreduce.reduce.java.opts
    mapreduce.reduce.memory.mb

Memory - Allocation Example

- **mapreduce.map.memory.mb** = 1536MB
- **mapreduce.reduce.memory.mb** = 3072MB
- **mapreduce.map.java.opts** = -Xmx1024m
- **yarn.nodemanager.vmem-pmem-ratio = 2.1**

```
Current usage: 2.1gb of 2.0gb physical memory used; 1.6gb of 3.15gb virtual memory used. Killing container.
```

Heap Size

Each container is actually a JVM process, and above "-Xmx" of java-opts should fit in the allocated memory size.

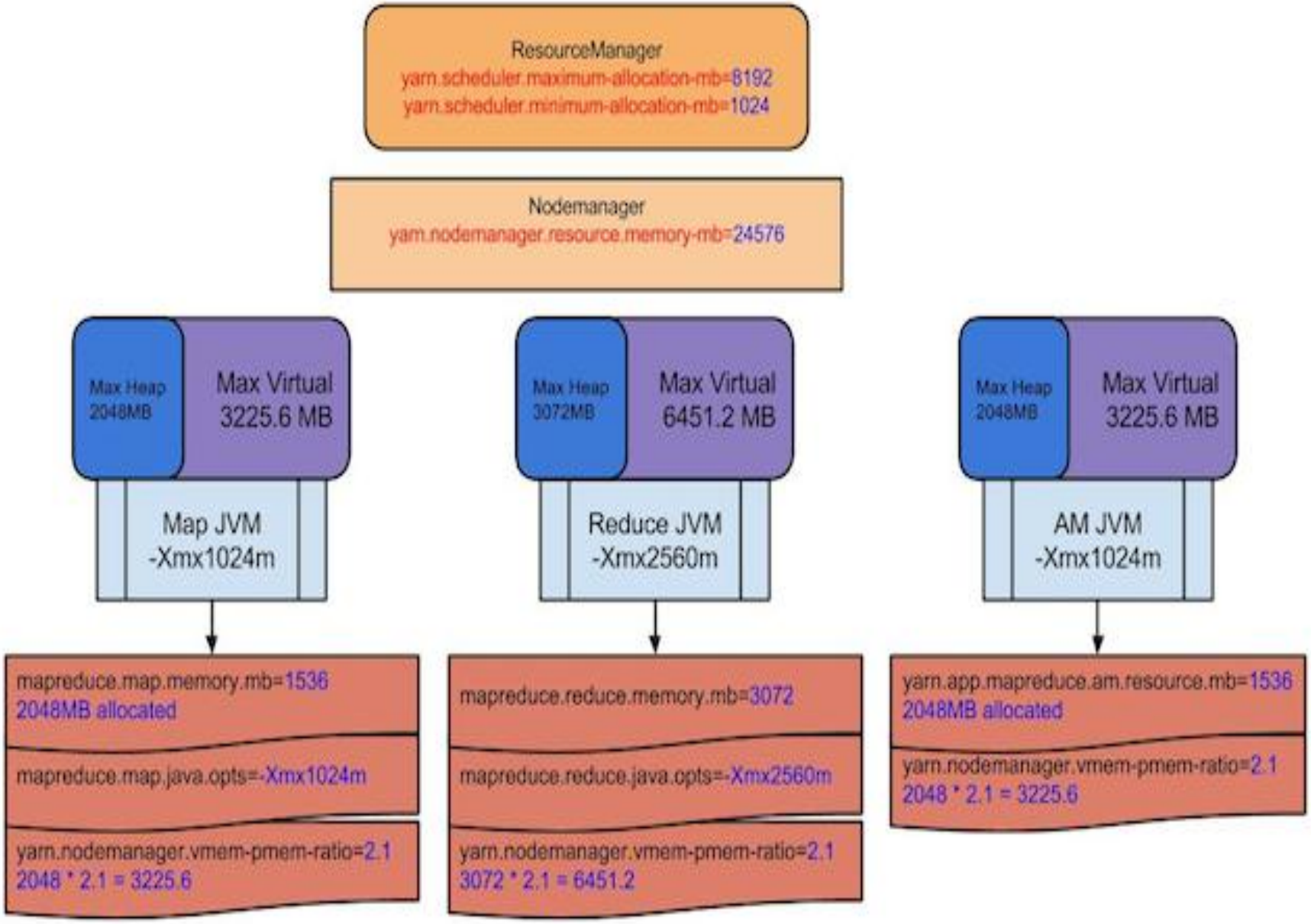
One best practice is to set it to **0.8 * (container memory allocation)**.

For example:

mapreduce.map.memory.mb=4096,

we can set

mapreduce.map.java.opts=-Xmx3277m.


ResourceManager
yarn.scheduler.maximum-allocation-mb=8192
yarn.scheduler.minimum-allocation-mb=1024
Nodemanager
yarn.nodemanager.resource.memory-mb=24576
Max Heap 2048MB
Max Virtual 3225.6 MB
Map JVM -Xmx1024m
mapreduce.map.memory.mb=1536
2048MB allocated
mapreduce.map.java.opts=-Xmx1024m
yarn.nodemanager.vmem-pmem-ratio=2.1
2048 * 2.1 = 3225.6
Max Heap 3072MB
Max Virtual 6451.2 MB
Reduce JVM -Xmx2560m
mapreduce.reduce.memory.mb=3072
mapreduce.reduce.java.opts=-Xmx2560m
yarn.nodemanager.vmem-pmem-ratio=2.1
3072 * 2.1 = 6451.2
Max Heap 2048MB
Max Virtual 3225.6 MB
AM JVM -Xmx1024m
yarn.app.mapreduce.am.resource.mb=1536
2048MB allocated
yarn.nodemanager.vmem-pmem-ratio=2.1
2048 * 2.1 = 3225.6

vCore -Allocation Example

For instance, if a single node has 24 processors, then an appropriate set of params can be set as follows:

Name	Value
mapreduce.map.cpu.vcores	1
mapreduce.reduce.cpu.vcores	1
yarn.app.mapreduce.am.resource.cpu-vcores	1
yarn.scheduler.maximum-allocation-vcores	24
yarn.scheduler.minimum-allocation-vcores	1
yarn.nodemanager.resource.cpu-vcores	24

vCore -Allocation Example

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;
2.2 GB of 2.1 GB virtual memory used. Killing container.
```

And this is a sample error for physical memory checker:

```
Current usage: 2.1gb of 2.0gb physical memory used;
1.1gb of 3.15gb virtual memory used. Killing container.
```

Other Factors

There are many factors which can affect the memory requirement for each container.

- 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

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.

Best is Check AM logs !!

Identify Bottleneck

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.

840 GB is wasted or free

Key Takeaways

- Be familiar with lower and upper bound of resource requirements for mapper and reducer.
- Be aware of the virtual and physical memory checker.
- Set -Xmx of java-opts of each container to 0.8 * (container memory allocation).
- Make sure each type of container meets proper resource requirement.
- Fully utilize bottleneck resource.

Let’s look all these practically !!

- YARN Installation
- Configuration
- Memory parameters settings.