# 相关参数

## Yarn参数

打开yarn-site.xml文件（/etc/hadoop/conf）

|  |
| --- |
| <property>  <name>yarn.scheduler.minimum-allocation-mb</name>  <value>1024</value>  </property>  <property>  <name>yarn.scheduler.increment-allocation-mb</name>  <value>512</value>  </property> |

找到这两个配置

|  |  |
| --- | --- |
| 配置项 | 说明 |
| yarn.scheduler.minimum-allocation-mb | 调度时，一个容器能够申请的最小资源 |
| yarn.scheduler.increment-allocation-mb | 调度规整值 |

< yarn.scheduler.minimum-allocation-mb >参数主要和driver.Memory/spark.yarn.am.memory有关。这两个参数都能影响到< yarn.scheduler.minimum-allocation-mb >。如果driver.Memory/spark.yarn.am.memory两个参数同时存在，则以driver.Memory配置为主。

## Executor参数

|  |  |
| --- | --- |
| 配置项 | 说明 |
| num-executors | 执行器数量 |
| executor-memory | 单个执行器内存 |
| spark.yarn.executor.memoryOverhead | 单个执行器的堆内内存大小 |
| driver-memory | Driver的内存 |

这里只列出对内存有影响的参数。

## 计算方法

Spark On Yarn条件下，提交任务后，该任务从逻辑上来看，则分为两个部分：第一是应用的主节点负责资源调度；第二是应用的从节点，负责计算处理。这两个部分会创建不同的资源容器（container）。

为了更加方便的阐述计算方法，这里使用一个例子，如图 1所示：

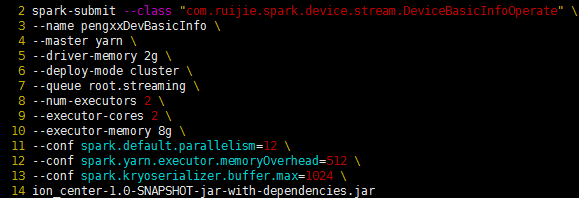


图 1 Spark提交任务到 Yarn 集群的例子

### AM container

该容器是spark的主节点产生的容器，主要负责资源调度，也即 Application Master Container。在我们提交任务的时候实际上就能够感觉到它的存在，如图 2所示：

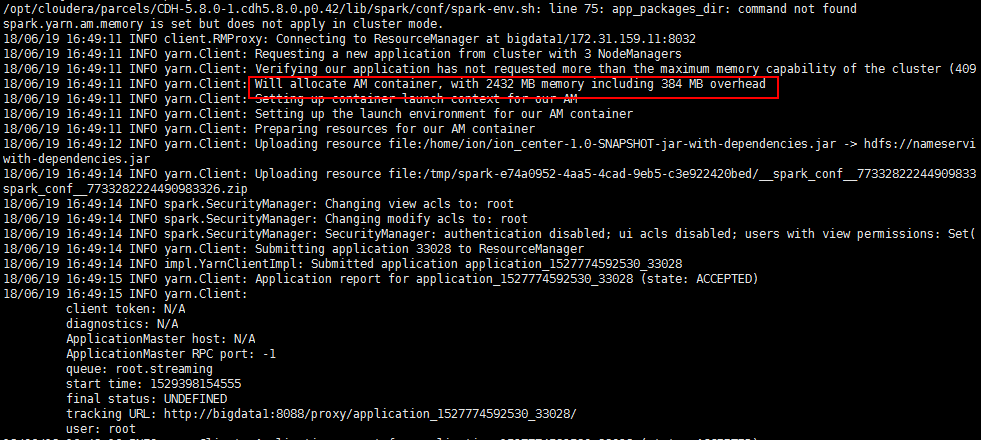


图 2 AM Container相关信息

在图 2中可以发现它创建了一个 AM 容器，而且使用 2432MB 的内存，其中 384MB是堆内内存。

这个 2432MB 的内存是如何计算出来的？

在图 1例子中，设置了

* driver-memory 2g

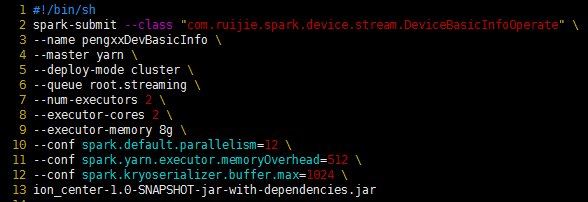
以及yarn-site.xml中的配置

* yarn.scheduler.minimum-allocation-mb=1024
* yarn.scheduler.increment-allocation-mb=512

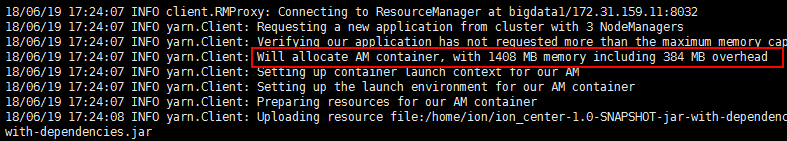
yarn集群中AM的最小调度内存为 1024MB ，现在设置了driver-memory和spark.yarn.am.memory，这里将以driver-memory为主，因此AM container实际内存就是2048+max(384,0.1\*2048)=2432。

考虑不同的情况：

1. 不设置driver-memory

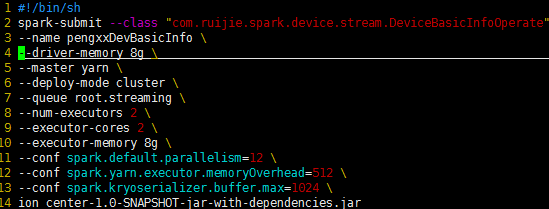


运行结果为：

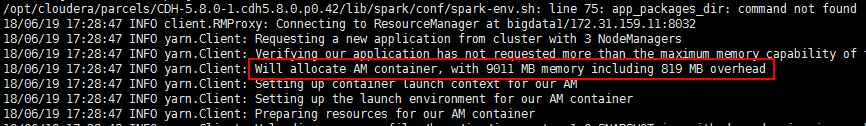


在该结果中发现 AM 容器的内存使用量为1408MB，这里则是使用了默认的资源调度最小参数yarn.scheduler.minimum-allocation-mb。

②driver-memory给非常大（8G）

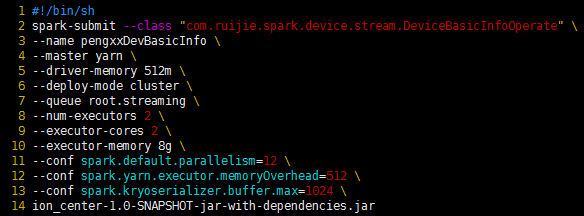


运行结果为：

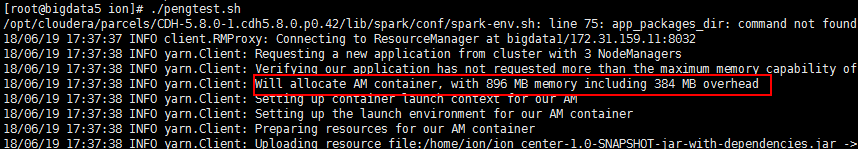


在该结果中发现 AM 容器的内存使用量为9011MB 且堆内内存不在是384MB，这个堆内内存的计算方式就是在 384MB 和0.1\*8G中选择最大值。

③driver-memory给非常小（512MB）



运行结果为：

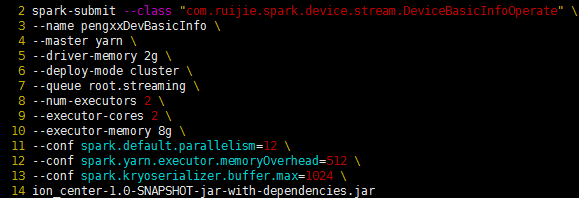


在该结果中可以发现 AM 容器的使用内存减小了，这说明driver-memory能够影响yarn.site.xml的配置。

**结论：**AM容器在申请内存资源时其主要参考了driver-memory配置，以及yarn配置文件中的默认最小资源调度值。而其真实内存则是：

### Executor container

该类型容器主要是指计算逻辑执行器产生的容器，该容器的数量则是执行器的数量（num-executors）。其内存计算方式和 AM container的相似。这里还是以图 1的数据举例：



运行结果为：



在这个结果中，内存使用量为19.5GB，那么怎么计算的呢？

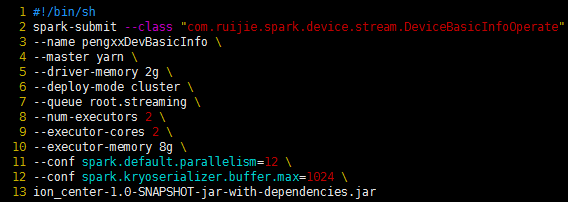
对于每个容器而言都应该包含逻辑处理单元内存以及堆内内存，逻辑处理单元的内存可以由executor-memory参数决定，堆内内存则是 max(384, executor.memoryOverhead)，这两个的和才是一个执行器容器的内存消耗，也就是8g+max(384,512)=8.5G。

在yarn-site.xml中有一个调度规整值的配置yarn.scheduler.increment-allocation-mb。该配置默认是512mb，也就是说，容器的内存使用量应该按照调度规整值的倍数增长，那么实际上执行器容器的内存消耗为ROUNDUP((8\*1024+max(384,512))/512)=8.5GB。这里由于设置了2个执行器，因此会有四个执行器容器，即总占用2\*8.5GB=17GB内存。

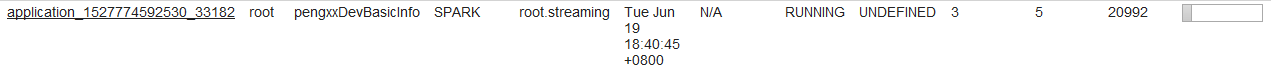
从yarn集群的资源消耗来看，总消耗为19.5GB内存，是因为将 AM container消耗内存和executor container消耗内存相加。即：

* AM container 内存 = ROUNDUP((2\*1024+max(384,0.1\*2\*1024))/512) = 2.5G
* executor container内存= ROUNDUP((8\*1024+max(384,512))/512)=17G

现在考虑另外一种情况，就是不设置executor.memoryOverhead。配置为：



运行结果为：



可以看到该结果中的内存使用总量反而增加了，变成了20.5G，其计算思路是：

* AM container 内存 = ROUNDUP(2\*1024+max(384,0.1\*2\*1024))/512)\*512 = 2.5G
* executor container内存= ROUNDUP((8\*1024+max(384,0.1\*8\*1024))/512)\*512=18G

### :总结：

对于一个Spark任务，当提交后会产生两种类型的容器，这两种容器分别是 AM container 和 Executor container，AM container主要资源调度主节点，也会占用内存，同时消耗一个内核。

若作了以下配置：

|  |  |  |
| --- | --- | --- |
| 参数 | 值 | 缩写假设 |
| driver-memory | 2G | Dm |
| num-executors | 2个 | Ne |
| executor-cores | 2个 | Ec |
| executor-memory | 8G | Em |
| spark.yarn.executor.memoryOverhead | 512mb | Emo |
| yarn.scheduler.minimum-allocation-mb | 1024mb | Smam |
| yarn.scheduler.increment-allocation-mb | 512mb | Siam |

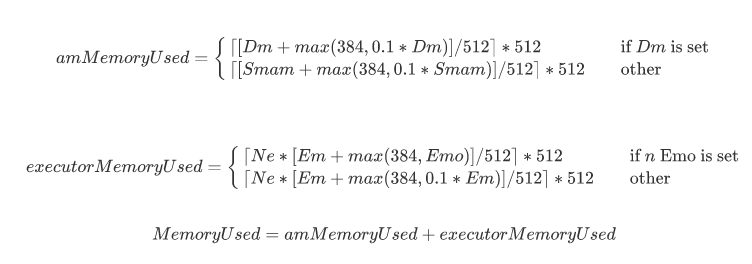
那么在提交到yarn集群后的资源消耗情况则是：

|  |  |
| --- | --- |
| 资源 | 值 |
| Running Containers | 3个 |
| Allocated CPU VCores | 5个 |
| Allocated Memory MB | 20992 |

消耗3个容器，是因为 AM 创建了1个容器，一个Executor创建一个容器，这里有两个Executor。

CPU虚拟核数为5个，是因为 AM 使用了1个核，一个Executor通过参数配置了2个核，这里有两个Executor，总消耗4个核。

总内存消耗量计算公式如下：



# 内存最少占用方案

## 动态资源调整

根据内存计算公式可得，Spark的内存占用下限和调度规整值有极大的关联度。例如在设置了driver.memory=512m和executor.memory=512m，调度规整值默认是512mb，当提交到yarn集群后最少占用内存则为2GB，因为AM和Executor要分别占用1GB内存。当降低调度规整值到128m时，内存使用量是1792MB=1.75GB。因此发现降低调度规整值的效果并不明显。实际上，Spark任务应该结合实际环境的数据量和算法复杂度来决定内存配置。

对于Spark-Streaming任务，可以使用Spark动态资源分配(Dynamic Resource Allocation,DRA)进行内存动态调整，仅仅设置最大和最小内存使用量即可。

1. 将$SPARK\_HOME/lib/spark-1.5.2-yarn-shuffle.jar 文件拷贝到hadoop/lib （hadoop lib）目录下
2. 在所有节点的yarn-site.xml中NodeManager上运行的附属服务中加入spark\_shuffle及对应的class类，如下：

|  |
| --- |
| <property>  <name>yarn.nodemanager.aux-services</name>  <value>mapreduce\_shuffle,spark\_shuffle</value>  </property>  <property>  <name>yarn.nodemanager.aux-services.spark\_shuffle.class</name>  <value>org.apache.spark.network.yarn.YarnShuffleService</value>  </property> |

1. 重启所有的Nodemanager，即可配置安装好spark的动态资源的分配。
2. 在spark-submit命令中添加以下配置：

|  |
| --- |
| spark-submit \  --master yarn \  --deploy-mode cluster \  --executor-cores 3 \  --executor-memory 10G \  --driver-memory 4G \  # 开启自动调节  --conf spark.dynamicAllocation.enabled=true \  --conf spark.shuffle.service.enabled=true \  # 初始执行器数量  --conf spark.dynamicAllocation.initialExecutors=5 \  --conf spark.dynamicAllocation.maxExecutors=40 \  --conf spark.dynamicAllocation.minExecutors=0 \  # executor空闲多长时间就被移除释放  --conf spark.dynamicAllocation.executorIdleTimeout=30s \  # 积压多久任务的时候要新申请executor  --conf spark.dynamicAllocation.schedulerBacklogTimeout=10s \ |