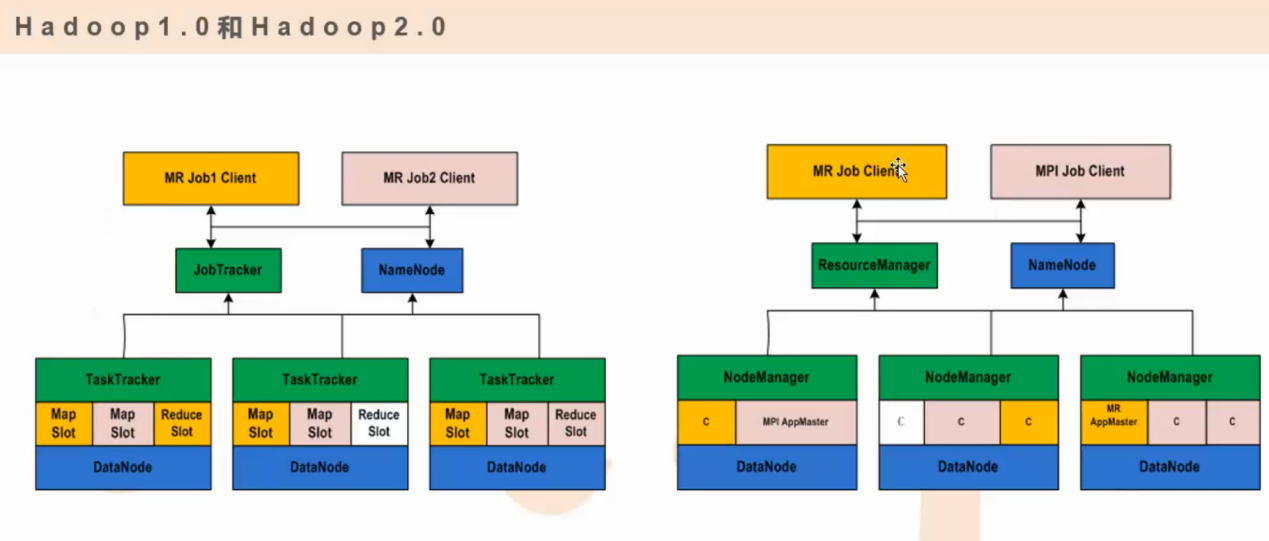
我们来看一下这个图，这个图也是hadoop1.0和hadoop2.0架构的一个差别（如下图1所示）

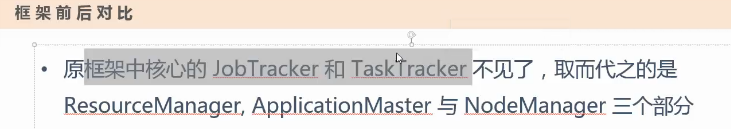
图1：



左边和右边你看一下对比，确实是差别很大对吧？你相当于是这个左边的JobTracker变成了右边的ResourceManger，功能也不一样了，这个ResourceManager的权限只是这个JobTracker的一半，然后原来TaskTracker就变成了一个NodeManager，但是你HDFS这块没有什么变化，HDFS的主仍然是Namanode，在2.0里面也是NameNode，然后存数据HDFS的数据节点在1.0里面是DataNode，在2.0里面也是DataNode，所以这个进程是没有变的。

还有上面提交客户端有一些变化，就是说你之前提交的这个计算框架或者提交这个任务都是这个Mapreduce对吧？那么现在的Yarn不仅仅是Mapreduce，你也可以是MPI，也可以是一个spark等等对吧？然后你在1.0里面你去给任务分配资源的时候，这个资源是一个Slot，这个就是代表了一个cpu，那么JobTracker另外一半对于这种任务级别的一些管理方面的这个master被下放到了各个的从上，就是这个MRIappMaster和MapreduceappMaster和还有一个sparkappMaster这样的一个情况。

我们来看一下架构前后对比，就是说原来框架核心的JobTracker和TaskTracker不见了（如下图2所示）

图2：  


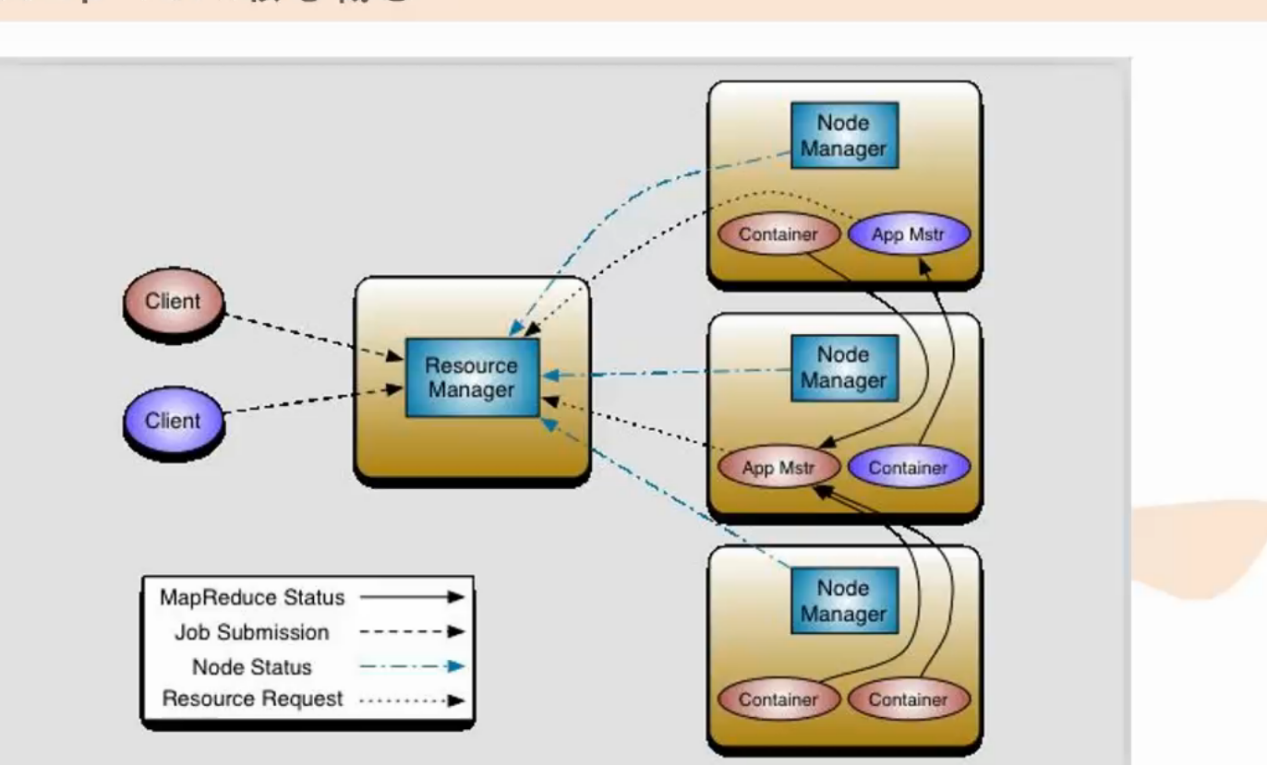
那么取代他们的是RM,AM,NM这三个部分，刚才也说过了这具体之间的逻辑，还有就是这个mapreduce经历了完全的重构，也不再是Hadoop的核心组件了，而是成为了Yarn上的很普通的应用框架，叫做MRV2，也就是mapreduce的第二版本（如下图3所示）

图3：

  
那你看整个架构其实跟之前的1.0就不太一样了，底层没变HDFS，但是你这个左边这块zookeeper，在HDFS也好还是mapreduce也好，那zookeeper在1.0里面是根本没有用得上的，因为zookeeper的诞生是比那个1.0要晚的，所以主和从的一些协调还是仍然用传统任务master，slave那种架构管理起来的，然后自从有了这种分布式锁之后，就让这个集群变得更加简单，然后扩展性就更好了，那所以在整个的2.0里面就把这个zookeeper引入进来。那么这个HDFS上面原来还记得把？在1.0时代里面HDFS上面直接就是mapreduce，那么现在不一样了，现在上面直接是一个Yarn系统，在这个Yarn上面才支持了一些各种各样的集成框架。

然后下面这个图，其实这个图我们刚刚基本上讲过的了，所以在这里我就不在花时间去讲了（如下图4所示）

图4：



那么接下来针对这几个细节点然后再做进一步的分析（如下图5所示）

图5：  


我们对每一个组件再重点的说一下，原来1.0里面在接收客户端的请求是有jobTracker对吧？现在RM来处理客户端的请求，也就是说你去提交了一个任务提交了一个作业提交了一个应用首先你要和你的RM进行联系，通过你客户端上提交这么一个作业，但是你提交作业的时候就是你已经在你的配置里面已经写好了，大概需要申请多少资源，然后这个RM会根据你的这个资源的要求然后并且它会评估一下你这个任务到底需要用多少资源，然后给你去找合适的从节点NodeManager上去收集相应的各个从节点上的状态信息，各个从节点上的资源的利用使用情况，然后去调度整个过程，分配一个Container作为一个App Mstr，首先你先把这个AM先伺候好对吧？你先把AM启动起来，比如说这三个进程（如下图6所示）

图6：  


这三个进程，有哪个进程启动以后不能停掉？有哪个进程是可以随时关闭和停止？RM是必须得启动起来，因为你这个是核心，然后NM也是必须要启动起来，AM可以没有，就是你提交任务的时候AM才会帮你启动，如果你任务执行完成之后，AM这个进程就自动消失，因为你任务没有了，你就不需要监控了，就是你任务存在的时候你的AM才启动。

好了刚才也说过了RM就是一个纯粹的一个调度器，实际这个RM就是一个独立的守护进程，运行在专有的机器上，就是说这个机器要求各方面的配置还是需要一些高级一点对吧？毕竟你这个RM是整个系统的核心对吧？但是你从这个上图4的架构上看的出来，你这个RM有问题对吧？这个RM是一个单点，是一个master，但是你这个单点的话你虽然原来这个jobTracker也是一个单点对吧？那么现在RM也是单点，虽然你是工作没有以前那么重了，就是系统比较稳定了，但是你必然还会存在挂掉的风险对吧？那这怎么办呢？在这个Hadoop2.4版本之后，就是它这个系统可以支持你RM的高可用，我们继续看这个图（如下图7）

图7：



其实这个图里面通过zookeeper的方式当你的RM主挂了的话，通过zookeeper的方式来协调然后调度进行一些个故障转移，然后RM的工作也比之前那个JobTracker也是轻多了是吧？仅仅是一个资源调度了，那么也不会耗太多的资源，所以从单点的角度运转也比之前稳定多了。

好了还有一个心跳，我们之前是默认的TaskTracker每三秒钟会去向JobTracker去上报一个心跳来汇报它当前这个从节点的一些状态，在Yarn里面一样，NodeManages也是这样，上报它的状态给你的RM，也是通过心跳达到了一个应该是被监控的作用，就是说我被你管理，然后要经常汇报我的状态，然后让你监控我是吧？

另外一个RM是有一个可插拔的调度器组件这个Scheduler，这个刚才说过了是不是？它不负责任何应用程序的监控和状态跟踪，它仅仅是一个资源调度，也不保证你应用程序失败或者硬件失败情况下对task的重启，那么这个事也不是Scheduler去干的，所以RM它主要的职责就是调度，因为你这个集群是大家共用的对吧？你每个人可能会提交一个任务来去竞争你的这个资源对不对？但是这个RM不关心每一个应用程序的一个状态管理，调度却只处理这个应用程序整体的一个资源分配，它目的就是忽略内部程序的一些状态，那之前的1.0里面对map和reduce等是做了静态分配，但是由于有这个调度器它相当于让你的这个Yarn将整个的集群看作一个统一的资源池，让你这个蛋糕切的更加的合理一点。

好了，我们看一下NM，NM跟RM之间的联系是通过心跳来保证的，这个刚才也已经说了哈！然后NM是来负责本地和用资源的监控（如下图8所示）

图8：

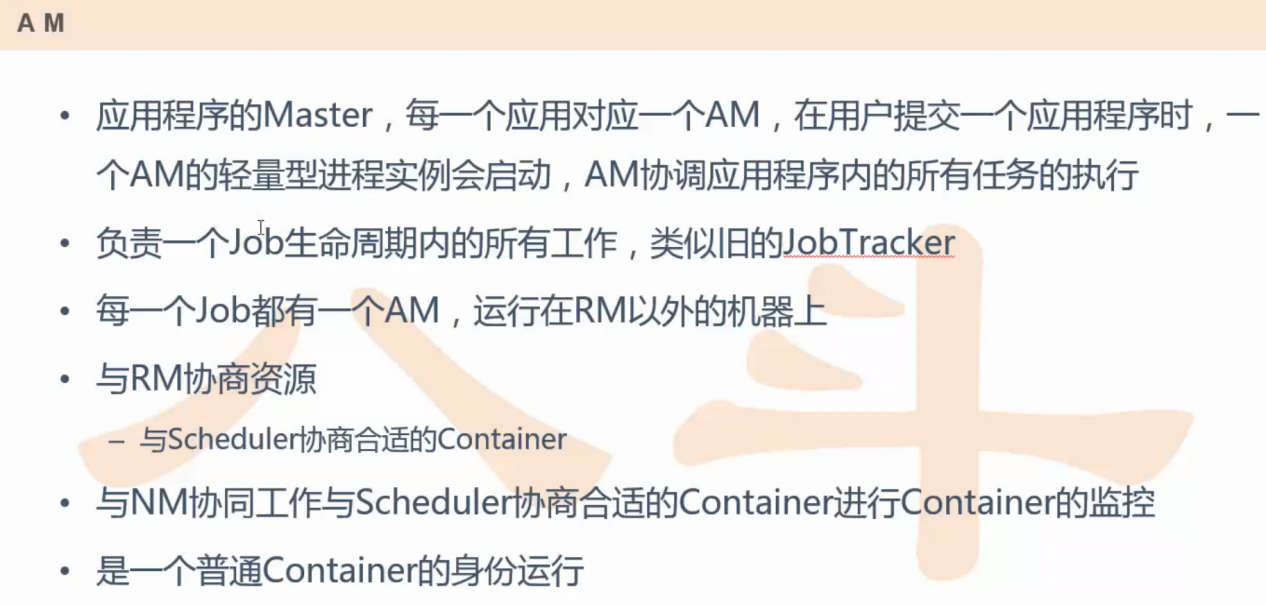


NodeManager是用来做一个本地可用资源的一个监控，然后RM依赖于你的NM来维护整个集群的一个全局视图，也就是说你这个RM有点类似于山高皇帝远是不是？但是你这个每一个地区都得安排一个省长或者是地区的一个负责人，那么这个就是一个NM，那你这个NM要定期的上报给这个中央这个皇上（RM），然后这个RM才能够了解到我这个祖国这么大，我这个祖国每一个角落原来是这样子的一个情况，然后在这个RM虽然它可以不用出远门，这个RM就可以了解他所管辖的这个大好江山是一个怎么样的情况，就是这个NM主动上报情况，这个RM才能知道全景的这么一个视图。

另外一个就是说RM我有这个任务请求，比如说你来了一个客户，这个客户就是想来租房的，那我这个RM就找几个这个地区的销售然后直接分配给你，然后你这个客户就直接去找销售，然后销售就给你相应的Container，这就是任务请求嘛对吧？那么这个NM是接收并处理来自AM的Container启动，停止各种需求，这个Container永远是这个NM来管理的，然后RM管理NM，NM管理Container，负责启动应用程序的Container并且监控他们，并且把这个状态上报给RM。然后总的来说就是在单节点上进行资源管理和任务管理。

那我们继续看一下AM（如下图9所示）

图9：

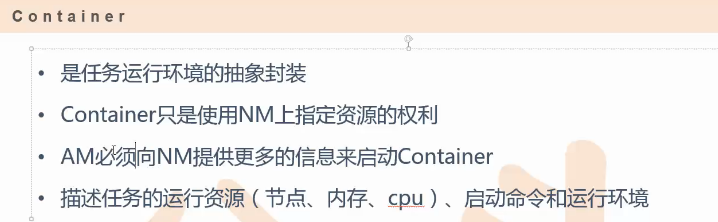


AM是每一个用户作业的主进程，跟资源没有关系，它只是一个作业层面的一个进程，负责你整个作业的一个运转周期，就是说当你RM将一些职责转义给了AM，就是原来你这个AM你不是负责作业的嘛对吧？本来这个AM这个功能本身是属于人家RM的，人家RM是从这个JobTracker里面演化过来的，你这个RM是未来把这个权限下放，就下放给了AM，把你这个任务方面级别的工作就是让你AM来去做了，然后这样做的目的就是说让你这个Yarn具有更好的扩展性，然后AM需要利用多台服务器的处理能力完成一个作业，就是你提交一个任务，这个任务不可能在一个机器上运行的嘛对吧？那肯定需要多个机器多个节点并行的完成你的作业，那为了实现这么一个目标就把你的AM启动起来了，你AM就开始向RM发出资源请求了，就是你这个客户首先在你当地的一个总部，你这个总部已经建好了，就相当于是这个美容院这个权力中心这个管理层已经有窝可住了对吧？然后这个时候这帮管理者才能够开始想办法才能开一些分店，那么这个时候AM就像RM请求了，发出一些资源请求，然后这些资源请求是以Container的方式来运转AM的运用。

这个时候大家估计有点懵了对吧？其实就是AM利用多台机器的处理能力完成一个作业，为了实现该目标，AM向RM申请资源，资源就是Container，集群以Container的形式运转AM的应用，就是说你这个AM，你想开5个美容院的这个分店或者开10个，那就是每一个美容院就是一个Container对吧？就是你的分店对吧？就是真正去接触客户，你这个AM是不接触客户的。AM申请资源是跟RM申请，然后RM分配资源让NM来管理。

那么运行中的Container进程，可能是一个map也可能是一个reduce，并且你这个Container起来之后，你这个Container也不能只管执行，你这个Container也得与AM通信，报告任务的状态和健康信息。然后NM和AM进行协调工作，然后与Scheduler进行协调资源，然后找到相应的Container，然后对这个Container进行监控，然后AM就相当于是一个普通的Container。

Container就是对你任务环境的一个封装，然后Container只是由NM上，NM就是一个从节点就是一台机器，Container就是在这台机器上启动起来的，然后这个进程是由你的NM来去启动的，然后并且你NM虽然我申请好了这些Container资源，但是我资源申请好了之后，就相当于是说你要开分店，比如你在某一个商场里面你开了一个美容院，这个美容院你得去装修对吧？这个装修得有一定的风格是吧？得必须按照我这个客户的要求统一包装，不能说你这个NM你想怎么搞就怎么搞，所以这个AM也得向这个NM提供相应的信息，然后才启动一个合适的Container（如下图10所示）

图10：  


所以这里面要注意，Container是NM来启动的，所以很多人是会搞混，总有人会认为Container是由AM来启动的，其实不是的，Container就是NM来启动的，所以从这个基本上层面来看这个Container它是一个单节点上的内存CPU，什么磁盘的一些物理集合，那单节点上可以有多个Container，就是你一个从节点上可以有多个Container，比如说内存是512M，1个CPU，不管内存是512还是一个CPU还是一个你在单台节点上的一个最小的一个单位，你不可能是1m一个Container或者是2m一个Container，所以起码你得有一个单位对吧？你就像这个HDFS上每一个数据的一个单位是64M，你必须统一下这个数据的单位，那么这里面你可以通过相应的配置去配，然后内存比如说512一个CPU，然后相当于这是一个最小的一个单位了，对于一个资源来说是最小的一个单位了，不能再往下做一个细分了，然后这个Container有可能是一个内存512M，也有可能是一个CPU这么一个情况，然后这个时候就是说一个任务如果需要多个Container，那么通过最小容量单位的多个Container来组成，那么这里面的CPU是一个虚拟的CPU，这个CPU是Yarn引入的概念，那么为什么存在一个Yarn虚拟的概念呢？就是说你这每台机器上有可能这个每一个机器的CPU本来它的性能就不一样，然后每个CPU具有的计算能力也是不一样，有可能你这个服务器上它是用了一个因特尔的CPU，那个机器可能用了一个其他类型的CPU是吧？比如说某一个物理的CPU计算能力可能是另外一个物理CPU的两倍，那这个时候如果你通过一个物理CPU去多配置几个CPU来拟补这样的一个差异，那当然这个CPU的个数也要和你的物理CPU的个数相同。

就是说你这个CPU的个数虽然我说的是一个虚拟的CPU，但是你不是说我一共有10台机器，我每台机器是个四核，我这整个集群就40个CPU，那我去跑任务的时候我不可能配400个，我再怎么虚拟我也不能虚拟出400个是不是？就是你这个虚拟的CPU它的个数也是和你物理的CPU个数也要保持一致。这个是要通过相应的配置去配的。

还有一个刚才我们也说过了，就是在那个Yarn里面有个优化，就是说当MR执行结束，可以释放map的资源Container，这个时候你的reduce就开始启动了，那我可以用释放出来的Container来请求更多的reduceContainer，这是不是感觉集群的利用率提高了是不是？

好了我们继续看一下Yarn框架对于整个的mapreduce框架的一个优势（如下图11所示）

图11：  

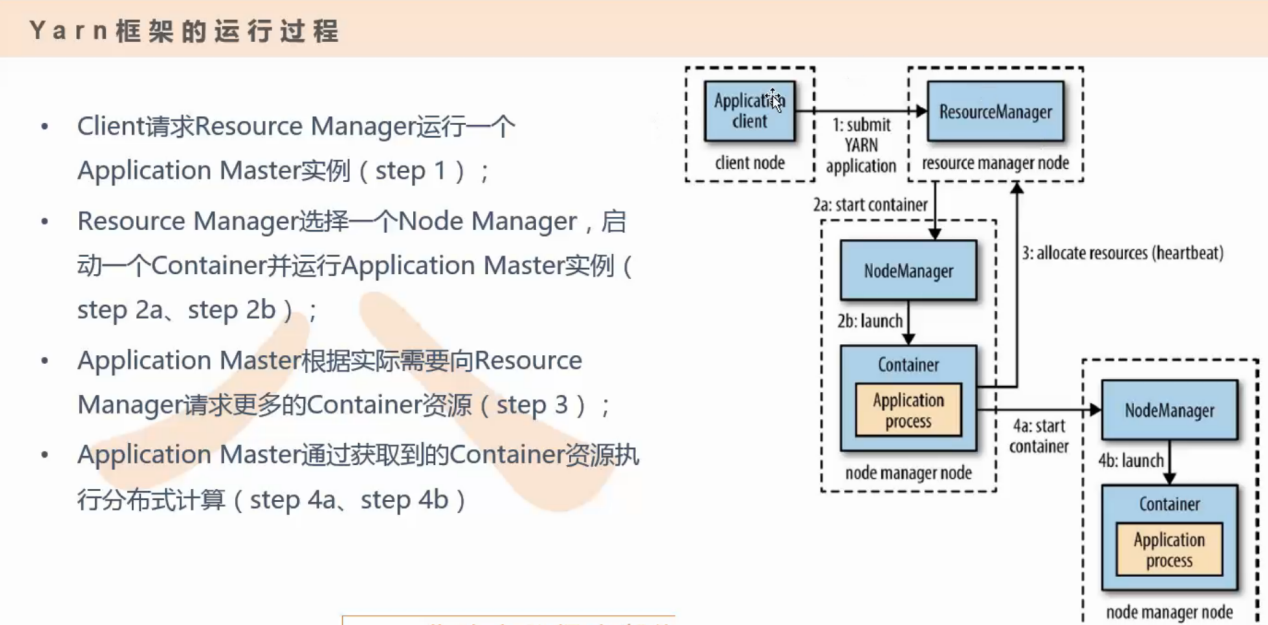

Yarn就是减少JobTracker的一些资源消耗，然后让检测每一个Job的任务就进行分布式化，更安全更优美。这个就相当于是分列出了一个AM了对吧？

AM上面可以跑你的mapreduce任务也可以跑你的spark任务，所以这就是可变更的意思

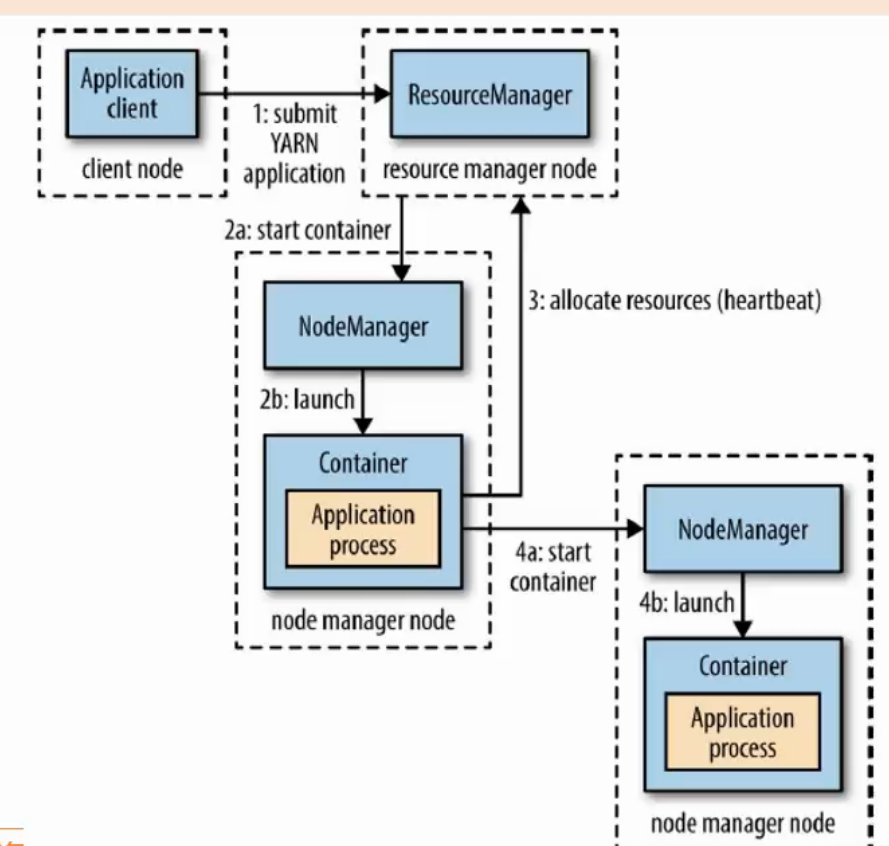
那么这些我们之前也说过了对吧？在这里就不在多说了。

那么我们看一下整个Yarn的一个运行过程(如下图12所示)

图12：



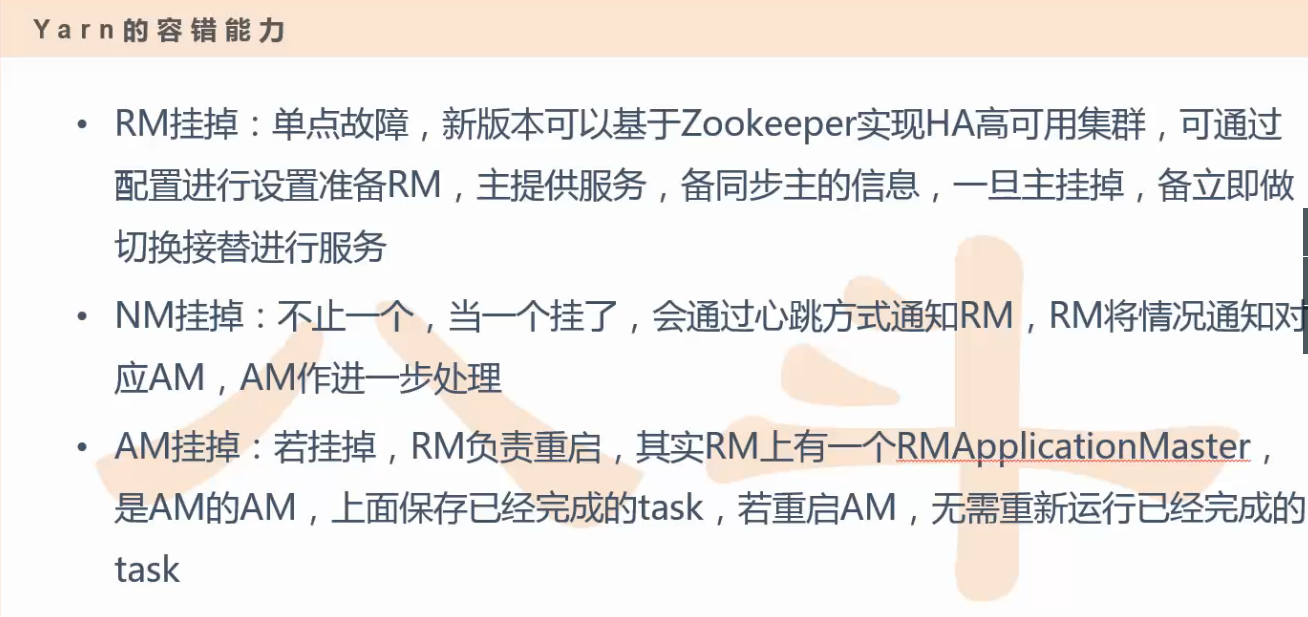
首先你作为客户端来说，你作为客户端去提交一个任务（如下图13所示）

图13：  


这个客户端client在上图13中的Application client这里，这个client首先你想提交一个任务，那这个任务第一步先和RM建立联系，让你的RM启动一个AM，那这个时候RM就选择了一个NM，然后这个NM就帮你启动了一个Container，并且在这个Container内部帮你启动一个Application process对不对？然后这个时候AM启动之后，这时候AM就可以根据它的这个任务的需求去要求资源的申请与RM建立一定的关系，这是第三步。AM根据它实际的需求和RM申请更多的IMG_256Container资源，这个Container是要存它即将要用来跑实际计算任务的那些容器，那这个时候RM同样就会给它分配相应的资源，然后找到了NM并且启动相应的Container，那这个时候AM就通过相应获取了Container一些资源的一些控制权力，就可以开始对这些的资源进行一个任务的管理。这个流程应该还是比较清晰的对吧？

然后Yarn的容错能力（如下图14所示）

图14：



就是说这个Yarn如果是这个RM挂掉的话就是单点故障了对吧？之前我们说过这个RM它有单点的一个风险对吧？好了那么在新版本里面我们之前已经说过了是吧？Zookeeper是可以实现了一个HA的一个高可用集群，可以通过相应的配置可以设置一个从的RM，就是做一个主从切换这么一个配置，主提供服务的时候，备份就同步主的信息，一旦主挂掉之后，那么这个备份就立即切换到了主的这么一个状态来对外提供服务是把？相当于主被做了一个容灾。

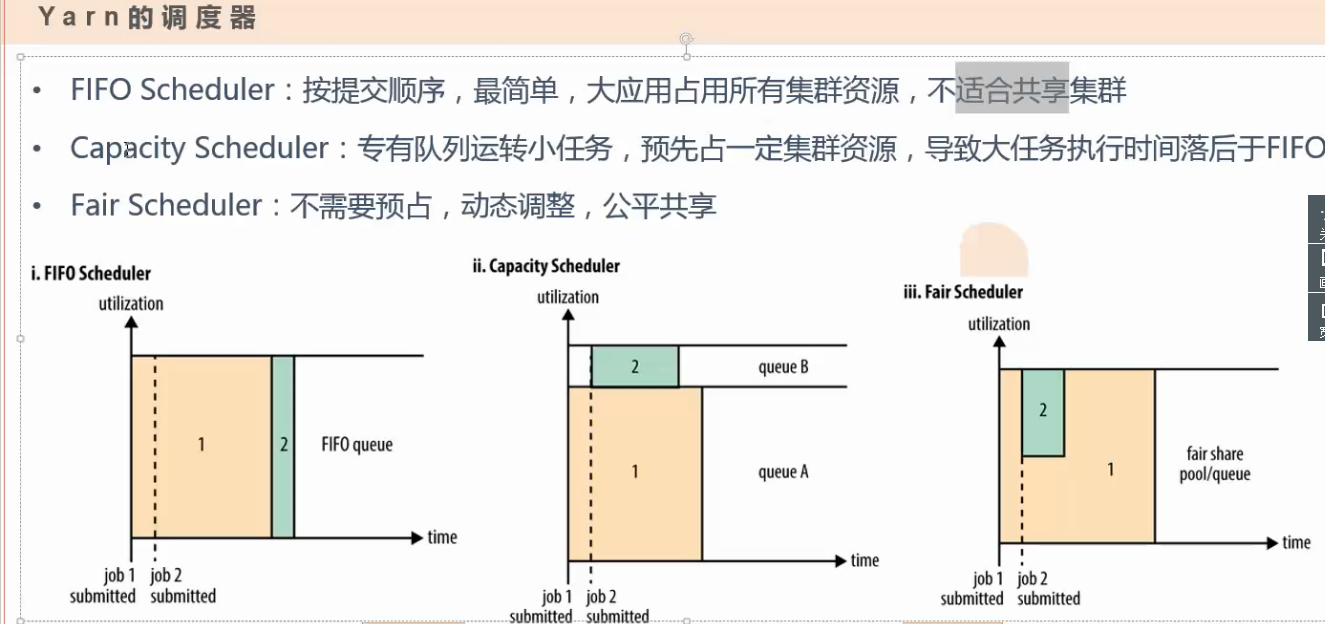
NM挂掉了，因为你NM是一个从节点上的一个进程，而且这个NM不仅仅是一个进程，当你有一个NM挂了的话，那它会通过心跳的方式通过RM，RM就知道你这个NM挂了，那这个时候你RM，比如说你这个NM上正在跑任务，然后你这个机器挂了，你任务也就中断了对吧？你任务还没执行完，那这个时候RM就得告诉你AM，AM就做进一步的处理。

那么你的任务失败的话，就会再帮你重新申请资源。

那么如果是AM挂掉的话，那么这个时候RM来负责重启，其实这个AM和这个RM也是保持一个通信的关系，就在RM上有一个这么一个服务就是RMApplicationMaster，这个的话会容易和你的AM混淆是吧？这个是你的AM的AM。就是说你AM就是一个普通的任务的话，那我相当于是对这个AM的任务又做了一个进一步的监控，让上面的保存已经完成的task，就是说如果你这个AM这个任务已经执行了百分之50了，突然你这个AM挂掉了，那我这个RM肯定是要求把你的AM重新启动对吧？但是已经完成的这个task，也就是已经完成任务的那百分之50就不需要再重新执行了，只需要把那些没有完成的任务执行起来就可以了。

还有一个调度器（如下图15所示）

图15：

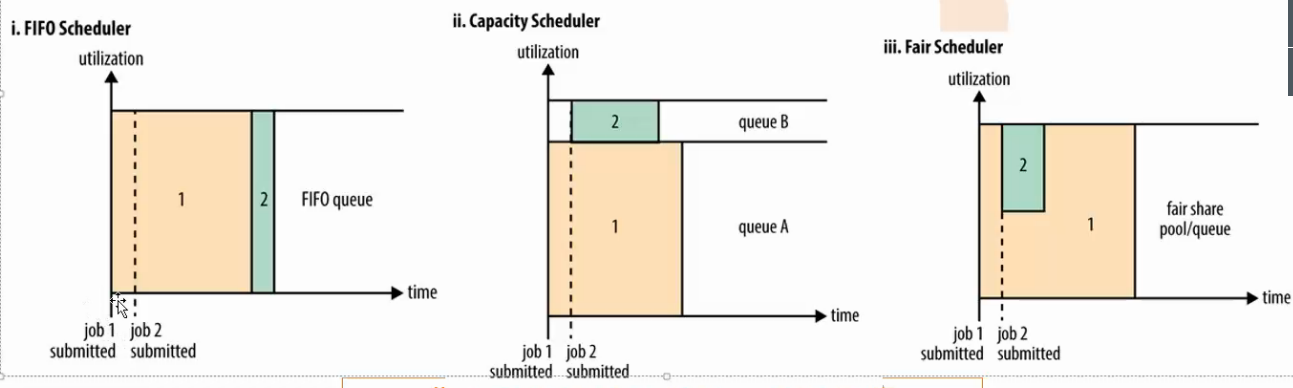


调度器这块是什么意思呢？就是比如说你去银行排队，比如说你是拿着普通的一个普通个人用户，那么你必须是叫号对不对？你这来得早就先对你提供服务是吧？后来的就后服务。你可能去取个钱或者存个款，那这个在银行的一个处理过程中，其实这个根本花不了多长时间对吧？那像有一些业务是比较花时间的，你比如说要开一个网银等，那么这就会比较慢了对吧？毕竟要填表啊等等要求才能办理到一个网银，那么这个就是有的人提交的任务就是一个大任务，有的人提交的任务是小任务，那么还有一个是属于那种VIP会员用户，那么它就有一个单独的专属通道，我不跟你这个普通用户共享一个队列，我有一个单独的队列，即使这个队列没人用，你普通个人用户也不允许占用，所以这就是一个银行调度策略。

那么我们在大数据也有调度策略，最简单的调度策略就是我刚刚最一开始介绍的那种先来先服务，后来后服务的策略，那么这个就是FIFO Scheduler，就是按照你提交的顺序最简单，但是这里虽然这个任务调度起来很简单，但是这里面的问题也很严重，因为可能有一个大的任务，那这个大的任务就会把整个的集群给你阻塞掉，因为它需要占很多的资源，那你后面即使排队的都是一些小的任务，那没办法，你只能等这个大的任务执行完之后，才能执行小的任务，所以你如果是通过这样的一个很简单的调度方式去分配你集群的资源的话，它是不利于你的集群资源共享的。

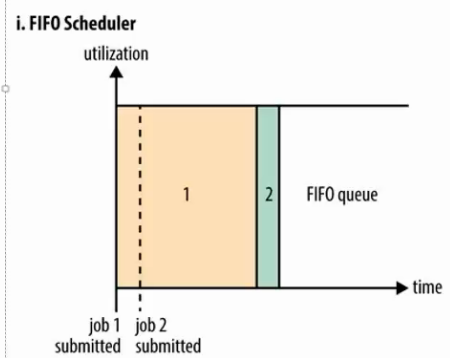
那为了进一步的这个集群能够共享起来，那就有了接下来两种策略，就好比下面的三张图（如下图16所示）

图16：



那我左边第一个张图FIFO Scheduler这个横坐标是一个时间（如下图17所示）

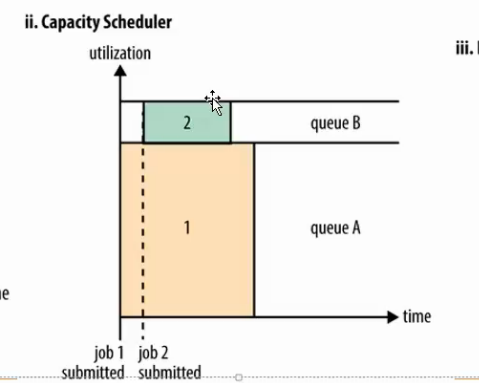
图17：



我第一个job1直线这一块是我第一个任务提交的一个作业，第二个job2虚线这块是我第二个任务提交的时候，因为我这个属于一个先来先服务，那我这个虚线第二个任务提交上来了，但是我没办法，我运行不了，我只能等你这个Job1执行完之后，我的job2第二个任务才能执行。

第二张图就是Capacity Scheduler（如下图17所示）

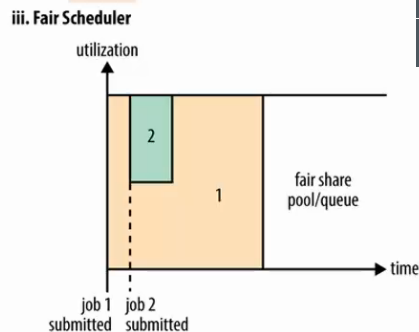
图17：



刚刚所说的VIP通道，就是说他有专门的一个队列，这个专门的队列是只运转你的小任务的，而且这个队列是提前要占一定的集群资源，就是你的大的任务不允许进这个队列的，我这个队列是专门给那些小任务来运转的，就比如上图16的第二张图的job1开始是请求了启动了，我job2提交了没关系，我有自己一个类似于VIP通道是吧？对我来说没有影响，然后我2也可以同步来运行，但是有一个问题，就是说我用了FIFO Scheduler的方式，我Job1我可能用了10个小时完成了我这个工作对不对？那我同样的这个任务在这个Capacity Scheduler里面你觉得10个小时能执行完吗？肯定执行不完对吧？那为何执行不完呢？因为你这一开始FIFO Scheduler的这个1里面是把所有的集群资源给独占了，那么Capacity Scheduler是它有一部分资源你是不能用的，所以你这里面10个小时肯定是执行不完的，所以导致了你同样的一个大的任务你的执行时间要落后于你的FIFO Scheduler，那么其实Capacity Scheduler也有一定的问题对吧？虽然可以让你的资源共享，但是总觉得运转起来性能还是不够高效。

那么最后一种公平的调度方式Fair Scheduler （如下图18所示）

图18：



就是说你这个资源我也不需要预占，你既然1来了，你一开始我这个集群上没有其他任何任务，那你1来了，那么1来了以后，资源我全都给你，那么这个时候我2来了，我2来了我不能让你等对吧？如果让你等的话，那不是就变成了FIFO Scheduler了对吧？我为了这个集群这个大家都能够公平共享这样的资源的话，那我就需要在这个1里面就把它那执行完的那些个资源能够空闲出来，然后给你这个2同步的给它那个均匀分配过去，所以这个时候这个2也可以运行起来，然后1又不影响，目的就是在这个资源在两个用户之间进行一个平衡。

就是说你这个2提交的时候，你这个资源肯定获得的资源不能像Capacity Scheduler，你来了以后我就马上执行，所以其实这个里面它也是对资源获取有一定的延迟，你必须要等这个1里面有相应的任务释放出来Container了，你这个小任务才能用，那如果你的小任务执行完之后，你就把你自己的资源再吐出来，然后这个1如果是它要是启动比如一些reduce等等，那么它可能就会吸收回去，就是这么一个过程。