

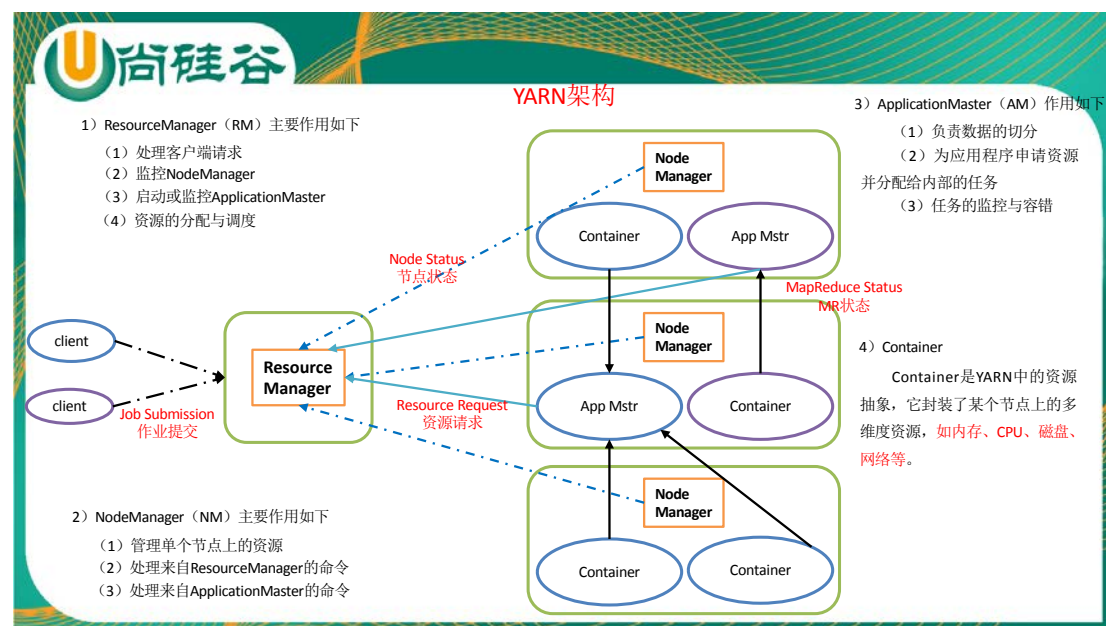
五 Yarn

5.1 Yarn 概述

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。

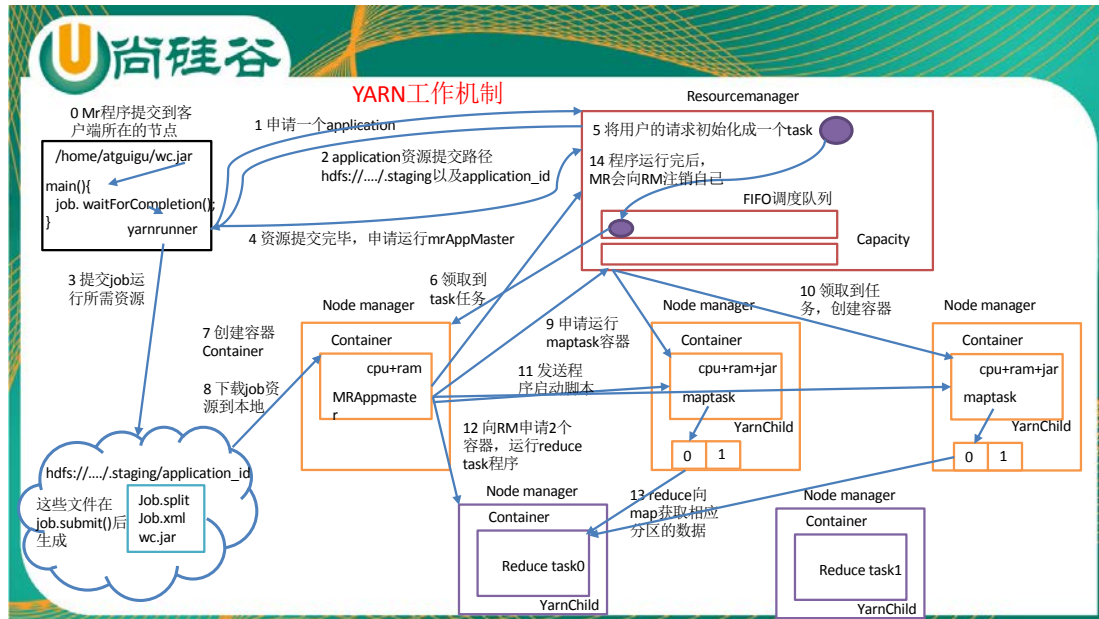
5.2 Yarn 基本架构

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



5.3 Yarn 工作机制

1) Yarn 运行机制

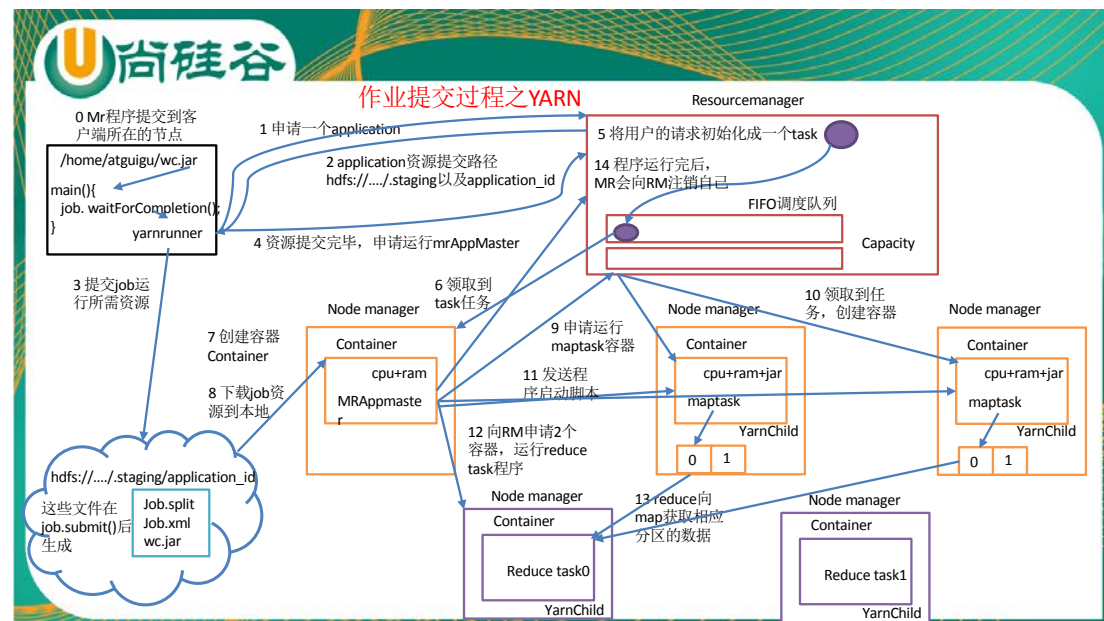


2) 工作机制详解

- (0) Mr 程序提交到客户端所在的节点。
- (1) Yarnrunner 向 Resourcemanager 申请一个 Application。
- (2) rm 将该应用程序的资源路径返回给 yarnrunner。
- (3) 该程序将运行所需资源提交到 HDFS 上。
- (4) 程序资源提交完毕后，申请运行 mrAppMaster。
- (5) RM 将用户的请求初始化成一个 task。
- (6) 其中一个 NodeManager 领取到 task 任务。
- (7) 该 NodeManager 创建容器 Container，并产生 MRAppmaster。
- (8) Container 从 HDFS 上拷贝资源到本地。
- (9) MRAppmaster 向 RM 申请运行 maptask 资源。
- (10) 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。
- (11) 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。
- (12) MrAppMaster 等待所有 maptask 运行完毕后，向 RM 申请容器，运行 reduce task。
- (13) reduce task 向 maptask 获取相应分区的数据。
- (14) 程序运行完毕后，MR 会向 RM 申请注销自己。

5.4 作业提交全过程

1) 作业提交过程之 YARN



作业提交全过程详解

(1) 作业提交

第 0 步: client 调用 `job.waitForCompletion` 方法, 向整个集群提交 MapReduce 作业。

第 1 步: client 向 RM 申请一个作业 id。

第 2 步: RM 给 client 返回该 job 资源的提交路径和作业 id。

第 3 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 4 步: client 提交完资源后, 向 RM 申请运行 MrAppMaster。

(2) 作业初始化

第 5 步: 当 RM 收到 client 的请求后, 将该 job 添加到容量调度器中。

第 6 步: 某一个空闲的 NM 领取到该 job。

第 7 步: 该 NM 创建 Container, 并产生 MRAppmaster。

第 8 步: 下载 client 提交的资源到本地。

(3) 任务分配

第 9 步: MrAppMaster 向 RM 申请运行多个 maptask 任务资源。

第 10 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 11 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 12 步: MrAppMaster 等待所有 maptask 运行完毕后, 向 RM 申请容器, 运行 reduce task。

第 13 步: reduce task 向 maptask 获取相应分区的数据。

第 14 步: 程序运行完毕后, MR 会向 RM 申请注销自己。

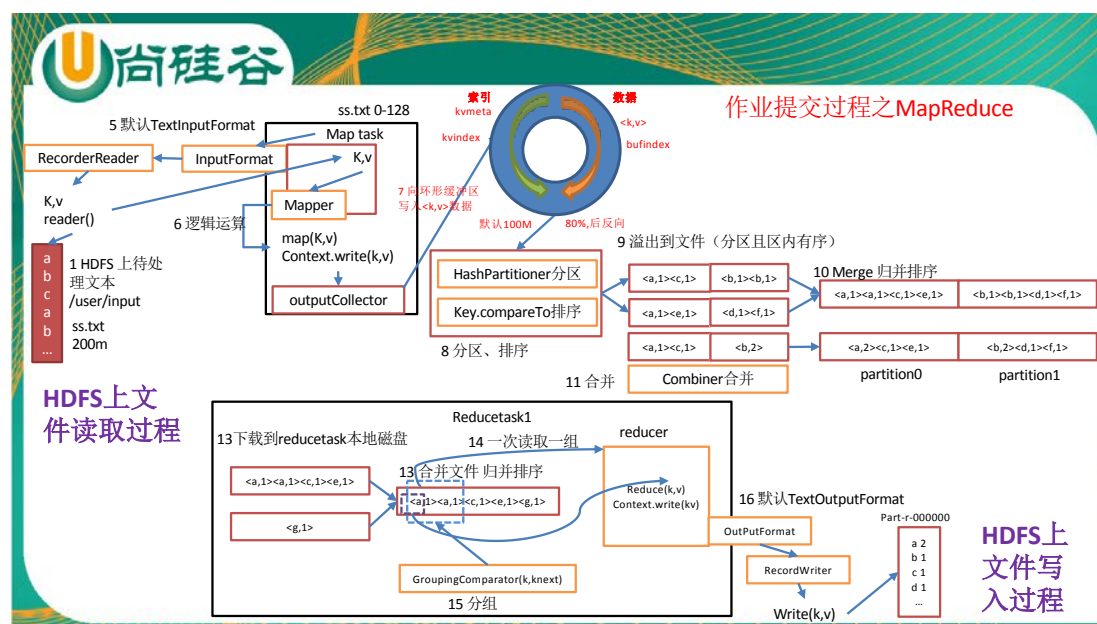
(5) 进度和状态更新

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 分钟都会通过调用 `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。

2) 作业提交过程之 MapReduce



5.5 资源调度器

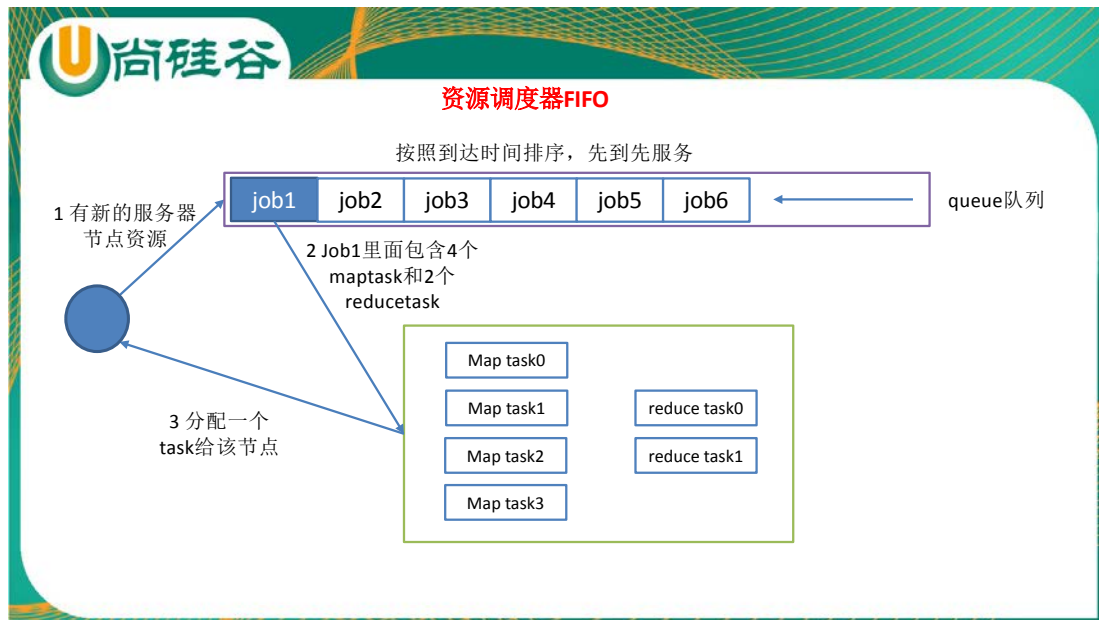
目前, Hadoop 作业调度器主要有三种: FIFO、Capacity Scheduler 和 Fair Scheduler。Hadoop2.7.2 默认的资源调度器是 Capacity Scheduler。

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

1) 先进先出调度器 (FIFO)



2) 容量调度器 (Capacity Scheduler)



3) 公平调度器 (Fair Scheduler)



5.6 任务的推测执行

1) 作业完成时间取决于最慢的任务完成时间

一个作业由若干个 Map 任务和 Reduce 任务构成。因硬件老化、软件 Bug 等，某些任务可能运行非常慢。

典型案例：系统中有 99% 的 Map 任务都完成了，只有少数几个 Map 老是进度很慢，完不成，怎么办？

2) 推测执行机制：

发现拖后腿的任务，比如某个任务运行速度远慢于任务平均速度。为拖后腿任务启动一个备份任务，同时运行。谁先运行完，则采用谁的结果。

3) 执行推测任务的前提条件

- (1) 每个 task 只能有一个备份任务；
- (2) 当前 job 已完成的 task 必须不小于 0.05 (5%)
- (3) 开启推测执行参数设置。Hadoop2.7.2 mapred-site.xml 文件中默认是打开的。

```
<property>
  <name>mapreduce.map.speculative</name>
  <value>true</value>
  <description>If true, then multiple instances of some map tasks
may be executed in parallel.</description>
</property>

<property>
  <name>mapreduce.reduce.speculative</name>
```



```
<value>true</value>
<description>If true, then multiple instances of some reduce tasks
may be executed in parallel.</description>
</property>
```

4) 不能启用推测执行机制情况

- (1) 任务间存在严重的负载倾斜;
- (2) 特殊任务, 比如任务向数据库中写数据。

5) 算法原理



推测执行算法原理

假设某一时刻, 任务T的执行进度为`progress`, 则可通过一定的算法推测出该任务的最终完成时刻`estimateEndTime`。另一方面, 如果此刻为该任务启动一个备份任务, 则可推断出它可能的完成时刻`estimateEndTime``, 于是可得出以下几个公式:

$$\text{estimateEndTime} = \text{estimatedRunTime} + \text{taskStartTime}$$
$$\text{推测执行完时刻 } 60 = \text{推测运行时间 (60s)} + \text{任务启动时刻 (0)}$$
$$\text{estimatedRunTime} = (\text{currentTimestamp} - \text{taskStartTime}) / \text{progress}$$
$$\text{推测运行时间 (60s)} = (\text{当前时刻 (6)} - \text{任务启动时刻 (0)}) / \text{任务运行比例 (10\%)}$$
$$\text{estimateEndTime`} = \text{currentTimestamp} + \text{averageRunTime}$$
$$\text{备份任务推测完成时刻 (16)} = \text{当前时刻 (6)} + \text{运行完成任务的平均时间 (10s)}$$

- 1 MR总是选择 (`estimateEndTime - estimateEndTime``) 差值最大的任务, 并为之启动备份任务。
- 2 为了防止大量任务同时启动备份任务造成的资源浪费, MR为每个作业设置了同时启动的备份任务数目上限。
- 3 推测执行机制实际上采用了经典的优化算法: 以空间换时间, 它同时启动多个相同任务处理相同的数据, 并让这些任务竞争以缩短数据处理时间。显然, 这种方法需要占用更多的计算资源。在集群资源紧缺的情况下, 应合理使用该机制, 争取在多用少量资源的情况下, 减少作业的计算时间。

