## Yarn通俗介绍

Apache Hadoop YARN （Yet Another Resource Negotiator，另一种资源协调者）是一种新的 Hadoop 资源管理器，它是一个通用资源管理系统和调度平台，可为上层应用提供统一的资源管理和调度，它的引入为集群在利用率、资源统一管理和数据共享等方面带来了巨大好处。

可以把yarn理解为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，Yarn为这些程序提供运算所需的资源（内存、cpu）。

* yarn并不清楚用户提交的程序的运行机制
* yarn只提供运算资源的调度（用户程序向yarn申请资源，yarn就负责分配资源）
* yarn中的主管角色叫ResourceManager
* yarn中具体提供运算资源的角色叫NodeManager
* yarn与运行的用户程序完全解耦，意味着yarn上可以运行各种类型的分布式运算程序，比如mapreduce、storm，spark，tez ……
* spark、storm等运算框架都可以整合在yarn上运行，只要他们各自的框架中有符合yarn规范的资源请求机制即可
* yarn成为一个通用的资源调度平台.企业中以前存在的各种运算集群都可以整合在一个物理集群上，提高资源利用率，方便数据共享

## Yarn基本架构

YARN是一个资源管理、任务调度的框架，主要包含三大模块：ResourceManager（RM）、NodeManager（NM）、ApplicationMaster（AM）。

ResourceManager负责所有资源的监控、分配和管理；

ApplicationMaster负责每一个具体应用程序的调度和协调；

NodeManager负责每一个节点的维护。

对于所有的applications，RM拥有绝对的控制权和对资源的分配权。而每个AM则会和RM协商资源，同时和NodeManager通信来执行和监控task。

## Yarn三大组件介绍

### ResourceManager

* ResourceManager负责整个集群的资源管理和分配，是一个全局的资源管理系统。
* NodeManager以心跳的方式向ResourceManager汇报资源使用情况（目前主要是CPU和内存的使用情况）。RM只接受NM的资源回报信息，对于具体的资源处理则交给NM自己处理。
* YARN Scheduler根据application的请求为其分配资源，不负责application job的监控、追踪、运行状态反馈、启动等工作。

### NodeManager

* NodeManager是每个节点上的资源和任务管理器，它是管理这台机器的代理，负责该节点程序的运行，以及该节点资源的管理和监控。YARN集群每个节点都运行一个NodeManager。
* NodeManager定时向ResourceManager汇报本节点资源（CPU、内存）的使用情况和Container的运行状态。当ResourceManager宕机时NodeManager自动连接RM备用节点。
* NodeManager接收并处理来自ApplicationMaster的Container启动、停止等各种请求。

### ApplicationMaster

* 用户提交的每个应用程序均包含一个ApplicationMaster，它可以运行在ResourceManager以外的机器上。
* 负责与RM调度器协商以获取资源（用Container表示）。
* 将得到的任务进一步分配给内部的任务(资源的二次分配)。
* 与NM通信以启动/停止任务。
* 监控所有任务运行状态，并在任务运行失败时重新为任务申请资源以重启任务。

## Yarn运行流程

* client向RM提交应用程序，其中包括启动该应用的ApplicationMaster的必须信息，例如ApplicationMaster程序、启动ApplicationMaster的命令、用户程序等。
* ResourceManager启动一个container用于运行ApplicationMaster。
* 启动中的ApplicationMaster向ResourceManager注册自己，启动成功后与RM保持心跳。
* ApplicationMaster向ResourceManager发送请求，申请相应数目的container。
* ResourceManager返回ApplicationMaster的申请的containers信息。申请成功的container，由ApplicationMaster进行初始化。container的启动信息初始化后，AM与对应的NodeManager通信，要求NM启动container。AM与NM保持心跳，从而对NM上运行的任务进行监控和管理。
* container运行期间，ApplicationMaster对container进行监控。container通过RPC协议向对应的AM汇报自己的进度和状态等信息。
* 应用运行期间，client直接与AM通信获取应用的状态、进度更新等信息。
* 应用运行结束后，ApplicationMaster向ResourceManager注销自己，并允许属于它的container被收回。

## Yarn的调度器Scheduler

理想情况下，我们应用对Yarn资源的请求应该立刻得到满足，但现实情况资源往往是有限的，特别是在一个很繁忙的集群，一个应用资源的请求经常需要等待一段时间才能的到相应的资源。在**Yarn中，负责给应用分配资源的就是Scheduler**。其实调度本身就是一个难题，很难找到一个完美的策略可以解决所有的应用场景。为此，Yarn提供了多种调度器和可配置的策略供我们选择。

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

### 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。

### Capacity Scheduler

Capacity 调度器允许多个组织共享整个集群，每个组织可以获得集群的一部分计算能力。通过为每个组织分配专门的队列，然后再为每个队列分配一定的集群资源，这样整个集群就可以通过设置多个队列的方式给多个组织提供服务了。除此之外，队列内部又可以垂直划分，这样一个组织内部的多个成员就可以共享这个队列资源了，在一个队列内部，资源的调度是采用的是先进先出(FIFO)策略。

容量调度器 Capacity Scheduler 最初是由 Yahoo 最初开发设计使得 Hadoop 应用能够被多用户使用，且最大化整个集群资源的吞吐量，现被 IBM BigInsights 和 Hortonworks HDP 所采用。

Capacity Scheduler 被设计为允许应用程序在一个可预见的和简单的方式共享集群资源，即"作业队列"。Capacity Scheduler 是根据租户的需要和要求把现有的资源分配给运行的应用程序。Capacity Scheduler 同时允许应用程序访问还没有被使用的资源，以确保队列之间共享其它队列被允许的使用资源。管理员可以控制每个队列的容量，Capacity Scheduler 负责把作业提交到队列中。

### 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。

公平调度器 Fair Scheduler 最初是由 Facebook 开发设计使得 Hadoop 应用能够被多用户公平地共享整个集群资源，现被 Cloudera CDH 所采用。

Fair Scheduler 不需要保留集群的资源，因为它会动态在所有正在运行的作业之间平衡资源。

### 示例：Capacity调度器配置使用

调度器的使用是通过yarn-site.xml配置文件中的

yarn.resourcemanager.scheduler.class参数进行配置的，默认采用Capacity Scheduler调度器。

假设我们有如下层次的队列：

root

├── prod

└── dev

├── mapreduce

└── spark

下面是一个简单的Capacity调度器的配置文件，文件名为capacity-scheduler.xml。在这个配置中，在root队列下面定义了两个子队列prod和dev，分别占40%和60%的容量。需要注意，一个队列的配置是通过属性yarn.sheduler.capacity.<queue-path>.<sub-property>指定的，<queue-path>代表的是队列的继承树，如root.prod队列，<sub-property>一般指capacity和maximum-capacity。

|  |
| --- |
| <configuration>  <property>  <name>**yarn.scheduler.capacity.root.queues**</name>  <value>**prod,dev**</value>  </property>  <property>  <name>**yarn.scheduler.capacity.root.dev.queues**</name>  <value>**mapreduce,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.mapreduce.capacity**</name>  <value>**50**</value>  </property>  <property>  <name>**yarn.scheduler.capacity.root.dev.spark.capacity**</name>  <value>**50**</value>  </property>  </configuration> |

我们可以看到，dev队列又被分成了mapreduce和spark两个相同容量的子队列。dev的maximum-capacity属性被设置成了75%，所以即使prod队列完全空闲dev也不会占用全部集群资源，也就是说，prod队列仍有25%的可用资源用来应急。我们注意到，mapreduce和spark两个队列没有设置maximum-capacity属性，也就是说mapreduce或spark队列中的job可能会用到整个dev队列的所有资源（最多为集群的75%）。而类似的，prod由于没有设置maximum-capacity属性，它有可能会占用集群全部资源。

关于队列的设置，这取决于我们具体的应用。比如，在MapReduce中，我们可以通过mapreduce.job.queuename属性指定要用的队列。如果队列不存在，我们在提交任务时就会收到错误。如果我们没有定义任何队列，所有的应用将会放在一个default队列中。

注意：对于Capacity调度器，我们的队列名必须是队列树中的最后一部分，如果我们使用队列树则不会被识别。比如，在上面配置中，我们使用prod和mapreduce作为队列名是可以的，但是如果我们用root.dev.mapreduce或者dev. mapreduce是无效的。

## .关于yarn常用参数设置

**设置container分配最小内存**

yarn.scheduler.minimum-allocation-mb 1024 给应用程序container分配的最小内存

**设置container分配最大内存**

yarn.scheduler.maximum-allocation-mb 8192 给应用程序container分配的最大内存

**设置每个container的最小虚拟内核个数**

yarn.scheduler.minimum-allocation-vcores 1 每个container默认给分配的最小的虚拟内核个数

**设置每个container的最大虚拟内核个数**

yarn.scheduler.maximum-allocation-vcores 32 每个container可以分配的最大的虚拟内核的个数

**设置NodeManager可以分配的内存大小**

yarn.nodemanager.resource.memory-mb 8192 nodemanager 可以分配的最大内存大小，默认8192Mb

**定义每台机器的内存使用大小**

yarn.nodemanager.resource.memory-mb 8192

**定义交换区空间可以使用的大小**

交换区空间就是讲一块硬盘拿出来做内存使用,这里指定的是nodemanager的2.1倍

yarn.nodemanager.vmem-pmem-ratio 2.

3. yarn基本介绍: 统一资源调度平台

资源: 内存 CPU 磁盘

yarn管理资源: 内存 和 CPU

hdfs管理资源: 磁盘

在实际生产中, 每一个datanode上都会运行着一个nodemanager

datanode: 出磁盘

nodeManager: 出 内存 和 CPU

hadoop1.x中: MapReduce平台

jobTracker : 主节点 仅支持单节点部署

功能:

任务的接收, 资源分配 以及任务分配

TaskTracker: 从节点 可以部署多台

功能:

负责任务最终执行

存在什么弊端:

1) jobTracker 存在单点故障问题

2) jobTracker工作任务有点繁多, 容易导致jobTracker宕机问题

3) 整个MR平台 只能允许 mr任务, 无法支持其他组件的计算任务

hadoop2.x: yarn平台

resourceManager: 主节点 支持有多台的

功能: 任务的接收 ,资源分配

nodeManager: 从节点 支持多台

功能: 负责任务最终执行

application master(app Master): 每一个job任务. 都会启动一个 appMaster

主要负责: 当前job任务的 任务分配工作, 以及负责向resourcemanager申请资源

并对当前这个任务, 进行全生命周期管理

带来的好处:

1) 解决了 原有的单点故障问题

2) 降低了 主节点的任务量, 从而提升其可用性

3) yarn平台支持可以运行多种计算程序: MR spark storm ....

4) yarn运行流程:

1) 由客户端向RM提交任务(MR,Spark...)

2) RM接收任务, 并根据任务随机找一台NM,启动AppMaster ,通知以container方式

container: 资源信息容器

节点信息

内存信息

CPU信息

运行: AppMaster

3) 指定NM启动AppMaster, 启动后和RM保持心跳机制, 用于报告当前已经启动了,并且通过心跳来传递相关信息

4) 根据RM给定任务信息. 根据任务信息, 对任务进行分配, 主要会分配出要启动多少个Map和多少个reduce, 以及

每个map 和每个reduce需要使用多大资源空间, 然后将资源申请相关信息发送给RM(心跳发送)

5) RM接收到资源申请信息后, 将申请信息交给内部资源调度器, 由资源调度器, 根据相关的资源调度方案, 进行资源

分配即可, 如果当下没有资源, 在此处等待

注意: 资源并不是一次性全部给到AppMaster, 一般会采用极可能满足方案, 如果满足不了, 会先给与一定资源

进行运行, 如果空闲资源连一个container都不足, 就会将这些资源挂起, 等待资源充足

6) AppMaster基于心跳机制, 不断询问RM是否已经准备好了资源了, 如果发现已经准备好了, 然后直接将资源信息获取

7) 根据资源信息说明,到指定的NM上启动container资源容器, 开始运行相关任务

8) NM 接收启动的信息后, 开始启动执行, 此时会和AppMaster以及RM保持心跳连接

将任务的相关信息根据心跳通知AppMaster

将资源的使用信息根据心跳通知RM

9) 当NM运行完成后, 会通知AppMaster并将资源使用完成情况通知给RM

10) AppMaster告知给RM 任务已经运行完成了, RM 回收资源, 通知AppMaster进行自毁即可

注意:

当NM在运行过程中, 如果发生错误了, 此时RM会立即将资源回收,此时AppMaster就需要重新和RM申请资源

5) yarn的资源调度器: scheduler

FIFO scheduler : 先进先出调度方案

当一个调度任务进入到调度器之后, 那么调度器会优先满足第一个MR任务全部资源,此时就有可能将资源全部都获取到了

导致后续的任务, 本身的运行时间很短, 但是由于第一个MR将资源全部抢走了, 导致后续任务全部等待

此种调度器在生产中 一般不会使用, 因为 生产中yarn平台不是你自己的

capacity scheduler : 容量调度器

此种调度器是有 Yahoo 提供一种调度方案, 同时也是当下Apache版本的hadoop默认调度方案

可以预先分配出多个队列, 相当于对资源进行预先的划分

每个队列, 可以指定占用多少的百分比的资源, 从而保证, 大的任务可以有单独的队列来运行, 并且小的任务, 也可以正常的运行

fair scheduler : 公平调度器

不需要预先的对资源进行划分操作

可以保证每个MR任务都可以获取到相对均衡的资源来运行MR任务,从而让各个MR都有资源可运行

目前 是 CDH版本的hadoop默认采用调度方案

在生产中, 一般采用的公平调度器, 采用版本一般都是CDH商业化版本hadoop