# HADOOP

## YARN

### YARN调度

#### （★★★）yarn流程

1． 用户向YARN中提交应用程序，其中包括ApplicationMaster程序、启动ApplicationMaster的命令、用户程序等。

2． ResourceManager为该应用程序分配第一个Container，并与对应的NodeManager通信，要求它在这个Container中启动应用程序的ApplicationMaster。

3． ApplicationMaster首先向ResourceManager注册，这样用户可以直接通过ResourceManage查看应用程序的运行状态，然后它将为各个任务申请资源，并监控它的运行状态，直到运行结束，即重复步骤4~7。

4． ApplicationMaster采用轮询的方式通过RPC协议向ResourceManager申请和领取资源。

5． 一旦ApplicationMaster申请到资源后，便与对应的NodeManager通信，要求它启动任务。

6． NodeManager为任务设置好运行环境（包括环境变量、JAR包、二进制程序等）后，将任务启动命令写到一个脚本中，并通过运行该脚本启动任务。

7． 各个任务通过某个RPC协议向ApplicationMaster汇报自己的状态和进度，以让ApplicationMaster随时掌握各个任务的运行状态，从而可以在任务失败时重新启动任务。在应用程序运行过程中，用户可随时通过RPC向ApplicationMaster查询应用程序的当前运行状态。

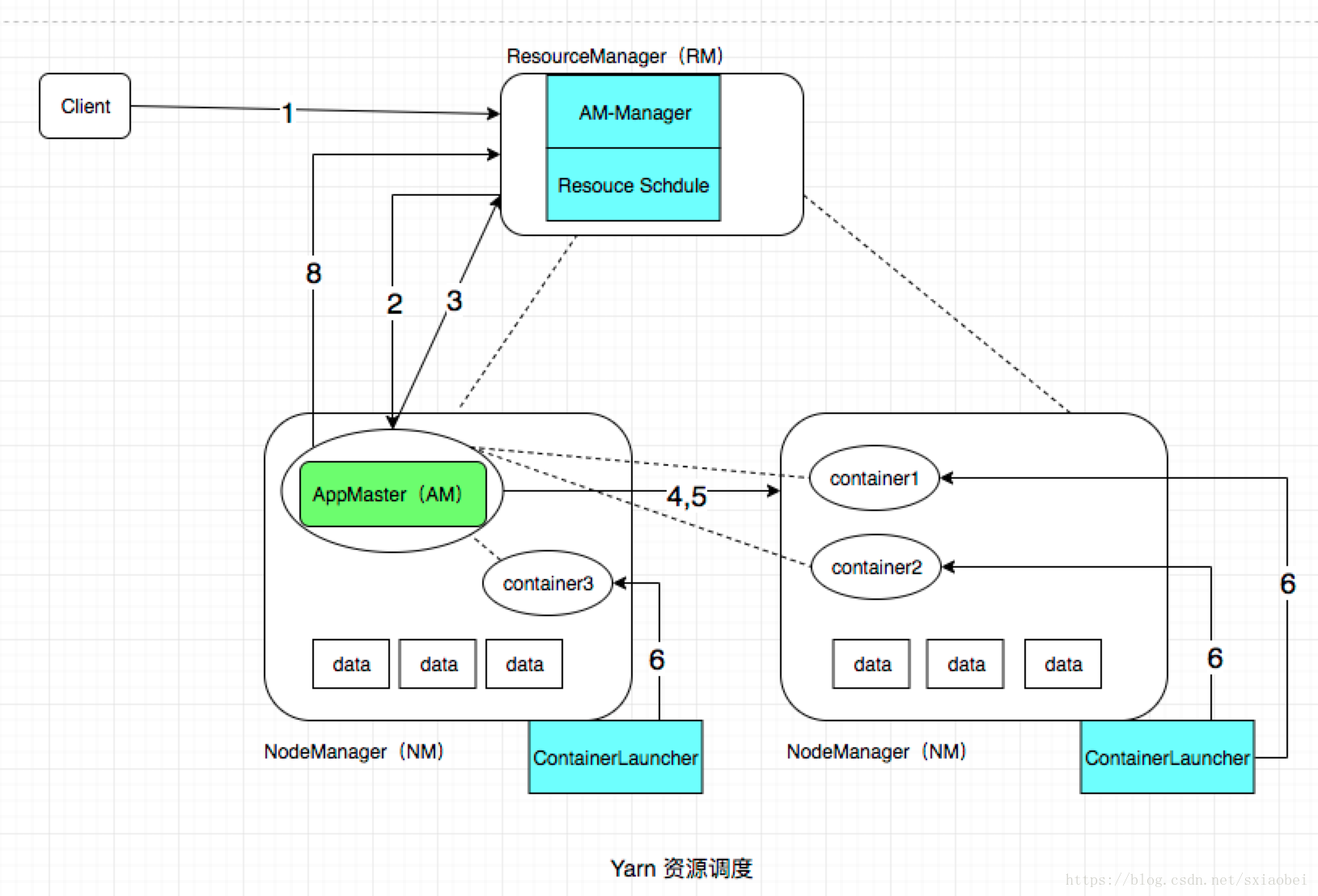
8． 应用程序运行完成后，ApplicationMaster向ResourceManager注销并关闭自己。

#### （★★）yarn中调度原则？

FIFO调度器的优点是，简单易懂，不需要任何配置，但是不适合共享集群。大的应用会占用集群中的所有资源，所以每个应用都必须等待知道轮到自己运行。在一个共享集群中，更适合使用容量调度器(CapacityScheduler)和公平调度器(FairScheduler)。这两种调度器都允许长时间运行的作业能及时完成，同时也允许正在运行较小临时查询的用户能够在合理时间内得到返回结果。

#### （★★★）yarn调度过程？

yarn主要由四个重要角色组成：

1. **ResourceManager**：顾名思义资源管理器，主要负责资源管理和调度，ResourceManager主要由两个组件构成：ApplicationManager,主要负责两类工作：1.管理监控各个系统的应用，包括启动ApplicaitonMaster，监控ApplicaitonMaster运行状态（为了容错）2.跟踪分配给ApplicationMaster的进度和状态。Scheduler，主要负责分配Container给ApplicaitonMaster，分配算法有多种（如公平调度等等）可以根据需求不同选择适合的调度策略。
2. **NodeManager**：节点管理器，主要负责维护本节点的资源情况和任务管理。首先NodeManager需要定期向ResourceManager汇报本节点资源使用情况，以便ResourceManager，根据资源使用情况，来分配资源给ApplicationMaster，其次，需要管理ApplicaitonMaster提交来的task，比如接收ApplicaitonMaster启动或停止task的请求(启动和停止有NodeManager的组件ContainersLanuncher完成)。
3. **ApplicaitonMaster**：用户提交的每个program都会对应一个ApplicationMaster，主要负责监控应用，任务容错（重启失败的task）等。它同时会和ResourceManager和NodeManager有交互，向ResourceManager申请资源，请求NodeManager启动或提示task
4. **Container**：容器是资源调度的单位，它是内存、cpu、磁盘、和IO的集合。ApplicationMaster会给task分配Container，task只能只用分配给它的Container的资源。分配流程为ResourceManager->ApplicationMaster->task

**Yarn调度主要分为8个步骤如上图所示：**

1.有YarnClient提交program信息打拼ResourceManager，包括（应用代码和应用需要的一切参数和环境信息）

2.ResourceManager收到请求之后，调用ApplicationMasterManager向NodeManager发送请求，申请一个资源（Container），并且要求Container启动ApplicationMaster.

3.ApplicationMaster启动之后，首先注册自己到ResourceManager，然后为自己的Task申请Container，这个过程是轮训的，循环申请资源，ResourceManager收到请求之后，会要求NodeManager分配资源

4.资源分配完毕之后，ApplicationMaster发送请求到NodeManager，启动任务。

5.NodeManager设置Container的运行时环境（jar包，环境变量，任务启动脚本），NodeManager会通过脚本启动任务

6.启动的过程是由NodeManager的ContainerLauncher负责的，ContainerLauncher完成启动任务的工作

7.这一步是在作业执行过程中持续发生的，我用虚线代表，主要包括两类交互，第一，task和ApplicationMaster的交互，task会向AM汇报任务状态和进度信息，比如任务启动，停止，状态更新。ApplicaitonMaster利用这些信息监控task整个执行过程。第二，是NodeManager和ResourceManager的交互，这个过程和任务没有关系，主要是两者之间保持的心跳信息（状态的变化等等）

8.ApplicationMaster在检测到作业运行完毕之后，ApplicationMaster想ResourceManager删除自己，并且停止自己执行。

**说明：**

1.图中蓝色的部分，是NodeManager和ResourceManager的组件，主要负责控制整个流程

2.图中绿色的部分，核心资源分配流程，不属于yarn，是由具体的数据引擎实现的（如mr，spark，strom等），由此可见Yarn采用了资源双层调度模型，第一层是Yarn分配资源（Container）给ApplicaitonMaster，这是由Yarn实现的，第二层，是ApplicaitonMaster分配资源给task，这是有具体工作引擎实现，这样的好处是将，yarn的调度和应用分隔开，便于yarn和应用实现、优化或升级自己的调度策略。

上述内容是Yarn调度的基本过程，从调度过程可以看到，Yarn将资源封装抽象为Container，将应用抽象为ApplicationMaster,两个关键模型的抽象，实现了对资源和应用的统一管理，进而实现了调度平台和执行引擎的解耦。至于Application怎么样把Container分配给task，怎么样监控task的执行过程，完全是由执行引擎根据自身特性实现的。充分解耦的好处是，Yarn可以同时部署不同的执行引擎，集群不受限制，并且当一个执行引擎升级的时候，不会影响到别的引擎，这对于生产是至关重要的。并且Yarn提供了多种资源调度模式，以满足不同的生产环境

#### （★★）yarn你们用的哪种调度模式(先进先出FIFO/公平/计算能力)?

看实际情况选择调度模式，每种模式都有自己的优缺点，以下为他们各自特点表述：

**1.FIFOScheduler-先进先出调度策略**

即所有的应用程序将按照提交顺序来执行，这些应用程序都放在一个队列里，只有在执行完了一个之后，在执行顺序执行下一个。

**缺点：**

耗时长的任务会导致后提交的一直处于等待状态，资源利用率不高；如果集群多人共享，显然不太合理。

**2.CapacityScheduler-容量调度**

1．由于FIFOScheduler的局限性，后来衍生出来了多个用户可以共享集群资源，然后集群资源按照队列为单位进行划分的调度器。

2．还可以控制每一个队列资源最低保障和最高使用限制，最高使用限制是防止过多占用空闲资源，而导致其他队列资源紧张。

3．还可以针对用户设置每个用户最高资源使用限制，以防止该用户滥用或者频繁使用资源。

4．每一个队列内部也是按照先进先出的原则调度资源。

5．如果某个队列资源使用紧张，某个队列比较资源有剩余，那么可以暂时把剩余的资源共享给资源比较紧张的队列，一旦共享资源的队列有应用程序提交，那么被共享资源的队列释放的资源会归还给该共享资源的队列(弹性队列)。

6．每一个队列有严格的访问控制，只有那些被允许的用户才可以查看该队列应用程序的状态。

假设我们有如下层次的队列：

root

|-uat

|-dev

|-ecomerce

|-scicne

CapacityScheduler有自己的配置文件，即conf目录下的capacity-scheduler.xml

配置选项：

**capacity:**队列的资源容量比,所有对列的资源容量比之后不能超过100

**maximum-capacity:**该队列最多使用到集群资源的上限比

**minimum-user-limit-percent:**每一个用户最低资源百分比

**user-limit-factor:**每一个用户最多可用的资源百分比

**maximum-applications:**集群或者队列资源同时处于等待的和运行状态的应用程序数目上限，一旦超过此配置，后需提交的应用程序将会被拒绝。默认10000.

**yarn.scheduler.capacity.maximum-applications：**集群最大允许的应用程序总数

**yarn.scheduler.capacity.<queue-path>.maximum-applications：**队列允许最大的应用程序总数

**maximum-am-resource-percent:**集群中用于运行应用程序Application

Master的资源比例上限。用浮点数表示，0.1表示10%。所有队列的ApplicationMaster资源比例上限可通过参数yarn.scheduler.capacity.maximum-am-resource-percent.设置

下面是一个简单的capacity-scheduler配置文件：

修改完成后，需要执行下面的命令：

$HADOOP\_YARN\_HOME/bin/yarnrmadmin-refreshQueues使功能动态生效。

参考博客：http://www.cnblogs.com/yanghaolie/p/6274098.html

**3.FairScheduler-公平调度器**

试图为每一个任务均匀分配资源，比如当前集群只有一个任务A，那么A拥有整个集群资源，这时候又提交了一个任务B，这时候任务A和B平分资源。

**主要特点：**

1．也是将集群资源以队列为单位进行划分

2．我们也可以设定队列的所使用container资源的最低保障百分比和最高上限

3．我们也可以针对每一个用户设置资源使用的最低保障和使用上限

4．当一个队列资源紧缺，可以共享其他队列的资源

5．我们可以根据的不同策略为应用程序分配资源，比如FIFO,Fair或者DRF,默认是Fair.这就说明2个应用程序可以个占用1/2的资源，三个应用程序可以个占用1/3的资源

6．当某个队列有剩余资源时，调度器会将这些资源共享给其他队列；如果当该队列有新的应用程序提交时，调度器为他回收资源，采用的是先等待在抢占，等待一段时间后，尚有为归还的资源，则会进行资源抢占，怎么抢占呢？从那些超额的资源的队列杀死一部分任务，进而释放资源

7．负载均衡：提供一个基于任务数目负载均衡机制，该机制尽可能将系统的任务均匀分配到各个节点上

yarn-site.xml关于公平调度的配置：

**yarn.scheduler.fair.allocation.file:**自定义XML配置文件所在位置，该文件主要用于描述各个队列属性，比如资源量，权重

**yarn.scheduler.fair.user-as-default-name:**当应用程序未指定队列名的时候，是否指定用户名作为应用程序所在队列名；如果未设置，那么所有的应用程序提交大default队列中，默认为true

**yarn.scheduler.fair.preemption:**是否启用抢占机制，默认为false

**yarn.scheduler.fair.sizebaseweight:**在一个队列分配资源时默认情况采用公平轮询的方式将资源分配给各个应用程序，而该参数提供了另外一种分配方式，按照应用程序资源需求数目进行资源分配，默认为fasle

**yarn.scheduler.fair.assignmultiple:**是否启动批量分配

**yarn.scheduler.fair.max.assign:**如果开启批量分配，可指定一次分配的Container数量

**yarn.scheduler.increment-allocation-mb:**内存资源规整化单位，默认1024，这意味着一个Container请求资源是1.5GB,则将被调整为ceiling

(1.5/1)\*1GB=2GB

自定义配置文件，默认是fair-scheduler.xml

**minResources:**最少资源保证量，格式为Xmb,Yvcores,当一个队列最小资源保证量为满足，他将优先于其他队列获取资源。

**maxResources:**最多可以使用的资源量

**maxRunningApps:**最多同时运行的应用程序数目

**schedulingMode/schedulingPolicy:**队列采用的调度模式

**aclSubmitApps:**可向队列中提交应用程序的用户列表

**minSharePreemptionTimeout：**最小公向量抢占时间：即一个资源池在该时间内使用的资源量一直低于最小资源量，则开始抢占资源

<allocations>

<defaultqueueschedulingpolicy>fair</defaultqueueschedulingpolicy>

<queuename="uat">

<weight>40<weight>

<schedulingpolicy>fifo</schedulingpocliy>

<minResources>100mb,100vcores</minResources>

<maxResources>150mb,200vcores</maxResources>

<maxRunningApps>200</maxRunningApps>

<minSharePreemptionTimeout>300</minSharePreemptionTimeout>

</queue>

<queuename="dev">

<weight>60<weight>

<minResources>30mb,30vcores</minResources>

<maxResources>50mb,50vcores</maxResources>

</queue>

<queuename="eng"/>

<queuename="science"/>

<queueplacementpolicy>

<rulename="specified"create="false"></rule>

<rulename="primarygroup"create="false"></rule>

<rulename="default"queue="dev.eng"></rule>

</queueplacementpolicy>

<username="userA">

<maxRunningApps>400</maxRunningApps>

</user>

<userMaxAppsDefault>40</userMaxAppsDefault>

<fairSharePreemptionTimeout>6000</fairSharePreemptionTimeout>

</allocations>

自定义2队列uat和dev,权重比40:60,也就是说不采用均分的策略。

每一个队列可以有不同的调度策略，默认是fair,此外还有fifo和DRF

#### （★）CapacityScheduler和FairScheduler的比较

1．相同点

1.都允许多用户共享集群资源

2.都是以队列对资源进行划分

3.都支持层级队列

4.都可以进行队列所需资源最低保证和使用资源的最高上限

5.资源紧缺的队列可以共享其他队列的空闲资源(或者集群的空闲资源)

6.都可以针对用户设置最低资源保障和资源使用最高上限

7.都可以对用户设置

2．不同点

1.是否支持负载均衡：

CapacityScheduler:不支持负载均衡

FairScheduler:支持负载均衡

2.Container请求资源粒度不一样

CapacityScheduler:请求资源的最小整数倍

FairScheduler:可以根据内存规整化参数控制，粒度更小

3.队列之间的资源分配方式：

CapacityScheduler优先选择资源使用率低的队列，然后队列中通过FIFO或者DRF策略进行调度

FairScheduler:使用公平算法选择队列，然后在在队列中通过Fair、FIFO或者DRF调度

Ps：关于DRF调度算法见：https://blog.csdn.net/opensure/article/details/51591478

### YARN常见问题

#### （★★★）“MapReduce2.0”与“YARN”是否等同，尝试解释说明。

MapReduce2.0--àmapreduce+yarn

YARN是“YetAnotherResourceNegotiator”的简称。在进一步了解YARN框架之前我们需要知道，相比较而言，MapReduce则是YARN的一个特例。YARN则是MapReduce的一个更加通用和高级的框架形式，并在其上增加了更多的功能。例如通过加载分布式执行脚本可以在集群节点上执行独立的脚本任务，并且更多功能正在被追加中。所以我们可以看到，YARN可以直接运行在MapReduce运行的框架上而不会造成更多的干扰，并且会为集群的运算带来更多的好处。更一步的开发显示了YARN会允许开发者根据自己的需求运行不同版本的MapReduce在集群中，这将为开发者提供更为便捷的服务

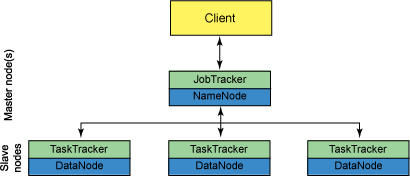
#### （★★）为什么会产生yarn,它解决了什么问题，有什么优势？

简单来讲是因为Mrv1的缺陷，产生yarn。下面详细介绍

**Hadoop和MRv1简单介绍**

Hadoop集群可从单一节点（其中所有Hadoop实体都在同一个节点上运行）扩展到数千个节点（其中的功能分散在各个节点之间，以增加并行处理活动）。

下图演示了一个Hadoop集群的高级组件。



一个Hadoop集群可分解为两个抽象实体：MapReduce引擎和分布式文件系统。MapReduce引擎能够在整个集群上执行Map和Reduce任务并报告结果，其中分布式文件系统提供了一种存储模式，可跨节点复制数据以进行处理。Hadoop分布式文件系统(HDFS)通过定义来支持大型文件（其中每个文件通常为64MB的倍数）。

当一个客户端向一个Hadoop集群发出一个请求时，此请求由JobTracker管理。JobTracker与NameNode联合将工作分发到离它所处理的数据尽可能近的位置。NameNode是文件系统的主系统，提供元数据服务来执行数据分发和复制。JobTracker将Map和Reduce任务安排到一个或多个TaskTracker上的可用插槽中。TaskTracker与DataNode（分布式文件系统）一起对来自DataNode的数据执行Map和Reduce任务。当Map和Reduce任务完成时，TaskTracker会告知JobTracker，后者确定所有任务何时完成并最终告知客户作业已完成。

从上图中可以看到，MRv1实现了一个相对简单的集群管理器来执行MapReduce处理。MRv1提供了一种分层的集群管理模式，其中大数据作业以单个Map和Reduce任务的形式渗入一个集群，并最后聚合成作业来报告给用户。但这种简单性有一些隐秘，不过也不是很隐秘的问题。

**MRv1的缺陷**

MapReduce的第一个版本既有优点也有缺点。MRv1是目前使用的标准的大数据处理系统。但是，这种架构存在不足，主要表现在大型集群上。当集群包含的节点超过4,000个时（其中每个节点可能是多核的），就会表现出一定的不可预测性。其中一个最大的问题是级联故障，由于要尝试复制数据和重载活动的节点，所以一个故障会通过网络泛洪形式导致整个集群严重恶化。

但MRv1的最大问题是多租户。随着集群规模的增加，一种可取的方式是为这些集群采用各种不同的模型。MRv1的节点专用于Hadoop，所以可以改变它们的用途以用于其他应用程序和工作负载。当大数据和Hadoop成为云部署中一个更重要的使用模型时，这种能力也会增强，因为它允许在服务器上对Hadoop进行物理化，而无需虚拟化且不会增加管理、计算和输入/输出开销。

我们现在看看YARN的新架构，看看它如何支持MRv2和其他使用不同处理模型的应用程序。

**YARN(MRv2)简介**

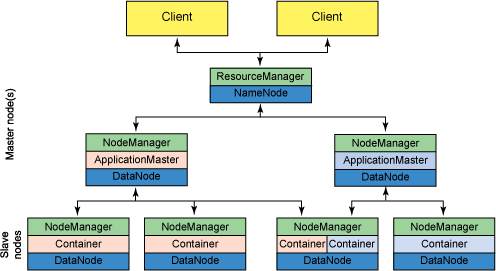
为了实现一个Hadoop集群的集群共享、可伸缩性和可靠性。设计人员采用了一种分层的集群框架方法。具体来讲，特定于MapReduce的功能已替换为一组新的守护程序，将该框架向新的处理模型开放。

由于限制了扩展以及网络开销所导致的某些故障模式，MRv1JobTracker和TaskTracker方法曾是一个重要的缺陷。这些守护程序也是MapReduce处理模型所独有的。为了消除这一限制，JobTracker和TaskTracker已从YARN中删除，取而代之的是一组对应用程序不可知的新守护程序。

YARN分层结构的本质是ResourceManager。这个实体控制整个集群并管理应用程序向基础计算资源的分配。ResourceManager将各个资源部分（计算、内存、带宽等）精心安排给基础NodeManager（YARN的每节点代理）。ResourceManager还与ApplicationMaster一起分配资源，与NodeManager一起启动和监视它们的基础应用程序。在此上下文中，ApplicationMaster承担了以前的TaskTracker的一些角色，ResourceManager承担了JobTracker的角色。

ApplicationMaster管理一个在YARN内运行的应用程序的每个实例。ApplicationMaster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（CPU、内存等的资源分配）。请注意，尽管目前的资源更加传统（CPU核心、内存），但未来会带来基于手头任务的新资源类型（比如图形处理单元或专用处理设备）。从YARN角度讲，ApplicationMaster是用户代码，因此存在潜在的安全问题。YARN假设ApplicationMaster存在错误或者甚至是恶意的，因此将它们当作无特权的代码对待。

NodeManager管理一个YARN集群中的每个节点。NodeManager提供针对集群中每个节点的服务，从监督对一个容器的终生管理到监视资源和跟踪节点健康。MRv1通过插槽管理Map和Reduce任务的执行，而NodeManager管理抽象容器，这些容器代表着可供一个特定应用程序使用的针对每个节点的资源。YARN继续使用HDFS层。它的主要NameNode用于元数据服务，而DataNode用于分散在一个集群中的复制存储服务。

要使用一个YARN集群，首先需要来自包含一个应用程序的客户的请求。ResourceManager协商一个容器的必要资源，启动一个ApplicationMaster来表示已提交的应用程序。通过使用一个资源请求协议，ApplicationMaster协商每个节点上供应用程序使用的资源容器。执行应用程序时，ApplicationMaster监视容器直到完成。当应用程序完成时，ApplicationMaster从ResourceManager注销其容器，执行周期就完成了。

通过这些讨论，应该明确的一点是，旧的Hadoop架构受到了JobTracker的高度约束，JobTracker负责整个集群的资源管理和作业调度。新的YARN架构打破了这种模型，允许一个新ResourceManager管理跨应用程序的资源使用，ApplicationMaster负责管理作业的执行。这一更改消除了一处瓶颈，还改善了将Hadoop集群扩展到比以前大得多的配置的能力。此外，不同于传统的MapReduce，YARN允许使用MessagePassingInterface等标准通信模式，同时执行各种不同的编程模型，包括图形处理、迭代式处理、机器学习和一般集群计算。

随着YARN的出现，我们不再受到更简单的MapReduce开发模式约束，而是可以创建更复杂的分布式应用程序。实际上，我们可以将MapReduce模型视为YARN架构可运行的一些应用程序中的其中一个，只是为自定义开发公开了基础框架的更多功能。这种能力非常强大，因为YARN的使用模型几乎没有限制，不再需要与一个集群上可能存在的其他更复杂的分布式应用程序框架相隔离，就像MRv1一样。甚至可以说，随着YARN变得更加健全，它有能力取代其他一些分布式处理框架，从而完全消除了专用于其他框架的资源开销，同时还简化了整个系统。

为了演示YARN相对于MRv1的效率提升，可考虑蛮力测试旧版本的LANManagerHash的并行问题，这是旧版Windows®用于密码散列运算的典型方法。在此场景中，MapReduce方法没有多大意义，因为Mapping/Reducing阶段涉及到太多开销。相反，更合理的方法是抽象化作业分配，以便每个容器拥有密码搜索空间的一部分，在其之上进行枚举，并通知您是否找到了正确的密码。这里的重点是，密码将通过一个函数来动态确定（这确实有点棘手），而不需要将所有可能性映射到一个数据结构中，这就使得MapReduce风格显得不必要且不实用。

归结而言，MRv1框架下的问题仅是需要一个关联数组，而且这些问题有专门朝大数据操作方向演变的倾向。但是，问题一定不会永远仅局限于此范式中，因为您现在可以更为简单地将它们抽象化，编写自定义客户端、应用程序主程序，以及符合任何您想要的设计的应用程序。

博客位置：http://www.aboutyun.com/thread-6785-1-1.html

#### （★★★）Hadoop2：体系结构、HDFSHA、YARN？

就是HDFS体系+yarn体系

HDFS体系结构

**采用Master-Slaver模式：**

NameNode中心服务器（Master）:维护文件系统树、以及整棵树内的文件目录、负责整个数据集群的管理。

DataNode分布在不同的机架上（Slaver）：在客户端或者NameNode的调度下，存储并检索数据块，并且定期向NameNode发送所存储的数据块的列表。

客户端与NameNode获取元数据；

与DataNode交互获取数据。

默认情况下，每个DataNode都保存了3个副本，其中两个保存在同一个机架的两个不同的节点上。另一个副本放在不同机架上的节点上。

**基本概念**

**机架：**HDFS集群，由分布在多个机架上的大量DataNode组成，不同机架之间节点通过交换机通信，HDFS通过机架感知策略，使NameNode能够确定每个DataNode所属的机架ID，使用副本存放策略，来改进数据的可靠性、可用性和网络带宽的利用率。

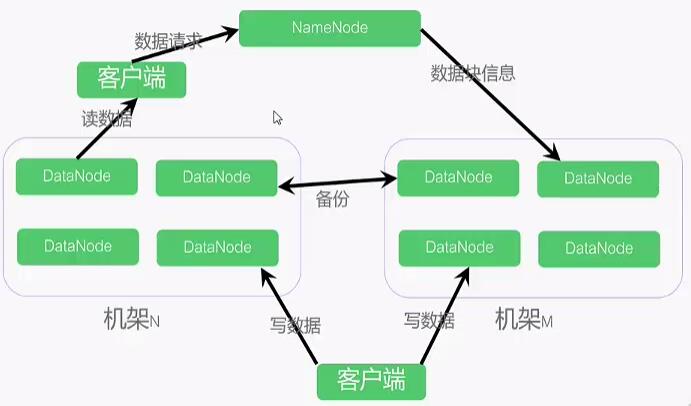
**数据块(block)：**HDFS最基本的存储单元，默认为64M，用户可以自行设置大小。

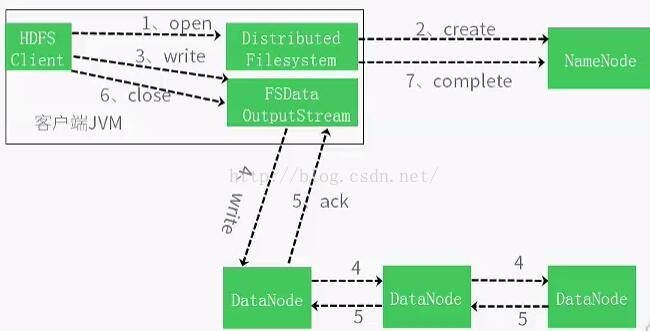
**元数据：**指HDFS文件系统中，文件和目录的属性信息。HDFS实现时，采用了镜像文件（Fsimage）+日志文件（EditLog）的备份机制。文件的镜像文件中内容包括：修改时间、访问时间、数据块大小、组成文件的数据块的存储位置信息。目录的镜像文件内容包括：修改时间、访问控制权限等信息。日志文件记录的是：HDFS的更新操作。

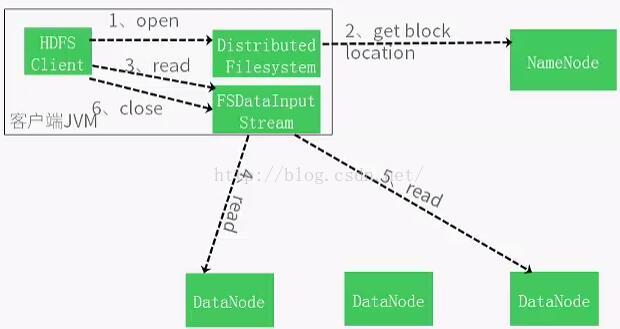
NameNode启动的时候，会将镜像文件和日志文件的内容在内存中合并。把内存中的元数据更新到最新状态。

**用户数据：**HDFS存储的大部分都是用户数据，以数据块的形式存放在DataNode上。

在HDFS中，NameNode和DataNode之间使用TCP协议进行通信。DataNode每3s向NameNode发送一个心跳。每10次心跳后，向NameNode发送一个数据块报告自己的信息，通过这些信息，NameNode能够重建元数据，并确保每个数据块有足够的副本。

Yarn体系（详见第二题）

HDFS读取数据的流程



#### （★★★）yarn的参数mr参数

**Yarn的参数：**

**1）ResourceManager相关配置参数**

**①yarn.resourcemanager.address**

参数解释：ResourceManager对客户端暴露的地址。客户端通过该地址向RM提交应用程序，杀死应用程序等。

默认值：${yarn.resourcemanager.hostname}:8032

**②yarn.resourcemanager.scheduler.address**

参数解释：ResourceManager对ApplicationMaster暴露的访问地址。ApplicationMaster通过该地址向RM申请资源、释放资源等。

默认值：${yarn.resourcemanager.hostname}:8030

**③yarn.resourcemanager.resource-tracker.address**

参数解释：ResourceManager对NodeManager暴露的地址.。NodeManager通过该地址向RM汇报心跳，领取任务等。

默认值：${yarn.resourcemanager.hostname}:8031

**④yarn.resourcemanager.admin.address**

参数解释：ResourceManager对管理员暴露的访问地址。管理员通过该地址向RM发送管理命令等。

默认值：${yarn.resourcemanager.hostname}:8033

**⑤yarn.resourcemanager.webapp.address**

参数解释：ResourceManager对外webui地址。用户可通过该地址在浏览器中查看集群各类信息。

默认值：${yarn.resourcemanager.hostname}:8088

**⑥yarn.resourcemanager.scheduler.class**

参数解释：启用的资源调度器主类。目前可用的有FIFO、CapacityScheduler和FairScheduler。

默认值：

org.apache.hadoop.yarn.server.resourcemanager.scheduler.capacity.CapacityScheduler

**⑦yarn.resourcemanager.resource-tracker.client.thread-count**

参数解释：处理来自NodeManager的RPC请求的Handler数目。

默认值：50

**⑧yarn.resourcemanager.scheduler.client.thread-count**

参数解释：处理来自ApplicationMaster的RPC请求的Handler数目。

默认值：50

**⑨yarn.scheduler.minimum-allocation-mb/yarn.scheduler.maximum-allocation-mb**

参数解释：单个可申请的最小/最大内存资源量。比如设置为1024和3072，则运行MapRedce作业时，每个Task最少可申请1024MB内存，最多可申请3072MB内存。

默认值：1024/8192

**⑩yarn.scheduler.minimum-allocation-vcores/yarn.scheduler.maximum-allocation-vcores**

参数解释：单个可申请的最小/最大虚拟CPU个数。比如设置为1和4，则运行MapRedce作业时，每个Task最少可申请1个虚拟CPU，最多可申请4个虚拟CPU。什么是虚拟CPU，可阅读我的这篇文章：“YARN资源调度器剖析”。

默认值：1/32

**⑪yarn.resourcemanager.nodes.include-path/yarn.resourcemanager.nodes.exclude-path**

参数解释：NodeManager黑白名单。如果发现若干个NodeManager存在问题，比如故障率很高，任务运行失败率高，则可以将之加入黑名单中。注意，这两个配置参数可以动态生效。（调用一个refresh命令即可）

默认值：“”

**⑫yarn.resourcemanager.nodemanagers.heartbeat-interval-ms**

参数解释：NodeManager心跳间隔

默认值：1000（毫秒）

**2）NodeManager相关配置参数**

**①yarn.nodemanager.resource.memory-mb**

参数解释：NodeManager总的可用物理内存。注意，该参数是不可修改的，一旦设置，整个运行过程中不可动态修改。另外，该参数的默认值是8192MB，即使你的机器内存不够8192MB，YARN也会按照这些内存来使用（傻不傻？），因此，这个值通过一定要配置。不过，Apache已经正在尝试将该参数做成可动态修改的。

默认值：8192

**②yarn.nodemanager.vmem-pmem-ratio**

参数解释：每使用1MB物理内存，最多可用的虚拟内存数。

默认值：2.1

**③yarn.nodemanager.resource.cpu-vcores**

参数解释：NodeManager总的可用虚拟CPU个数。

默认值：8

**④yarn.nodemanager.local-dirs**

参数解释：中间结果存放位置，类似于1.0中的mapred.local.dir。注意，这个参数通常会配置多个目录，已分摊磁盘IO负载。

默认值：${hadoop.tmp.dir}/nm-local-dir

**⑤yarn.nodemanager.log-dirs**

参数解释：日志存放地址（可配置多个目录）。

默认值：${yarn.log.dir}/userlogs

**⑥yarn.nodemanager.log.retain-seconds**

参数解释：NodeManager上日志最多存放时间（不启用日志聚集功能时有效）。

默认值：10800（3小时）

**⑦yarn.nodemanager.aux-services**

参数解释：NodeManager上运行的附属服务。需配置成mapreduce\_shuffle，才可运行MapReduce程序

默认值：“”