

尚硅谷大数据技术之 Hadoop (Yarn)

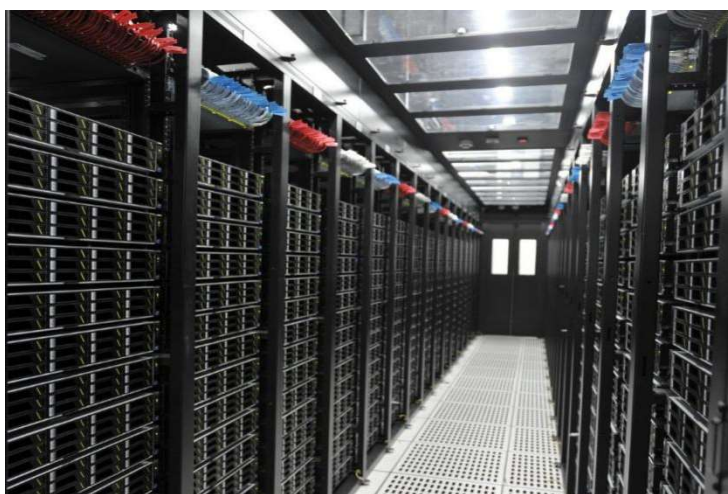
(作者：尚硅谷大数据研发部)

版本：V3.3

第 1 章 Yarn 资源调度器

思考：

- 1) 如何管理集群资源？
- 2) 如何给任务合理分配资源？



Yarn 是一个资源调度平台，负责为运算程序提供服务器运算资源，相当于一个分布式的操作系统平台，而 **MapReduce** 等运算程序则相当于运行于操作系统之上的应用程序。

1.1 Yarn 基础架构

YARN 主要由 ResourceManager、NodeManager、ApplicationMaster 和 Container 等组件构成。

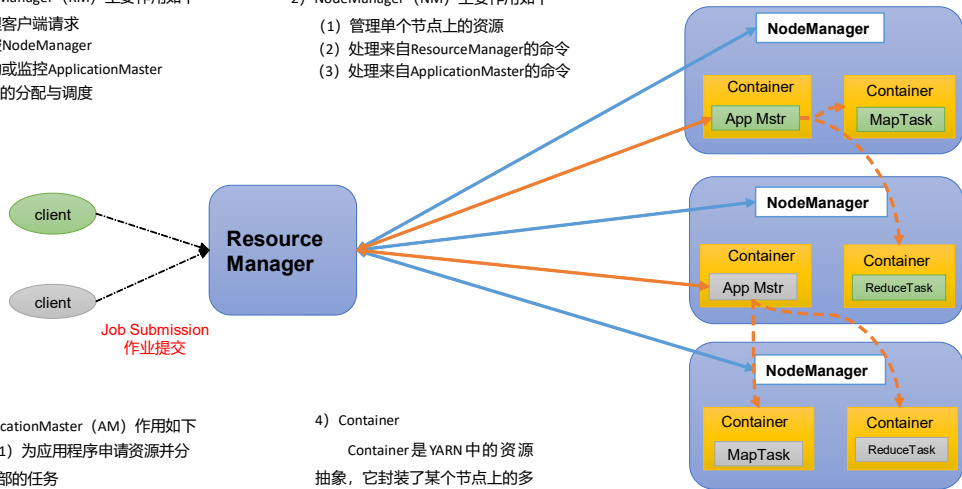
YARN基础架构

- 1) ResourceManager (RM) 主要作用如下
- (1) 处理客户端请求
 - (2) 监控NodeManager
 - (3) 启动或监控ApplicationMaster
 - (4) 资源的分配与调度

- 2) NodeManager (NM) 主要作用如下
- (1) 管理单个节点上的资源
 - (2) 处理来自ResourceManager的命令
 - (3) 处理来自ApplicationMaster的命令

- 3) ApplicationMaster (AM) 作用如下
- (1) 为应用程序申请资源并分配给内部的任务
 - (2) 任务的监控与容错

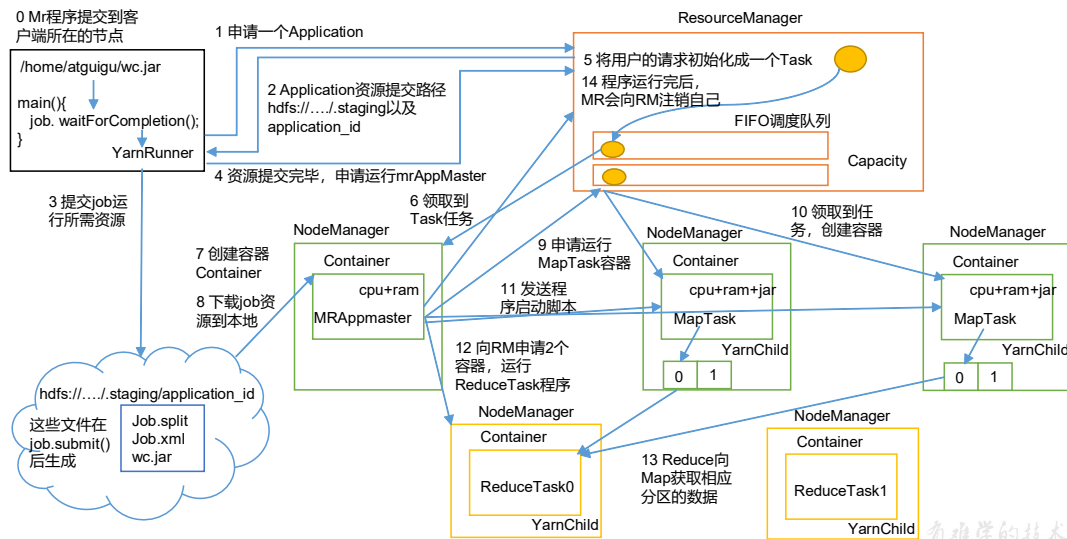
- 4) Container
- Container 是 YARN 中的资源抽象，它封装了某个节点上的多维度资源，如内存、CPU、磁盘、网络等。



让天下没有难学的技术

1.2 Yarn 工作机制

YARN工作机制



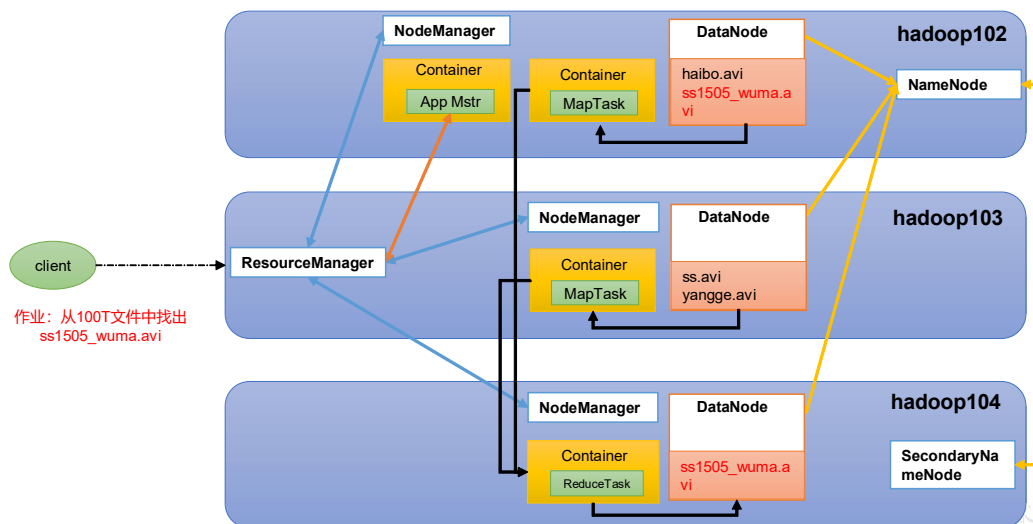
有难学的技术

- (1) MR 程序提交到客户端所在的节点。
- (2) YarnRunner 向 ResourceManager 申请一个 Application。
- (3) RM 将该应用程序的资源路径返回给 YarnRunner。
- (4) 该程序将运行所需资源提交到 HDFS 上。
- (5) 程序资源提交完毕后，申请运行 mrAppMaster。
- (6) RM 将用户的请求初始化成一个 Task。
- (7) 其中一个 NodeManager 领取到 Task 任务。
- (8) 该 NodeManager 创建容器 Container，并产生 MRAppmaster。

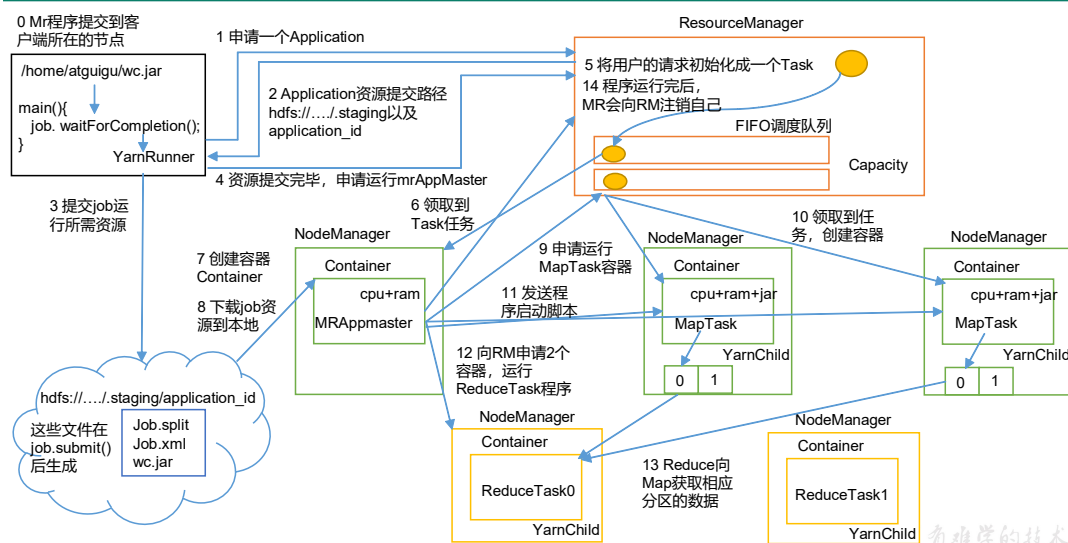
- (9) Container 从 HDFS 上拷贝资源到本地。
- (10) MRAppmaster 向 RM 申请运行 MapTask 资源。
- (11) RM 将运行 MapTask 任务分配给另外两个 NodeManager, 另两个 NodeManager 分别领取任务并创建容器。
- (12) MR 向两个接收到任务的 NodeManager 发送程序启动脚本, 这两个 NodeManager 分别启动 MapTask, MapTask 对数据分区排序。
- (13) MRAppMaster 等待所有 MapTask 运行完毕后, 向 RM 申请容器, 运行 ReduceTask。
- (14) ReduceTask 向 MapTask 获取相应分区的数据。
- (15) 程序运行完毕后, MR 会向 RM 申请注销自己。

1.3 作业提交全过程

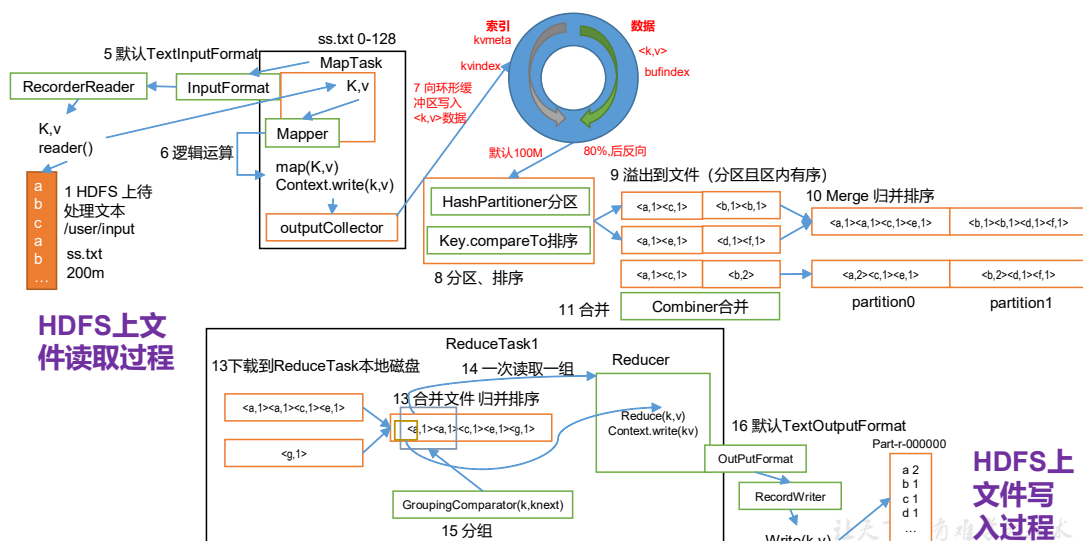
HDFS、YARN、MapReduce三者关系



作业提交过程之YARN



作业提交过程之HDFS & MapReduce



作业提交全过程详解

(1) 作业提交

- 第1步: Client 调用 `job.waitForCompletion()` 方法, 向整个集群提交 MapReduce 作业。
- 第2步: Client 向 RM 申请一个作业 id。
- 第3步: RM 给 Client 返回该 job 资源的提交路径和作业 id。
- 第4步: Client 提交 jar 包、切片信息和配置文件到指定的资源提交路径。
- 第5步: Client 提交完资源后, 向 RM 申请运行 MrAppMaster。

(2) 作业初始化

- 第6步: 当 RM 收到 Client 的请求后, 将该 job 添加到容量调度器中。

第 7 步：某一个空闲的 NM 领取到该 Job。

第 8 步：该 NM 创建 Container，并产生 MRAppmaster。

第 9 步：下载 Client 提交的资源到本地。

(3) 任务分配

第 10 步：MrAppMaster 向 RM 申请运行多个 MapTask 任务资源。

第 11 步：RM 将运行 MapTask 任务分配给另外两个 NodeManager，另两个 NodeManager 分别领取任务并创建容器。

(4) 任务运行

第 12 步：MR 向两个接收到任务的 NodeManager 发送程序启动脚本，这两个 NodeManager 分别启动 MapTask，MapTask 对数据分区排序。

第 13 步：MrAppMaster 等待所有 MapTask 运行完毕后，向 RM 申请容器，运行 ReduceTask。

第 14 步：ReduceTask 向 MapTask 获取相应分区的数据。

第 15 步：程序运行完毕后，MR 会向 RM 申请注销自己。

(5) 进度和状态更新

YARN 中的任务将其进度和状态(包括 counter)返回给应用管理器，客户端每秒(通过 `mapreduce.client.progressmonitor.pollinterval` 设置)向应用管理器请求进度更新，展示给用户。

(6) 作业完成

除了向应用管理器请求作业进度外，客户端每 5 秒都会通过调用 `waitForCompletion()` 来检查作业是否完成。时间间隔可以通过 `mapreduce.client.completion.pollinterval` 来设置。作业完成之后，应用管理器和 Container 会清理工作状态。作业的信息会被作业历史服务器存储以备之后用户核查。

1.4 Yarn 调度器和调度算法

目前，Hadoop 作业调度器主要有三种：FIFO、容量（Capacity Scheduler）和公平（Fair Scheduler）。Apache Hadoop3.1.3 默认的资源调度器是 Capacity Scheduler。

CDH 框架默认调度器是 Fair Scheduler。

具体设置详见：yarn-default.xml 文件

```
<property>
  <description>The class to use as the resource scheduler.</description>
  <name>yarn.resourcemanager.scheduler.class</name>
  <value>org.apache.hadoop.yarn.server.resourcemanager.scheduler.capaci
```

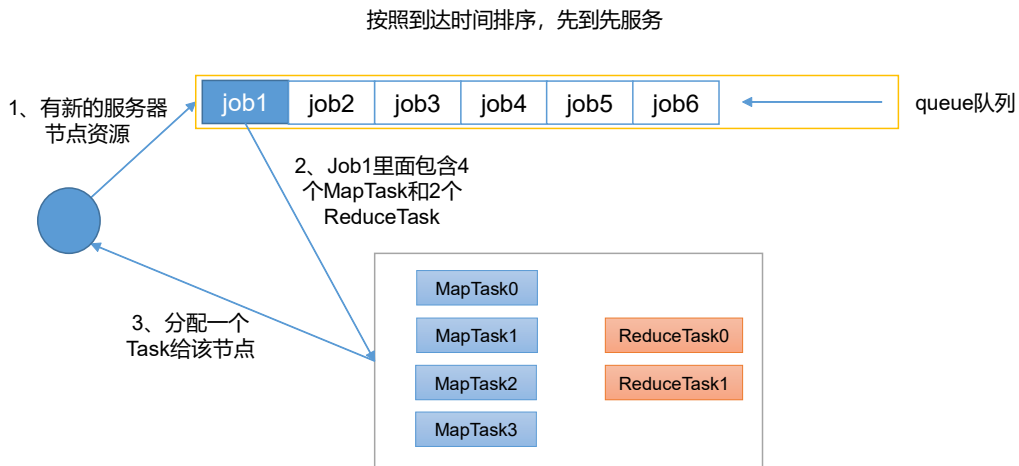
```
ty.CapacityScheduler</value>
</property>
```

1.4.1 先进先出调度器 (FIFO)

FIFO 调度器 (First In First Out)：单队列，根据提交作业的先后顺序，先来先服务。



FIFO调度器



让天下没有难学的技术

优点：简单易懂；

缺点：不支持多队列，生产环境很少使用；

1.4.2 容量调度器 (Capacity Scheduler)

Capacity Scheduler 是 Yahoo 开发的多用户调度器。



容量调度器特点



- 1、多队列：每个队列可配置一定的资源量，每个队列采用FIFO调度策略。
- 2、容量保证：管理员可为每个队列设置资源最低保证和资源使用上限
- 3、灵活性：如果一个队列中的资源有剩余，可以暂时共享给那些需要资源的队列，而一旦该队列有新的应用程序提交，则其他队列借调的资源会归还给该队列。
- 4、多租户：
 - 支持多用户共享集群和多应用程序同时运行。
 - 为了防止同一个用户的作业独占队列中的资源，该调度器会对同一用户提交的作业所占资源量进行限定。

让天下没有难学的技术

容量调度器资源分配算法

```

root
|--queueA 20%
|--queueB 50%
|--queueC 30%
|   |--ss 50%
|   |--cls 50%

```

1) 队列资源分配

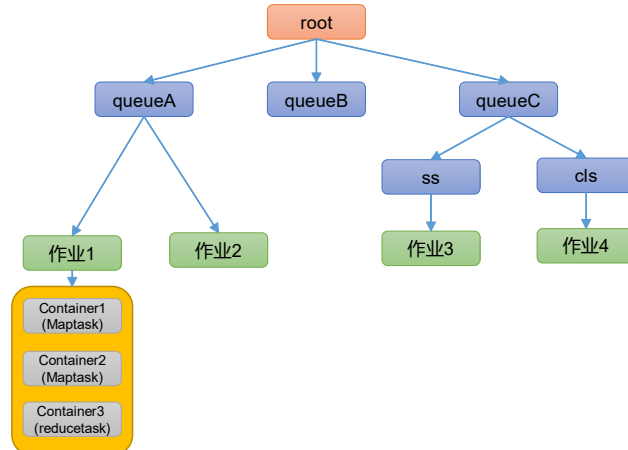
从root开始，使用深度优先算法，**优先选择资源占用率最低的队列**分配资源。

2) 作业资源分配

默认按照提交作业的**优先级**和**提交时间**顺序分配资源。

3) 容器资源分配

按照容器的**优先级**分配资源；
如果优先级相同，按照**数据本地性原则**：
(1) 任务和数据在同一节点
(2) 任务和数据在同一机架
(3) 任务和数据不在同一节点也不在同一机架



让天下没有难学的技术

1.4.3 公平调度器 (Fair Scheduler)

Fair Scheduler 是 Facebook 开发的多用户调度器。

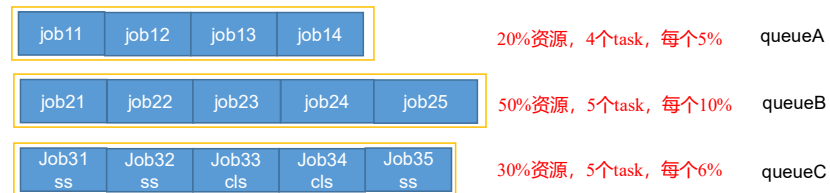
公平调度器特点

```

root
|--queueA 20%
|--queueB 50%
|--queueC 30%
|   |--ss 50%
|   |--cls 50%

```

同队列所有任务共享资源，在时间尺度上获得公平的资源



1) 与容量调度器相同点

- (1) 多队列：支持多队列多作业
- (2) 容量保证：管理员可为每个队列设置资源最低保证和资源使用上线
- (3) 灵活性：如果一个队列中的资源有剩余，可以暂时共享给那些需要资源的队列，而一旦该队列有新的应用程序提交，则其他队列借调的资源会归还给该队列。
- (4) 多租户：支持多用户共享集群和多应用程序同时运行；为了防止同一个用户的作业独占队列中的资源，该调度器会对同一用户提交的作业所占资源量进行限定。

2) 与容量调度器不同点

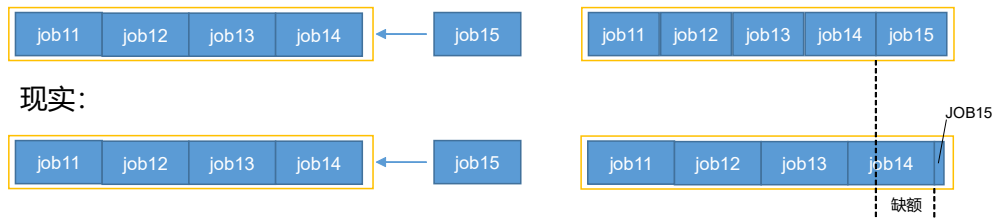
- (1) 核心调度策略不同
 - 容量调度器：优先选择**资源利用率低**的队列
 - 公平调度器：优先选择对资源的**缺额**比例大的
- (2) 每个队列可以单独设置资源分配方式
 - 容量调度器：FIFO、**DRF**
 - 公平调度器：FIFO、**FAIR**、**DRF**

让天下没有难学的技术



公平调度器——缺额

理想：



- 公平调度器设计目标是：在时间尺度上，所有作业获得公平的资源。某一时刻一个作业应获资源和实际获取资源的差距叫“缺额”
- 调度器会**优先为缺额大的作业分配资源**

让天下没有难学的技术



公平调度器队列资源分配方式

1) FIFO策略

公平调度器每个队列资源分配策略如果选择FIFO的话，此时公平调度器相当于上面讲过的容量调度器。

2) Fair策略

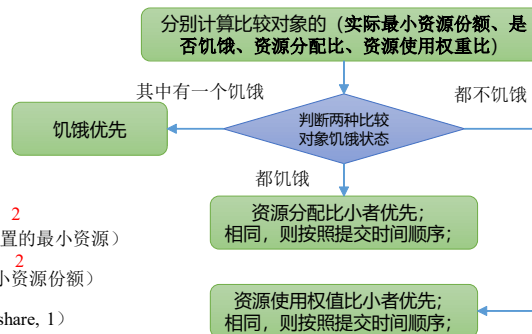
Fair 策略（默认）是一种基于最大最小公平算法实现的资源多路复用方式，默认情况下，每个队列内部采用该方式分配资源。这意味着，如果一个队列中有两个应用程序同时运行，则每个应用程序可得到1/2的资源；如果三个应用程序同时运行，则每个应用程序可得到1/3的资源。

具体资源分配流程和容量调度器一致：

- 选择队列
- 选择作业
- 选择容器

以上三步，每一步都是按照公平策略分配资源

- **实际最小资源份额**：mindshare = Min（资源需求量，配置的最小资源）
- **是否饥饿**：isNeedy = 资源使用量 / 配置的最小资源 < mindshare（实际最小资源份额）
- **资源分配比**：minShareRatio = 资源使用量 / Max（mindshare, 1）
- **资源使用权重比**：useToWeightRatio = 资源使用量 / 权重



让天下没有难学的技术

公平调度器资源分配算法

```

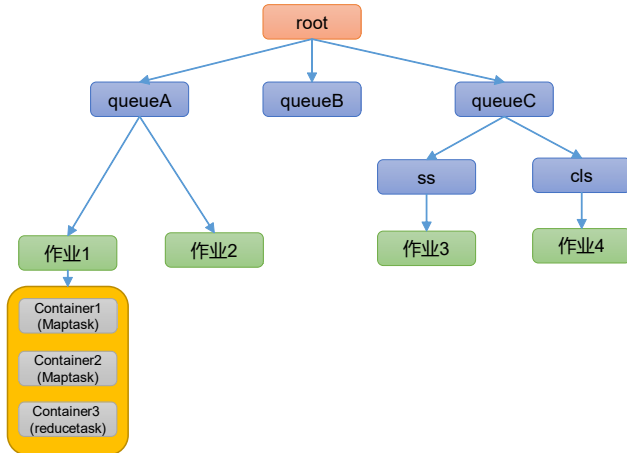
root
|--queueA 20%
|--queueB 50%
|--queueC 30%
|   |--ss 50%
|   |--cls 50%
    
```

(1) 队列资源分配

需求：集群总资源100，有三个队列，对资源的需求分别是：
queueA -> 20, queueB -> 50, queueC -> 30

第一次算： $100 / 3 = 33.33$
queueA: 分33.33 -> 多13.33
queueB: 分33.33 -> 少16.67
queueC: 分33.33 -> 多3.33

第二次算： $(13.33 + 3.33) / 1 = 16.66$
queueA: 分20
queueB: 分33.33 + 16.66 = 50
queueC: 分30



让天下没有难学的技术

公平调度器队列资源分配方式

(2) 作业资源分配

(a) 不加权（关注点是Job的个数）：

需求：有一条队列总资源12个，有4个job，对资源的需求分别是：
job1->1, job2->2, job3->6, job4->5

第一次算： $12 / 4 = 3$
job1: 分3 --> 多2个
job2: 分3 --> 多1个
job3: 分3 --> 差3个
job4: 分3 --> 差2个

第二次算： $3 / 2 = 1.5$
job1: 分1
job2: 分2
job3: 分3 --> 差3个 --> 分1.5 --> 最终：4.5
job4: 分3 --> 差2个 --> 分1.5 --> 最终：4.5

第n次算：一直算到没有空闲资源

(b) 加权（关注点是Job的权重）：

需求：有一条队列总资源16，有4个job
对资源的需求分别是：
job1->4 job2->2 job3->10 job4->4
每个job的权重为：
job1->5 job2->8 job3->1 job4->2

第一次算： $16 / (5+8+1+2) = 1$
job1: 分5 --> 多1
job2: 分8 --> 多6
job3: 分1 --> 少9
job4: 分2 --> 少2

第二次算： $7 / (1+2) = 7/3$
job1: 分4
job2: 分2
job3: 分1 --> 分7/3 (2.33) --> 少6.67
job4: 分2 --> 分14/3 (4.66) --> 多2.66

第三次算： $2.66 / 1 = 2.66$
job1: 分4
job2: 分2
job3: 分1 --> 分2.66/1 --> 分2.66
job4: 分4

第n次算：一直算到没有空闲资源

让天下没有难学的技术



3) DRF策略

DRF (Dominant Resource Fairness)，我们之前说的资源，都是单一标准，例如只考虑内存（也是Yarn默认的情况）。但是很多时候我们资源有很多种，例如内存，CPU，网络带宽等，这样我们很难衡量两个应用应该分配的资源比例。

那么在YARN中，我们用DRF来决定如何调度：

假设集群一共有100 CPU和10T 内存，而应用A需要（2 CPU, 300GB），应用B需要（6 CPU, 100GB）。则两个应用分别需要A（2%CPU, 3%内存）和B（6%CPU, 1%内存）的资源，这就意味着A是内存主导的，B是CPU主导的，针对这种情况，我们可以选择DRF策略对不同应用进行不同资源（CPU和内存）的一个不同比例的限制。

让天下没有难学的技术

1.5 Yarn 常用命令

Yarn 状态的查询，除了可以在 `hadoop103:8088` 页面查看外，还可以通过命令操作。常见的命令操作如下所示：

需求：执行 WordCount 案例，并用 Yarn 命令查看任务运行情况。

```
[atguigu@hadoop102 hadoop-3.1.3]$ myhadoop.sh start

[atguigu@hadoop102 hadoop-3.1.3]$ hadoop jar
share/hadoop/mapreduce/hadoop-mapreduce-examples-3.1.3.jar wordcount
/input /output
```

1.5.1 yarn application 查看任务

(1) 列出所有 Application:

```
[atguigu@hadoop102 hadoop-3.1.3]$ yarn application -list
2021-02-06 10:21:19,238 INFO client.RMPProxy: Connecting to ResourceManager
at hadoop103/192.168.10.103:8032
Total number of applications (application-types: [], states: [SUBMITTED,
ACCEPTED, RUNNING] and tags: []):0

Application-Id      Application-Name      Application-Type
User      Queue      State      Final-State      Progress
Tracking-URL
```

(2) 根据 Application 状态过滤: `yarn application -list -appStates` （所有状态: ALL、NEW、NEW_SAVING、SUBMITTED、ACCEPTED、RUNNING、FINISHED、FAILED、KILLED）

```
[atguigu@hadoop102 hadoop-3.1.3]$ yarn application -list -appStates
FINISHED
2021-02-06 10:22:20,029 INFO client.RMPProxy: Connecting to ResourceManager
at hadoop103/192.168.10.103:8032
Total number of applications (application-types: [], states: [FINISHED]
and tags: []):1

Application-Id      Application-Name      Application-Type
User      Queue      State      Final-State      Progress
Tracking-URL
application_1612577921195_0001      word count      MAPREDUCE
```

```
atguigu default FINISHED SUCCEEDED 100%
http://hadoop102:19888/jobhistory/job/job_1612577921195_0001
```

(3) Kill 掉 Application:

```
[atguigu@hadoop102 hadoop-3.1.3]$ yarn application -kill
application_1612577921195_0001
2021-02-06 10:23:48,530 INFO client.RMProxy: Connecting to ResourceManager
at hadoop103/192.168.10.103:8032
Application application_1612577921195_0001 has already finished
```

1.5.2 yarn logs 查看日志

(1) 查询 Application 日志: yarn logs -applicationId <ApplicationId>

```
[atguigu@hadoop102 hadoop-3.1.3]$ yarn logs -applicationId
application_1612577921195_0001
```

(2) 查询 Container 日志: yarn logs -applicationId <ApplicationId> -containerId <ContainerId>

```
[atguigu@hadoop102 hadoop-3.1.3]$ yarn logs -applicationId
application_1612577921195_0001 -containerId
container_1612577921195_0001_01_000001
```

1.5.3 yarn applicationattempt 查看尝试运行的任务

(1) 列出所有 Application 尝试的列表: yarn applicationattempt -list <ApplicationId>

```
[atguigu@hadoop102 hadoop-3.1.3]$ yarn applicationattempt -list
application_1612577921195_0001
2021-02-06 10:26:54,195 INFO client.RMProxy: Connecting to ResourceManager
at hadoop103/192.168.10.103:8032
Total number of application attempts :1
ApplicationAttempt-Id State AM-
Container-Id Tracking-URL
appattempt_1612577921195_0001_000001 FINISHED
container_1612577921195_0001_01_000001
http://hadoop103:8088/proxy/application_1612577921195_0001/
```

(2) 打印 ApplicationAttempt 状态: yarn applicationattempt -status <ApplicationAttemptId>

```
[atguigu@hadoop102 hadoop-3.1.3]$ yarn applicationattempt -status
appattempt_1612577921195_0001_000001
2021-02-06 10:27:55,896 INFO client.RMProxy: Connecting to ResourceManager
at hadoop103/192.168.10.103:8032
Application Attempt Report :
ApplicationAttempt-Id : appattempt_1612577921195_0001_000001
State : FINISHED
AMContainer : container_1612577921195_0001_01_000001
Tracking-URL :
http://hadoop103:8088/proxy/application_1612577921195_0001/
RPC Port : 34756
AM Host : hadoop104
Diagnostics :
```

1.5.4 yarn container 查看容器

(1) 列出所有 Container: yarn container -list <ApplicationAttemptId>

```
[atguigu@hadoop102 hadoop-3.1.3]$ yarn container -list
appattempt_1612577921195_0001_000001
2021-02-06 10:28:41,396 INFO client.RMProxy: Connecting to ResourceManager
at hadoop103/192.168.10.103:8032
Total number of containers :0
Container-Id Start Time Finish Time
```

| State | Host | Node | Http | Address |
|-------|------|------|------|---------|
|-------|------|------|------|---------|

(2) 打印 Container 状态: `yarn container -status <ContainerId>`

```
[atguigu@hadoop102 ~]$ yarn container -status container_1612577921195_0001_01_000001
2021-02-06 10:29:58,554 INFO client.RMProxy: Connecting to ResourceManager at hadoop103/192.168.10.103:8032
Container with id 'container_1612577921195_0001_01_000001' doesn't exist in RM or Timeline Server.
```

注：只有在任务跑的途中才能看到 container 的状态

1.5.5 yarn node 查看节点状态

列出所有节点: `yarn node -list -all`

```
[atguigu@hadoop102 ~]$ yarn node -list -all
2021-02-06 10:31:36,962 INFO client.RMProxy: Connecting to ResourceManager at hadoop103/192.168.10.103:8032
Total Nodes:3
Node-Id      Node-State  Node-Http-Address  Number-of-Running-Containers
hadoop103:38168  RUNNING    hadoop103:8042     0
hadoop102:42012  RUNNING    hadoop102:8042     0
hadoop104:39702  RUNNING    hadoop104:8042     0
```

1.5.6 yarn rmadmin 更新配置

加载队列配置: `yarn rmadmin -refreshQueues`

```
[atguigu@hadoop102 ~]$ yarn rmadmin -refreshQueues
2021-02-06 10:32:03,331 INFO client.RMProxy: Connecting to ResourceManager at hadoop103/192.168.10.103:8033
```

1.5.7 yarn queue 查看队列

打印队列信息: `yarn queue -status <QueueName>`

```
[atguigu@hadoop102 ~]$ yarn queue -status default
2021-02-06 10:32:33,403 INFO client.RMProxy: Connecting to ResourceManager at hadoop103/192.168.10.103:8032
Queue Information :
Queue Name : default
State : RUNNING
Capacity : 100.0%
Current Capacity : .0%
Maximum Capacity : 100.0%
Default Node Label expression : <DEFAULT_PARTITION>
Accessible Node Labels : *
Preemption : disabled
Intra-queue Preemption : disabled
```

1.6 Yarn 生产环境核心参数

YARN生产环境核心参数

1) ResourceManager相关

yarn.resourcemanager.scheduler.class 配置调度器, 默认容量
yarn.resourcemanager.scheduler.client.thread-count ResourceManager处理调度器请求的线程数量, 默认50

2) NodeManager相关

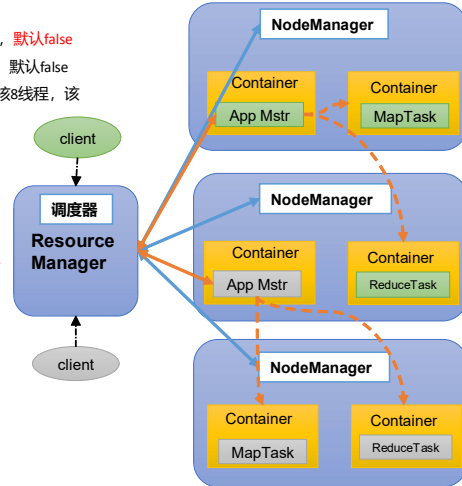
yarn.nodemanager.resource.detect-hardware-capabilities 是否让yarn自己检测硬件进行配置, 默认false
yarn.nodemanager.resource.count-logical-processors-as-cores 是否将虚拟核数当作CPU核数, 默认false
yarn.nodemanager.resource.pcores-vcores-multiplier 虚拟核数和物理核数乘数, 例如: 4核8线程, 该参数就应设为2, 默认1.0

yarn.nodemanager.resource.memory-mb NodeManager使用内存, 默认8G
yarn.nodemanager.resource.system-reserved-memory-mb NodeManager为系统保留多少内存
以上二个参数配置一个即可

yarn.nodemanager.resource.cpu-vcores NodeManager使用CPU核数, 默认8个
yarn.nodemanager.pmem-check-enabled 是否开启物理内存检查限制container, 默认打开
yarn.nodemanager.vmem-check-enabled 是否开启虚拟内存检查限制container, 默认打开
yarn.nodemanager.vmem-pmem-ratio 虚拟内存物理内存比例, 默认2.1

3) Container相关

yarn.scheduler.minimum-allocation-mb 容器最最小内存, 默认1G
yarn.scheduler.maximum-allocation-mb 容器最大内存, 默认8G
yarn.scheduler.minimum-allocation-vcores 容器最小CPU核数, 默认1个
yarn.scheduler.maximum-allocation-vcores 容器最大CPU核数, 默认4个



第 2 章 Yarn 案例实操

注：调整下列参数之前尽量拍摄 Linux 快照，否则后续的案例，还需要重写准备集群。

2.1 Yarn 生产环境核心参数配置案例

1) 需求：从 1G 数据中，统计每个单词出现次数。服务器 3 台，每台配置 4G 内存，4 核 CPU，4 线程。

2) 需求分析：

$1G / 128m = 8$ 个 MapTask；1 个 ReduceTask；1 个 mrAppMaster

平均每个节点运行 10 个 / 3 台 \approx 3 个任务 (4 3 3)

3) 修改 yarn-site.xml 配置参数如下：

```
<!-- 选择调度器, 默认容量 -->
<property>
  <description>The class to use as the resource scheduler.</description>
  <name>yarn.resourcemanager.scheduler.class</name>
  <value>org.apache.hadoop.yarn.server.resourcemanager.scheduler.capacity.CapacityScheduler</value>
</property>

<!-- ResourceManager 处理调度器请求的线程数量, 默认 50; 如果提交的任务数大于 50, 可以增加该值, 但是不能超过 3 台 * 4 线程 = 12 线程 (去除其他应用程序实际不能超过 8) -->
<property>
  <description>Number of threads to handle scheduler interface.</description>
  <name>yarn.resourcemanager.scheduler.client.thread-count</name>
  <value>8</value>
</property>
```

<!-- 是否让 yarn 自动检测硬件进行配置, 默认是 false, 如果该节点有很多其他应用程序, 建议手动配置。如果该节点没有其他应用程序, 可以采用自动 -->

```
<property>
  <description>Enable auto-detection of node capabilities such as
memory and CPU.
</description>
<name>yarn.nodemanager.resource.detect-hardware-capabilities</name>
<value>false</value>
</property>
```

<!-- 是否将虚拟核数当作 CPU 核数, 默认是 false, 采用物理 CPU 核数 -->

```
<property>
  <description>Flag to determine if logical processors(such as
hyperthreads) should be counted as cores. Only applicable on Linux
when yarn.nodemanager.resource.cpu-vcores is set to -1 and
yarn.nodemanager.resource.detect-hardware-capabilities is true.
</description>
<name>yarn.nodemanager.resource.count-logical-processors-as-
cores</name>
<value>false</value>
</property>
```

<!-- 虚拟核数和物理核数乘数, 默认是 1.0 -->

```
<property>
  <description>Multiplier to determine how to convert physical cores to
vcores. This value is used if yarn.nodemanager.resource.cpu-vcores
is set to -1(which implies auto-calculate vcores) and
yarn.nodemanager.resource.detect-hardware-capabilities is set to true.
The number of vcores will be calculated as number of CPUs * multiplier.
</description>
<name>yarn.nodemanager.resource.pcores-vcores-multiplier</name>
<value>1.0</value>
</property>
```

<!-- NodeManager 使用内存数, 默认 8G, 修改为 4G 内存 -->

```
<property>
  <description>Amount of physical memory, in MB, that can be allocated
for containers. If set to -1 and
yarn.nodemanager.resource.detect-hardware-capabilities is true, it is
automatically calculated(in case of Windows and Linux).
In other cases, the default is 8192MB.
</description>
<name>yarn.nodemanager.resource.memory-mb</name>
<value>4096</value>
</property>
```

<!-- nodemanager 的 CPU 核数, 不按照硬件环境自动设定时默认是 8 个, 修改为 4 个 -->

```
<property>
  <description>Number of vcores that can be allocated
for containers. This is used by the RM scheduler when allocating
resources for containers. This is not used to limit the number of
CPUs used by YARN containers. If it is set to -1 and
yarn.nodemanager.resource.detect-hardware-capabilities is true, it is
automatically determined from the hardware in case of Windows and Linux.
In other cases, number of vcores is 8 by default.</description>
<name>yarn.nodemanager.resource.cpu-vcores</name>
<value>4</value>
</property>
```

<!-- 容器最小内存, 默认 1G -->

```
<property>
  <description>The minimum allocation for every container request at the
```

```
RM in MBs. Memory requests lower than this will be set to the value of
this property. Additionally, a node manager that is configured to have
less memory than this value will be shut down by the resource manager.
</description>
<name>yarn.scheduler.minimum-allocation-mb</name>
<value>1024</value>
</property>

<!-- 容器最大内存，默认 8G，修改为 2G -->
<property>
  <description>The maximum allocation for every container request at the
RM in MBs. Memory requests higher than this will throw an
InvalidResourceRequestException.
</description>
<name>yarn.scheduler.maximum-allocation-mb</name>
<value>2048</value>
</property>

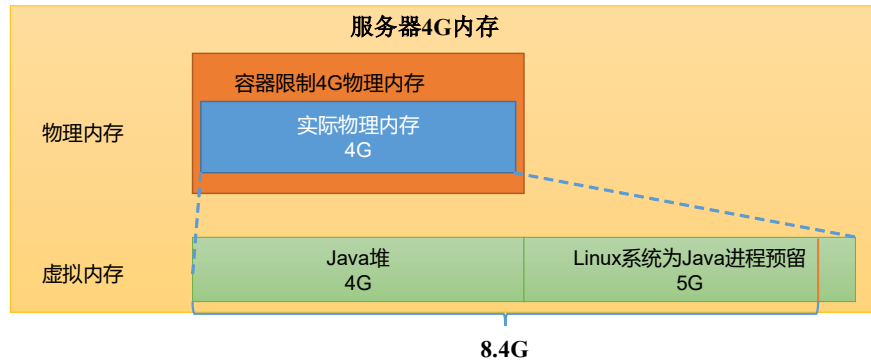
<!-- 容器最小 CPU 核数，默认 1 个 -->
<property>
  <description>The minimum allocation for every container request at the
RM in terms of virtual CPU cores. Requests lower than this will be set to
the value of this property. Additionally, a node manager that is configured
to have fewer virtual cores than this value will be shut down by the
resource manager.
</description>
<name>yarn.scheduler.minimum-allocation-vcores</name>
<value>1</value>
</property>

<!-- 容器最大 CPU 核数，默认 4 个，修改为 2 个 -->
<property>
  <description>The maximum allocation for every container request at the
RM in terms of virtual CPU cores. Requests higher than this will throw an
InvalidResourceRequestException.</description>
<name>yarn.scheduler.maximum-allocation-vcores</name>
<value>2</value>
</property>

<!-- 虚拟内存检查，默认打开，修改为关闭 -->
<property>
  <description>Whether virtual memory limits will be enforced for
containers.</description>
<name>yarn.nodemanager.vmem-check-enabled</name>
<value>false</value>
</property>

<!-- 虚拟内存和物理内存设置比例，默认 2.1 -->
<property>
  <description>Ratio between virtual memory to physical memory when
setting memory limits for containers. Container allocations are
expressed in terms of physical memory, and virtual memory usage is
allowed to exceed this allocation by this ratio.
</description>
<name>yarn.nodemanager.vmem-pmem-ratio</name>
<value>2.1</value>
</property>
```


关闭虚拟内存检查原因



```
<property>
  <description>Ratio between virtual memory to physical memory when setting memory limits for containers. Container
  allocations are expressed in terms of physical memory, and virtual memory usage is allowed to exceed this allocation by this ratio.
</description>
  <name>yarn.nodemanager.vmem-pmem-ratio</name>
  <value>2.1</value>
</property>
```

让天下没有难学的技术

4) 分发配置。

注意：如果集群的硬件资源不一致，要每个 NodeManager 单独配置

5) 重启集群

```
[atguigu@hadoop102 hadoop-3.1.3]$ sbin/stop-yarn.sh
[atguigu@hadoop103 hadoop-3.1.3]$ sbin/start-yarn.sh
```

6) 执行 WordCount 程序

```
[atguigu@hadoop102 hadoop-3.1.3]$ hadoop jar
share/hadoop/mapreduce/hadoop-mapreduce-examples-3.1.3.jar wordcount
/input /output
```

7) 观察 Yarn 任务执行页面

<http://hadoop103:8088/cluster/apps>

2.2 容量调度器多队列提交案例

1) 在生产环境怎么创建队列？

- (1) 调度器默认就 1 个 default 队列，不能满足生产要求。
- (2) 按照框架：hive/spark/flink 每个框架的任务放入指定的队列（企业用的不是特别多）
- (3) 按照业务模块：登录注册、购物车、下单、业务部门 1、业务部门 2

2) 创建多队列的好处？

- (1) 因为担心员工不小心，写递归死循环代码，把所有资源全部耗尽。
 - (2) 实现任务的降级使用，特殊时期保证重要的任务队列资源充足。11.11 6.18
- 业务部门 1（重要）=》业务部门 2（比较重要）=》下单（一般）=》购物车（一般）=》登录注册（次要）

更多 Java - 大数据 - 前端 - python 人工智能资料下载，可百度访问：尚硅谷官网

2.2.1 需求

需求 1: default 队列占总内存的 40%，最大资源容量占总资源 60%，hive 队列占总内存的 60%，最大资源容量占总资源 80%。

需求 2: 配置队列优先级

2.2.2 配置多队列的容量调度器

1) 在 capacity-scheduler.xml 中配置如下:

(1) 修改如下配置

```
<!-- 指定多队列，增加 hive 队列 -->
<property>
  <name>yarn.scheduler.capacity.root.queues</name>
  <value>default,hive</value>
  <description>
    The queues at the this level (root is the root queue).
  </description>
</property>

<!-- 降低 default 队列资源额定容量为 40%，默认 100% -->
<property>
  <name>yarn.scheduler.capacity.root.default.capacity</name>
  <value>40</value>
</property>

<!-- 降低 default 队列资源最大容量为 60%，默认 100% -->
<property>
  <name>yarn.scheduler.capacity.root.default.maximum-capacity</name>
  <value>60</value>
</property>
```

(2) 为新加队列添加必要属性:

```
<!-- 指定 hive 队列的资源额定容量 -->
<property>
  <name>yarn.scheduler.capacity.root.hive.capacity</name>
  <value>60</value>
</property>

<!-- 用户最多可以使用队列多少资源，1 表示 -->
<property>
  <name>yarn.scheduler.capacity.root.hive.user-limit-factor</name>
  <value>1</value>
</property>

<!-- 指定 hive 队列的资源最大容量 -->
<property>
  <name>yarn.scheduler.capacity.root.hive.maximum-capacity</name>
  <value>80</value>
</property>

<!-- 启动 hive 队列 -->
<property>
  <name>yarn.scheduler.capacity.root.hive.state</name>
  <value>RUNNING</value>
</property>
```

```

<!-- 哪些用户有权向队列提交作业 -->
<property>
  <name>yarn.scheduler.capacity.root.hive.acl_submit_applications</name>
  <value>*</value>
</property>

<!-- 哪些用户有权操作队列，管理员权限（查看/杀死） -->
<property>
  <name>yarn.scheduler.capacity.root.hive.acl_administer_queue</name>
  <value>*</value>
</property>

<!-- 哪些用户有权配置提交任务优先级 -->
<property>
  <name>yarn.scheduler.capacity.root.hive.acl_application_max_priority</name>
  <value>*</value>
</property>

<!-- 任务的超时时间设置: yarn application -appId appId -updateLifetime Timeout
参考资料: https://blog.cloudera.com/enforcing-application-lifetime-slas-yarn/ -->

<!-- 如果 application 指定了超时时间，则提交到该队列的 application 能够指定的最大超时
时间不能超过该值。
-->
<property>
  <name>yarn.scheduler.capacity.root.hive.maximum-application-
lifetime</name>
  <value>-1</value>
</property>

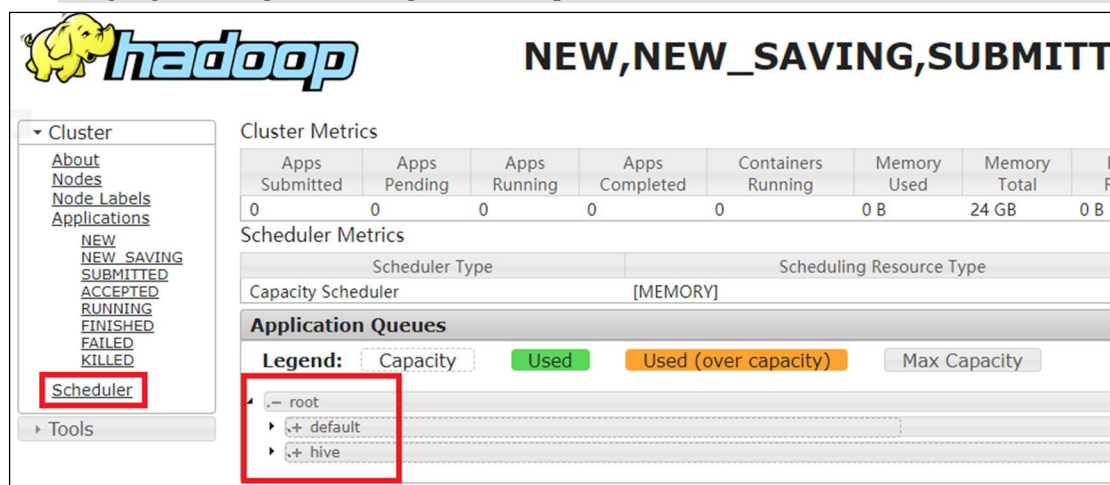
<!-- 如果 application 没指定超时时间，则用 default-application-lifetime 作为默认
值 -->
<property>
  <name>yarn.scheduler.capacity.root.hive.default-application-
lifetime</name>
  <value>-1</value>
</property>

```

2) 分发配置文件

3) 重启 Yarn 或者执行 `yarn rmadmin -refreshQueues` 刷新队列，就可以看到两条队列：

```
[atguigu@hadoop102 hadoop-3.1.3]$ yarn rmadmin -refreshQueues
```



The screenshot shows the Hadoop Yarn Web UI. On the left, the 'Cluster' sidebar has 'Scheduler' highlighted. The main content area shows 'Cluster Metrics' with a table of application states (Submitted, Pending, Running, Completed) and resource usage (Containers, Memory). Below this, 'Scheduler Metrics' shows the 'Capacity Scheduler' with '[MEMORY]' resource type. The 'Application Queues' section includes a legend for Capacity, Used, and Used (over capacity) states, and a list of queues: 'root', 'default', and 'hive'. The 'default' and 'hive' queues are highlighted with a red box.

2.2.3 向 Hive 队列提交任务

1) hadoop jar 的方式

```
[atguigu@hadoop102      hadoop-3.1.3]$      hadoop      jar
share/hadoop/mapreduce/hadoop-mapreduce-examples-3.1.3.jar wordcount -D
mapreduce.job.queueName=hive /input /output
```

注: -D 表示运行时改变参数值

2) 打 jar 包的方式

默认的任务提交都是提交到 default 队列的。如果希望向其他队列提交任务，需要在 Driver 中声明：

```
public class WcDriver {

    public static void main(String[] args) throws IOException,
        ClassNotFoundException, InterruptedException {

        Configuration conf = new Configuration();

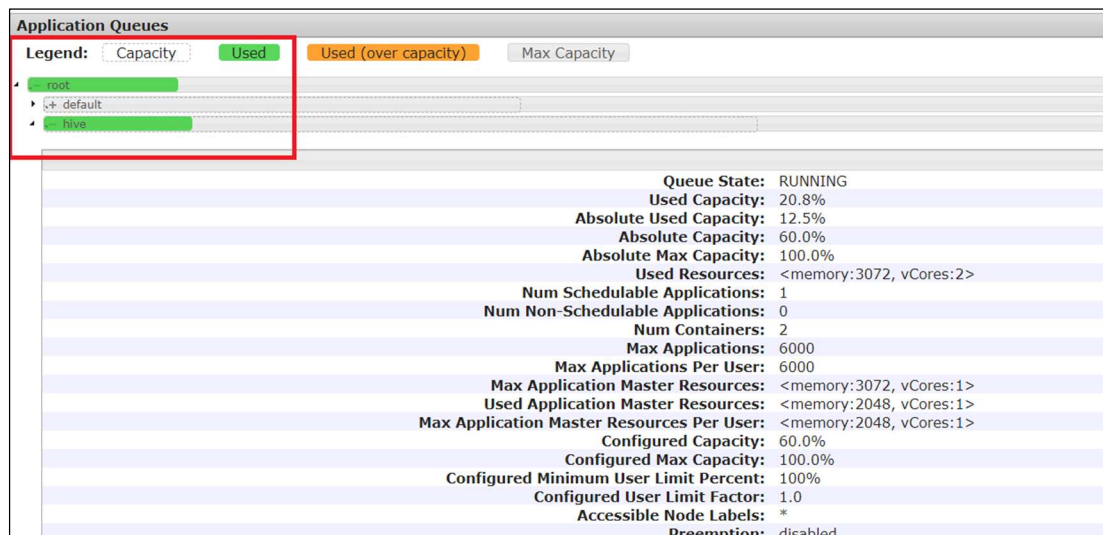
        conf.set("mapreduce.job.queueName", "hive");

        //1. 获取一个 Job 实例
        Job job = Job.getInstance(conf);

        . . . . .

        //6. 提交 Job
        boolean b = job.waitForCompletion(true);
        System.exit(b ? 0 : 1);
    }
}
```

这样，这个任务在集群提交时，就会提交到 hive 队列：



2.2.4 任务优先级

容量调度器，支持任务优先级的配置，在资源紧张时，优先级高的任务将优先获取资源。默认情况，Yarn 将所有任务的优先级限制为 0，若想使用任务的优先级功能，须开放该限制。

更多 Java - 大数据 - 前端 - python 人工智能资料下载，可百度访问：尚硅谷官网

1) 修改 yarn-site.xml 文件，增加以下参数

```
<property>
  <name>yarn.cluster.max-application-priority</name>
  <value>5</value>
</property>
```

2) 分发配置，并重启 Yarn

```
[atguigu@hadoop102 hadoop]$ xsync yarn-site.xml
[atguigu@hadoop103 hadoop-3.1.3]$ sbin/stop-yarn.sh
[atguigu@hadoop103 hadoop-3.1.3]$ sbin/start-yarn.sh
```

3) 模拟资源紧张环境，可连续提交以下任务，直到新提交的任务申请不到资源为止。

```
[atguigu@hadoop102 hadoop-3.1.3]$ hadoop jar /opt/module/hadoop-
3.1.3/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.1.3.jar pi 5
2000000
```

| Application Priority | StartTime | LaunchTime | FinishTime | State | FinalStatus | Running Containers | Allocated CPU Vcores | Allocated Memory MB | Reserved CPU Vcores | Reserved Memory MB | % of Queue | % of Cluster | Progress | Tracking UI | Blacklisted Nodes |
|----------------------|--------------------------------|--------------------------------|------------|----------|-------------|--------------------|----------------------|---------------------|---------------------|--------------------|------------|--------------|-------------|-------------------|-------------------|
| 0 | Wed Jan 20 17:22:46 +0800 2021 | N/A | N/A | ACCEPTED | UNDEFINED | 0 | 0 | 0 | 0 | 0 | 0.0 | 0.0 | <div></div> | ApplicationMaster | 0 |
| 已无空余资源 | | | | | | | | | | | | | | | |
| 0 | Wed Jan 20 17:21:22 +0800 2021 | Wed Jan 20 17:21:23 +0800 2021 | N/A | RUNNING | UNDEFINED | 1 | 1 | 1536 | 0 | 0 | 25.0 | 12.5 | <div></div> | ApplicationMaster | 0 |
| 0 | Wed Jan 20 17:20:32 +0800 2021 | Wed Jan 20 17:20:32 +0800 2021 | N/A | RUNNING | UNDEFINED | 5 | 5 | 5632 | 0 | 0 | 91.7 | 45.8 | <div></div> | ApplicationMaster | 0 |

4) 再次重新提交优先级高的任务

```
[atguigu@hadoop102 hadoop-3.1.3]$ hadoop jar /opt/module/hadoop-
3.1.3/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.1.3.jar pi -D
mapreduce.job.priority=5 5 2000000
```

| Application Priority | StartTime | LaunchTime | FinishTime | State | FinalStatus | Running Containers | Allocated CPU Vcores | Allocated Memory MB | Reserved CPU Vcores | Reserved Memory MB | % of Queue | % of Cluster | Progress | Tracking UI | Blacklisted Nodes |
|----------------------|--------------------------------|--------------------------------|------------|----------|-------------|--------------------|----------------------|---------------------|---------------------|--------------------|------------|--------------|-------------|-------------------|-------------------|
| 5 | Wed Jan 20 17:23:59 +0800 2021 | Wed Jan 20 17:23:59 +0800 2021 | N/A | RUNNING | UNDEFINED | 3 | 3 | 3584 | 0 | 0 | 58.3 | 29.2 | <div></div> | ApplicationMaster | 0 |
| 高优先级任务，优先获取资源 | | | | | | | | | | | | | | | |
| 0 | Wed Jan 20 17:22:46 +0800 2021 | N/A | N/A | ACCEPTED | UNDEFINED | 0 | 0 | 0 | 0 | 0 | 0.0 | 0.0 | <div></div> | ApplicationMaster | 0 |
| 0 | Wed Jan 20 17:21:22 +0800 2021 | Wed Jan 20 17:21:23 +0800 2021 | N/A | RUNNING | UNDEFINED | 1 | 1 | 1536 | 0 | 0 | 25.0 | 12.5 | <div></div> | ApplicationMaster | 0 |
| 0 | Wed Jan 20 17:20:32 +0800 2021 | Wed Jan 20 17:20:32 +0800 2021 | N/A | RUNNING | UNDEFINED | 1 | 1 | 1536 | 0 | 0 | 25.0 | 12.5 | <div></div> | ApplicationMaster | 0 |

5) 也可以通过以下命令修改正在执行的任务的优先级。

yarn application -appID <ApplicationID> -updatePriority 优先级

```
[atguigu@hadoop102 hadoop-3.1.3]$ yarn application -appID
application_1611133087930_0009 -updatePriority 5
```

2.3 公平调度器案例

2.3.1 需求

创建两个队列，分别是 test 和 atguigu（以用户所属组命名）。期望实现以下效果：若用户提交任务时指定队列，则任务提交到指定队列运行；若未指定队列，test 用户提交的任务到 root.group.test 队列运行，atguigu 提交的任务到 root.group.atguigu 队列运行（注：group 为

用户所属组)。

公平调度器的配置涉及到两个文件，一个是 yarn-site.xml，另一个是公平调度器队列分配文件 fair-scheduler.xml（文件名可自定义）。

(1) 配置文件参考资料：

<https://hadoop.apache.org/docs/r3.1.3/hadoop-yarn/hadoop-yarn-site/FairScheduler.html>

(2) 任务队列放置规则参考资料：

<https://blog.cloudera.com/untangling-apache-hadoop-yarn-part-4-fair-scheduler-queue-basics/>

2.3.2 配置多队列的公平调度器

1) 修改 yarn-site.xml 文件，加入以下参数

```
<property>
  <name>yarn.resourcemanager.scheduler.class</name>

  <value>org.apache.hadoop.yarn.server.resourcemanager.scheduler.fair.FairScheduler</value>
  <description>配置使用公平调度器</description>
</property>

<property>
  <name>yarn.scheduler.fair.allocation.file</name>
  <value>/opt/module/hadoop-3.1.3/etc/hadoop/fair-scheduler.xml</value>
  <description>指明公平调度器队列分配配置文件</description>
</property>

<property>
  <name>yarn.scheduler.fair.preemption</name>
  <value>false</value>
  <description>禁止队列间资源抢占</description>
</property>
```

2) 配置 fair-scheduler.xml

```
<?xml version="1.0"?>
<allocations>
  <!-- 单个队列中 Application Master 占用资源的最大比例,取值 0-1 ，企业一般配置 0.1 -->
  <queueMaxAMShareDefault>0.5</queueMaxAMShareDefault>
  <!-- 单个队列最大资源的默认值 test atguigu default -->
  <queueMaxResourcesDefault>4096mb,4vcores</queueMaxResourcesDefault>

  <!-- 增加一个队列 test -->
  <queue name="test">
    <!-- 队列最小资源 -->
    <minResources>2048mb,2vcores</minResources>
    <!-- 队列最大资源 -->
    <maxResources>4096mb,4vcores</maxResources>
    <!-- 队列中最多同时运行的应用数，默认 50，根据线程数配置 -->
    <maxRunningApps>4</maxRunningApps>
    <!-- 队列中 Application Master 占用资源的最大比例 -->
    <maxAMShare>0.5</maxAMShare>
    <!-- 该队列资源权重，默认值为 1.0 -->
```

```

<weight>1.0</weight>
<!-- 队列内部的资源分配策略 -->
<schedulingPolicy>fair</schedulingPolicy>
</queue>
<!-- 增加一个队列 atguigu -->
<queue name="atguigu" type="parent">
  <!-- 队列最小资源 -->
  <minResources>2048mb,2vcores</minResources>
  <!-- 队列最大资源 -->
  <maxResources>4096mb,4vcores</maxResources>
  <!-- 队列中最多同时运行的应用数, 默认 50, 根据线程数配置 -->
  <maxRunningApps>4</maxRunningApps>
  <!-- 队列中 Application Master 占用资源的最大比例 -->
  <maxAMShare>0.5</maxAMShare>
  <!-- 该队列资源权重, 默认值为 1.0 -->
  <weight>1.0</weight>
  <!-- 队列内部的资源分配策略 -->
  <schedulingPolicy>fair</schedulingPolicy>
</queue>

<!-- 任务队列分配策略, 可配置多层规则, 从第一个规则开始匹配, 直到匹配成功 -->
<queuePlacementPolicy>
  <!-- 提交任务时指定队列, 如未指定提交队列, 则继续匹配下一个规则; false 表示: 如果指定队列不存在, 不允许自动创建 -->
  <rule name="specified" create="false"/>
  <!-- 提交到 root.group.username 队列, 若 root.group 不存在, 不允许自动创建; 若 root.group.user 不存在, 允许自动创建 -->
  <rule name="nestedUserQueue" create="true">
    <rule name="primaryGroup" create="false"/>
  </rule>
  <!-- 最后一个规则必须为 reject 或者 default. Reject 表示拒绝创建提交失败, default 表示把任务提交到 default 队列 -->
  <rule name="reject" />
</queuePlacementPolicy>
</allocations>

```

3) 分发配置并重启 Yarn

```

[atguigu@hadoop102 hadoop]$ xsync yarn-site.xml
[atguigu@hadoop102 hadoop]$ xsync fair-scheduler.xml

[atguigu@hadoop103 hadoop-3.1.3]$ sbin/stop-yarn.sh
[atguigu@hadoop103 hadoop-3.1.3]$ sbin/start-yarn.sh

```

2.3.3 测试提交任务

1) 提交任务时指定队列, 按照配置规则, 任务会到指定的 root.test 队列

```

[atguigu@hadoop102 hadoop-3.1.3]$ hadoop jar /opt/module/hadoop-3.1.3/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.1.3.jar pi -Dmapreduce.job.queueName=root.test 1 1

```

| | | | | |
|--------------------------------|---------|-----------------|-----------|-----------|
| application_1611023120675_0002 | atguigu | QuasiMonteCarlo | MAPREDUCE | root.test |
|--------------------------------|---------|-----------------|-----------|-----------|

2) 提交任务时不指定队列, 按照配置规则, 任务会到 root.atguigu.atguigu 队列

```

[atguigu@hadoop102 hadoop-3.1.3]$ hadoop jar /opt/module/hadoop-3.1.3/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.1.3.jar pi 1 1

```

| | | | | |
|--------------------------------|---------|-----------------|-----------|----------------------|
| application_1611023120675_0003 | atguigu | QuasiMonteCarlo | MAPREDUCE | root.atguigu.atguigu |
|--------------------------------|---------|-----------------|-----------|----------------------|

2.4 Yarn 的 Tool 接口案例

0) 回顾:

```
[atguigu@hadoop102 hadoop-3.1.3]$ hadoop jar wc.jar  
com.atguigu.mapreduce.wordcount2.WordCountDriver /input  
/output1
```

期望可以动态传参, 结果报错, 误认为是第一个输入参数。

```
[atguigu@hadoop102 hadoop-3.1.3]$ hadoop jar wc.jar  
com.atguigu.mapreduce.wordcount2.WordCountDriver -  
Dmapreduce.job.queueName=root.test /input /output1
```

1) 需求: 自己写的程序也可以动态修改参数。编写 Yarn 的 Tool 接口。

2) 具体步骤:

(1) 新建 Maven 项目 YarnDemo, pom 如下:

```
<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0"  
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0  
http://maven.apache.org/xsd/maven-4.0.0.xsd">  
    <modelVersion>4.0.0</modelVersion>  
  
    <groupId>com.atguigu.hadoop</groupId>  
    <artifactId>yarn_tool_test</artifactId>  
    <version>1.0-SNAPSHOT</version>  
  
    <dependencies>  
        <dependency>  
            <groupId>org.apache.hadoop</groupId>  
            <artifactId>hadoop-client</artifactId>  
            <version>3.1.3</version>  
        </dependency>  
    </dependencies>  
</project>
```

(2) 新建 com.atguigu.yarn 报名

(3) 创建类 WordCount 并实现 Tool 接口:

```
package com.atguigu.yarn;  
  
import org.apache.hadoop.conf.Configuration;  
import org.apache.hadoop.fs.Path;  
import org.apache.hadoop.io.IntWritable;  
import org.apache.hadoop.io.LongWritable;  
import org.apache.hadoop.io.Text;  
import org.apache.hadoop.mapreduce.Job;  
import org.apache.hadoop.mapreduce.Mapper;  
import org.apache.hadoop.mapreduce.Reducer;  
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;  
import  
org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  
import org.apache.hadoop.util.Tool;  
  
import java.io.IOException;
```

```
public class WordCount implements Tool {

    private Configuration conf;

    @Override
    public int run(String[] args) throws Exception {

        Job job = Job.getInstance(conf);

        job.setJarByClass(WordCountDriver.class);

        job.setMapperClass(WordCountMapper.class);
        job.setReducerClass(WordCountReducer.class);

        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(IntWritable.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        FileInputFormat.setInputPaths(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        return job.waitForCompletion(true) ? 0 : 1;
    }

    @Override
    public void setConf(Configuration conf) {
        this.conf = conf;
    }

    @Override
    public Configuration getConf() {
        return conf;
    }

    public static class WordCountMapper extends
Mapper<LongWritable, Text, Text, IntWritable> {

        private Text outK = new Text();
        private IntWritable outV = new IntWritable(1);

        @Override
        protected void map(LongWritable key, Text value,
Context context) throws IOException, InterruptedException {

            String line = value.toString();
            String[] words = line.split(" ");

            for (String word : words) {
                outK.set(word);

                context.write(outK, outV);
            }
        }
    }

    public static class WordCountReducer extends Reducer<Text,
```

```
IntWritable, Text, IntWritable> {
    private IntWritable outV = new IntWritable();

    @Override
    protected void reduce(Text key, Iterable<IntWritable>
values, Context context) throws IOException,
InterruptedException {

        int sum = 0;

        for (IntWritable value : values) {
            sum += value.get();
        }
        outV.set(sum);

        context.write(key, outV);
    }
}
```

(4) 新建 WordCountDriver

```
package com.atguigu.yarn;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
import java.util.Arrays;

public class WordCountDriver {

    private static Tool tool;

    public static void main(String[] args) throws Exception {
        // 1. 创建配置文件
        Configuration conf = new Configuration();

        // 2. 判断是否有 tool 接口
        switch (args[0]){
            case "wordcount":
                tool = new WordCount();
                break;
            default:
                throw new RuntimeException(" No such tool: "+
args[0] );
        }
        // 3. 用 Tool 执行程序
        // Arrays.copyOfRange 将老数组的元素放到新数组里面
        int run = ToolRunner.run(conf, tool,
Arrays.copyOfRange(args, 1, args.length));

        System.exit(run);
    }
}
```

3) 在 HDFS 上准备输入文件, 假设为 /input 目录, 向集群提交该 Jar 包

```
[atguigu@hadoop102 hadoop-3.1.3]$ yarn jar YarnDemo.jar
com.atguigu.yarn.WordCountDriver wordcount /input /output
```

注意此时提交的 3 个参数，第一个用于生成特定的 Tool，第二个和第三个为输入输出目录。此时如果我们希望加入设置参数，可以在 wordcount 后面添加参数，例如：

```
[atguigu@hadoop102 hadoop-3.1.3]$ yarn jar YarnDemo.jar  
com.atguigu.yarn.WordCountDriver wordcount -  
Dmapreduce.job.queueName=root.test /input /output1
```

- 4) 注：以上操作全部做完过后，快照回去或者手动将配置文件修改成之前的状态，因为本身资源就不够，分成了这么多，不方便以后测试。