day03 YARN

一、内容回顾

```
- Zookeeper的简介
- Zookeeper的API
- Zookeeper的命令
- Zookeeper的数据模型
- Zookeeper的安装配置
- Zookeeper的节点类型
- Zookeeper的选举制度
- Zookeeper的监听原理
- Zookeeper的写数据流程
- HA集群搭建
```

二、教学目标

三、教学导读

我们已经搭建了一个HDFS的分布式文件系统，在将数据存储到HDFS的时候，会将数据进行切块，分布在不同的数据节点进行存储。那么存储的问题就解决了，然后计算呢？如何计算这些节点上存储的数据文件呢？

还记得HDFS是怎么来的吗？谷歌发表了一篇论文《GFS》，Nutch团队对这个论文使用Java进行了实现，命名为NDFS，也就是后来的HDFS。谷歌还发表过另外一篇论文《MapReduce》，介绍的就是如何解决分布式文件系统上存储的数据进行计算的问题。也就是一个分布式计算的框架。虽然我们现在还没学到MapReduce，不过这个分布式的计算思想还是可以先了

解一下的。

如果我们需要计算的文件分布在不同的节点上，在进行数据计算的时候有两种方式:

- 将数据移动到一个节点上，在这个节点上进行数据的计算。
- 将计算程序分发到每一个数据节点，在每一个节点计算自己的数据。

在这里使用的是第二种计算方式，也就是将计算程序分发到不同的数据节点进行计算。还记得我们在搭建完HDFS的集群之后演示的Hadoop官方案例吗？这其中就是一个分布式的计算程序。那么问题来了: 我们如何为每一个计算任务分配计算资源(内存、CPU)，如何协调每一个节点上的计算任务？如何监控每一个节点上的计算任务等等。而这些，就是我们今天要学习的 YARN。

四、教学内容

4.1. YARN的简介

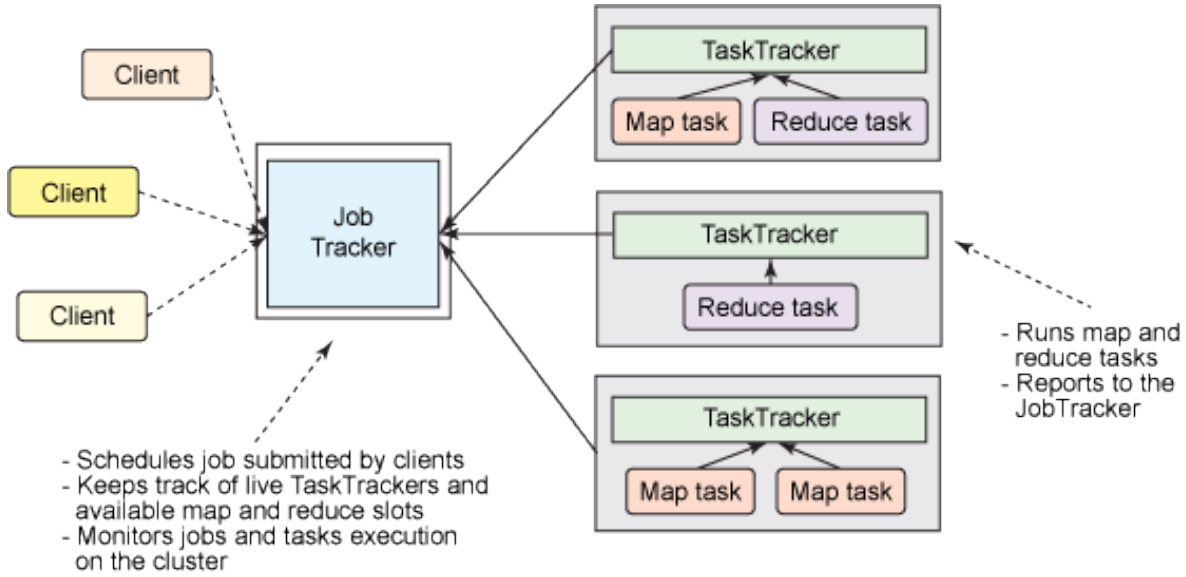
4.1.1. MapReduce 1.x

第一代Hadoop，由分布式存储系统HDFS和分布式计算框架MapReduce组成，其中，HDFS由一个NameNode和多个DataNode组成，MapReduce由一个JobTracker和多个TaskTracker组成。对应Hadoop版本为Hadoop 1.x和0.21.X，0.22.x。

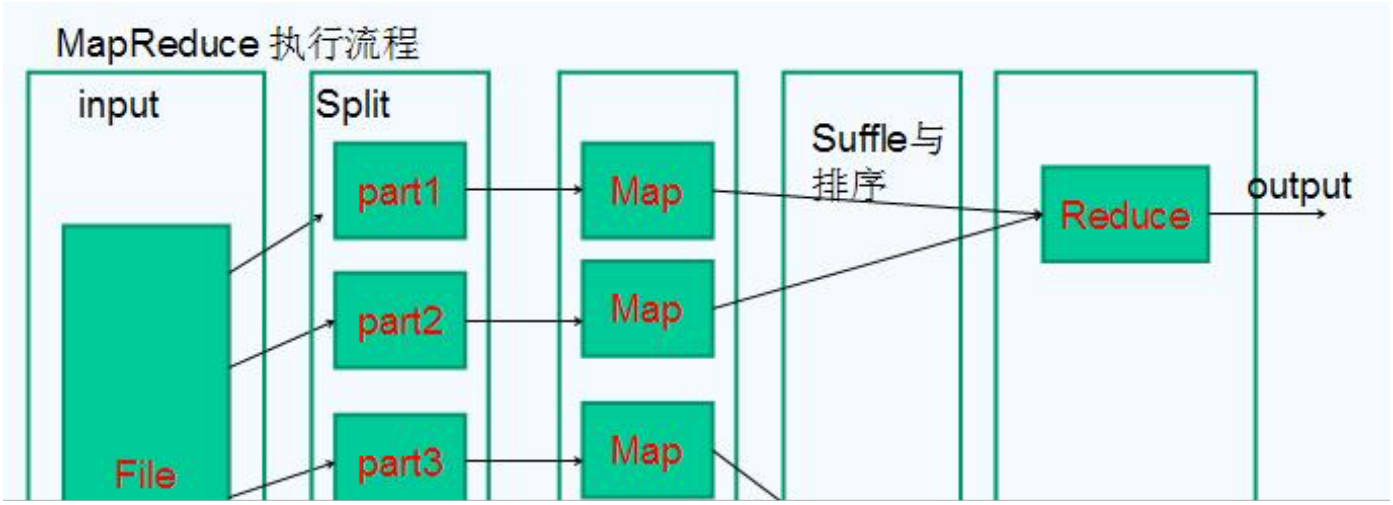
4.1.1.1. MapReduce 1.x的角色

- Client: 作业提交发起者
- JobTracker: 初始化作业，分配作业，与TaskTracker通信，协调整个作业。

- TaskTracker: 保持JobTracker通信，在分配的数据片段上执行MapReduce任务。



4.1.1.2. MapReduce 1.x执行流程



1. 提交作业

```
编写MapReduce程序代码,创建job对象，并进行配置，比如输入和输出路径，压缩格式等，然后通过JobClinet来提交作业。
```

2. 初始化作业

```
客户端提交完成后，JobTracker会将作业加入队列，然后进行调度，默认的调度方法是FIFO调试方式。
```

3. 分配任务

```
TaskTracker和JobTracker之间的通信与任务的分配是通过心跳机制完成的。

TaskTracker会主动向JobTracker询问是否有作业要做，如果自己可以做，那么就会申请到作业任务，这个任务可以是MapTask也可能是ReduceTask。
```

4. 执行任务

```
申请到任务后，TaskTracker会做如下事情：

1. 拷贝代码到本地
2. 拷贝任务的信息到本地
3. 启动JVM运行任务
```

5. 状态与任务的更新

```
任务在运行过程中，首先会将自己的状态汇报给TaskTracker，然后由TaskTracker汇总告之JobTracker。任务进度是通过计数器来实现的。
```

6. 作业的完成

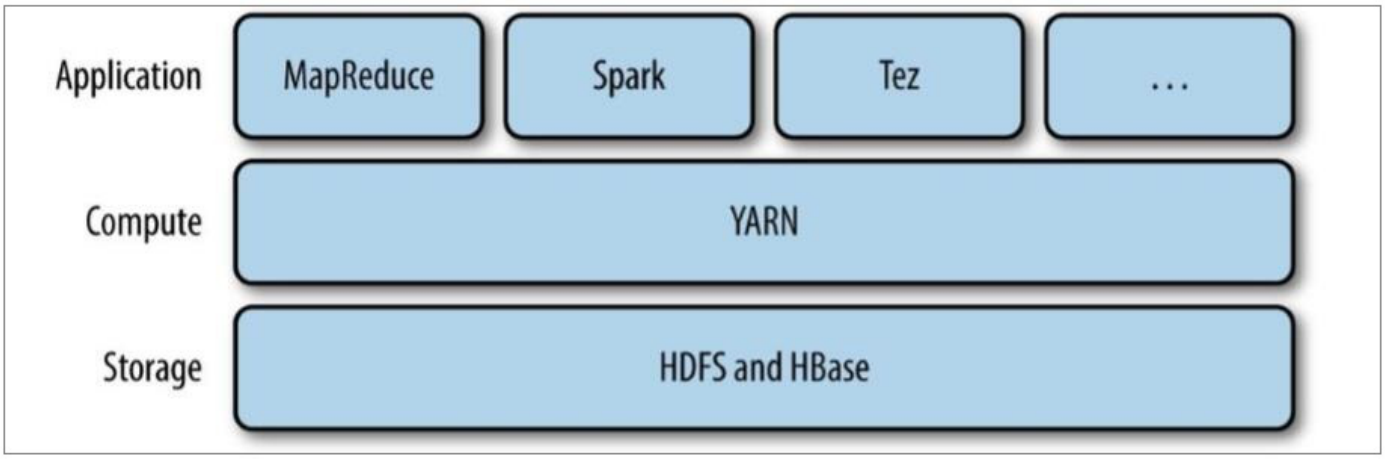
```
JobTracker是在接受到最后一个任务运行完成后，才会将任务标记为成功。此时会做删除中间结果等善后处理工作。
```

4.1.2. YARN的介绍

为克服Hadoop 1.0中HDFS和MapReduce存在的各种问题而提出的，针对Hadoop 1.0中的MapReduce在扩展性和多框架支持方面的不足，提出了全新的资源管理框架YARN.

Apache YARN（Yet another Resource Negotiator的缩写）是Hadoop集群的资源管理系统，负责为计算程序提供服务器计算资源，相当于一个分布式的操作系统平台，而MapReduce等计算程序则相当于运行于操作系统之上的应用程序。

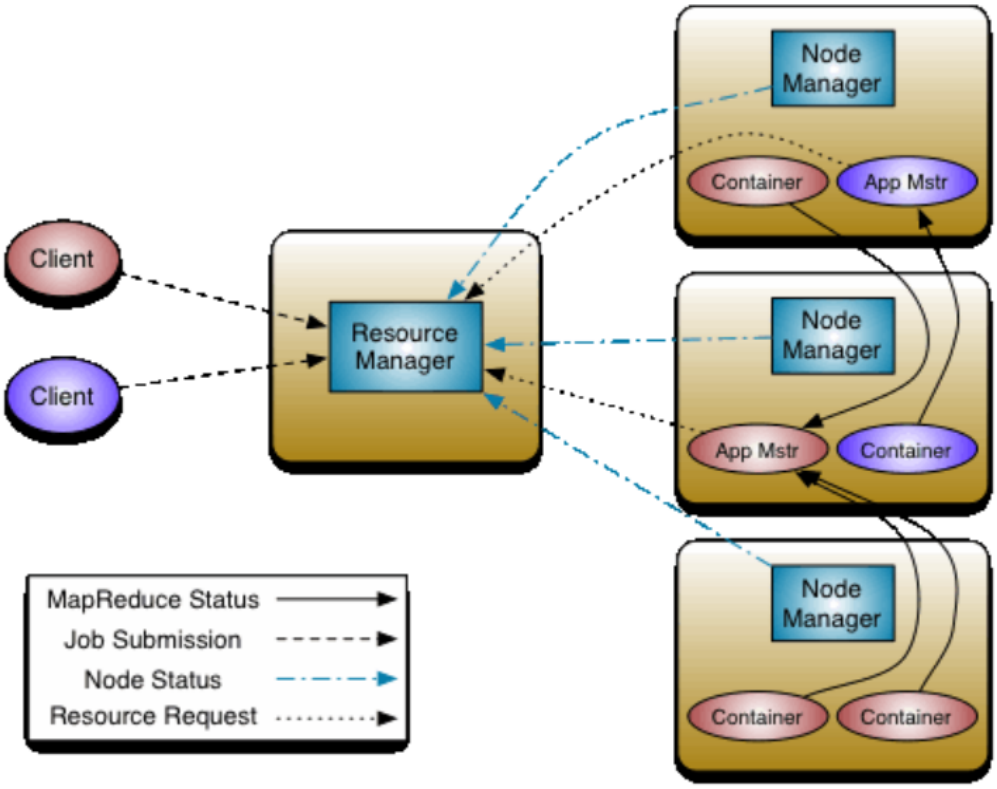
YARN被引入Hadoop2,最初是为了改善MapReduce的实现，但是因为具有足够的通用性，同样可以支持其他的分布式计算模式，比如Spark，Tez等计算框架。



> 注意：还有一层应用是运行在MapReduce，Spark或者Tez之上的处理框架，如Pig,Hive和Crunch等。

4.2. YARN的设计思想

YARN的基本思想是将资源管理和作业调度/监视功能划分为单独的守护进程。其思想是拥有一个全局ResourceManager (RM)，以及每个应用程序拥有一个ApplicationMaster (AM)。应用程序可以是单个作业，也可以是一组作业。



一个ResourceManager和多个NodeManager构成了YARN资源管理框架。他们是YARN启动后长期运行的守护进程，来提供核心服务。

- ResourceManager

```
1 是在系统中的所有应用程序之间仲裁资源的最终权威，即管理整个集群上的所有资源分配,内部含有一个Scheduler(资源调度器)
```

- NodeManager

```
1 是每台机器的资源管理器，也就是单个节点的管理者，负责启动和监视容器(container)资源使用情况，并向ResourceManager及其Scheduler报告使用情况
```

- container

```
1 即集群上的可使用资源，包含cpu、内存、磁盘、网络等
```

- ApplicationMaster（简称AM）

  ```
  实际上是框架的特定的库，每启动一个应用程序，都会启动一个AM，它的任务是与ResourceManager协商资源，并与NodeManager一起执行和监视任务
  ```

YARN的角色	MapReduce 1.x的角色
ResourceManager、Application Master、Timeline Server	JobTracker
NodeManager	TaskTracker
Container	Slot

4.3. YARN的配置

YARN属于Hadoop的核心组件，不需要单独安装，只需要修改一些配置文件即可。

4.3.1. mapred-site.xml

```
<configuration>
    <!-- 指定MapReduce作业执行时，使用YARN进行资源调度 -->
    <property>
        <name>mapreduce.framework.name</name>
        <value>yarn</value>
    </property>
</configuration>
```

4.3.2. yarn-site.xml

```
<configuration>
    <!-- 设置ResourceManager -->
    <property>
        <name>yarn.resourcemanager.hostname</name>
        <value>qianfeng01</value>
    </property>

    <!--配置yarn的shuffle服务-->
    <property>
        <name>yarn.nodemanager.aux-services</name>
        <value>mapreduce_shuffle</value>
    </property>
</configuration>
```

4.3.3. hadoop-env.sh

```
# 添加如下:
export YARN_RESOURCEMANAGER_USER=root
export YARN_NODEMANAGER_USER=root
```

4.3.4. 分发到其他节点

```
[root@qianfeng01 ~]# cd $HADOOP_HOME/etc/
[root@qianfeng01 etc]# scp -r hadoop qianfeng02:$PWD
[root@qianfeng01 etc]# scp -r hadoop qianfeng03:$PWD
```

4.3.5. YARN的服务启停

描述	命令
开启YARN全部服务	start-yarn.sh
停止YARN全部服务	stop-yarn.sh
单点开启YARN相关进程	yarn --daemon start resourcemanager yarn --daemon start nodemanager
单点停止YARN相关进程	yarn --daemon stop resourcemanager yarn --daemon stop nodemanager

> 当YARN的进程开启之后，我们可以在WebUI上查看到集群的资源信息、任务的运行状态等
>
> http://192.168.10.101:8088

4.3.6. 任务测试

当开启所有的YARN的进程之后，我们再次运行之前的Hadoop的官方案例: wordcount

```
[root@qianfeng01 ~]# hadoop jar $HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.3.1.jar wordcount /input /output
```

当任务运行起来之后，我们可以在WebUI上查看到任务的运行状态: http://192.168.10.101:8088

4.4. YARN的历史日志

4.4.1. 历史日志概述

我们在YARN运行MapReduce的程序的时候，任务会被分发到不同的节点，在不同的Container内去执行。如果一个程序执行结束后，我们想去查看这个程序的运行状态呢？每一个MapTask的执行细节？每一个ReduceTask的执行细节？这个时候我们是查看不到的，因此我们需要开启记录历史日志的服务。

历史日志服务开启之后，Container在运行任务的过程中，会将日志记录下来，保存到当前的节点。例如: 在qianfeng02节点上开启了一个Container去执行MapTask，那么此时就会在qianfeng02的$HADOOP_HOME/logs/userlogs中记录下来日志。我们可以到不同的节点上去查看日志。虽然这样可以查看，但是很不方便！因此，我们一般还会开启另外的一个服务: **日志聚合**。顾名思义，就是将不同节点的日志聚合到一起保存起来。

4.4.2. mr-historyserver

顾名思义，就是去记录MapReduce的历史日志的。接下来我们从配置开始、到日志聚合、运行任务去讲解。

4.4.2.1. 配置文件

1. mapred-site.xml

```
<!-- 添加如下配置 -->

```

```
<!-- 历史任务的内部通讯地址 -->
<property>
    <name>MapReduce.jobhistory.address</name>
    <value>qianfeng01:10020</value>
</property>

<!--历史任务的外部监听页面-->
<property>
    <name>MapReduce.jobhistory.webapp.address</name>
    <value>qianfeng01:19888</value>
</property>

<property>
  <name>yarn.app.mapreduce.am.env</name>
  <value>HADOOP_MAPRED_HOME=/usr/local/hadoop-3.3.1</value>
</property>
<property>
  <name>mapreduce.map.env</name>
  <value>HADOOP_MAPRED_HOME=/usr/local/hadoop-3.3.1</value>
</property>
<property>
  <name>mapreduce.reduce.env</name>
  <value>HADOOP_MAPRED_HOME=/usr/local/hadoop-3.3.1</value>
</property>
```

2. yarn-site.xml

```
<!-- 添加如下配置 -->

```

```
<!-- 是否需要开启日志聚合 -->
<!-- 开启日志聚合后，将会将各个Container的日志保存在yarn.nodemanager.remote-app-log-dir的位置 -->
<!-- 默认保存在/tmp/logs -->
<property>
    <name>yarn.log-aggregation-enable</name>
    <value>true</value>
</property>

<!-- 历史日志在HDFS保存的时间，单位是秒 -->
<!-- 默认的是-1，表示永久保存 -->
<property>
    <name>yarn.log-aggregation.retain-seconds</name>
    <value>604800</value>
</property>

<property>
    <name>yarn.log.server.url</name>

<value>http://qianfeng01:19888/jobhistory/logs</value>
</property>
```

4.4.2.2. 分发配置

```
[root@qianfeng01 ~]# cd $HADOOP_HOME/etc/hadoop
[root@qianfeng01 hadoop]# scp mapred-site.xml yarn-site.xml qianfeng02:$PWD
[root@qianfeng01 hadoop]# scp mapred-site.xml yarn-site.xml qianfeng03:$PWD
```

4.4.2.3. 开启历史服务

```
# 重启YARN集群
[root@qianfeng01 ~]# stop-yarn.sh
[root@qianfeng01 ~]# start-yarn.sh
# 打开历史服务
[root@qianfeng01 ~]# mapred --daemon start historyserver

# 开启之后，通过jps可以查看到 JobHistoryServer 进程，表示开启成功
```

4.4.2.4. 执行任务

```
[root@qianfeng01 ~]# hadoop jar $HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.3.1.jar wordcount /input /output
```

我们以官方案例wordcount为例，现在在运行完这个任务后，就会在WebUI上看到这个任务。我们可以点击任务的ID进入到任务的详情页，此时可以查看日志。


翻译可用
Logged in as: dr.who
Application application_ _0001
Cluster
Application Overview

也可以在http://192.168.10.101:19888查看每一个MapTask、ReduceTask的日志

4.4.3. timeline

4.4.3.1. timeline服务简介

我们在前面配置完成了mr-jobhistoryserver服务，从名字也可以看出来，这个服务只能够保存MapReduce任务的日志信息。但是YARN是一个通用的资源调度框架，运行在YARN上的任务不仅仅是MapReduce的任务，还有Spark的任务、Flink的任务等。假如有一天我在YARN上运行了一个Spark的任务，那么此时mr-jobhistoryserver将不会为我保存日志信息。当然Spark会有自己的服务来保存历史日志。但是如果每一种作业我们都需要考虑单独的历史服务是不是比较麻烦呢？于是timeline就出现了。

YARN中有一个Timeline Server的组件，我们通常称为**时间轴服务**，可以以通用的方式存储程序的运行日志信息。无论是MapReduce的任务，还是Spark的任务，或者是Flink的任务等等，都可以将日志保存下来。mr-jobhistoryserver只是实现了时间轴服务的一部分功能。

目前位置，Timeline Server有V1、V1.5、V2三种版本，其中V2的版本正在测试中，功能尚不完善，且后端依赖HBase。所以目前我们以V1.5的版本为例。

4.4.3.2. 配置文件

只需要配置yarn-site.xml即可

```
<!-- 是否需要开启Timeline服务 -->
```

```
<property>
    <name>yarn.timeline-service.enabled</name>
    <value>true</value>
</property>

<!-- Timeline Web服务的主机，通过8188端口访问 -->
<property>
    <name>yarn.timeline-service.hostname</name>
    <value>qianfeng01</value>
</property>

<!-- 设置ResourceManager是否发送指标信息到Timeline服务 -->
<property>
    <name>yarn.system-metrics-publisher.enabled</name>
    <value>false</value>
</property>
```

4.4.3.3. 分发并启动服务

```
[root@qianfeng01 ~]# cd $HADOOP_HOME/etc/hadoop
[root@qianfeng01 hadoop]# scp yarn-site.xml qianfeng02:$PWD
[root@qianfeng01 hadoop]# scp yarn-site.xml qianfeng03:$PWD

# 因为修改了yarn-site.xml的内容，因此需要重启YARN
[root@qianfeng01 ~]# stop-yarn.sh
[root@qianfeng01 ~]# start-yarn.sh

# 开启Timeline服务
[root@qianfeng01 ~]# yarn --daemon start timelineserver
```

4.4.3.4. 执行任务

```
[root@qianfeng01 ~]# hadoop jar $HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.3.1.jar wordcount /input /output
```

此时，可以到TimelineServer的WebUI上查看任务的状态和历史服务 http://192.168.10.101:8188

4.5. YARN的Job提交

在MR程序运行时，有五个独立的进程：

- YarnRunner: 用于提交作业的客户端程序
- ResourceManager: yarn资源管理器，负责协调集群上计算机资源的分配
- NodeManager: yarn节点管理器，负责启动和监视集群中机器上的计算容器（container）
- Application Master: 负责协调运行MapReduce作业的任务，他和任务都在容器中运行，这些容器由资源管理器分配并由节点管理器进行管理。
- HDFS:用于共享作业所需文件。

1. 调用waitForCompletion方法每秒轮询作业的进度，内部封装了submit()方法，用于创建JobCommiter实例，并且调用其的submitJobInternal方法。提交成功后，如果有状态改变，就会把进度报告到控制台。错误也会报告到控制台

2. JobCommiter实例会向ResourceManager申请一个新应用ID，用于MapReduce作业ID。这期间JobCommiter也会进行检查输出路径的情况，以及计算输入分片。

3. 如果成功申请到ID,就会将运行作业所需要的资源（包括作业jar文件，配置文件和计算所得的输入分片元数据文件）上传到一个用ID命名的目录下的HDFS上。此时副本个数默认是10.

4．准备工作已经做好，再通知ResourceManager调用submitApplication方法提交作业。

5．ResourceManager调用submitApplication方法后，会通知Yarn调度器（Scheduler），调度器分配一个容器，在节点管理器的管理下在容器中启动 application master进程。

6．application master的主类是MRAppMaster，其主要作用是初始化任务，并接受来自任务的进度和完成报告。

7．然后从HDFS上接受资源，主要是split。然后为每一个split创建MapTask以及参数指定的ReduceTask，任务ID在此时分配

8．然后Application Master会向资源管理器请求容器，首先为MapTask申请容器，然后再为ReduceTask申请容器。（5%）

9．一旦ResourceManager中的调度器（Scheduler），为Task分配了一个特定节点上的容器，Application Master就会与NodeManager进行通信来启动容器。

10．运行任务是由YarnChild来执行的，运行任务前，先将资源本地化（jar文件，配置文件，缓存文件）

11．然后开始运行MapTask或ReduceTask。

12．当收到最后一个任务已经完成的通知后，application master会把作业状态设置为success。然后Job轮询时，知道成功完成，就会通知客户端，并把统计信息输出到控制台

4.6. YARN的命令

- yarn top

```
类似于Linux的top命令，查看正在运行的程序资源占用情况。
```

- yarn queue -status root.default

```
查看指定队列使用情况，下文会讲解任务队列
```

- yarn application
  - -list

```
# 通过任务的状态，列举YARN的任务。使用 -appStates 指定状态
# 任务状态：ALL、NEW、NEW_SAVING、SUBMITTED、ACCEPTED、RUNNING、FINISHED、FAILED、KILLED

# e.g.
# 查看所有正在运行的任务
[root@qianfeng01 ~]# yarn application -list -appStates RUNNING
# 查看所有的失败的任务
[root@qianfeng01 ~]# yarn application -list -appStates FAILED
```

  - -movetoqueue

```
# 将一个任务移动到指定的队列中
[root@qianfeng01 ~]# yarn application -movetoqueue application_xxxxxx_xxx -queue root.small
```

- -kill

```
# 杀死指定的任务
[root@qianfeng01 ~]# yarn application -kill application_xxxxxx_xxx
```

- yarn container
  - -list

```
# 查看正在执行的任务的容器信息
[root@qianfeng01 ~]# yarn container -list application_xxxxxx_xxx
```

  - -status

```
# 查看指定容器信息
[root@qianfeng01 ~]# yarn container -status container_xxxxx
```

- yarn jar

```
# 提交任务到YARN执行
[root@qianfeng01 ~]# yarn jar $HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.3.1.jar /input /output
```

- yarn logs

```
# 查看YARN的程序运行时的日志信息
[root@qianfeng01 ~]# yarn logs -applicationId application_1528080031923_0064
```

- yarn node -all -list

```
查看所有节点信息
```

4.7. YARN的三种调度器

4.7.1 什么是Scheduler（调度器）

```
Scheduler即调度器，根据容量、队列等限制条件（如每个队列分配一定的资源，最多执行一定数量的作业等），将系统中的资源分配给各个正在运行的应用程序。
```

4.7.2 YARN提供的三种内置调度器：

4.7.2.1. FIFO Scheduler（FIFO调度器）

```
FIFO 为 First Input First Output 的缩写，先进先出。FIFO 调度器将应用放在一个队列中，按照先 后顺序
运行应用。这种策略较为简单，但不适合共享集群，因为大的应用会占用集群的所有资源，每个应用必须等待直到轮到自己。

优点：简单易懂，不需要任何配置
缺点：不适合共享集群，大的应用会占据集群中的所有资源，所以每个应用都必须等待，直到轮到自己执行。
```

如下图所示，只有当job1全部执行完毕，才能开始执行job2

4.7.2.2. Capacity Scheduler（容量调度器）

```
容量调度器 Capacity Scheduler 允许多个组织共享一个 Hadoop 集群。使用容量调度器时，一个独立的专门队列保证小作业一提交就可以启动。

优点：小任务不会因为前面有大任务在执行，而只能一直等下去
缺点：这种策略是以整个集群利用率为代价的，这意味着与使用FIFO调度器相比，大作业执行的时间要长上一些。
```

如图所示，专门留了一部分资源给小任务，可以在执行job1的同时，不会阻塞job2的执行，但是因为这部分资源是一直保留给其他任务的，所以就算只有一个任务，也无法为其分配全部资源，只能让这部分保留资源闲置着，有着一定的资源浪费问题。

4.7.2.3. Fair Scheduler（公平调度器）

```
公平调度器的目的就是为所有运行的应用公平分配资源。使用公平调度器时，不需要预留一定量的资源，因为调度器
会在所有运行的作业之间动态平衡资源，第一个（大）作业启动时，它也是唯一运行的作业，因而获得集群中的所有
资源，当第二个（小）作业启动时，它被分配到集群的一半资源，这样每个作业都能公平共享资源。
```

如图所示，就像是把好几个任务拼接成了一个任务，可以充分利用资源，同时又不会因为大任务在前面执行而导致小任务一直无法完成

4.8. YARN的队列配置

YARN默认采用的调度器是容量调度，且默认只有一个任务队列。该调度器内单个队列的调度策略为FIFO，因此在单个队列中的任务并行度为1。那么就会出现单个任务阻塞的情况，如果随着业务的增长，充分的利用到集群的使用率，我们就需要手动的配置多条任务队列。

4.8.1. 配置任务队列

> 默认YARN只有一个default任务队列，现在我们添加一个small的任务队列。

修改配置文件: **$HADOOP_HOME/etc/hadoop/capacity-scheduler.xml**

```
<configuration>
    <!-- 不需要修改 -->
```

```
    <!-- 容量调度器中最多容纳多少个Job -->
    <property>
        <name>yarn.scheduler.capacity.maximum-applications</name>
        <value>10000</value>
        <description>
            Maximum number of applications that can be pending and running.
        </description>
    </property>

    <!-- 不需要修改 -->
    <!-- MRAppMaster进程所占的资源可以占用队列总资源的百分比，可以通过修改这个参数来限制队列中提交Job的数量 -->
    <property>
        <name>yarn.scheduler.capacity.maximum-am-resource-percent</name>
        <value>0.1</value>
        <description>
            Maximum percent of resources in the cluster which can be used to run
            application masters i.e. controls number of concurrent running
            applications.
        </description>
    </property>

    <!-- 不需要修改 -->
    <!-- 为Job分配资源的时候，使用什么策略 -->
    <property>
        <name>yarn.scheduler.capacity.resource-calculator</name>
```

```
28
<value>org.apache.hadoop.yarn.util.resource.DefaultResourceCalculator</value>
29          <description>
30              The ResourceCalculator implementation to be used to compare
31              Resources in the scheduler.
32              The default i.e. DefaultResourceCalculator only uses Memory while
33              DominantResourceCalculator uses dominant-resource to compare
34              multi-dimensional resources such as Memory, CPU etc.
35          </description>
36      </property>
37
38      <!-- 修改!!! -->
39      <!-- 调度器中有什么队列，我们添加一个small队列 -->
40      <property>
41
<name>yarn.scheduler.capacity.root.queues</name>
42          <value>default,small</value>
43          <description>
44              The queues at the this level (root is the root queue).
45          </description>
46      </property>
47
48      <!-- 修改!!! -->
49      <!-- 配置default队列的容量百分比 -->
50      <property>
```

```
51
<name>yarn.scheduler.capacity.root.default.capacity</name>
52          <value>70</value>
53          <description>Default queue target capacity.</description>
54      </property>
55
56      <!-- 新增!!! -->
57      <!-- 新增small队列的容量百分比 -->
58      <!-- 所有的队列容量百分比和需要是100 -->
59      <property>
60
<name>yarn.scheduler.capacity.root.small.capacity</name>
61          <value>30</value>
62          <description>Default queue target capacity.</description>
63      </property>
64
65      <!-- 不需要修改 -->
66      <!-- default队列用户能使用的容量最大百分比 -->
67      <property>
68
<name>yarn.scheduler.capacity.root.default.user-limit-factor</name>
69          <value>1</value>
70          <description>
71              Default queue user limit a percentage from 0.0 to 1.0.
72          </description>
73      </property>
74
```

```
    <!-- 添加!!! -->
    <!-- small队列用户能使用的容量最大百分比 -->
    <property>
        <name>yarn.scheduler.capacity.root.small.user-limit-factor</name>
        <value>1</value>
        <description>
            Default queue user limit a percentage from 0.0 to 1.0.
        </description>
    </property>

    <!-- 不需要修改 -->
    <!-- default队列能使用的容量最大百分比 -->
    <property>

<name>yarn.scheduler.capacity.root.default.maximum-capacity</name>
        <value>100</value>
        <description>
            The maximum capacity of the default queue.
        </description>
    </property>

    <!-- 添加!!! -->
    <!-- small队列能使用的容量最大百分比 -->
    <property>

<name>yarn.scheduler.capacity.root.small.maximum-capacity</name>
        <value>100</value>
        <description>
            The maximum capacity of the default queue.
```

```
        </description>
    </property>

    <!-- 不需要修改 -->
    <!-- default队列的状态 -->
    <property>

<name>yarn.scheduler.capacity.root.default.state</name
>
        <value>RUNNING</value>
        <description>
            The state of the default queue. State can
be one of RUNNING or STOPPED.
        </description>
    </property>

    <!-- 添加!!! -->
    <!-- small队列的状态 -->
    <property>

<name>yarn.scheduler.capacity.root.small.state</name>
        <value>RUNNING</value>
        <description>
            The state of the default queue. State can
be one of RUNNING or STOPPED.
        </description>
    </property>

    <!-- 不需要修改 -->
    <!-- 限制向队列提交的用户-->
    <property>
```

```
128
    <name>yarn.scheduler.capacity.root.default.acl_submit_
    applications</name>
129         <value>*</value>
130         <description>
131             The ACL of who can submit jobs to the
    default queue.
132         </description>
133     </property>
134     <!-- 添加!!! -->
135     <property>
136
    <name>yarn.scheduler.capacity.root.small.acl_submit_ap
    plications</name>
137         <value>*</value>
138         <description>
139             The ACL of who can submit jobs to the
    default queue.
140         </description>
141     </property>
142
143     <!-- 不需要修改 -->
144     <property>
145
    <name>yarn.scheduler.capacity.root.default.acl_adminis
    ter_queue</name>
146         <value>*</value>
147         <description>
148             The ACL of who can administer jobs on the
    default queue.
149         </description>
150     </property>
151     <!-- 添加!!! -->
```

```
152     <property>
153 
    <name>yarn.scheduler.capacity.root.small.acl_administe
    r_queue</name>
154         <value>*</value>
155         <description>
156             The ACL of who can administer jobs on the
    default queue.
157         </description>
158     </property>
159 
160 
161     <!-- 不需要修改 -->
162     <property>
163         <name>yarn.scheduler.capacity.node-locality-
    delay</name>
164         <value>40</value>
165         <description>
166             Number of missed scheduling opportunities
    after which the CapacityScheduler
167             attempts to schedule rack-local
    containers.
168             Typically this should be set to number of
    nodes in the cluster, By default is setting
169             approximately number of nodes in one rack
    which is 40.
170         </description>
171     </property>
172     <!-- 不需要修改 -->
173     <property>
174         <name>yarn.scheduler.capacity.queue-
    mappings</name>
175         <value></value>
```

```
        <description>
            A list of mappings that will be used to assign jobs to queues
            The syntax for this list is [u|g]:[name]:[queue_name][,next mapping]*
            Typically this list will be used to map users to queues,
            for example, u:%user:%user maps all users to queues with the same name
            as the user.
        </description>
    </property>
    <!-- 不需要修改 -->
    <property>
        <name>yarn.scheduler.capacity.queue-mappings-override.enable</name>
        <value>false</value>
        <description>
            If a queue mapping is present, will it override the value specified
            by the user? This can be used by administrators to place jobs in queues
            that are different than the one specified by the user.
            The default is false.
        </description>
    </property>
</configuration>
```

4.8.2. 分发配置到各个节点

```
[root@qianfeng01 ~]# cd $HADOOP_HOME/etc/hadoop
[root@qianfeng01 hadoop]# scp capacity-scheduler.xml qianfeng02:$PWD
[root@qianfeng01 hadoop]# scp capacity-scheduler.xml qianfeng03:$PWD

# 重启集群
[root@qianfeng01 hadoop]# stop-yarn.sh
[root@qianfeng01 hadoop]# start-yarn.sh
```

4.8.3. 提交任务

```
# 提交Job到default队列，其中的-Dmapreduce.job.queuename可以不指定，因为现在默认向default队列提交Job
[root@qianfeng01 ~]# hadoop jar $HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.3.1.jar wordcount -Dmapreduce.job.queuename=default /input /output1

# 提交Job到small队列
[root@qianfeng01 ~]# hadoop jar $HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.3.1.jar wordcount -Dmapreduce.job.queuename=small /input /output2
```

4.8.4. 查看任务

http://192.168.10.101:8088/cluster/scheduler

4.8.5. 默认队列设置

YARN默认将任务提交到default队列，我们如果需要提交到其他的队列中，可以使用-Dmapreduce.job.queuename指定提交的队列。也可以设置默认的任务提交队列。

> 例如: Hive的底层会把HQL语句翻译成MapReduce的程序执行，我们可以创建一个hive队列，将这个队列的容量设置的大一些。我们可以设置默认将任务提交到这个队列中。如果需要往其他的队列中提交任务的话，可以再使用-Dmapreduce.job.queuename去提交了。

- 临时生效

```
set mapreduce.job.queuename=small;
set mapreduce.job.priority=HIGH;
```

- 永久生效

```
<!--配置hive默认的提交队列-->
<property>
    <name>mapreduce.job.queuename</name>
    <value>small</value>
</property>
```

4.9. YARN的Node Label机制

4.9.1. Node Label的介绍

官网对NodeLabel的介绍如下:

```
Node label is a way to group nodes with similar characteristics and applications can specify where to run.
节点标签是一种对具有相似特征的节点进行分组的方法，应用程序可以指定在哪里运行。
```

那么标签到底是做什么的?

```
我们可以创建多个队列，划分集群的总的资源。例如队列hive占集群总资源的70%，那么这个70%具体会落地在哪一个节点上呢？没办法确定，有可能在qianfeng01，也有可能在qianfeng03。而节点标签，可以为每一个NodeManager打上标签，可以限定某一个程序只能够运行在哪些节点上。例如我希望我的wordcount的程序只会运行在qianfeng02和qianfeng03节点上，这就是标签的作用。
```

4.9.2. 开启标签

修改 **yarn-site.xml** 文件，添加如下配置:

```
<!-- 启用节点标签 -->
<property>
    <name>yarn.node-labels.enabled</name>
    <value>true</value>
</property>

<!-- 节点标签存储的路径，可以是HDFS，也可以是本地文件系统 -->
```

```
<!-- 如果是本地文件系统，使用类似file:///home/yarn/node-label这样的路径 -->
<!-- 无论是HDFS，还是本地文件系统，需要保证RM有权限去访问 -->
<property>
    <name>yarn.node-labels.fs-store.root-dir</name>
    <value>hdfs://qianfeng01:9820/tmp/yarn/node-labels/</value>
</property>

<!-- 保持默认即可，也可以不配置这个选项 -->
<property>
    <name>yarn.node-labels.configuration-type</name>
    <value>centralized</value>
</property>
```

分发到每一个节点，重启YARN即可。

4.9.3. 标签管理

4.9.3.1. 添加标签

```
yarn rmadmin -addToClusterNodeLabels "label_1"
yarn rmadmin -addToClusterNodeLabels "label_2,label_3"
```

4.9.3.2. 查看标签

```
yarn cluster --list-node-labels
```

4.9.3.3. 删除标签

```
yarn rmadmin -removeFromClusterNodeLabels label_1
```

4.9.3.4. 为节点打上标签

```
# 绑定一个NodeManager与Label
yarn rmadmin -replaceLabelsOnNode "qianfeng02=label_2"
# 绑定多个NodeManager与Label的关系，中间用空格分隔
yarn rmadmin -replaceLabelsOnNode "qianfeng02=label_2 qianfeng03=label_2"

# 一个标签可以绑定多个NodeManager，一个NodeManager只能绑定一个标签。
# 例如上方的，label_2就绑定在了qianfeng02和qianfeng03的NodeManager上。

# 绑定完成后，可以使用WebUI进行查看。
# 在WebUI的左侧，有Node Labels的查看，可以查看到所有的标签，以及对应的节点信息。
# 需要注意的是，如果某节点没有进行标签的绑定，则其在一个默认的<DEFAULT_PARTITION>上绑定。
```

4.9.4. 为队列绑定标签

通过修改**capacity-scheduler.xml**实现:

```
<!-- 前文，我们已经新增了一个队列，现在共有两个队列：default、small -->

```

```
<!-- 设置某个队列可以使用的标签，*表示通配，可以使用所有标签 -->
<property>

<name>yarn.scheduler.capacity.root.default.accessible-node-labels</name>
    <value>*</value>
</property>

<!-- 设置small队列可以使用label_3标签的节点资源 -->
<property>

<name>yarn.scheduler.capacity.root.small.accessible-node-labels</name>
    <value>label_3</value>
</property>

<!-- 设置default队列可以使用label_2标签的节点资源最多60% -->
<property>

<name>yarn.scheduler.capacity.root.default.accessible-node-labels.label_2.capacity</name>
    <value>60</value>
</property>

<!-- 设置small队列可以使用label_3标签的节点资源最多80% -->
<property>

<name>yarn.scheduler.capacity.root.small.accessible-node-labels.label_3.capacity</name>
    <value>80</value>
</property>

```

```
<!-- 设置default队列，如果没有明确的标签指向，则默认使用label_3 -->
<property>
    <name>yarn.scheduler.capacity.root.default.default-node-label-expression</name>
    <value>label_3</value>
</property>
```

修改之后，无需重启，直接刷新一下队列即可: **yarn rmadmin -refreshQueues**

4.9.5. 测试

```
hadoop jar $HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.3.1.jar pi -Dmapreduce.job.queuename=small 10 10
```

4.10. 企业级调优

五、案例实战

六、教学总结

七、课后作业

八、解决方案

8.1. container的生命周期是什么，是整个job运行完成，还是说container上的任务完成后

```
1. Container启动过程主要经历三个阶段：资源本地化、启动并运行container、资源回收，其中，资源本地化指创建container工作目录，从HDFS下载运行container所需的各种资源（jar包、可执行文件等）等，而资源回收则是资源本地化的逆过程，它负责清理各种资源，它们均由ResourceLocalizationService服务完成的。启动container是由ContainersLauncher服务完成的，而运行container是由插拔式组件ContainerExecutor完成的，YARN提供了两种ContainerExecutor实现，一种是 DefaultContainerExecutor，另一种是 LinuxContainerExecutor

2. container的生命周期是这样的：

    nm先去申请资源，然后是locallizing-downloading-localized->running-exit with failure(success)—>kill->clearnup
```

8.2. 基于yarn的任务运行时报错，用Linux命令行查看错误日志信息

1、查看某个job的日志，例如：

```
yarn logs -applicationId application_1529513682598_0009
```

2、查看某个job的状态，例如：

```
yarn application -status application_1529513682598_0009
```

3、终止某个job　　注意：一般不要直接在UI界面或者是终端kill掉任务，该任务可能还会继续执行下去。

正确操作方法：停止job的执行命令如下：`yarn application -kill application_1515118561637_0439`