Hadoop-YARN

组件

- ResourceManager
  - 总管理者
  - 以后台进程运行，8042访问
  - 负责整个系统的资源管理和分配
  - 主要由调度器（Scheduler）和应用程序管理器（Application Manager）组成

  Scheduler：根据队列和容量等限制条件，将资源分配给运行的应用程序，不负责监控和跟踪应用执行状态，将资源分类封装在资源容器（Container）中

  ApplicationManager：负责监控、重启应用程序，与Scheduler协调资源

- NodeManager
  - 节点管理者
  - 启动时向RM注册，并定时发送心跳
  - 维护Container生命周期，监控其资源使用
  - 管理任务依赖，根据AM需要在Con启动前将程序依赖拷贝到本地

- ApplicationMaster
  - 应用管理
  - 向RM申请资源
  - 动态分配来自RM的资源，给内部任务
  - 与NM通信，任务失败重新申请资源重启任务

- Container
  - 是一个资源的分配单位
  - 资源封装运输，类似容器
  - 它封装了某个节点上的多维度资源，如内存、CPU、磁盘、网络等

通信协议

RPC协议是Yarn各组件之间的通信协议

❑ JobClient（作业提交客户端）与RM之间的协议—ApplicationClientProtocol：JobClient通过该RPC协议提交应用程序、查询应用程序状态等。

❑ Admin（管理员）与RM之间的通信协议—ResourceManagerAdministrationProtocol：Admin通过该RPC协议更新系统配置文件，比如节点黑白名单、用户队列权限等。

❑ AM与RM之间的协议—ApplicationMasterProtocol：AM通过该RPC协议向RM注册和撤销自己，并为各个任务申请资源。

❑ AM与NM之间的协议—ContainerManagementProtocol：AM通过该RPC要求NM启动或者停止Container，获取各个Container的使用状态等信息。

❑ NM与RM之间的协议—ResourceTracker：NM通过该RPC协议向RM注册，并定时发送心跳信息汇报当前节点的资源使用情况和Container运行情况。

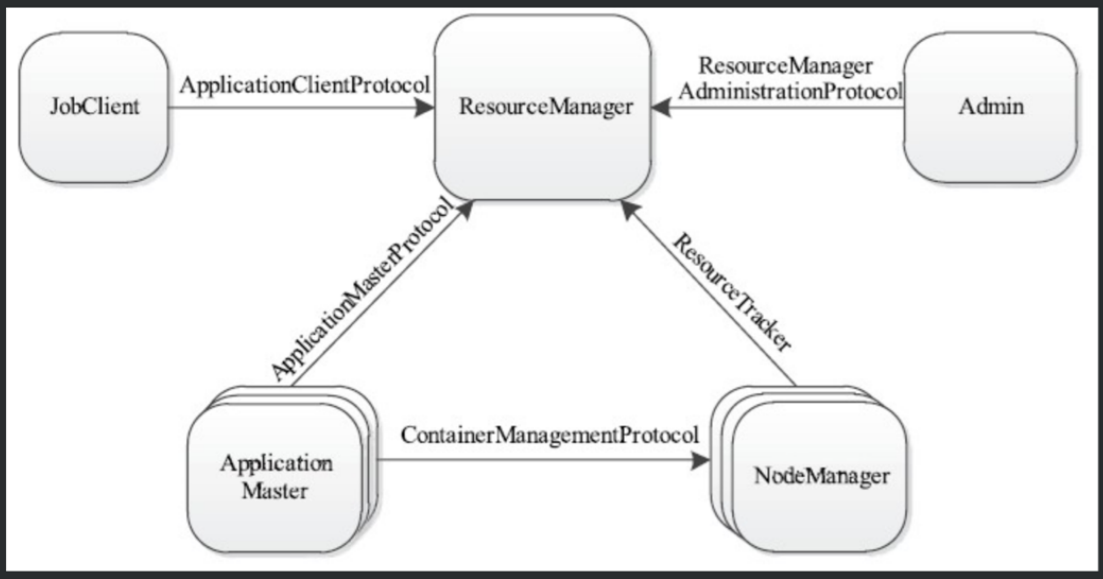

图2-10　Apache YARN的RPC协议

工作流程

1，用户提交应用程序，包括ApplicationMaster
2，RM分配Con并与NM通信
3，AM像RM注册，这一步完成后，用户可通过UI界面访问任务状态
4，AM通过RPC与RM通信，申请资源
5，AM得到资源，与NM通信，开启任务
6，NM设置运行环境，将任务写到脚本，通过脚本启动
7，个任务通过RPC像AM汇总状态和进度，AM可以在任务失败时得到消息并重启
8，任务完成，AM像RM注销并关闭自己

工作流程详解

- 作业提交

  1，客户端提交作业信息到MapReduce
  2，MapRecude向RM申请作业ID
  3，MapReduce将作业copy到HDFS，包括包括 Jar 包，配置文件, split 信息 （split是有客户端完成的）
  4，向RM提交作业

- 作业初始化

  5，RM开启一个Con传到NM，给RM分配资源
  6，NM开启进程监控资源使用、任务完成情况
  7，通过客户端计算的split结果创建map任务，根据map结果创建reduce对象

- 任务分配

  7，如果任务小，会在JVM中运行，任务大会在分布式系统中运行：通过心跳，向RM请求Con运行map和reduce任务

- 任务运行

  8，当任务分配了Con之后，AM启动Con
  9，一个YarnChlid的Java应用初始化任务的资源（配置、JAR包等）

- 作业完成

  10，客户端每5分钟调用waitForCompletion()检查作业是否完成，时间间隔可通过配置文件修改

- 提交作业命令

Yarn通信库

Hadoop&RPC

- 分布式Hadoop用RPC调度各个模块和程序

- RPC优点：
  - 用户感觉不到是在多机器运行
  - 高性能，master是高性能服务器，高效处理各种client的请求，保证并发
  - 可控，JDK中的RPC不优秀所以没用

RPC调用部分

RPC四层架构

- 序列化层：函数参数等序列化减少网络压力
- 传输层：在不同机器之间传数据
- 函数调用层：调用执行函数
- 服务器端处理框架：客户端与服务器的交互，直接决定并发能力，Hadoop采用了基于Reactor设计模式的事件驱动IO模型

RPC通信模型

- 客户程序/服务过程：请求发出者和处理者
- Stub：模拟远程函数的本地调用，将请求信息封装成包发送到服务器
- 调度程序：接受通信模块发来的信息，交给一个Stub，用线程池解决请求量大的问题
- 通信模块：单机模式无需考虑，分布式需要通信

RPC程序调用过程

1，客户端程序通过本地调用的方式产生stub程序
2，本地stub程序将函数调用信息封装成消息包发送给本地通信模块
3，本地通信模块将信息发送给服务器通信模块
4，服务器通信模块将信息发送到服务器端的stub模块
5，服务器端stub解析函数调用信息，将过程交给服务过程进行调用
6，服务过程执行函数，并返回值，逐级返回到客户端的客户程序

提交格式: hadoop jar jar包路径 主类名称 主类参数

```
hadoop jar hadoop-mapreduce-examples-2.6.0-cdh5.15.2.jar pi 3 3
```

调度选项

- FIFO：先进先出
- Fair：公平调度，集群只有一个任务时分配所有资源，任务多了均分
- Capacity：容量调度，根据应得资源和集群资源比值，计算实际资源分配

抢占（公平调度专有）

允许调度器终止那些占用资源超过分配给它的容器（Con）
被终止的Con需要重新执行（类似rollback）

延迟调度

- 繁忙的集群上，应用请求节点，这个节点可能正在运行其他Con，多等几秒可以增加请求节点上分配Con的机会，提高集群效率
- 容量和公平调度都支持延迟调度

资源公平（内存&CPU）

- 两个不同类型的应用（moreCPU or more 内存）分到的容器数是不同的
- 根据两任务CPU和内存需求占集群资源总量的比例不同，判断两任务的主类资源是什么
- 得到Con数量的占比等于主类资源的占比

数据压缩

- 数据压缩本质上减轻了数据存储、IO资源和网络带宽的消耗
- 数据压缩和解压的过程是CPU计算，增加了CPU资源的计算
- 基本原则
  - CPU密集型任务，少用压缩
  - IO密集型任务，多用压缩

压缩方式

1）**Gzip压缩**

优点：压缩率比较高，而且压缩/解压速度也比较快；Hadoop本身支持，在应用中处理gzip格式的文件就和直接处理文本一样；大部分linux系统都自带gzip命令，使用方便。

缺点：不支持split。

应用场景：当每个文件压缩之后在130M以内的（1个块大小内），都可以考虑用gzip压缩格式。例如说一天或者一个小时的日志压缩成一个gzip文件，运行mapreduce程序的时候通过多个gzip文件达到并发。hive程序，streaming程序，和java写的mapreduce程序完全和文本

处理一样，压缩之后原来的程序不需要做任何修改。

2）**Bzip2压缩**

优点：支持split；具有很高的压缩率，比gzip压缩率都高；hadoop本身支持，但不支持native；在linux系统下自带bzip2命令，使用方便。

缺点：压缩/解压速度慢。

应用场景：适合对速度要求不高，但需要较高的压缩率的时候，可以作为mapreduce作业的输出格式；或者输出之后的数据比较大，处理之后的数据需要压缩存档减少磁盘空间并且以后数据用得比较少的情况；或者对单个很大的文本文件想压缩减少存储空间，同时又需要支持split，而且兼容之前的应用程序（即应用程序不需要修改）的情况。

3）**Lzo压缩**

优点：压缩/解压速度也比较快，合理的压缩率；支持split，是hadoop中最流行的压缩格式；可以在linux系统下安装lzop命令，使用方便。

缺点：压缩率比gzip要低一些；hadoop本身不支持，需要安装；在应用中对lzo格式的文件需要做一些特殊处理（为了支持split需要建索引，还需要指定inputformat为lzo格式）。

应用场景：一个很大的文本文件，压缩之后还大于200M以上的可以考虑，而且单个文件越大，lzo优点越越明显。

4）**Snappy压缩**

优点：高速压缩速度和合理的压缩率。

缺点：不支持split；压缩率比gzip要低；hadoop本身不支持，需要安装；

应用场景：当Mapreduce作业的Map输出的数据比较大的时候，作为Map到Reduce的中间数据的压缩格式；或者作为一个Mapreduce作业的输出和另外一个Mapreduce作业的输入。