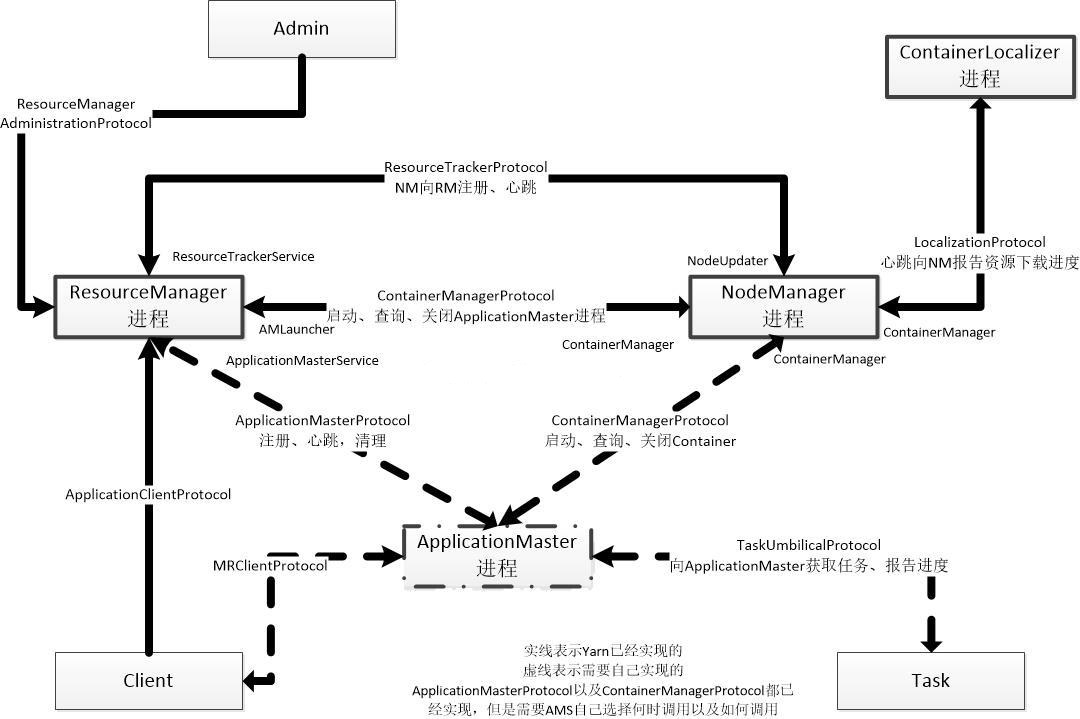
Yarn AppMaster

在Yarn中，Client向ResourceManager提交应用，每个应用都会启动一个ApplicationMa

ster。AM经过ResourceManager分配资源后，运行在某个Slave节点（NodeManager）的container上，执行计算的Task，也运行在Slave节点的container中。RM,NM,AM及Container之间的通信，都是通过RPC机制来完成的，详细框架如下图所示：



Yarn详细框架

1）通过sbin/yarn命令管理Yarn集群，Yarn Amin对应的默认端口是8033，在RM中对应的类为AdminService等服务，其实现接口RMAdministrationProtocol。Client，可以Yarn提交应用或者获取集群等信息，可以通过Yarn命令，但是实现的接口是AdminClientProtocol，对应的端口为8032等。Client及Admin对Yarn的访问及管理，都是与RM通信来完成。

2）ResourceManager(RM)负责管理调度资源。每个应用的ApplicationMaster与ResourceManager协商资源，并与NodeManager协同工作来执行和管理Task。ResourceManager内部有一个可插入的调度器，负责向各个应用分配资源以满足容量、组等限制，这个调度器不负责管理或追踪应用的状态，也不负责由于硬件错误或应用问题导致的task失败重启问题，调度器只一句应用的资源需求来执行调度工作，调度的内容是一个抽象的Resouce Container，其中包含资源元素，如内存,cpu,网络及磁盘等。

3）NodeManager是每个节点的slave，其负责管理应用的container，管理他们的资源使用，并向ResouceManater汇报整体的资源使用情况。NM会接收并且处理来自AM的container的启动和停止的各种请求。

4）ApplicationMaster，负责向RM的调度器协商合理的Resource Container并追踪他们的状态，管理进度，AM在系统中以Container的形式执行。AM同时与NodeManager进行通信以启动或者停止服务/监控所有任务的运行情况，并在任务失败的情况下，重新为任务申请资源并且重启任务、负责推测任务的执行。

5）Container，Yarn对系统资源的抽象，同时也是系统资源分配的基本单位，封装节点上多维度资源，包括cpu/内存/磁盘和网络等。Yarn会每个任务分配一个Container，并且该任务只能使用该container中所描述的资源。

Hadoop源码中实现了两个基于Yarn的application，一个是MapReduce，另一个是被当做写Application的示例程序--Distributedshell（可以看做是yarn的wordcount示例程序）。Distributedshell的作用是分布式shell执行，将用户提交的一串shell命令或者脚本由AppMaster控制，分配到不同的container中执行，下面分析DistributedShell的实现。

实现一个Application有三个要求：

1）客户端（Client.java）: 客户端提交application

2）ApplicationMaster: 向ResourceManager注册，并申请分配container；和NodeMa

-manger通信，启动Container

通过下面的命令创建10个Container并在container中执行/bin/date

hadoop jar hadoop-yarn-applications-distributedshell-2.6.0.jar org.apache.hadoop.yarn.applications.distributedshell

.Client -jar hadoop-yarn-applications-distributedshell-2.6.0.jar -shell\_command '/usr/bin/date' -num\_containers 10

执行该命令后：

在${hadoop-home}/logs/userlogs/applicationxxx/container-xxx/stdout中会有执行结果：

vim stdout

可以看到命令/usr/bin/date中输出的结果，该命令输出时间。

执行流程如下所示:

1. 客户端通过distributedshell.Client提交application到RM，在Client#run过程中，会根据配置创建ApplicationSubmissionContext
2. distributedshell.ApplicationMaster提交Container请求，执行用户提交的命令ContainerLaunchContext#commands

# 一、Client提交Application源码分析

Yarn Application客户端的主要作用是提供一系列的访问接口供用户与YARN交互，包括提交Application、查询Application运行状态，修改Application属性（如优先级）等。其中，最重要的是访问接口是提交Application函数。

客户端提交一个应用程序需要经过以下两个步骤：

1. Client通过RPC函数ApplicationClientProtocol#getNewApplication从ResourceManager中获取唯一的applicationID
2. Client通过RPC函数ApplicationClientProtocol#submitApplication将ApplicationMaster提交到ResourceManager

# 二、ApplicationMaster提交Container请求

ApplicationMaster（AM）需要与ResourceManager和NodeManager两个服务交互，通过与RM交互，ApplicationMaster可获得任务计算所需的资源。通过与NodeManager交互，Application可启动计算任务，并监控其直到运行完成。

1. AM-RM编写

ApplicationMaster通过RPC函数ApplicationMasterProtocol#registerApplicationMaster向ResourceManager注册

ApplicationMaster通过RPC函数ApplicationMasterProtocl#allocate向ResourceMana

-ger申请资源（以container形式表示）

ApplicationMaster通过RPC函数ApplicationMasterProtocol#finishApplicationMaster告诉ResourceManager应用程序执行完毕，并退出。

2. AM-NM编写流程

Application将申请到的资源二次分配给内部的任务，并通过RPC函数ContainerManage

-mentProtocol#startContainer与对应的NodeManager通信以启动Container。NMClientAsyn

-cImpl#startContainerAsync(container,ctx)来启动一个Container，最终的实现是通过NMClientAsyncImpl类中的状态机，调用StartContainerTransition来启动Container。

为了实时掌握各个Container运行状态，ApplicationMaster通过RPC函数ContainerMan

-agementProtocol#getContainerStatus向NodeManager询问Container运行状态，一旦发现某个Container运行失败，ApplicationMaster可尝试的对对应的任务申请资源。Application

-master#onContainerStarted中调用getContainerStatusAsync，然后AM发送ContainerStartEvent事件，NMClientAsyncImpl通过NMClient#getContainerStatus获取Cont

aner状态信息，然后根据状态信息进行Container的操作。

一旦Container运行完成，ApplicationMaster通过RPC函数ContainerManagementProto

col#stopContainer释放container。NMClientAsyncImpl#StopContainerTransition#transition中调用client.stopContainer