

# Yarn资源调度系统

## 一、课前准备

1. 搭建好至少有三个节点的hadoop集群.

## 二、课堂主题

1. yarn架构, 核心组件
2. yarn应用提交过程
3. yarn的调度策略
4. yarn的优化

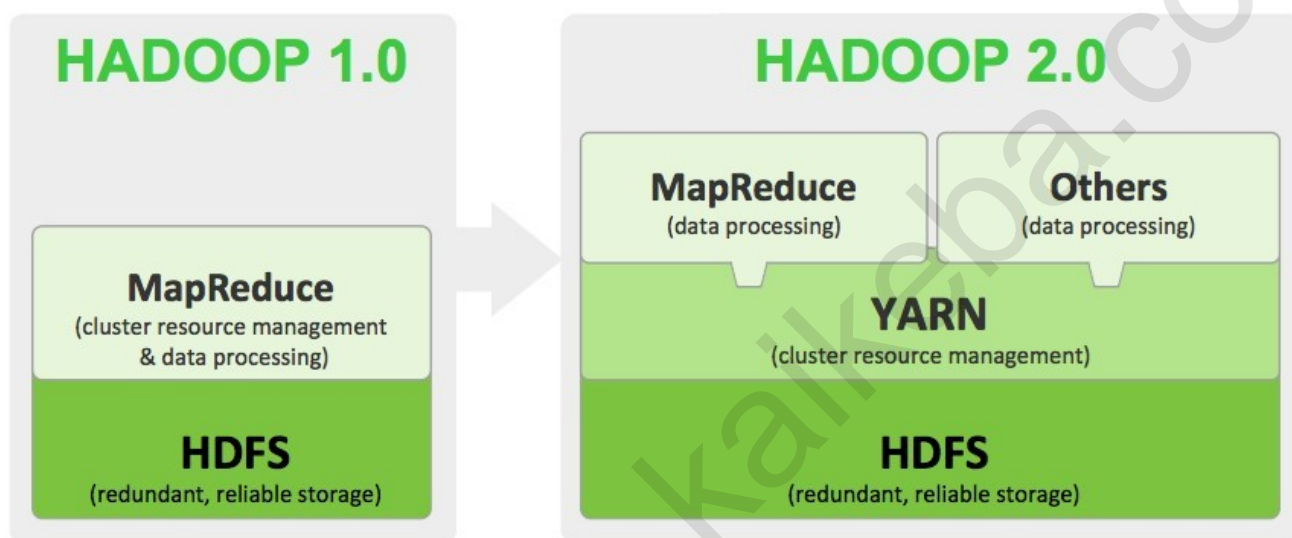
## 三、课堂目标

1. 数据yarn资源和任务调度原理
2. 熟练对yarn集群进行维护
3. 了解如何使用YARN的可扩展性, 效率和灵活性来增强集群性能

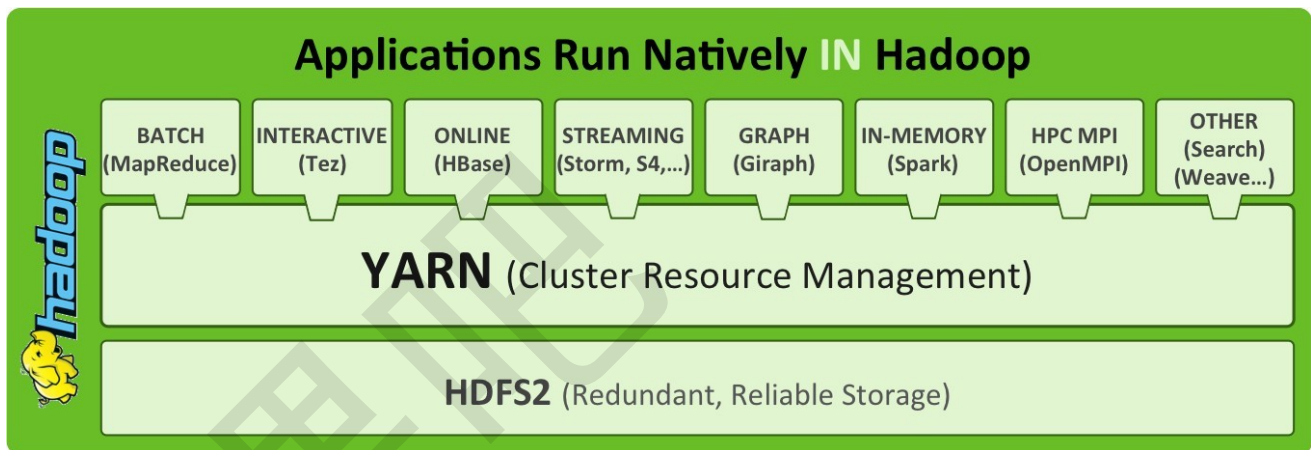
## 四、知识要点

### 1. yarn介绍

Apache Hadoop YARN 是 apache Software Foundation Hadoop的子项目, 为分离Hadoop2.0资源管理和计算组件而引入。YARN的诞生缘于存储于HDFS的数据需要更多的交互模式, 不单单是MapReduce模式。Hadoop2.0 的 YARN 架构提供了更多的处理框架, 不再强迫使用MapReduce框架。



从hadoop2.0的架构图可以看出，YARN承担着原本由MapReduce承担的资源管理的功能，同时将这部分的功能打包使得他们可以被新的数据处理引擎使用。这也同时简化了MapReduce的流程，使得MapReduce专注的将数据处理做到最好。使用YARN,可以用共同的资源管理，在Hadoop上跑很多应用程序。目前，很多机构已经开发基于YARN的应用程序。



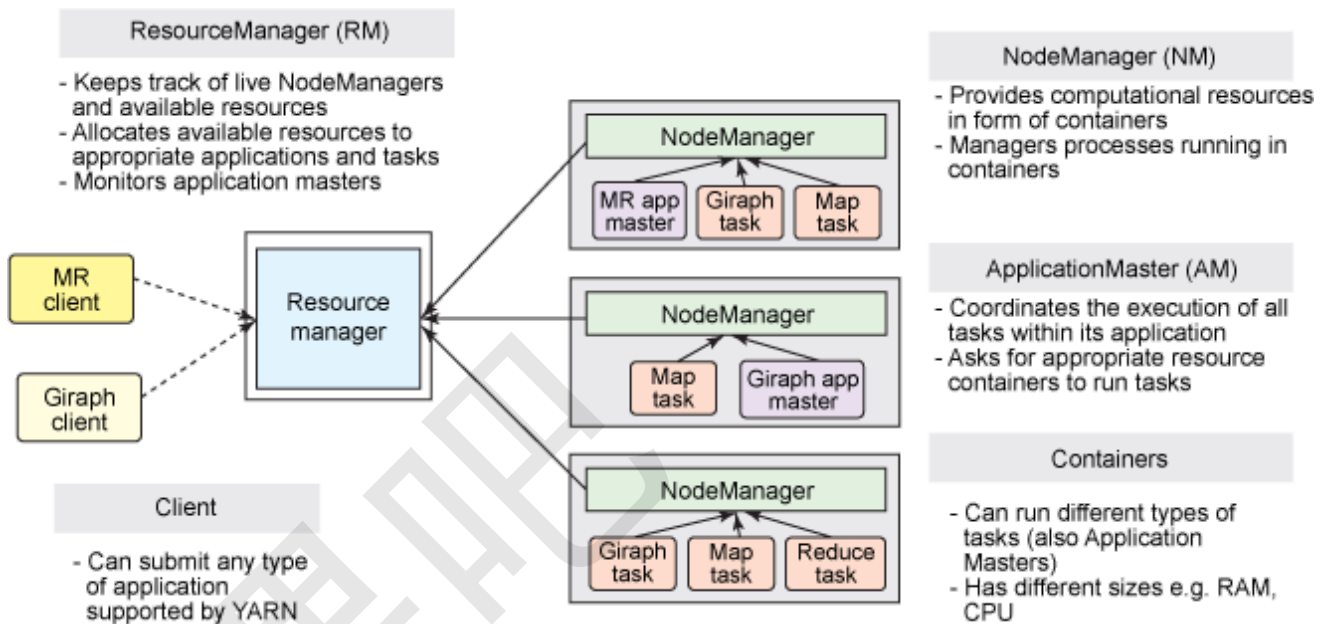
当企业的数据在HDFS中是可用的，有多种数据处理方式是非常重要的。有了Hadoop2.0和YARN,机构可以采用流处理、互动数据处理方式以及其他的基于Hadoop的应用程序。

当企业的数据在HDFS中是可用的，有多种数据处理方式是非常重要的。有了Hadoop2.0和YARN,机构可以采用流处理、互动数据处理方式以及其他的基于Hadoop的应用程序。

YARN 提供了些什么？YARN从以下几个方面提升了Hadoop的计算能力：1、可扩展性 数据中心的数据处理能力继续快速增长。因为YARN ResourceManager 仅仅专注于 调度，能将大集群的管理变得更加简单。2、兼容 MapReduce 现存的MapReduce 应用程序无需更改就能直接在YARN上运行 3、提高集群使用率 ResourceManager 是一个纯粹的调度系统根据capacity ,fair 或者SIAs等原则对集群进行优化利用。与之前不同的是，再也没有map slot和reduce slot,没有这两类资源的划分，有助于提高集群资源的利用。4、支持MapReduce 以外的计算框架 数据处理除了图形处理和迭代处理，还为企业添加了一些实时处理模型，从而提升企业对Hadoop 投资回报率，5、灵活 随着MapReduce 成为用户端库，它发展独立于底层的资源管理层，从而可以有多种灵活的方式。

## 2. yarn架构

YARN的架构还是经典的**主从 ( master/slave ) 结构**，如下图所示。大体上看，YARN服务由一个 ResourceManager ( RM ) 和多个NodeManager ( NM ) 构成，ResourceManager为主节点 ( master ) ，NodeManager为从节点 ( slave ) 。



在YARN体系结构中，全局ResourceManager作为主守护程序运行，通常在专用计算机上运行，该仲裁程序在各种竞争应用程序之间仲裁可用的群集资源。ResourceManager跟踪群集上可用的活动节点和资源数量，并协调用户提交的程序应获取这些资源的时间和数量。ResourceManager是具有此信息的单个进程，因此它可以以共享、安全和多租户的方式进行分配（或者更确切地说，调度）决策（例如，根据应用程序优先级，队列容量，ACL，数据位置等）。

当用户提交应用程序时，将启动名为ApplicationMaster的轻量级进程实例，以协调应用程序中所有任务的执行。这包括监视任务，重新启动失败的任务，推测性地运行慢速任务以及计算应用程序计数器的总值。这些职责先前已分配给所有工作的单个JobTracker。ApplicationMaster和属于其应用程序的任务在NodeManagers控制的资源容器中运行。

NodeManager是TaskTracker的更通用和高效的版本。NodeManager没有固定数量的map和reduce插槽，而是有许多动态创建的资源容器。容器的大小取决于它包含的资源量，例如内存，CPU，磁盘和网络IO。目前，仅支持内存和CPU（YARN-3）。cgroups可能会在将来用于控制磁盘和网络IO。节点上的容器数量是配置参数和专用于从属守护程序和OS的资源之外的节点资源总量（例如总CPU和总内存）的乘积。

有趣的是，ApplicationMaster可以在容器内运行任何类型的任务。例如，MapReduce ApplicationMaster请求容器启动map或reduce任务，而Giraph ApplicationMaster请求容器运行Giraph任务。您还可以实现运行特定任务的自定义ApplicationMaster，并以此方式创建一个闪亮的新分布式应用程序框架，该框架可以更改大数据世界。我鼓励您阅读Apache Twill，它旨在简化编写位于YARN之上的分布式应用程序。

在YARN中，MapReduce简单地降级为分布式应用程序的角色（但仍然是非常流行且有用的），现在称为MRv2。MRv2只是重新实现经典的MapReduce引擎，现在称为MRv1，它运行在YARN之上。

**一个可以运行任何分布式应用程序的集群** ResourceManager，NodeManager和容器不关心应用程序或任务的类型。所有特定于应用程序框架的代码都被简单地移动到了其ApplicationMaster，以便YARN可以支持任何分布式框架 - 只要有人为它实现适当的ApplicationMaster。

由于这种通用方法，运行许多不同工作负载的Hadoop YARN集群的梦想成真。想象一下：数据中心内的单个Hadoop集群可以运行MapReduce，Giraph，Storm，Spark，Tez / Impala，MPI等。

单集群方法显然具有许多优点，包括：

更高的集群利用率，一个框架未使用的资源可以被另一个框架使用 降低运营成本，因为只需要管理和调整一个“全能”集群 减少数据移动，因为不需要在Hadoop YARN和运行在不同机器群集上的系统之间移动数据 管理单个群集还可以为数据处理提供更环保的解决方案。使用的数据中心空间更少，浪费的硅更少，耗电更少，碳排放更少，因为我们在更小但更高效的Hadoop集群上运行相同的计算。

## 核心组件

简单地说，YARN 主要由 ResourceManager、NodeManager、ApplicationMaster和 Container 等几个组件构成。

组件名	作用
ResourceManager	是Master上一个独立运行的进程，负责集群统一的资源管理、调度、分配等等；
ApplicationManager	相当于这个Application的监护人和管理者，负责监控、管理这个Application的所有Attempt在cluster中各个节点上的具体运行，同时负责向Yarn ResourceManager申请资源、返还资源等；
NodeManager	是Slave上一个独立运行的进程，负责上报节点的状态(磁盘，内存，cpu等使用信息)；
Container	是yarn中分配资源的一个单位，包涵内存、CPU等等资源，YARN以Container为单位分配资源；

ResourceManager 负责对各 NodeManager 上资源进行统一管理和调度。当用户提交一个应用程序时，需要提供一个用以跟踪和管理这个程序的 ApplicationMaster，它负责向 ResourceManager 申请资源，并要求 NodeManager 启动可以占用一定资源的任务。由于不同的 ApplicationMaster 被分布到不同的节点上，因此它们之间不会相互影响。

Client 向 ResourceManager 提交的每一个应用程序都必须有一个 ApplicationMaster，它经过 ResourceManager 分配资源后，运行于某一个 Slave 节点的 Container 中，具体做事情的 Task，同样也运行与某一个 Slave 节点的 Container 中。

## 2.1 ResourceManager

RM是一个全局的资源管理器，集群只有一个，负责整个系统的资源管理和分配，包括处理客户端请求、启动/监控 ApplicationMaster、监控 NodeManager、资源的分配与调度。它主要由两个组件构成：调度器（Scheduler）和应用程序管理器（Applications Manager，ASM）。

### （1）调度器

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。需要注意的是，该调度器是一个“纯调度器”，它从事任何与具体应用程序相关的工作，比如不负责监控或者跟踪应用的执行状态等，也不负责重新启动因应用执行失败或者硬件故障而产生的失败任务，这些均交由应用程序相关的ApplicationMaster完成。



调度器仅根据各个应用程序的资源需求进行资源分配，而资源分配单位用一个抽象概念“资源容器”（Resource Container，简称Container）表示，Container是一个动态资源分配单位，它将内存、CPU、磁盘、网络等资源封装在一起，从而限定每个任务使用的资源量。

## （2）应用程序管理器

应用程序管理器主要负责管理整个系统中所有应用程序，接收job的提交请求，为应用分配第一个 Container 来运行 ApplicationMaster，包括应用程序提交、与调度器协商资源以启动 ApplicationMaster、监控 ApplicationMaster 运行状态并在失败时重新启动它等。

## 2.2 ApplicationMaster

管理 YARN 内运行的一个应用程序的每个实例。关于 job 或应用的管理都是由 ApplicationMaster 进程负责的，Yarn 允许我们以为自己的应用开发 ApplicationMaster。

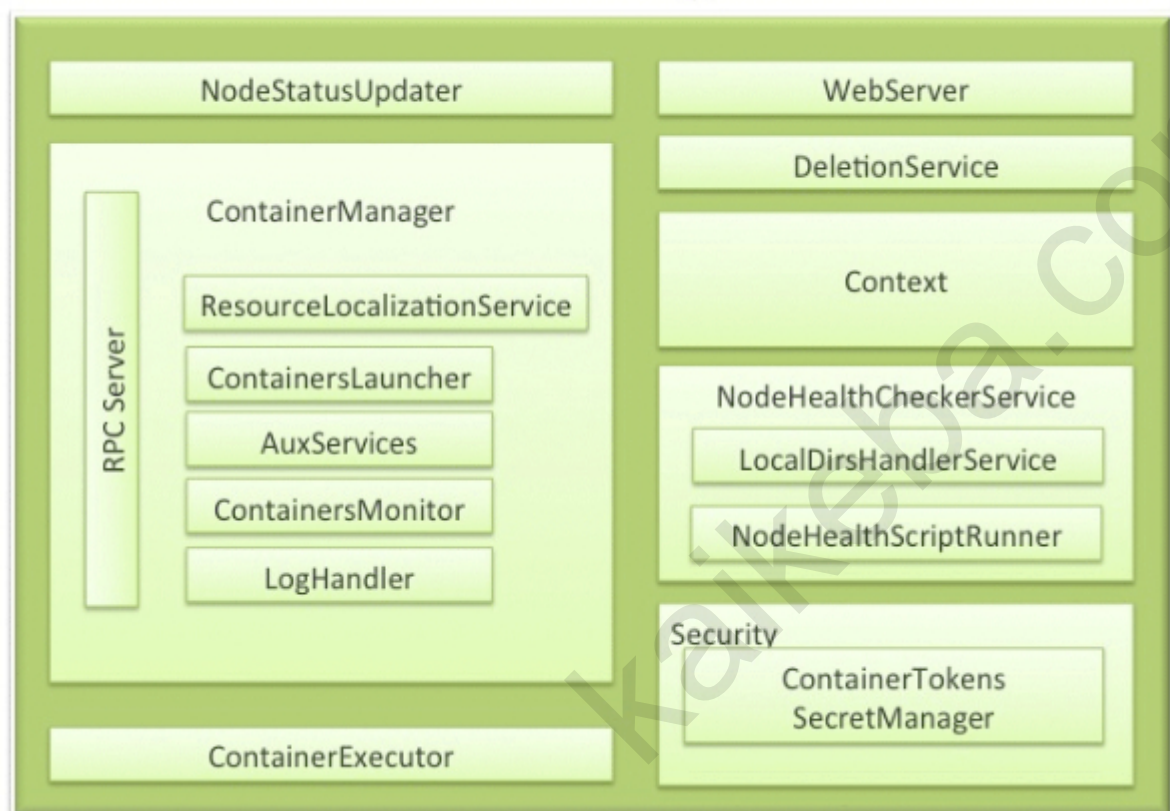
功能：

- 数据切分；
- 为应用程序申请资源并进一步分配给内部任务（TASK）；
- 任务监控与容错；
- 负责协调来自ResourceManager的资源，并通过NodeManager监视容器的执行和资源使用情况。

可以说，ApplicationMaster 与 ResourceManager 之间的通信是整个 Yarn 应用从提交到运行的最核心部分，是 Yarn 对整个集群进行动态资源管理的根本步骤，Yarn 的动态性，就是来源于多个 Application 的 ApplicationMaster 动态地和 ResourceManager 进行沟通，不断地申请、释放、再申请、再释放资源的过程。

## 2.3 NodeManager

### NodeManager



NodeManager 整个集群有多个，负责每个节点上的资源和使用。

NodeManager 是一个 slave 服务：它负责接收 ResourceManager 的资源分配请求，分配具体的 Container 给应用。同时，它还负责监控并报告 Container 使用信息给 ResourceManager。通过和 ResourceManager 配合，NodeManager 负责整个 Hadoop 集群中的资源分配工作。

功能：NodeManager 本节点上的资源使用情况和各个 Container 的运行状态（cpu和内存等资源）

- 接收及处理来自 ResourceManager 的命令请求，分配 Container 给应用的某个任务；
- 定时地向RM汇报以确保整个集群平稳运行，RM 通过收集每个 NodeManager 的报告信息来追踪整个集群健康状态的，而 NodeManager 负责监控自身的健康状态；
- 处理来自 ApplicationMaster 的请求；
- 管理着所在节点每个 Container 的生命周期；
- 管理每个节点上的日志；
- 执行 Yarn 上面应用的一些额外的服务，比如 MapReduce 的 shuffle 过程；

当一个节点启动时，它会向 ResourceManager 进行注册并告知 ResourceManager 自己有多少资源可用。在运行期，通过 NodeManager 和 ResourceManager 协同工作，这些信息会不断被更新并保障整个集群发挥出最佳状态。

NodeManager 只负责管理自身的 Container，它并不知道运行在它上面应用的信息。负责管理应用信息的组件是 ApplicationMaster

## 2.4 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，当 AM 向 RM 申请资源时，RM 为 AM 返回的资源便是用 Container 表示的。YARN 会为每个任务分配一个 Container，且该任务只能使用该 Container 中描述的资源。

Container 和集群节点的关系是：一个节点会运行多个 Container，但一个 Container 不会跨节点。任何一个 job 或 application 必须运行在一个或多个 Container 中，在 Yarn 框架中，ResourceManager 只负责告诉 ApplicationMaster 哪些 Containers 可以用，ApplicationMaster 还需要去找 NodeManager 请求分配具体的 Container。

需要注意的是，Container 是一个动态资源划分单位，是根据应用程序的需求动态生成的。目前为止，YARN 仅支持 CPU 和内存两种资源，且使用了轻量级资源隔离机制 Cgroups 进行资源隔离。

功能：

- 对task环境的抽象；
- 描述一系列信息；
- 任务运行资源的集合（cpu、内存、io等）；
- 任务运行环境

## 2.5 Resource Request 及 Container

### [引用连接](#)

Yarn的设计目标就是允许我们的各种应用以共享、安全、多租户的形式使用整个集群。并且，为了保证集群资源调度和数据访问的高效性，Yarn还必须能够感知整个集群拓扑结构。

为了实现这些目标，ResourceManager的调度器Scheduler为应用程序的资源请求定义了一些灵活的协议，通过它就可以对运行在集群中的各个应用做更好的调度，因此，这就诞生了**Resource Request**和**Container**。

一个应用先向ApplicationMaster发送一个满足自己需求的资源请求，然后ApplicationMaster把这个资源请求以resource-request的形式发送给ResourceManager的Scheduler，Scheduler再在这个原始的resource-request中返回分配到的资源描述Container。

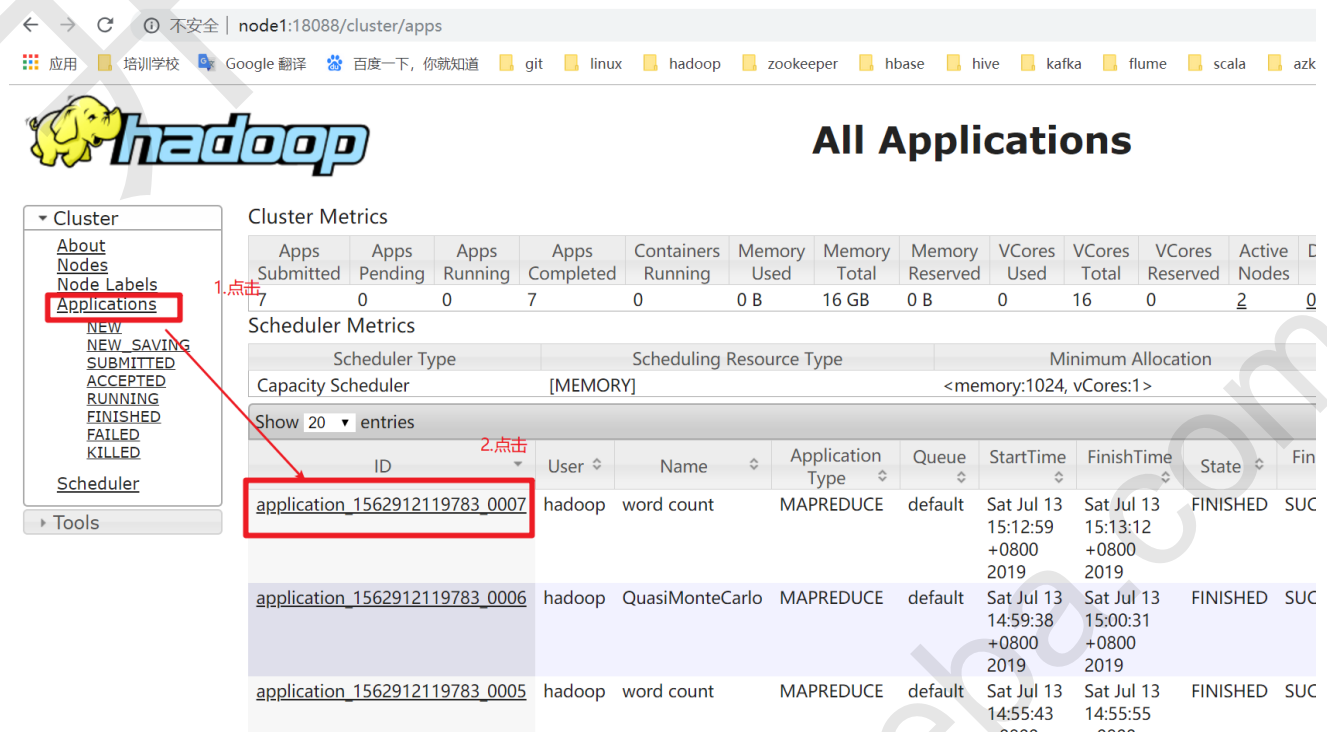
每个ResourceRequest可看做一个可序列化Java对象，包含的字段信息如下：

```
<!--
- resource-name : 资源名称，现阶段指的是资源所在的host和rack，后期可能还会支持虚拟机或者更复杂的网络结构
- priority : 资源的优先级
- resource-requirement : 资源的具体需求，现阶段指内存和cpu需求的数量
- number-of-containers : 满足需求的Container的集合
-->
<resource-name, priority, resource-requirement, number-of-containers>
```

## 2.6 JobHistoryServer

作业历史服务,记录在yarn中调度的作业历史运行情况,通过mr-jobhistory-daemon.sh start historyserver命令在集群中的数据节点机器上不需要做任何配置,单独使用命令启动直接启动即可,启动成功后会出现JobHistoryServer进程(使用jps命令查看,下面会有介绍),并且可以从19888端口进行查看日志详细信息

2.6.1 如果我们没有启动jobhistoryserver时我们运行一个mapreduce程序在yarn上看到什么呢？



The screenshot shows the Hadoop web interface at node1:18088/cluster/apps. The left sidebar has a 'Cluster' section with 'Applications' highlighted. The main content area shows 'All Applications' with a table of application metrics. The table has columns for App ID, User, Name, Application Type, Queue, Start Time, Finish Time, State, and Final Status. The first row is highlighted with a red box, showing 'application\_1562912119783\_0007' for user 'hadoop' and name 'word count'. A red arrow points from the 'Applications' menu item to this row. Another red arrow points from the 'Show 20 entries' dropdown to the same row.

App ID	User	Name	Application Type	Queue	Start Time	Finish Time	State	Final Status
application_1562912119783_0007	hadoop	word count	MAPREDUCE	default	Sat Jul 13 15:12:59 +0800 2019	Sat Jul 13 15:13:12 +0800 2019	FINISHED	SUC
application_1562912119783_0006	hadoop	QuasiMonteCarlo	MAPREDUCE	default	Sat Jul 13 14:59:38 +0800 2019	Sat Jul 13 15:00:31 +0800 2019	FINISHED	SUC
application_1562912119783_0005	hadoop	word count	MAPREDUCE	default	Sat Jul 13 14:55:43 +0800 2019	Sat Jul 13 14:55:55 +0800 2019	FINISHED	SUC

打开如下图界面，在下图中点击History，页面会进行一次跳转

node1:18088/cluster/app/application\_1562912119783\_0007

应用 培训学校 Google 翻译 百度一下, 你就知道 git linux hadoop zookeeper hbase hive kafka flume scala azkaban java

hadoop

Application  
application\_1562912119783\_0007

Cluster

- About
- Nodes
- Node Labels
- Applications
- NEW
- NEW\_SAVING
- SUBMITTED
- ACCEPTED
- RUNNING
- FINISHED
- FAILED
- KILLED
- Scheduler

Tools

Kill Application

Application Overview

User: hadoop

Name: word count

Application Type: MAPREDUCE

Application Tags:

YarnApplicationState: FINISHED

Queue: default

FinalStatus Reported by AM: SUCCEEDED

Started: Sat Jul 13 15:12:59 +0800 2019

Elapsed: 13sec

Tracking URL: [History](#)

Diagnostics:

Application Metrics

Total Resource Preempted: <memory:0, vCores:0>

Total Number of Non-AM Containers Preempted: 0

点击History之后 跳转后的页面如下图,是空白的, 这时因为我们没有启动jobhistoryserver所导致的。

node3:19888/jobhistory/job/job\_1562912119783\_0007

应用 培训学校 Google 翻译 百度一下, 你就知道 git linux hadoop zookeeper hbase hive kafka flume

2.6.2 在三台机器上执行mr-jobhistory-daemon.sh start historyserver命令依次启动jobhistoryserver。

在node1节点启动

```
[hadoop@node1 ~]$ mr-jobhistory-daemon.sh start historyserver
starting historyserver, logging to /opt/bigdata/hadoop-2.7.3/logs/ma
del.out
[hadoop@node1 ~]$ jps
104208 ResourceManager
104035 SecondaryNameNode
9799 Master
103820 NameNode
59565 Jps
59519 JobHistoryServer
[hadoop@node1 ~]$
```

启动后的进程

在node2节点启动



```
[hadoop@node2 ~]$ mr-jobhistory-daemon.sh start historyserver
starting historyserver, logging to /opt/bigdata/hadoop-2.7.3/logs/mapred
de2.out
[hadoop@node2 ~]$ jps
56800 Jps
101250 DataNode
101381 NodeManager
56758 JobHistoryServer
[hadoop@node2 ~]$
```

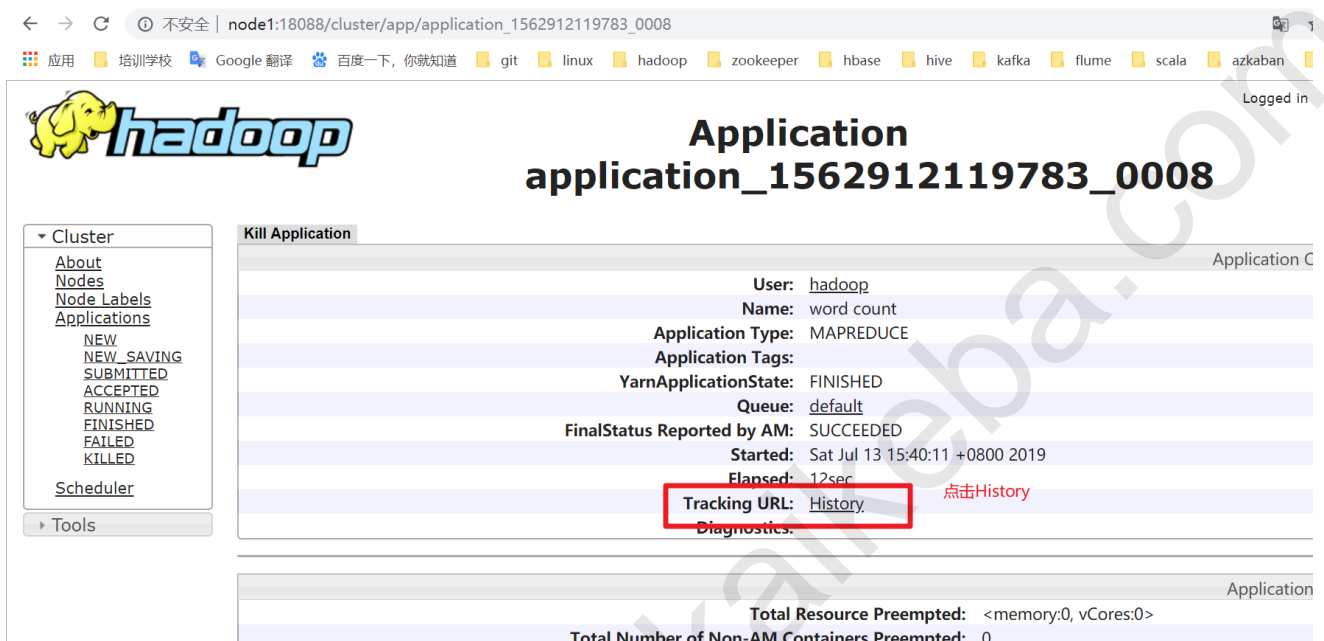
启动的进程

在node3节点启动

```
[hadoop@node3 ~]$ cd ~
[hadoop@node3 ~]$ jps
126420 DataNode
126568 NodeManager
108059 Jps
[hadoop@node3 ~]$ mr-jobhistory-daemon.sh start historyserver
starting historyserver, logging to /opt/bigdata/hadoop-2.7.3/logs/ma
de3.out
[hadoop@node3 ~]$ jps
126420 DataNode
108151 Jps
126568 NodeManager
108106 JobHistoryServer
[hadoop@node3 ~]$
```

启动后的进程

好，此时我们在三个节点把jobhistoryserver启动后，在此运行wordcount程序(记得启动前把输出目录删除掉)



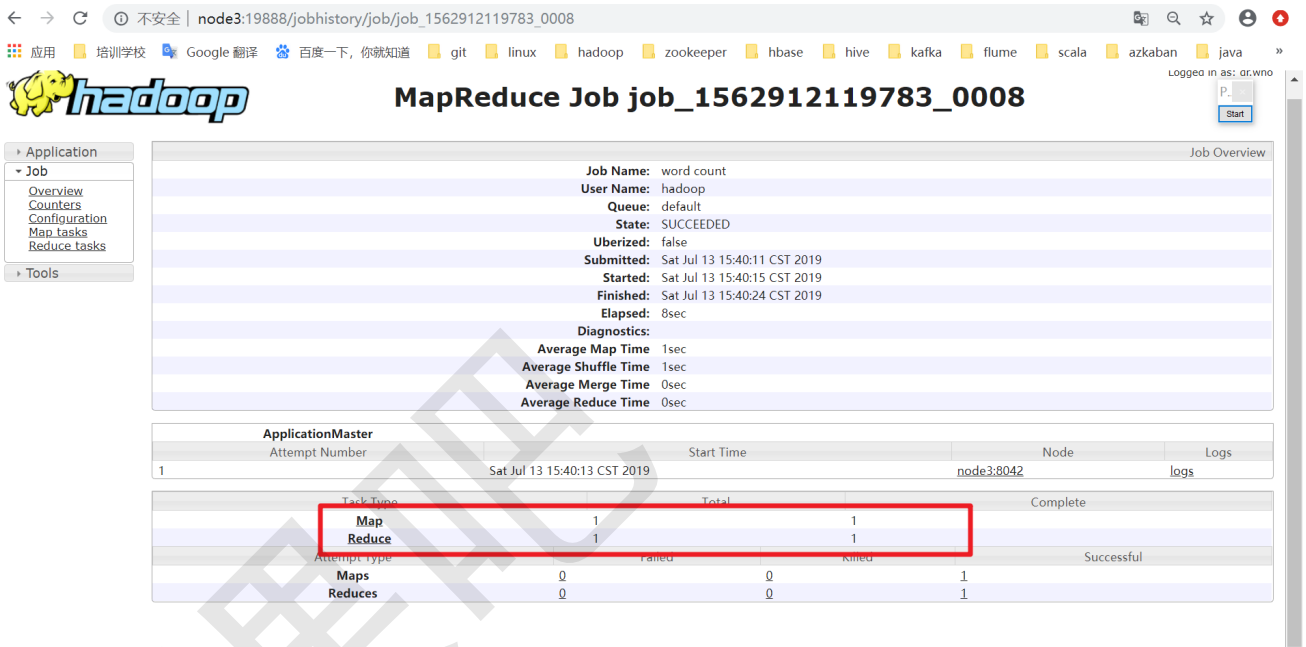
The screenshot shows the Hadoop Web UI for application `application_1562912119783_0008`. The interface includes a sidebar with navigation links like 'Cluster', 'About', 'Nodes', 'Node Labels', 'Applications', and 'Tools'. The main content area displays application details:

- User:** hadoop
- Name:** word count
- Application Type:** MAPREDUCE
- Application Tags:**
- YarnApplicationState:** FINISHED
- Queue:** default
- FinalStatus Reported by AM:** SUCCEEDED
- Started:** Sat Jul 13 15:40:11 +0800 2019
- Elapsed:** 12sec
- Tracking URL:** [History](#) (highlighted with a red box and labeled '点击History')
- Diagnostics:**

At the bottom, it shows resource information:

- Total Resource Preempted:** <memory:0, vCores:0>
- Total Number of Non-AM Containers Preempted:** 0

点击History连接会跳转一个全新的页面，在页面下方会看到TaskType中列举的map和reduce，Total表示此次运行的mapreduce程序执行所需要的map和reduce的任务数据。



MapReduce Job job\_1562912119783\_0008

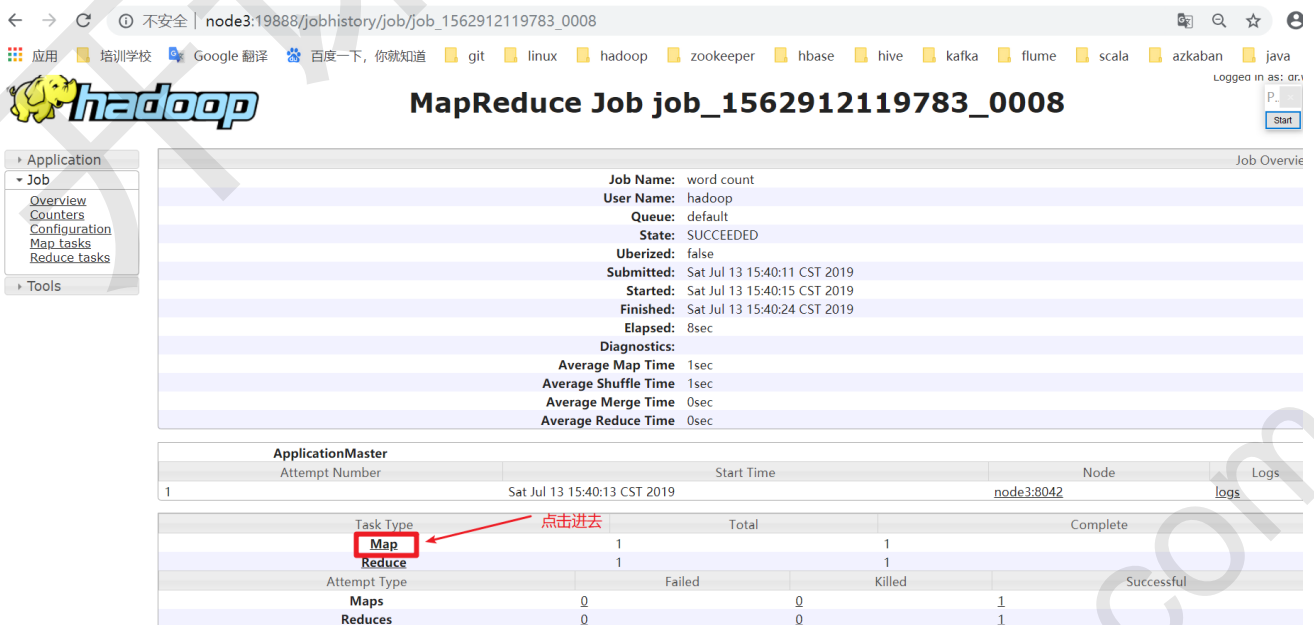
Job Name: word count  
 User Name: hadoop  
 Queue: default  
 State: SUCCEEDED  
 Uberized: false  
 Submitted: Sat Jul 13 15:40:11 CST 2019  
 Started: Sat Jul 13 15:40:15 CST 2019  
 Finished: Sat Jul 13 15:40:24 CST 2019  
 Elapsed: 8sec  
 Diagnostics:  
 Average Map Time: 1sec  
 Average Shuffle Time: 1sec  
 Average Merge Time: 0sec  
 Average Reduce Time: 0sec

Attempt Number	Start Time	Node	Logs
1	Sat Jul 13 15:40:13 CST 2019	node3:8042	logs

Task Type	Attempt	Failed	Killed	Complete
Map	1	0	0	1
Reduce	1	0	0	1

Attempt Type	Maps	Reduces
Maps	0	0
Reduces	0	0

如下图，我们在TaskType列中点击Map连接



MapReduce Job job\_1562912119783\_0008

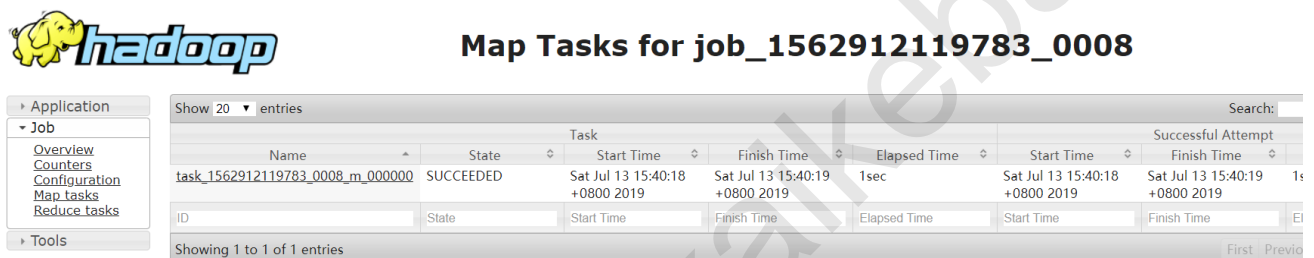
Job Name: word count  
 User Name: hadoop  
 Queue: default  
 State: SUCCEEDED  
 Uberized: false  
 Submitted: Sat Jul 13 15:40:11 CST 2019  
 Started: Sat Jul 13 15:40:15 CST 2019  
 Finished: Sat Jul 13 15:40:24 CST 2019  
 Elapsed: 8sec  
 Diagnostics:  
 Average Map Time: 1sec  
 Average Shuffle Time: 1sec  
 Average Merge Time: 0sec  
 Average Reduce Time: 0sec

Attempt Number	Start Time	Node	Logs
1	Sat Jul 13 15:40:13 CST 2019	node3:8042	logs

Task Type	Attempt	Failed	Killed	Complete
Map	1	0	0	1
Reduce	1	0	0	1

Attempt Type	Maps	Reduces
Maps	0	0
Reduces	0	0

如下图我们可以看到map任务的相关信息比如执行状态,启动时间，完成时间。



Map Tasks for job\_1562912119783\_0008

Name	State	Start Time	Finish Time	Elapsed Time	Start Time	Successful Attempt
task_1562912119783_0008_m_000000	SUCCEEDED	Sat Jul 13 15:40:18 +0800 2019	Sat Jul 13 15:40:19 +0800 2019	1sec	Sat Jul 13 15:40:18 +0800 2019	Sat Jul 13 15:40:19 +0800 2019

可以使用同样的方式我们查看reduce任务执行的详细信息，这里不再赘述。

从以上操作中我们可以看到jobhistoryserver就是进行作业运行过程中历史运行信息的记录，方便我们对作业进行分析。

## 2.7 Timeline Server

用来写日志服务数据，一般来写与第三方结合的日志服务数据(比如spark等),从官网的介绍看，它是对jobhistoryserver功能的有效补充，jobhistoryserver只能对mapreduce类型的作业信息进行记录，除了jobhistoryserver能够进行对作业运行过程中信息进行记录之外还有更细粒度的信息记录，比如任务在哪个队列中运行，运行任务时设置的用户是哪个用户。

根据官网的解释jobhistoryserver只能记录mapreduce应用程序的记录，timelineserver功能更强大,但不是替代jobhistory两者是功能间的互补关系。

### Persisting Generic Information about Completed Applications

Previously this was supported purely for MapReduce jobs by the Application History Server. With the introduction of the timeline server, the Application History Server becomes just one use of the Timeline Server.

Generic information includes application level data such as

- queue-name,
- user information and the like set in the ApplicationSubmissionContext,
- a list of application-attempts that ran for an application
- information about each application-attempt
- the list of containers run under each application-attempt
- information about each container.

Generic data is published by the YARN Resource Manager to the timeline store and used by its web-UI to display information about completed applications.

[官网教程](#)

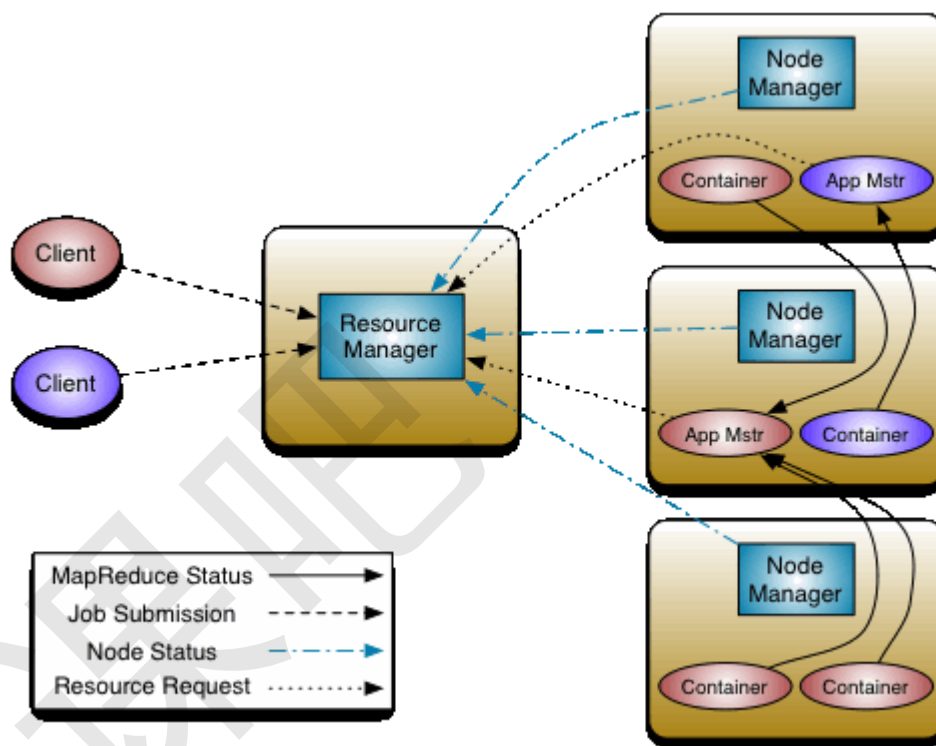
## 3. yarn应用运行原理

**YARN 是如何工作的？** YARN的基本理念是将JobTracker/TaskTracker 两大职能分割为以下几个实体：

1. 一个全局的资源管理ResourceManager
2. 每个应用程序一个ApplicationMaster
3. 每个从节点一个NodeManager
4. 每个应用程序一个运行在NodeManager上的Container

ResourceManager 和 NodeManager 组成了一个新的、通用的、用分布式管理应用程序的系统。

ResourceManager 对系统中的应用程序资源有终极仲裁的权限。ApplicationMaster 是一个特定于框架的实体，它的责任是同ResourceManager 谈判资源，同时为NodeManager(s)执行和监控组件任务。ResourceManager 有一个调度器，根据不同的约束条件，例如队列容量、用户限制等，将资源进行分配给各类运行着的应用程序。调度器执行调度功能是基于应用程序的资源申请。NodeManager 负责发布应用程序容器，监控资源的使用并向ResourceManager进行汇报。每个ApplicationMaster都有职责从调度器那谈判得到适当的资源容器，追踪它们的状态，并监控他们的进程。从系统的视图看，ApplicationMaster 作为一个普通的容器运行着。



### 3.1 yarn应用提交过程

Application在Yarn中的执行过程，整个执行过程可以总结为三步：

1. 应用程序提交
2. 启动应用的ApplicationMaster实例
3. ApplicationMaster 实例管理应用程序的执行

具体提交过程为：

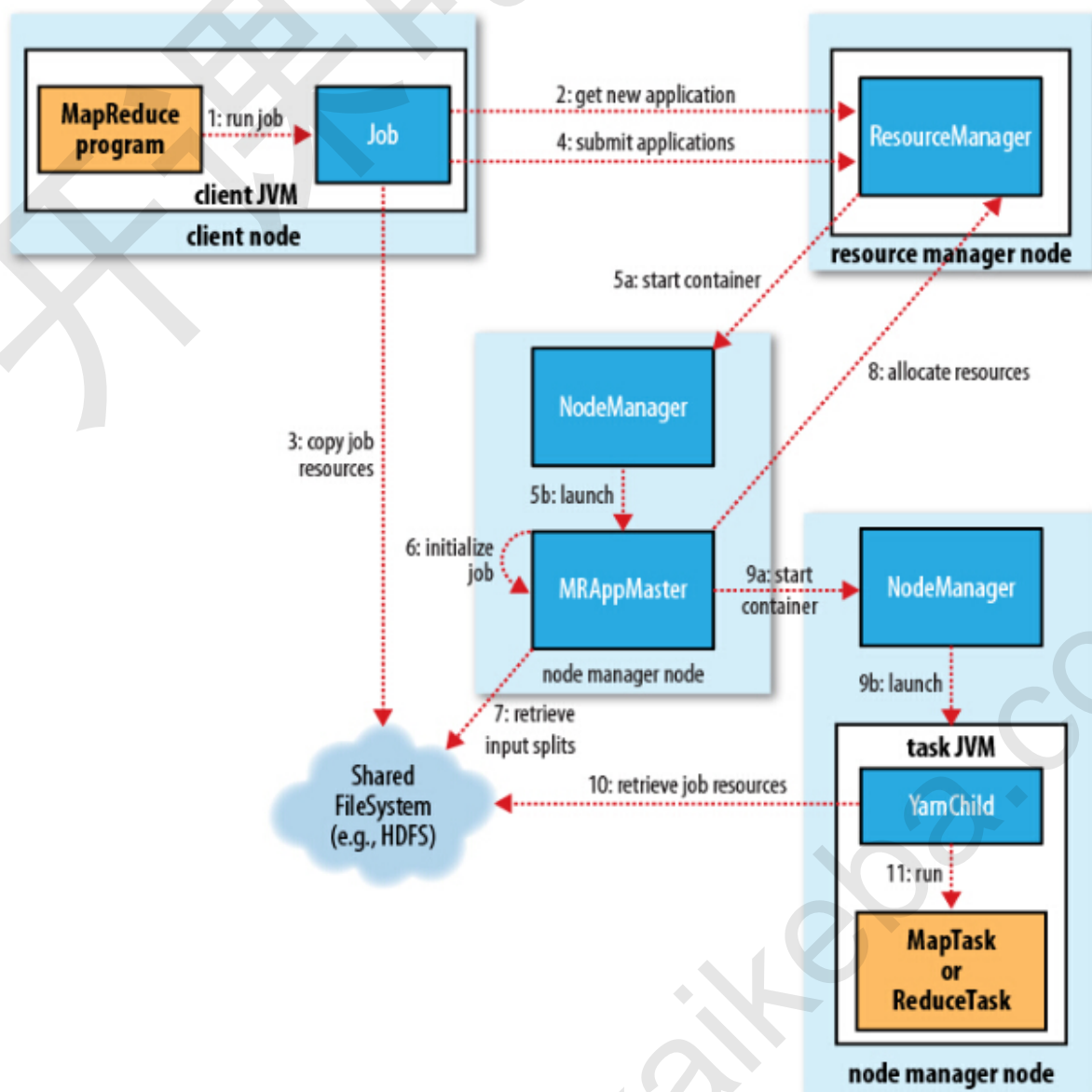
1. 客户端程序向 ResourceManager 提交应用并请求一个 ApplicationMaster 实例；
2. ResourceManager 找到一个可以运行一个 Container 的 NodeManager，并在这个 Container 中启动 ApplicationMaster 实例；
3. ApplicationMaster 向 ResourceManager 进行注册，注册之后客户端就可以查询 ResourceManager 获得自己 ApplicationMaster 的详细信息，以后就可以和自己的 ApplicationMaster 直接交互了（这个时候，客户端主动和 ApplicationMaster 交流，应用先向 ApplicationMaster 发送一个满足自己需求的资源请求）；
4. 在平常的操作过程中，ApplicationMaster 根据 resource-request协议 向 ResourceManager 发送 resource-request请求；
5. 当 Container 被成功分配后，ApplicationMaster 通过向 NodeManager 发送 container-launch-specification信息 来启动Container，container-launch-specification信息包含了能够让Container 和 ApplicationMaster 交流所需要的资料；
6. 应用程序的代码以 task 形式在启动的 Container 中运行，并把运行的进度、状态等信息通过 application-specific协议 发送给ApplicationMaster；
7. 在应用程序运行期间，提交应用的客户端主动和 ApplicationMaster 交流获得应用的运行状态、进度更新等信息，交流协议也是 application-specific协议；
8. 一旦应用程序执行完成并且所有相关工作也已经完成，ApplicationMaster 向 ResourceManager 取消注册然后关闭，用到所有的 Container 也归还给系统。



**精简版的：** 步骤1：用户将应用程序提交到 ResourceManager 上；步骤2：ResourceManager 为应用程序 ApplicationMaster 申请资源，并与某个 NodeManager 通信启动第一个 Container，以启动 ApplicationMaster；步骤3：ApplicationMaster 与 ResourceManager 注册进行通信，为内部要执行的任务申请资源，一旦得到资源后，将于 NodeManager 通信，以启动对应的 Task；步骤4：所有任务运行完成后，ApplicationMaster 向 ResourceManager 注销，整个应用程序运行结束。

## 3.2 mapreduce on yarn

MapReduce基于yarn的工作原理：我们通过提交jar包，进行MapReduce处理，那么整个运行过程分为五个环节：  
1、向client端提交MapReduce job. 2、随后yarn的ResourceManager进行资源的分配. 3、由NodeManager进行加载与监控containers. 4、通过applicationMaster与ResourceManager进行资源的申请及状态的交互，由NodeManagers进行MapReduce运行时job的管理. 5、通过hdfs进行job配置文件、jar包的各节点分发。



**Job 初始化过程** 1、当resourceManager收到了submitApplication()方法的调用通知后，scheduler开始分配container,随之ResourceManager发送applicationMaster进程，告知每个nodeManager管理器。 2、由applicationMaster决定如何运行tasks,如果job数据量比较小，applicationMaster便选择将tasks运行在一个JVM中。那么如何判别这个job是大是小呢？当一个job的mappers数量小于10个，只有一个reducer或者读取的文件大

小要小于一个HDFS block时, ( 可通过修改配置项

mapreduce.job.ubertask.maxmaps,mapreduce.job.ubertask.maxreduces以及

mapreduce.job.ubertask.maxbytes 进行调整) 3、在运行tasks之前, applicationMaster将会调用setupJob()方法, 随之创建output的输出路径(这就能够解释, 不管你的mapreduce一开始是否报错, 输出路径都会创建)

**Task 任务分配** 1、接下来applicationMaster向ResourceManager请求containers用于执行map与reduce的tasks ( step 8),这里map task的优先级要高于reduce task, 当所有的map tasks结束后, 随之进行sort(这里是shuffle过程后面再说),最后进行reduce task的开始。(这里有一点, 当map tasks执行了百分之5%的时候, 将会请求reduce, 具体下面再总结) 2、运行tasks的是需要消耗内存与CPU资源的, 默认情况下, map和reduce的task资源分配为1024MB与一个核, ( 可修改运行的最小与最大参数配

置,mapreduce.map.memory.mb,mapreduce.reduce.memory.mb,mapreduce.map.cpu.vcores,mapreduce.reduce.reduce.cpu.vcores.) **Task 任务执行** 1、这时一个task已经被ResourceManager分配到一个container

中, 由applicationMaster告知nodemanager启动container, 这个task将会被一个主函数为YarnChild的java application运行, 但在运行task之前, 首先定位task需要的jar包、配置文件以及加载在缓存中的文件。 2、YarnChild运行于一个专属的JVM中, 所以任何一个map或reduce任务出现问题, 都不会影响整个nodemanager的crash或者hang。 3、每个task都可以在相同的JVM task中完成, 随之将完成的处理数据写入临时文件中。

**Mapreduce数据流 运行进度与状态更新** 1、MapReduce是一个较长运行时间的批处理过程, 可以是一小时、几小时甚至几天, 那么Job的运行状态监控就非常重要。每个job以及每个task都有一个包含

job ( running,successfully completed,failed ) 的状态, 以及value的计数器, 状态信息及描述信息( 描述信息一般都是在代码中加的打印信息 ), 那么, 这些信息是如何与客户端进行通信的呢? 2、当一个task开始执行,

它将会保持运行记录, 记录task完成的比例, 对于map的任务, 将会记录其运行的百分比, 对于reduce来说可能复杂点, 但系统依旧会估计reduce的完成比例。当一个map或reduce任务执行时, 子进程会持续每三秒钟与applicationMaster进行交互。 Job 完成

### 3.3 yarn应用生命周期

- RM: Resource Manager
- AM: Application Master
- NM: Node Manager

1. Client向RM提交应用, 包括AM程序及启动AM的命令。
2. RM为AM分配第一个容器, 并与对应的NM通信, 令其在容器上启动应用的AM。
3. AM启动时向RM注册, 允许Client向RM获取AM信息然后直接和AM通信。
4. AM通过资源请求协议, 为应用协商容器资源。
5. 如容器分配成功, AM要求NM在容器中启动应用, 应用启动后可以和AM独立通信。
6. 应用程序在容器中执行, 并向AM汇报。
7. 在应用执行期间, Client和AM通信获取应用状态。
8. 应用执行完成, AM向RM注销并关闭, 释放资源。

**申请资源->启动appMaster->申请运行任务的container->分发Task->运行Task->Task结束->回收container**

## 4. 如何使用yarn

### 4.1 配置文件

```
<!-- $HADOOP_HOME/etc/hadoop/mapred-site.xml -->
<configuration>
  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>
</configuration>
```

```
<!-- $HADOOP_HOME/etc/hadoop/yarn-site.xml -->
<configuration>
  <property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
  </property>
</configuration>
```

## 4.2 yarn启动停止

启动 ResourceManager 和 NodeManager ( 以下分别简称RM、NM )

```
#主节点运行命令
$HADOOP_HOME/sbin/start-yarn.sh
```

停止 RM 和 NM

```
#主节点运行命令
$HADOOP_HOME/sbin/stop-yarn.sh
```

若RM没有启动起来，可以单独启动

```
#若RM没有启动，在主节点运行命令
$HADOOP_HOME/sbin/yarn-daemon.sh start resourcemanager
#相反，可单独关闭
$HADOOP_HOME/sbin/yarn-daemon.sh stop resourcemanager
```

若NM没有启动起来，可以单独启动

```
#若NM没有启动，在相应节点运行命令
$HADOOP_HOME/sbin/yarn-daemon.sh start nodemanager
#相反，可单独关闭
$HADOOP_HOME/sbin/yarn-daemon.sh stop nodemanager
```

## 4.3 yarn常用命令

### 4.3.1 yarn命令列表

```
[bruce@node-01 ~]$ yarn
Usage: yarn [--config confdir] [COMMAND | CLASSNAME]
  CLASSNAME                                run the class named CLASSNAME
or
where COMMAND is one of:
  resourcemanager -format-state-store      deletes the RMStateStore
  resourcemanager                          run the ResourceManager
  nodemanager                             run a nodemanager on each slave
  timelineserver                          run the timeline server
  rmadmin                                 admin tools
  sharedcachemanager                      run the SharedCacheManager daemon
  scmadmin                               SharedCacheManager admin tools
  version                                print the version
  jar <jar>                             run a jar file
  application                             prints application(s)
                                          report/kill application
  applicationattempt                     prints applicationattempt(s)
                                          report
  container                             prints container(s) report
  node                                  prints node report(s)
  queue                                prints queue information
  logs                                 dump container logs
  classpath                             prints the class path needed to
                                          get the Hadoop jar and the
                                          required libraries
  cluster                               prints cluster information
  daemonlog                             get/set the log level for each
                                          daemon
```

### 4.3.2 yarn application命令

```
[bruce@node-01 ~]$ yarn application
19/07/16 22:23:30 INFO client.RMPProxy: Connecting to ResourceManager at node-01/192.168.197.128:18040
Invalid Command Usage :
usage: application
  -appStates <States>                  Works with -list to filter applications
                                          based on input comma-separated list of
                                          application states. The valid application
                                          state can be one of the following:
                                          ALL,NEW,NEW_SAVING,SUBMITTED,ACCEPTED,RUN
                                          NING,FINISHED,FAILED,KILLED
  -appTypes <Types>                    Works with -list to filter applications
                                          based on input comma-separated list of
                                          application types.
  -help                                Displays help for all commands.
  -kill <Application ID>               Kills the application.
  -list                                List applications. Supports optional use
                                          of -appTypes to filter applications based
                                          on application type, and -appStates to
                                          filter applications based on application
                                          state.
  -movetoqueue <Application ID>        Moves the application to a different
                                          queue.
  -queue <Queue Name>                  Works with the movetoqueue command to
                                          specify which queue to move an
                                          application to.
  -status <Application ID>             Prints the status of the application.
```

#### #1. 查看正在运行的任务

```
yarn application -list
```

#### #2. 杀掉正在运行任务

```
yarn application -kill 任务id
```



### #3. 查看节点列表

```
yarn node -list
```

```
[bruce@node-01 ~]$ yarn node -list
19/07/16 22:51:33 INFO client.RMPProxy: Connecting to ResourceManager at node-01/192.168.197.128:18040
Total Nodes:3
      Node-Id          Node-State Node-Http-Address      Number-of-Running-Containers
node-02:46574         RUNNING   node-02:8042             0
node-03:45899         RUNNING   node-03:8042             0
node-01:44909         RUNNING   node-01:8042             0
[bruce@node-01 ~]$
```

### #4. 查看节点状况 TODO

```
yarn node -status node3:40652
```

### #5. 查看yarn依赖jar的环境变量

```
yarn classpath
```

## 5. yarn调度器

试想一下，你现在所在的公司有一个hadoop的集群。但是A项目组经常做一些定时的BI报表，B项目组则经常使用一些软件做一些临时需求。那么他们肯定会遇到同时提交任务的场景，这个时候到底如何分配资源满足这两个任务呢？是先执行A的任务，再执行B的任务，还是同时跑两个？

如果你存在上述的困惑，可以多了解一些yarn的资源调度器。

在Yarn框架中，调度器是一块很重要的内容。有了合适的调度规则，就可以保证多个应用可以在同一时间有条不紊的工作。最原始的调度规则就是FIFO，即按照用户提交任务的时间来决定哪个任务先执行，先提交的先执行。但是这样很可能一个大任务独占资源，其他的资源需要不断的等待。也可能一堆小任务占用资源，大任务一直无法得到适当的资源，造成饥饿。所以FIFO虽然很简单，但是并不能满足我们的需求。

理想情况下，yarn应用发出的资源请求应该立刻给予满足。然而现实中的资源有限，在一个繁忙的集群上，一个应用经常需要等待才能得到所需的资源。yarn调度器的工作就是根据既定的策略为应用分配资源。调度通常是一个难题，并且没有一个所谓的“最好”的策略，这也是为什么yarn提供了多重调度器和可配置策略供我们选择的原因。

**yarn分为一级调度管理和二级调度管理** 一级调度管理(更近底层,更接近于操作资源,更偏向于应用层和底层结合) 计算资源管理(cpu,内存等,计算复杂消耗的cpu多) App生命周期管理 二级调度管理(自己代码的算法等,更偏向于应用层) App内部的计算模型管理 多样化的计算模型

### 5.1 调度器

在Yarn中有三种调度器可以选择：FIFO Scheduler，Capacity Scheduler，FairS cheduler

### 5.2 FIFO Scheduler

FIFO Scheduler把应用按提交的顺序排成一个队列，这是一个先进先出队列，在进行资源分配的时候，先给队列中最头上的应用进行分配资源，待最头上的应用需求满足后再给下一个分配，以此类推。

FIFO Scheduler是最简单也是最容易理解的调度器，也不需要任何配置，但它并不适用于共享集群。大的应用可能会占用所有集群资源，这就导致其它应用被阻塞。在共享集群中，更适合采用Capacity Scheduler或Fair Scheduler，这两个调度器都允许大任务和小任务在提交的同时获得一定的系统资源。

下面“Yarn调度器对比图”展示了这几个调度器的区别，从图中可以看出，在FIFO 调度器中，小任务会被大任务阻塞。

## 5.3 Capacity Scheduler

而对于Capacity调度器，有一个专门的队列用来运行小任务，但是为小任务专门设置一个队列会预先占用一定的集群资源，这就导致大任务的执行时间会落后于使用FIFO调度器时的时间。

如何配置容量调度器

队列层级结构如下：

```
graph TD
    root --> prod
    root --> dev
    dev --> spark
    dev --> hdp
```

将HADOOP\_HOME/etc/hadoop/中的对应capacity - scheduler.xml配置文件备份到其它目录，在目录HADOOP\_HOME/etc/hadoop/中建立一个新的capacity-scheduler.xml；内容如下

```
<?xml version="1.0" encoding="utf-8"?>

<configuration>
  <property>
    <name>yarn.scheduler.capacity.root.queues</name>
    <value>prod,dev</value>
  </property>
  <property>
    <name>yarn.scheduler.capacity.root.dev.queues</name>
    <value>hdp,spark</value>
  </property>
  <property>
    <name>yarn.scheduler.capacity.root.prod.capacity</name>
    <value>40</value>
  </property>
  <property>
    <name>yarn.scheduler.capacity.root.dev.capacity</name>
    <value>60</value>
  </property>
  <property>
    <name>yarn.scheduler.capacity.root.dev.maximum-capacity</name>
    <value>75</value>
  </property>
  <property>
    <name>yarn.scheduler.capacity.root.dev.hdp.capacity</name>
    <value>50</value>
  </property>
</configuration>
```

```
<property>
  <name>yarn.scheduler.capacity.root.dev.spark.capacity</name>
  <value>50</value>
</property>
</configuration>
```

将应用放置在哪个队列中，取决于应用本身。

例如MR，可以通过设置属性mapreduce.job.queueName指定相应队列。以WordCount为例，如下

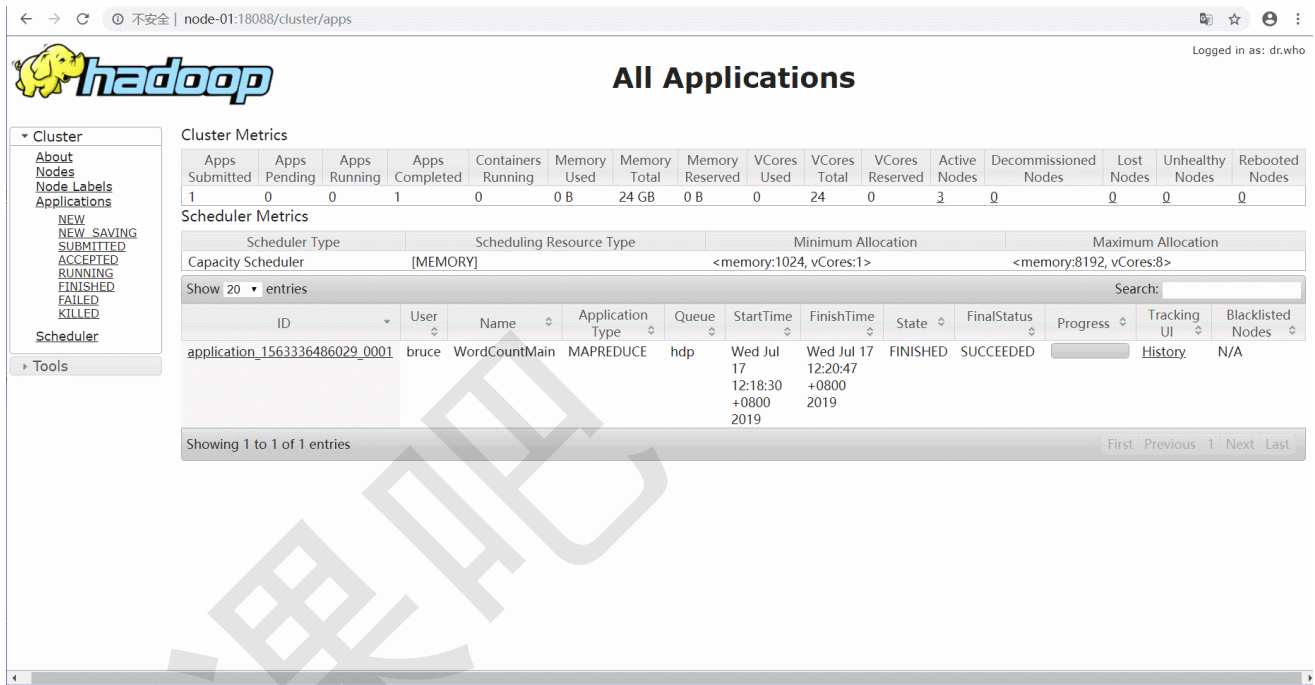
如果指定的队列不存在，则发生错误。如果不指定，默认使用"default"队列，如下图

https://hadoop.apache.org/docs/r2.7.3/hadoop-mapreduce-client/hadoop-mapreduce-client-core/mapred-default.xml	
mapreduce.reduce.skip.maxgroups	0
mapreduce.ifile.readahead	true
mapreduce.ifile.readahead.bytes	4194304
mapreduce.jobtracker.taskcache.levels	2
mapreduce.job.queueName	default

```
14 public class WordCountMain {
15     // 若在IDEA中本地执行MR程序，需要将mapred-site.xml中的mapreduce.framework.name值修改成local
16     public static void main(String[] args) throws IOException,
17         ClassNotFoundException, InterruptedException {
18         if (args.length != 2 || args == null) {
19             System.out.println("please input Path!");
20             System.exit(status: 0);
21         }
22
23         Configuration configuration = new Configuration();
24         //attention: 如果指定在队列不存在，则报错
25         configuration.set("mapreduce.job.queueName", "hdp");
26         //configuration.set("mapreduce.job.jar", "/home/bruce/project/kkbhdp01/target/com.kaika.hadoop
27
28         // 调用getInstance方法，生成job实例
29         Job job = Job.getInstance(configuration, WordCountMain.class.getSimpleName());
30     }
```

程序打包，提交集群运行

```
[bruce@node-01 Desktop]$ hadoop jar com.kaika.hadoop-1.0-SNAPSHOT.jar
com.kaika.hadoop.scheduler.WordCountMain /sogou.500w.utf8 /0717scheduler02
```



The screenshot shows the Hadoop web interface for 'All Applications'. The top navigation bar includes links for Cluster, About, Nodes, Node Labels, Applications, NEW, NEW SAVING, SUBMITTED, ACCEPTED, RUNNING, FINISHED, FAILED, KILLED, Scheduler, and Tools. The main content area displays Cluster Metrics, Scheduler Metrics, and a table of application entries.

**Cluster Metrics**

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved	Active Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes
1	0	0	1	0	0 B	24 GB	0 B	0	24	0	3	0	0	0	0

**Scheduler Metrics**

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation
Capacity Scheduler	[MEMORY]	<memory:1024, vCores:1>	<memory:8192, vCores:8>

**Application Entries**

ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI	Blacklisted Nodes
application_1563336486029_0001	bruce	WordCountMain	MAPREDUCE	hdp	Wed Jul 17 12:18:30 +0800 2019	Wed Jul 17 12:20:47 +0800 2019	FINISHED	SUCCEEDED		History	N/A

Showing 1 to 1 of 1 entries

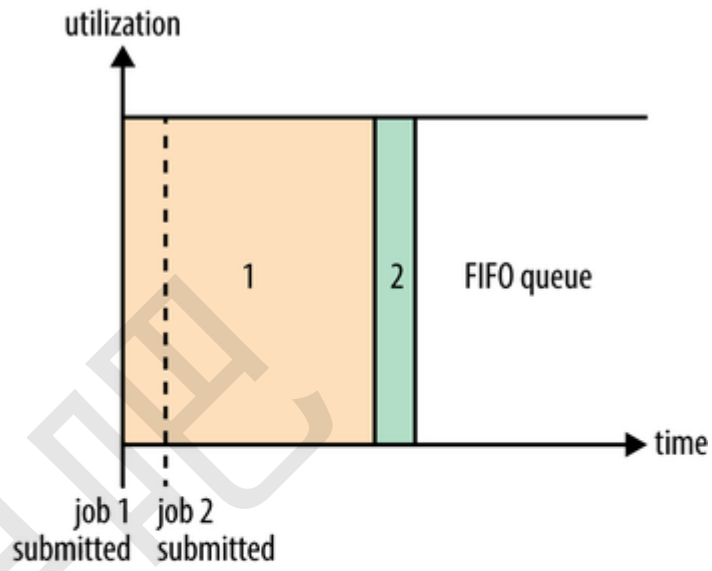
## 5.4 Fair Scheduler

在Fair调度器中，我们不需要预先占用一定的系统资源，Fair调度器会为所有运行的job动态的调整系统资源。如下图所示，当第一个大job提交时，只有这一个job在运行，此时它获得了所有集群资源；当第二个小任务提交后，Fair调度器会分配一半资源给这个小任务，让这两个任务公平的共享集群资源。

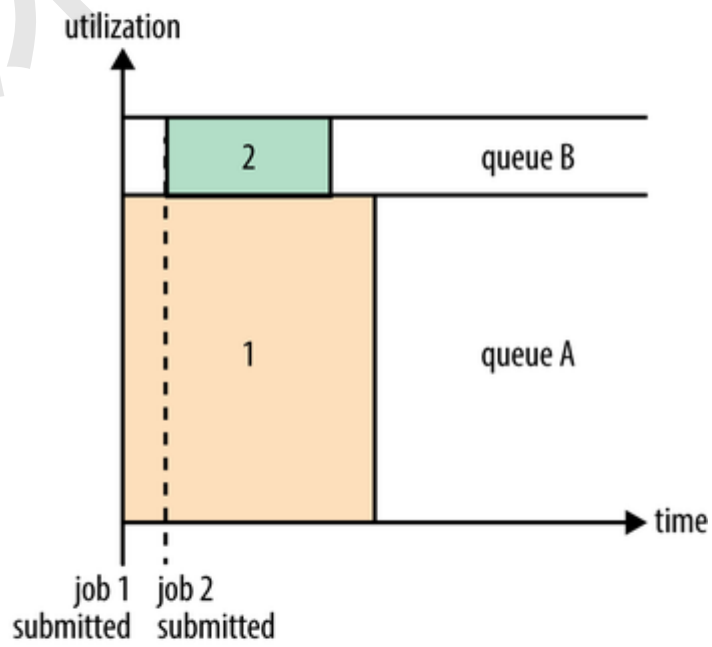
需要注意的是，在下图Fair调度器中，从第二个任务提交到获得资源会有一定的延迟，因为它需要等待第一个任务释放占用的Container。小任务执行完成之后也会释放自己占用的资源，大任务又获得了全部的系统资源。最终的效果就是Fair调度器即得到了高的资源利用率又能保证小任务及时完成。



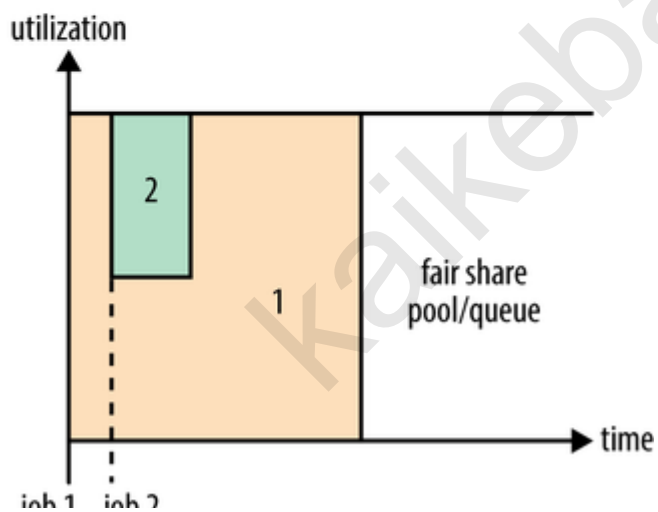
### i. FIFO Scheduler



### ii. Capacity Scheduler



### iii. Fair Scheduler

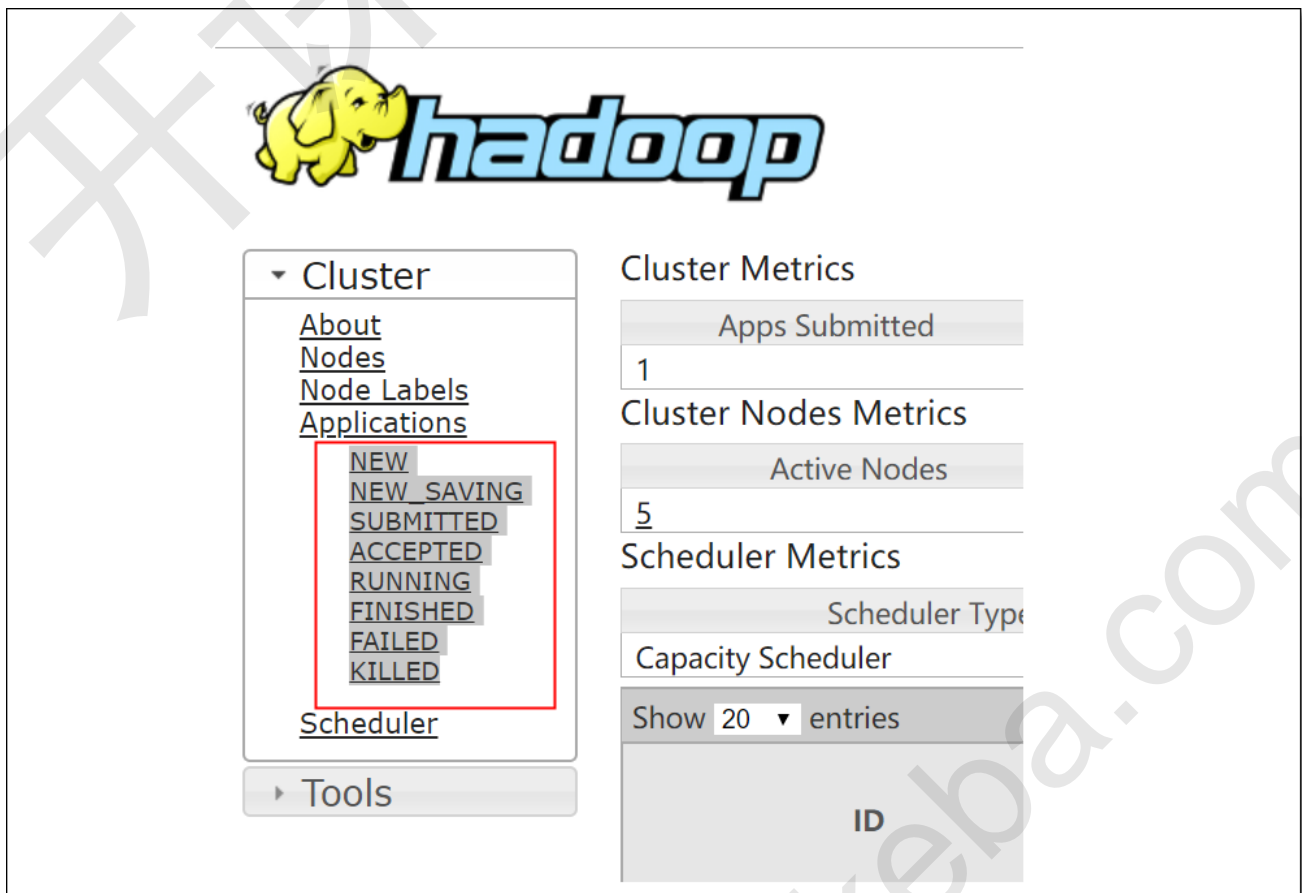


JOB 1    JOB 2  
submitted    submitted

## 6. yarn应用状态

我们在yarn 的web ui上能够看到yarn 应用程序分为如下几个状态:

- NEW ----新建状态
- NEW\_SAVING----新建保存状态
- SUBMITTED----提交状态
- ACCEPTED----接受状态
- RUNNING----运行状态
- FINISHED----完成状态
- FAILED----失败状态
- KILLED----杀掉状态



## 7. yarn调优

关于Yarn内存分配与管理，主要涉及到了ResourceManage、ApplicationMatser、NodeManager这几个概念，相关的优化也要紧紧围绕着这几方面来开展。这里还有一个Container的概念，现在可以先把它理解为运行map/reduce task的容器，后面有详细介绍。

## 7.1 RM的内存资源配置, 配置的是资源调度相关

RM1 : yarn.scheduler.minimum-allocation-mb 分配给AM单个容器可申请的最小内存 RM2 :  
yarn.scheduler.maximum-allocation-mb 分配给AM单个容器可申请的最大内存 注 :

最小值可以计算一个节点最大Container数量 一旦设置, 不可动态改变

## 7.2 NM的内存资源配置, 配置的是硬件资源相关

NM1 : yarn.nodemanager.resource.memory-mb 节点最大可用内存 NM2 : yarn.nodemanager.vmem-pmem-ratio 虚拟内存率, 默认2.1 注 :

RM1、RM2的值均不能大于NM1的值 NM1可以计算节点最大最大Container数量,  $\max(\text{Container}) = \text{NM1} / \text{RM1}$  一旦设置, 不可动态改变

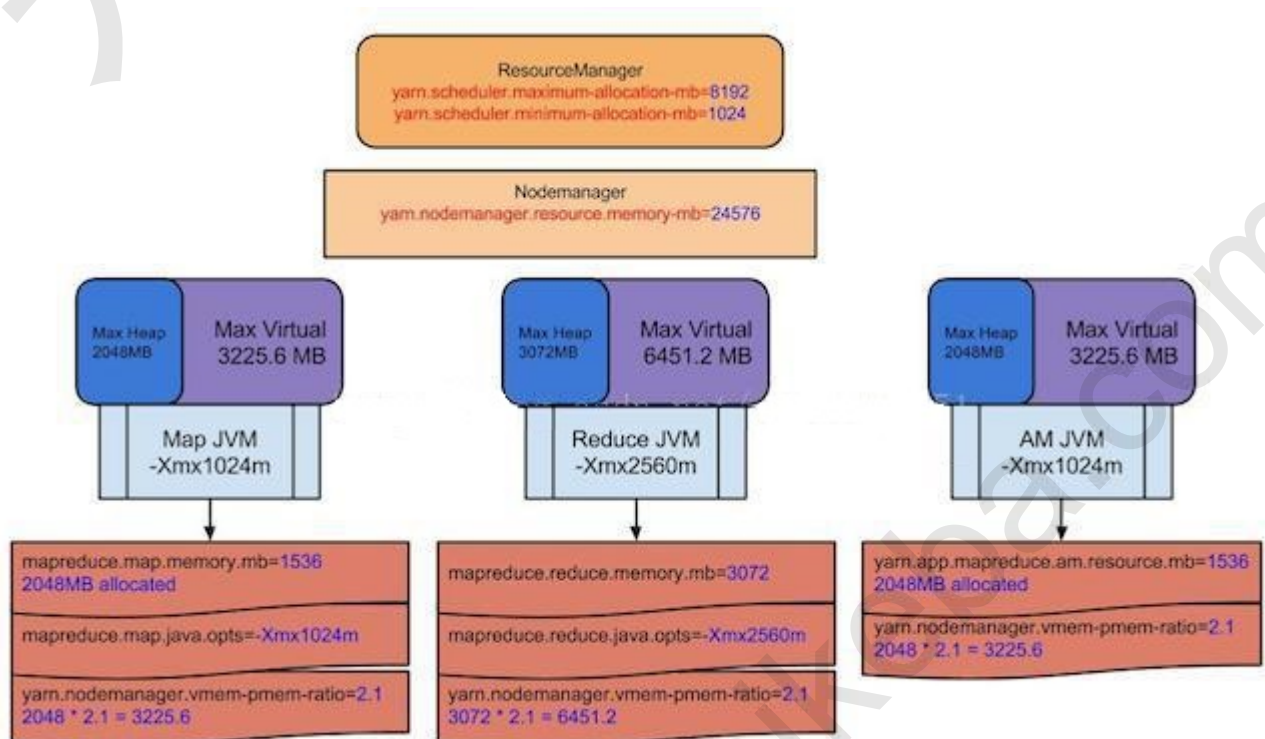
## 7.3 AM内存配置相关参数, 配置的是任务相关

AM1 : mapreduce.map.memory.mb 分配给map Container的内存大小 AM2 : mapreduce.reduce.memory.mb 分配给reduce Container的内存大小

这两个值应该在RM1和RM2这两个值之间 AM2的值最好为AM1的两倍 这两个值可以在启动时改变

AM3 : mapreduce.map.java.opts 运行map任务的jvm参数, 如-Xmx, -Xms等选项 AM4 :  
mapreduce.reduce.java.opts 运行reduce任务的jvm参数, 如-Xmx, -Xms等选项 注 :

这两个值应该在AM1和AM2之间

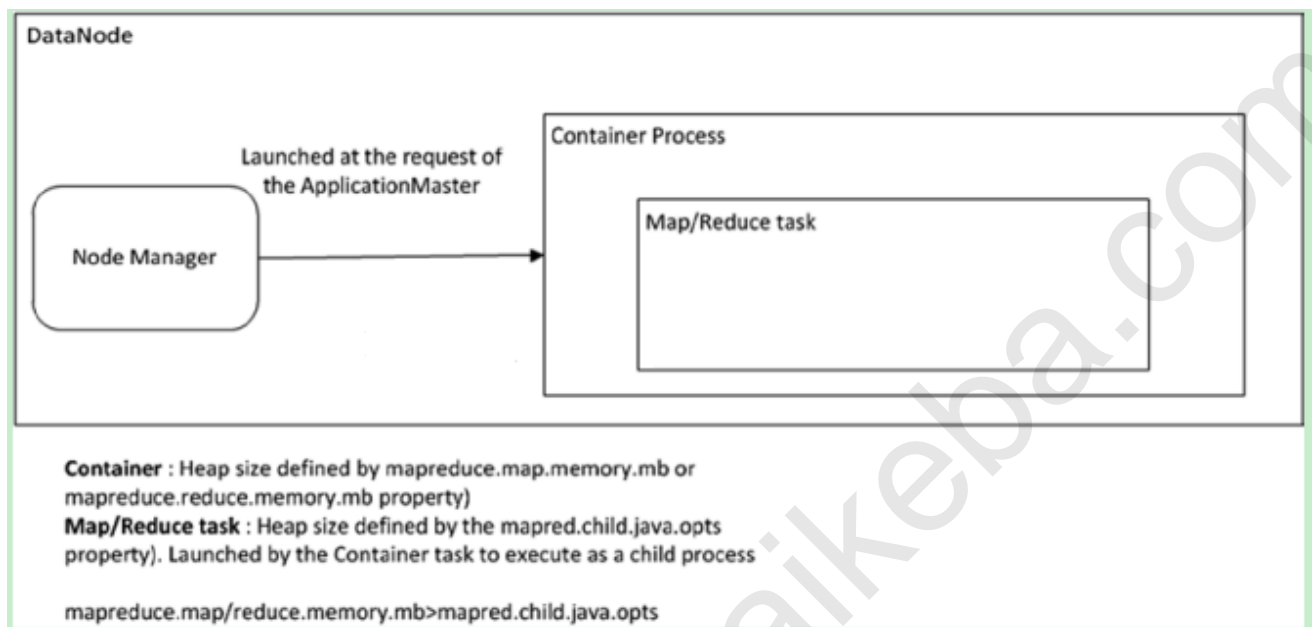


如上图所示, 先看最下面褐色部分, AM参数mapreduce.map.memory.mb=1536MB, 表示AM要为map Container申请1536MB资源, 但RM实际分配的内存却是2048MB, 因为yarn.scheduler.mininum-allocation-mb=1024MB, 这定义了RM最小要分配1024MB, 1536MB超过了这个值, 所以实际分配给AM的值为2048MB(这涉及到了规整化因子, 关于规整化因子, 在本文最后有介绍)。AM参数mapreduce.map.java.opts=-Xmx

1024m，表示运行map任务的jvm内存为1024MB，因为map任务要运行在Container里面，所以这个参数的值略微小于mapreduce.map.memory.mb=1536MB这个值。NM参数yarn.nodemanager.vmem-pmem-ratio=2.1，这表示NodeManager可以分配给map/reduce Container 2.1倍的虚拟内存，按照上面的配置，实际分配给map Container容器的虚拟内存大小为2048\*2.1=3225.6MB，若实际用到的内存超过这个值，NM就会kill掉这个map Container，任务执行过程就会出现异常。AM参数mapreduce.reduce.memory.mb=3072MB，表示分配给reduce Container的容器大小为3072MB，而map Container的大小分配的是1536MB，从这也看出，reduce Container容器的大小最好是map Container大小的两倍。NM参数yarn.nodemanager.resource.mem.mb=24576MB，这个值表示节点分配给NodeManager的可用内存，也就是节点用来执行yarn任务的内存大小。这个值要根据实际服务器内存大小来配置，比如我们hadoop集群机器内存是128GB，我们可以分配其中的80%给yarn，也就是102GB。上图中RM的两个参数分别1024MB和8192MB，分别表示分配给AM map/reduce Container的最大值和最小值。

## 7.4 关于Container

(1) Container是YARN中资源的抽象，它封装了某个节点上一定量的资源（CPU和内存两类资源）。它跟Linux Container没有任何关系，仅仅是YARN提出的一个概念（从实现上看，可看做一个可序列化/反序列化的Java类）。(2) Container由ApplicationMaster向ResourceManager申请的，由ResourceManager中的资源调度器异步分配给ApplicationMaster；(3) Container的运行是由ApplicationMaster向资源所在的NodeManager发起的，Container运行时需提供内部执行的任务命令（可以使任何命令，比如java、Python、C++进程启动命令均可）以及该命令执行所需的环境变量和外部资源（比如词典文件、可执行文件、jar包等）。另外，一个应用程序所需的Container分为两大类，如下：(1) 运行ApplicationMaster的Container：这是由ResourceManager（向内部的资源调度器）申请和启动的，用户提交应用程序时，可指定唯一的ApplicationMaster所需的资源；(2) 运行各类任务的Container：这是由ApplicationMaster向ResourceManager申请的，并由ApplicationMaster与NodeManager通信以启动之。以上两类Container可能在任意节点上，它们的位置通常而言是随机的，即ApplicationMaster可能与它管理的任务运行在一个节点上。Container是YARN中最重要的概念之一，懂得该概念对于理解YARN的资源模型至关重要，望大家好好理解。注意：如下图，map/reduce task是运行在Container之中的，所以上面提到的mapreduce.map(reduce).memory.mb大小都大于mapreduce.map(reduce).java.opts值的大小。



## 五、拓展点、未来计划、行业趋势（5分钟）



1. 查看官网capacity scheduler内容
2. [capacity scheduler参考资料](#)

## 六、总结（5分钟）

---

1. 介绍了yarn的应用场景
2. yarn的核心组件
3. yarn应用调度过程
4. yarn的典型应用

## 七、作业

---

## 八、互动问答

---

## 九、题库 - 本堂课知识点

---