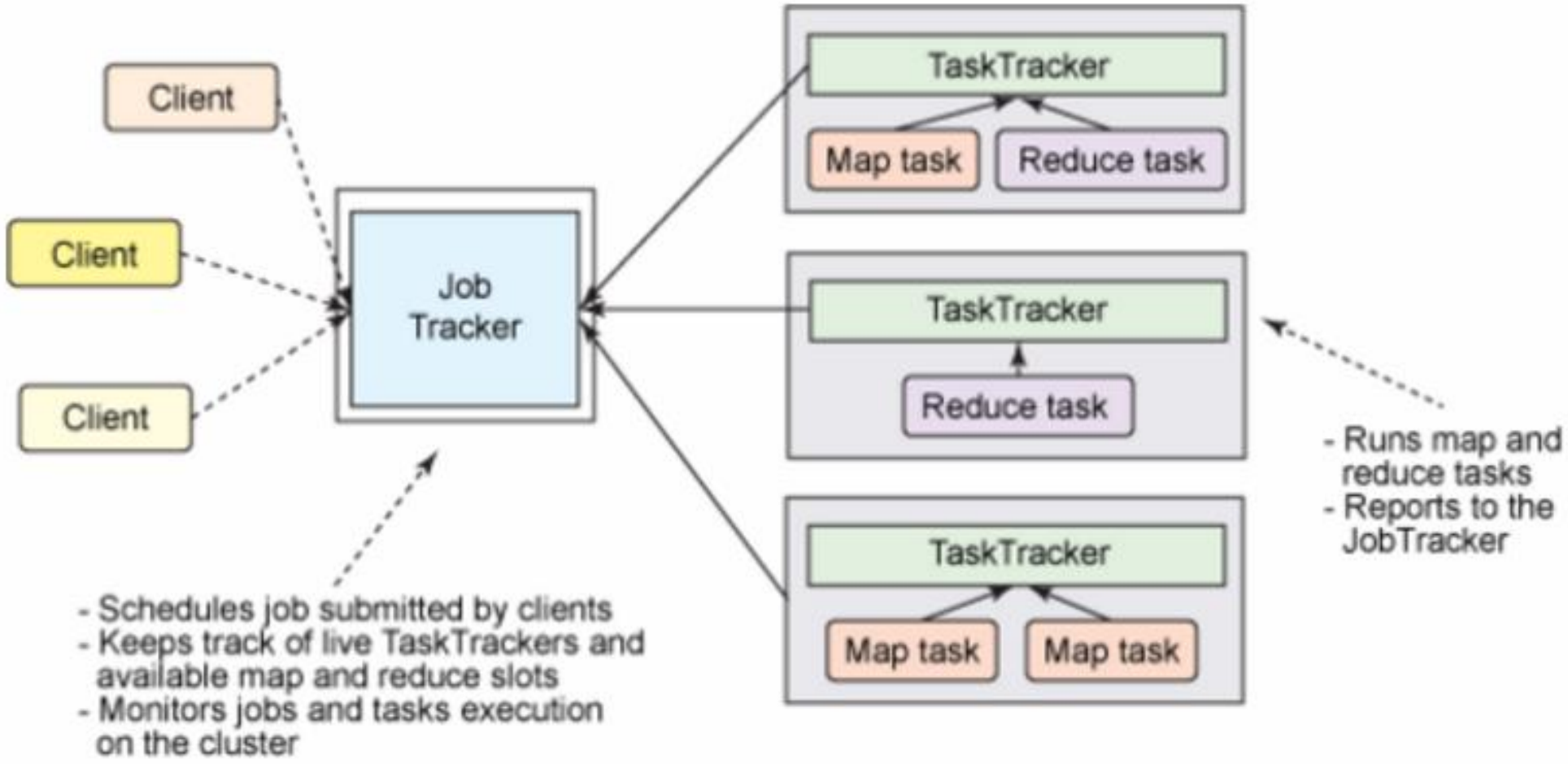
YARN学习总结

安思宇

经典MapReduce（MapReduce1）

- 客户端，提交MapReduce作业
- Jobtracker，协调作业的运行。Jobtracker是一个Java应用程序，它的主类是JobTracker
- Tasktracker，运行作业划分后的任务。Tasktracker是Java应用程序，它的主类是TaskTracker
- 分布式文件系统（HDFS），用来在其他实体间共享作业文件。

Apache Hadoop 的经典版本 (MRv1)



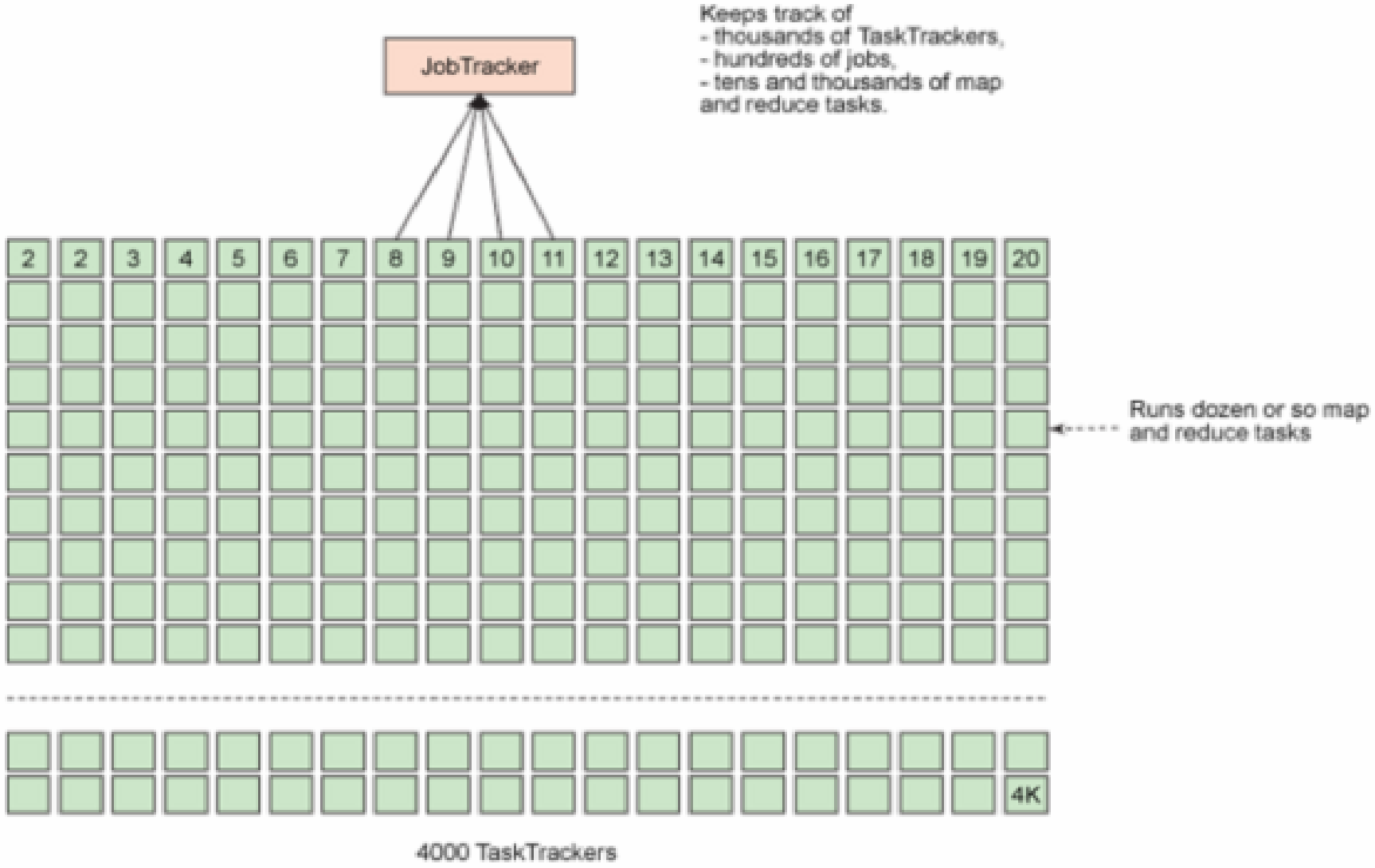
经典MapReduce设计思路

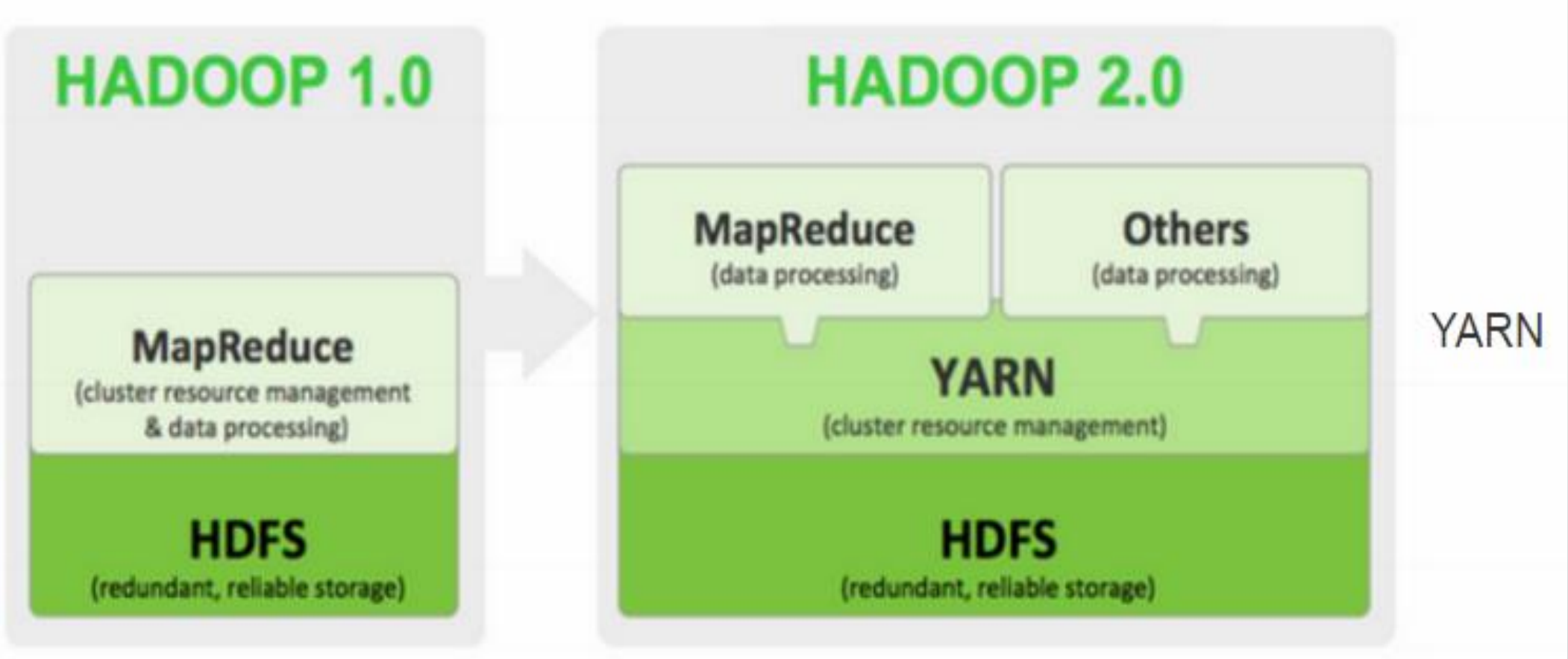
- 首先用户（JobClient）提交了一个job，job的信息会发送到JobTracker中，JobTracker是MapReduce框架的中心，他需要与集群中的机器定时通信（heartbeat），需要管理哪些程序应该跑在哪些机器上，需要管理所有job失败、重启等操作。
- TaskTracker是MapReduce集群中每台机器都有的一个部分，它做的主要事情是监视自己所在机器的资源情况。
- TaskTracker同时监视当前机器的tasks运行状况。TaskTracker需要把这些信息通过heartbeat发送给JobTracker，JobTracker会搜集这些信息以为新提交的job分配运行在哪些机器上。

经典MapReduce的主要问题

- JobTracker是MapReduce的集中处理点，存在单点故障
- JobTracker完成了太多任务，造成了过多的资源消耗，当MapReduce的job非常多的时候，会造成很大的内存开销，潜在来说，也增加了JobTracker fail的风险，因此业界总结经典MapReduce只能支持4000节点主机的上限。
- 在TaskTracker端，以map/reduce task的数目作为资源的表示过于简单，没有考虑到cpu和内存的占用情况，存在两个大内存消耗的task被调度到一块的情况。
- 在TaskTracker端，把资源强制划分为map task slot和reduce task slot，当应用程序不适合划分为map和reduce时效率较低
- ……

大型 Apache Hadoop 集群 (MRv1) 上繁忙的 JobTracker





Hadoop2.0中，YARN负责管理MapReduce中的资源（内存，CPU等）并且将其打包成Container，这样可以精简MapReduce，使之专注于其擅长的数据处理任务，无需考虑资源调度。

YARN会通过分配Container来给每个应用提供处理能力，Container是YARN中处理能力的基本单元，是对内存和CPU的封装。

YARN（MapReduce2）组成

- 提交MapReduce作业的客户端
- YARN资源管理器，负责协调集群上计算资源的分配
- YARN节点管理器，负责启动和监视集群中机器上的计算容器（container）
- MapReduce应用程序master负责协调运行MapReduce作业的任务它和MapReduce任务在容器中运行，这些容器由资源管理器分配并由节点管理器进行管理
- 分布式文件系统（HDFS）用来于其他实体间共享作业文件

Hadoop YARN框架原理及运作机制

- YARN的基本思想是将JobTracker的两大主要职能：资源管理、作业的调度/监控拆分为两个独立的进程：一个全局的ResourceManager（RM）和与每个应用对应的ApplicationMaster（AM）
- ResourceManager和每个节点上的NodeManager（NM）组成了全新的通用操作系统，以分布式的方式管理应用程序
- ResourceManager拥有为系统中所有应用的资源分配的决定权。ApplicationMaster负责与ResoureManager协商资源，以及与NodeManager协同工作来执行和监控各个任务

RM:Scheduler & ApplicationsManager

- Scheduler根据容量、队列等限制条件（如每个队列分配一定的资源，最多执行一定量的作业等），将系统中的资源分配给各个正在运行的程序
- Scheduler是一个纯调度器，它不再从事任何与具体应用程序相关的工作，比如不负责监控或者跟踪应用的执行状态等，也不负责重新启动

因应用执行失败或者硬件故障而产生的失败任务，这些均交由应用程序相关的AppliacationMaster完成。调度器仅根据各个应用程序的资源需求进行资源分配

- 资源分配单位用一个抽象概念:“资源容器”(Container)表示，Container是一个动态资源分配单位，它将内存、CPU、磁盘、网络等资源封装在一起从而限定每个任务使用的资源量。

- ASM（ApplicationsManager）主要负责接收作业，协商获取第一个容器用于执行AM和提供重启失败AM container的服务。

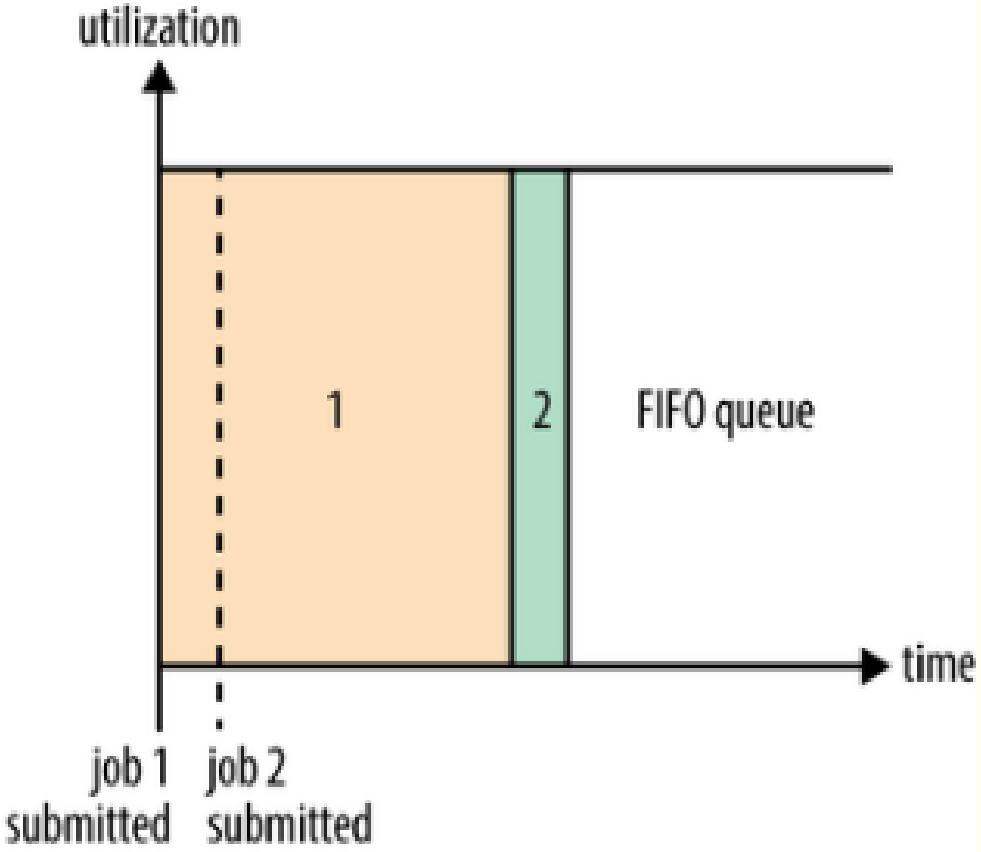
调度器（Scheduler）

- YARN自带Fifo Scheduler，Capacity Scheduler和Fair Scheduler三种常用资源调度器。
- 其中默认实现为Capacity Scheduler，Capacity Scheduler实现了资源更加细粒度的分配，可以设置多级队列，每个队列有一定的容量，然后对每一级队列分别再采用合适的调度策略（如FIFO）进行调度。
- 可以实现YARN的资源调度接口Resource Scheduler，然后修改yarn-site.xml配置项yarn.resourcemanager.scheduler.class

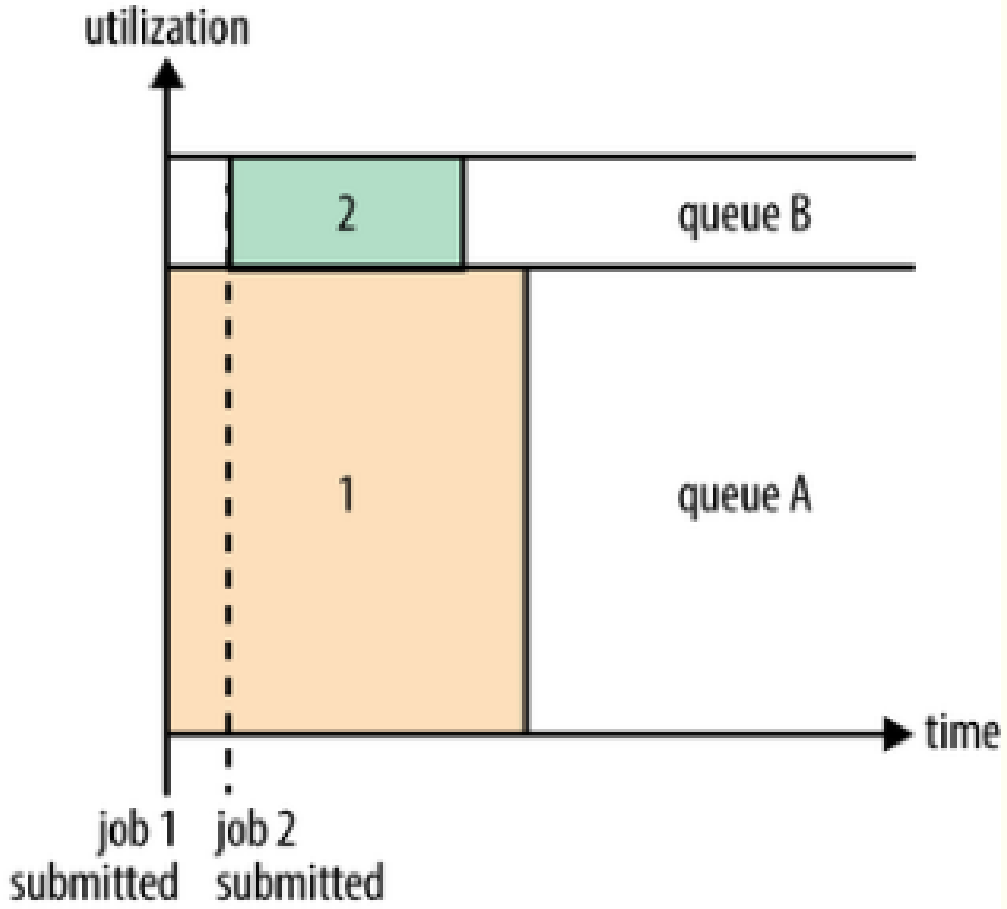
- FIFO Scheduler把应用按提交的顺序排成一个先进先出队列，在进行资源分配的时候，先给队列中最头上的应用进行分配资源，待最头上的应用需求满足后再给下一个分配。
- FIFO不适用于共享集群，在共享集群中大的应用可能会占用所有集群资源，会导致其他应用被阻塞
- 共享集群适合采用Capacity Scheduler和Fair Scheduler，它们允许大任务和小任务在提交的同时获得一定的资源

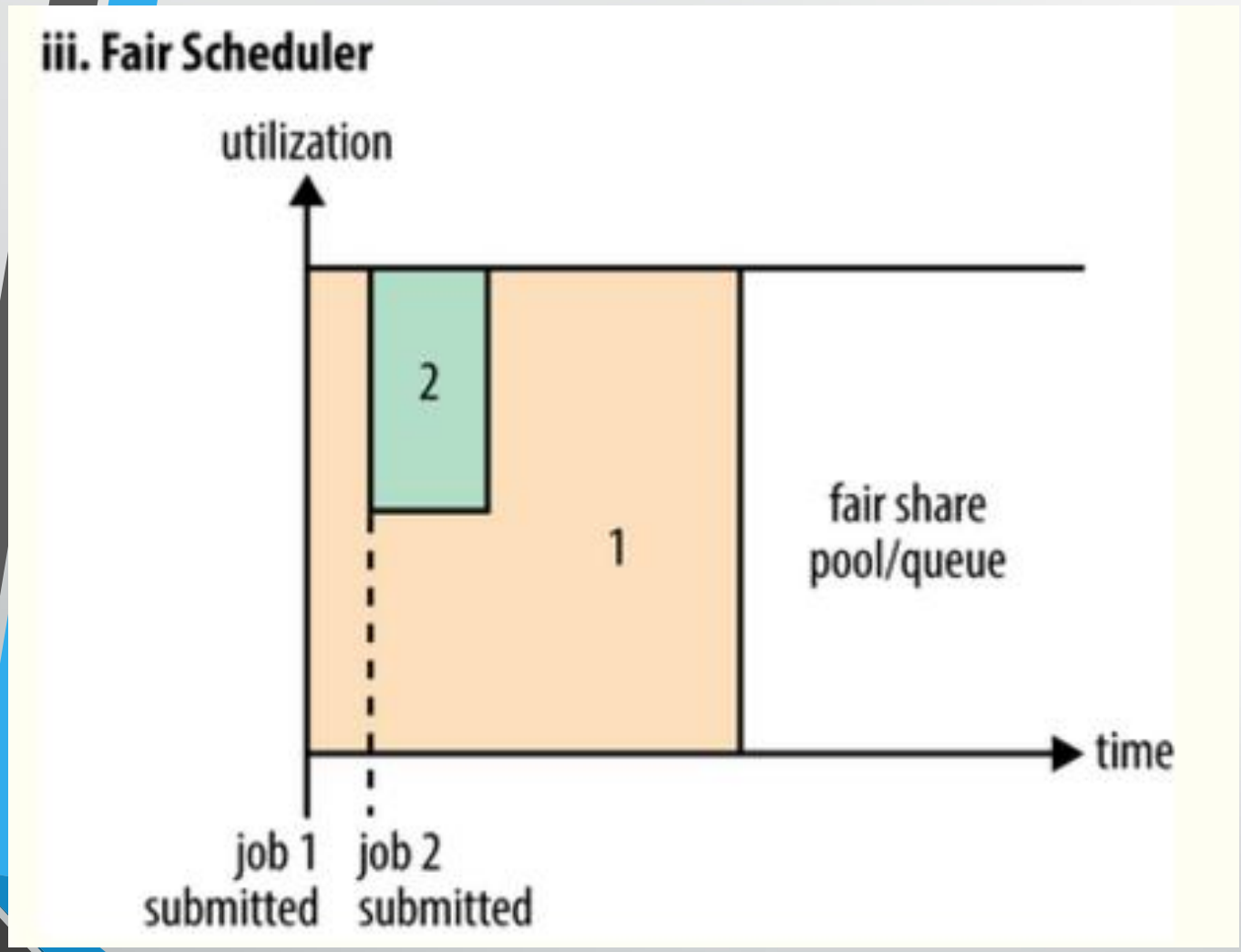
Yarn 调度器对比图:

i. FIFO Scheduler



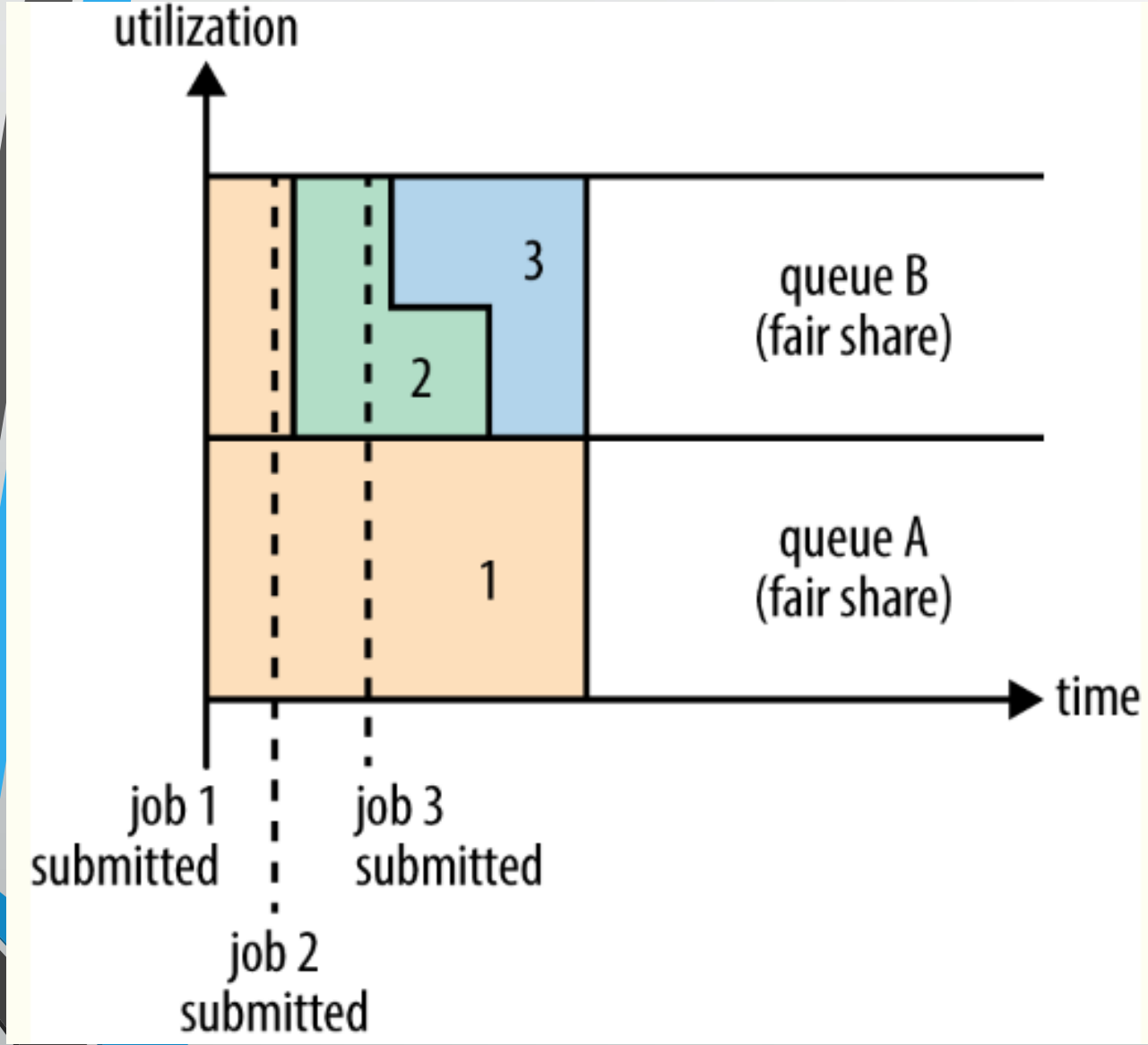
ii. Capacity Scheduler





在Capacity调度器中，有一个专门的队列来运行小任务，但是为小任务专门设置一个队列会预先占用一定的集群资源，这就导致大任务的执行时间会落后于使用FIFO调度器的时间。

在Fair调度器中，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动态调整系统资源。当一个大job提交时，只有一个job在运行，获得了所有集群资源；当有小任务提交后，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。



Fair调度器的设计目标是为所有的应用分配公平的资源，假设用户A和B，他们分别拥有一个队列。当A启动一个job而B没有任务时，A会获得全部集群资源；当B启动一个job后，A的job会继续运行，不过两个任务会各自获得一般的集群资源。此时如果B再启动第二个job并且其他job还在运行，则它将会和B的第一个job共享B这个队列的资源，即B的两个job会用1/4的集群资源，而A的job仍然用于集群一半的资源，因此资源在两个用户间平等共享

- 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。
- 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FIFO
- 弹性队列：如果一个job使用不了整个队列的资源，就可以分配其他job，如果队列的资源不够用了，capacity调度器还可以分配额外的资源给这个队列。

调度器处理6种类型的事件：

- NODE_REMOVED:表示集群中移除了一个计算节点
- NODE_ADDED:表示集群中增加了一个计算节点
- APPLICATION_ADDED:表示ResourceManager收到一个新的 Application
- APPLICATION_REMOVED:表示一个Application运行结束
- CONTAINER_EXPIRED:当资源调度器将一个Container分配给某个ApplicationMaster后，如果该ApplicationMaster在一定时间内没有使用该Container，则资源调度器会对该Container回收再分配
- NODE_UPDATE：ResourceManager收到NodeManager通过心跳机制汇报的信息后，会触发一个NODE_UPDATE事件，由于此时可能有新的Container得到释放，因此该事件会触发资源分配。

ApplicationMaster

- 协调集群中应用程序执行的进程，每个程序都有自己特有的ApplicationMaster，负责与ResourceManager协调资源
- ApplicationMaster实际上是特定框架库的一个实例，负责与ResourceManager协商资源，并和NodeManager协同工作来执行和监控Container以及它们的资源消耗
- 用户作业生命周期的管理者，用户应用程序驻留的地方
- RM只负责监控AM，在AM运行失败时候启动它，RM并不负责AM内部任务的容错，这由AM来完成
- 可以请求最小值整数倍的资源的Container

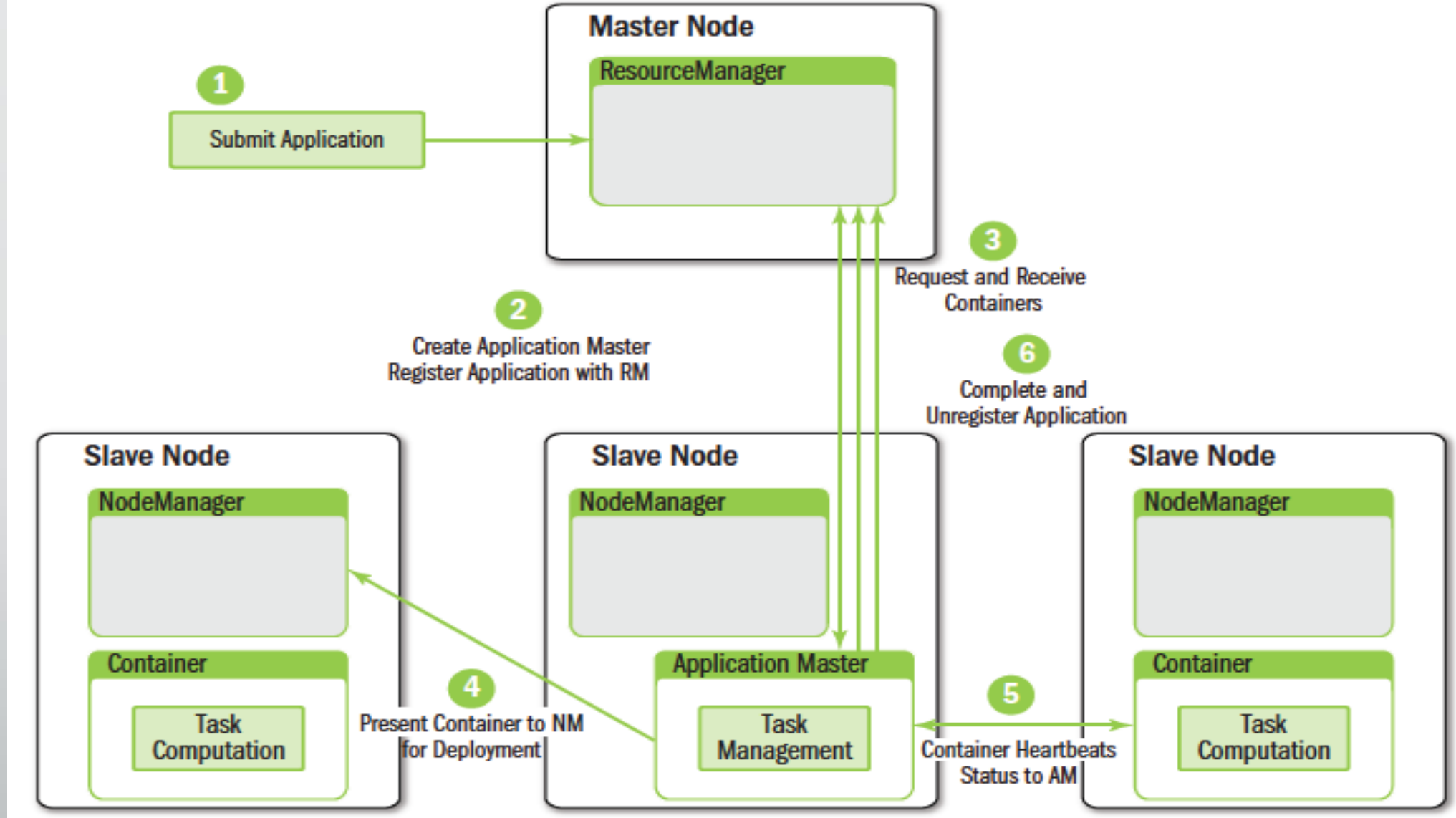


Figure 7.6 Application Master interactions with YARN

NodeManager

- NodeManager是与每台机器对应的从属进程（slave），负责启动应用程序的Container，监控它们的资源使用情况（CPU，内存，磁盘和网络），并且报告给ResourceManager
- ResourceManager和NodeManager之间通过心跳来通信，NodeManager负责本地资源的监控，故障报告以及Container生命周期的管理
- 接收并处理来自ApplicationMaster的Container启动/停止等请求。
- 中央的ResourceManager和所有NodeManager创建了集群统一的计算基础设施

- Container的管理是NodeManager的核心功能
- NodeManager接收来自AplicationMaster的启动或者停止Container的请求，对Container令牌进行授权（确保应用程序可以适当使用ResourceManager给定资源），管理Container执行依赖的库，监控Container的执行过程
- 所有Container都通过Container Launch Context（CLC）来描述。CLC包括环境变量，依赖的库，下载库文件以及Container自身运行需要的安全令牌等

Container

- Container是单个节点上如RAM，CPU核和磁盘等物理资源的集合。单个节点上可以有多个Container
- ApplicationMaster可以请求任何Container来占据最小容量的整数倍的资源
- Container代表了集群中单个节点上的一组资源（内存，CPU），由NodeManager监控，由ResourceManager调度
- 每一个应用程序从ApplicationMaster开始，它本身就是一个Container。一旦启动，ApplicationMaster就与ResourceManager协商更多的Container，并且在运行过程中，可以动态的请求和释放。

一个应用程序所需的Container分为两大类：

I. 运行ApplicationMaster的Container：这是由ResourceManager(向内部的资源调度器)申请和启动的，用户提交应用时，可指定唯一的ApplicationMaster所需的资源；

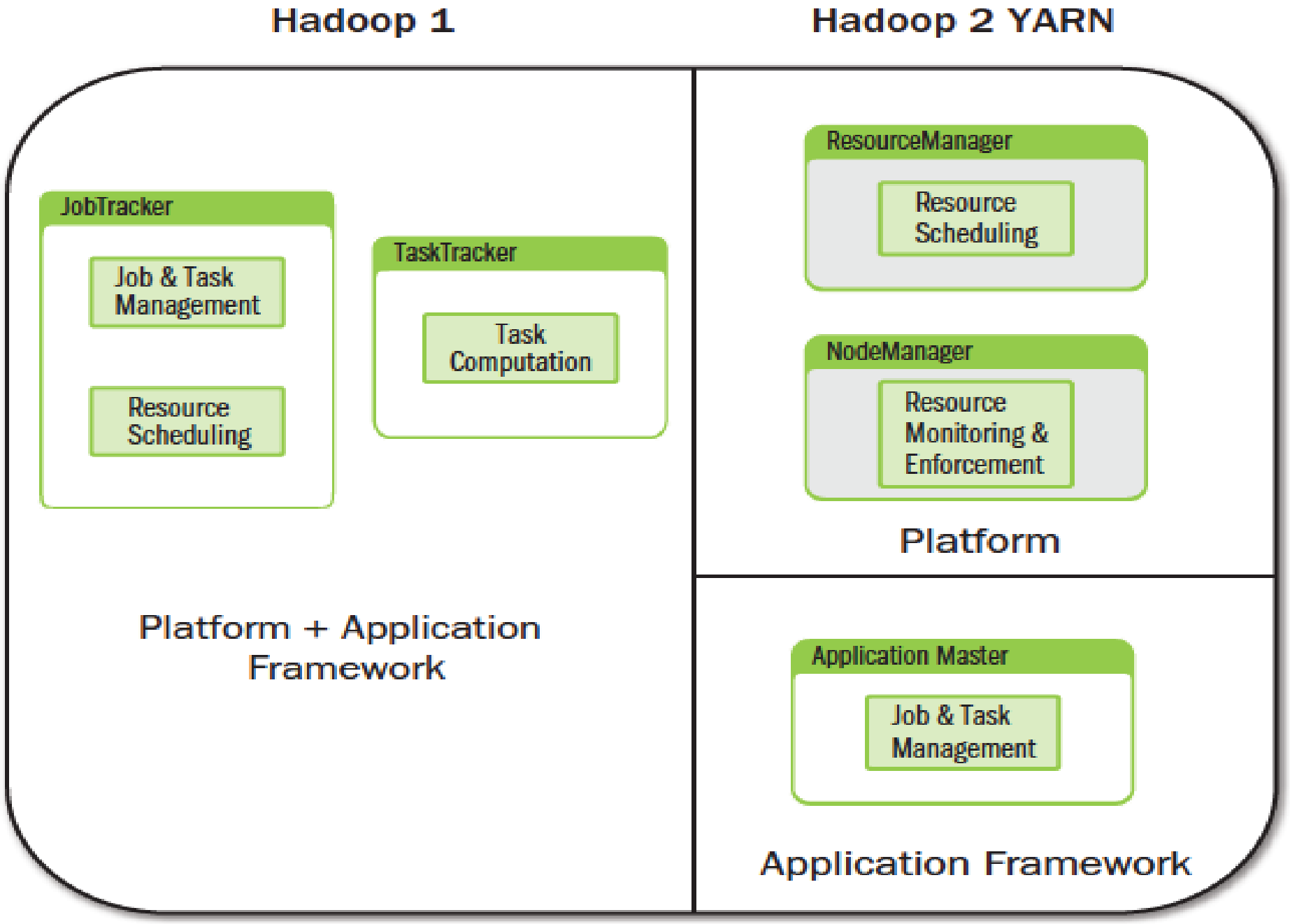
II. 运行各类任务的Container：这是由ApplicationMaster向ResourceManager申请的，并由ApplicationMaster与NodeManager通信以启动。

- 以上两类Container可能在任意节点上，它们的位置通常是随机的，即AplicatoinMaster可能与它管理的任务运行在一个节点上。

资源模型

- 第一层，ResourceManager中的资源调度器将资源分配给各个ApplicationMaster

- 第二层，ApplicationMaster再进一步将资源分配给它的内部任务


Hadoop 1
Hadoop 2 YARN
JobTracker
Job & Task Management
Resource Scheduling
TaskTracker
Task Computation
Platform + Application Framework
ResourceManager
Resource Scheduling
NodeManager
Resource Monitoring & Enforcement
Platform
Application Master
Job & Task Management
Application Framework

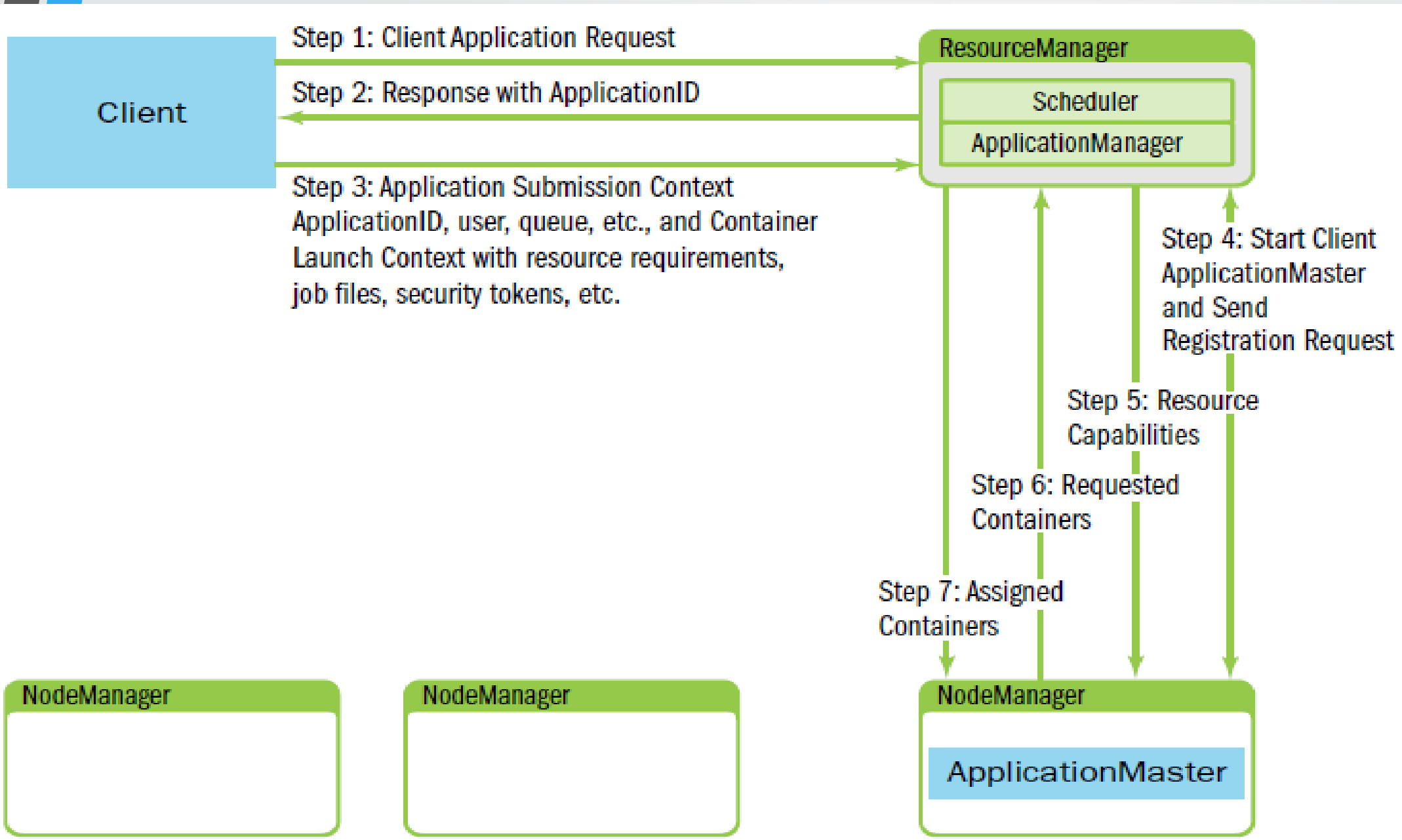


Figure 4.2 Example client resource request to ResourceManager

客户端资源请求过程

1.客户端必须先通知RM它要提交一个应用程序

2.RM在应答中给出一个ApplicationID

3.客户端使用“Application Submission Context”发出响应，包含了ApplicationID、用户名、队列以及其他启动ApplicationMaster所需的信息。“Container Launch Context”(CLC)也会发给ResourceManager。CLC提供了资源需求（内存/CPU）、作业文件、安全令牌以及节点上启动ApplicationMaster需要的其他信息。

4.当ResourceManager收到来自客户端的应用程序提交上下文，它就会调度一个可用Container，这个Container就是ApplicationMaster，如果没有适用的Container，这个请求必须等待。如果找到了合适的Container，ResourceManager就会联系相应的NodeManager并启动ApplicationMaster。同时建立ApplicationMaster的RPC端口和用于跟踪的URL，用来监控应用程序的状态。

5.ResourceManager会发送关于集群的最小和最大容量信息。在这一点上，ApplicationMaster必须决定如何使用当前可用的资源

6.基于ResourceManager的可用资源报告，ApplicationMaster会请求一定量的Container。该请求可以包括最小资源整数倍的Container

7.ResourceManager会基于调度策略，尽可能最优地为ApplicationMaster分配Container资源，作为资源请求的应答发给ApplicationMaster

随着作业的执行，ApplicationMaster将心跳和进度信息通过心跳发给ResourceManager。在这些心跳中，ApplicationMaster还可以请求和释放Container。当作业结束时，ApplicationMaster向Resourcemanager发出一个Finish消息，然后退出

ApplicationMaster与Container管理器的通信

- 此时ResourceManager已经将分配NodeManager的控制权交给了ApplicationMaster。ApplicationMaster将独立联系其指定的NodeManager并提供Container Launch Context(CLC)，CLC包括环境变量、远程存储依赖文件、安全令牌以及启动实际进程所需的命令。
- 当Container启动时，所有的数据文件，可执行文件以及必要的依赖文件都被拷贝到节点的本地存储上了。依赖文件可以被运行中的应用程序的Container之间共享。

- 一旦所有的Container都被启动，ApplicationMaster就可以检查它们的状态。ResourceManager不参与应用程序的执行，只处理调度和监控其他资源。ResourceManager根据需要可以命令NodeManager杀死Container，当Container被杀死后，NodeManager会清理它的本地工作目录。
- 作业结束后，ApplicationMaster通知ResourceManager该作业成功完成，然后ResourceManager通知NodeManager聚集日志并且清理Container专用的文件。如果Container还没有退出，NodeManager也可以接收指令杀死剩余的Container（包括ApplicationMaster）

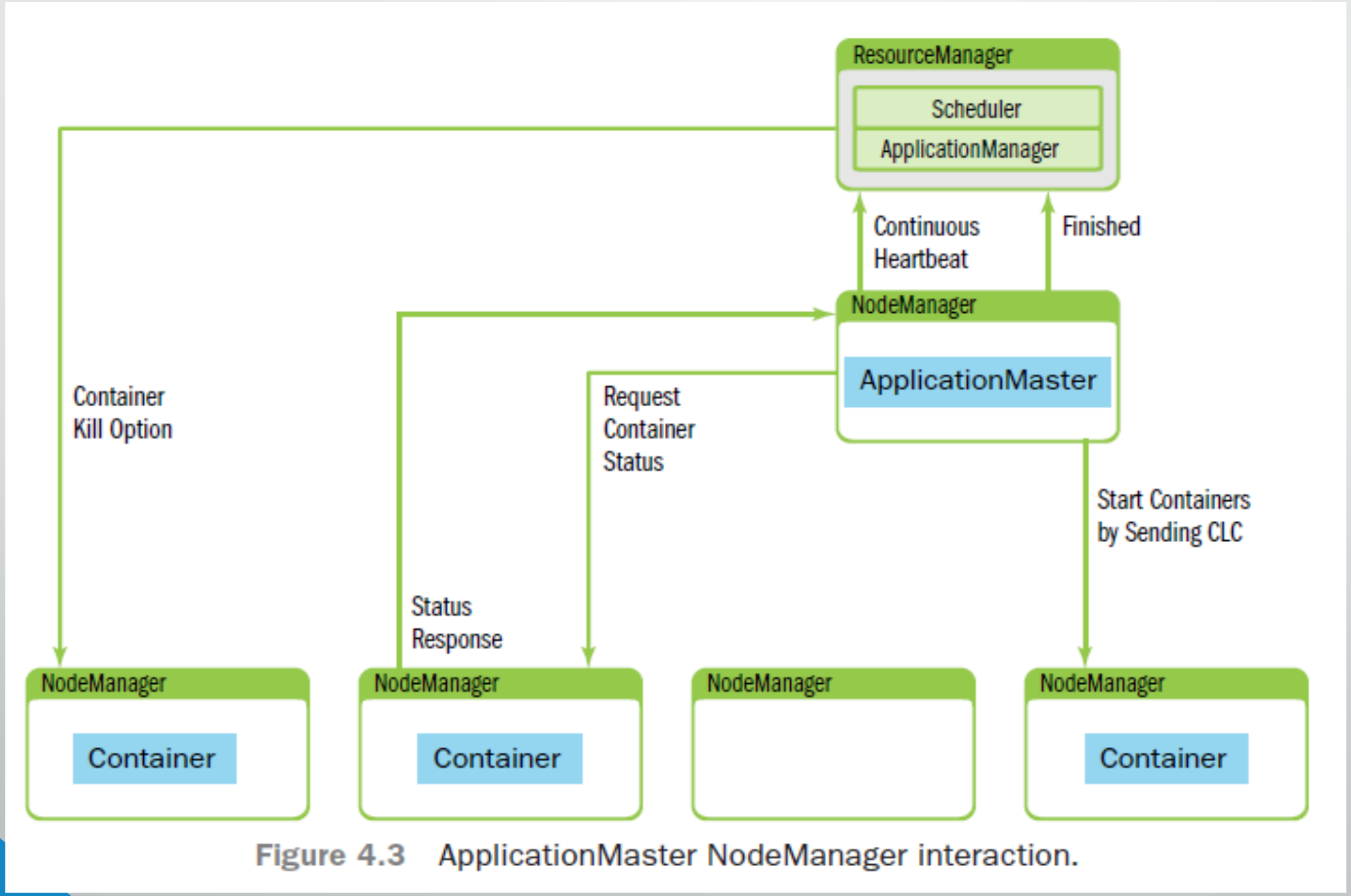
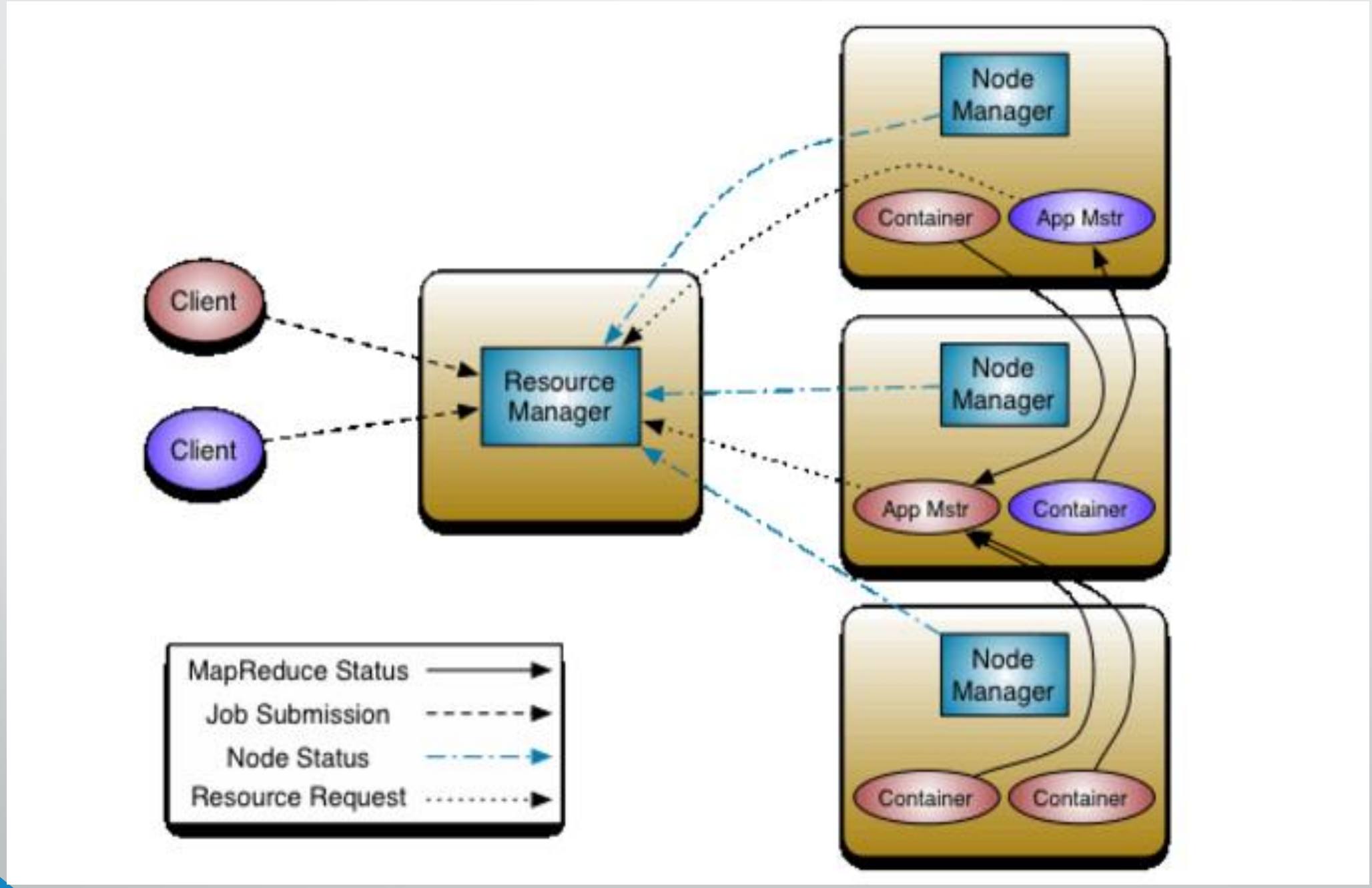


Figure 4.3 ApplicationMaster NodeManager interaction.


Client
Client
Resource Manager
Node Manager
Container
App Mstr
Node Manager
App Mstr
Container
Node Manager
Container
Container
MapReduce Status
Job Submission
Node Status
Resource Request

YARN容错

- ResourceManager
  - ✓存在单点故障
  - ✓基于ZooKeeper实现HA
- NodeManager
  - ✓失败后，RM将失败任务告诉对应AM
  - ✓AM决定如何处理失败的任务
- ApplicationMaster
  - ✓失败后，由RM负责重启
  - ✓AM需处理内部任务的容错问题
  - ✓RMAppMaster会保存已经运行完成的Task，重启后无需重新运行

管理应用程序的依赖文件-本地化

- YARN为应用程序提供了本地化其依赖文件的能力
- 当启动一个Container时，ApplicationMaster可以指定该Container需要的所有文件，这些文件都应该被本地化。
- YARN会负责指定文件的本地化，并且隐藏所有安全拷贝、管理以及后续的删除等引入的复杂性。

- Localization：Localization是拷贝/下载远程资源到本地文件系统的过程。资源被拷贝到本地机器上后可在本地访问，而不需要总是访问远程资源
- LocationResource：LocalResource代表运行Container所需的文件库。NodeManager负责在启动Container之前将这些资源本地化
  - 适合做LocalResource的有：启动Container所需的库文件，如jar等；Container启动后所需的配置文件，静态字典文件等
  - 不适合的有：未来可能被外部模块更新，并且当前Container要跟踪变化的共享文件，应用程序本身要直接更新的文件，应用程序要与外部服务共享更新信息的文件

- 一旦文件从远程位置被拷贝到NodeManager的本地磁盘，它就失去了除URL之外所有与原始文件的联系。对远程文件的任何修改都不再被跟踪，因此，为了避免不一致的问题，YARN会让依赖于被修改的远程文件的Container失败
- LocalResource三种可见性：
  a. PUBLIC-可以被任何用户的Container访问
  b. PRIVATE-被节点上同一用户的应用程序共享。LocalResource被复制到特定用户的私有缓存
  c. APPLICATION-被节点上同一个应用程序的Container，LocalResource被复制到应用程序专有的LocalCache
- 生命周期：
  a. PUBLIC和PRIVATE-只在磁盘容量紧张时删除
  b. APPLICATION：应用程序结束后立即删除

YARN的内存分配和管理

- ResourceManager
- ApplicationMaster
- NodeManager
- Container：运行map/reduce task的容器

RM

- RM的内存资源配置，配置的是资源调度相关
- RM1：yarn.scheduler.minimum-allocation-mb,分配给AM单个容器可申请的最小内存
- RM2：yarn.scheduler.maxmum-allocation-mb,分配给AM单个容器可申请的最大内存

NM

- NM的内存资源配置，配置的是硬件资源相关
- NM1：yarn.nodemanager.resource.memory-mb，节点最大可用内存
- NM2：yarn.nodemamager.vmem-pmem-ratio，虚拟内存率，默认2.1

RM1，RM2的值均不能大于NM1的值

NM1可以计算节点最大Container数量，max(Container)=NM1/RM1

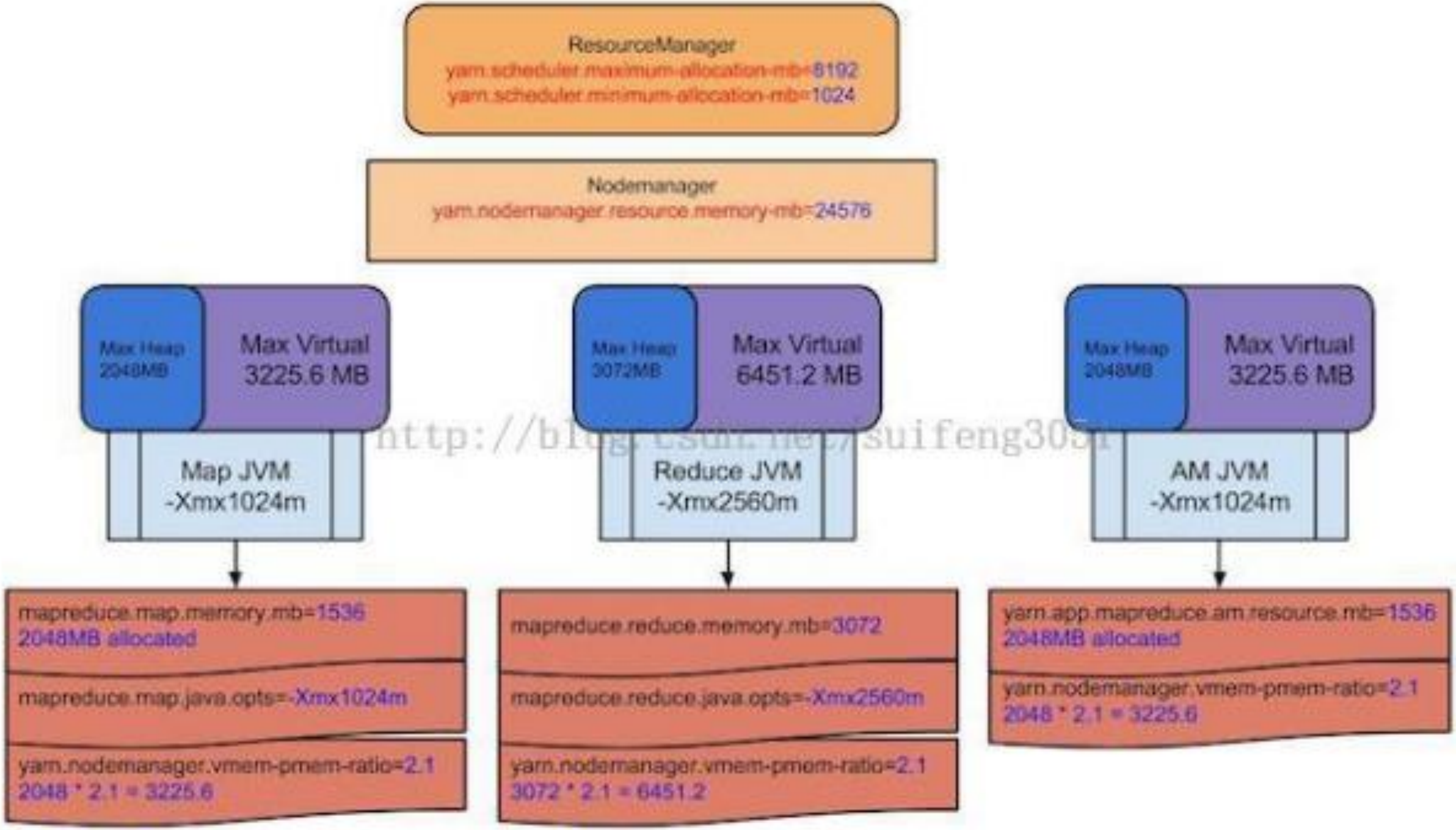
一旦设置不可动态改变

AM

- 内存配置相关参数，配置的是任务相关
- AM1：mapreduce.map.memory.mb 分配给map Container的内存大小
- AM2：mapreduce.reduce.memory.mb 分配给reduce Container的内存大小
- AM3：mapreduce.map.java.opts 运行map任务的jvm参数，如-Xmx，-Xms等
- AM4：mapreduce.reduce.java.opts 运行reduce任务的jvm参数，如-Xmx，-Xms等选项

AM1和AM2应该在RM1和RM2两个值之间，AM2的值最好为AM1的两倍，两个值可以在启动时改变

AM3和AM4应该在AM1和AM2之间



- 下面开始，AM参数mapreduce.map.memory.mb=1536MB，表示AM要为map Container申请1536MB资源，但RM实际分配的内存却是2048MB，因为yarn.scheduler.minimum-allocation-mb=1024MB，这定义了RM最小要分配1024MB，但是实际分配的内存更大，所以由规整化因子，
- AM参数mapreduce.map.java.opts=Xmx 1024m，表示运行map任务的jvm内存为1024MB，因为map任务要运行在Container里面，所以这个参数的值略小于mapreduce.map.memory.mb=1536
- NM参数yarn.nodemanager.vmem-pmem-radio=2.1，这表示NodeManager可以分配给map/reduce Container 2.1倍的虚拟内存，按照上面的配置，实际分配给map Container容器的虚拟内存大小为2048*2.1=3225.6MB，实际用到的内存如果超过这个值，NM就会kill掉这个map Container，任务执行过程中就会出现异常。

- AM参数mapreduce.reduce.memory.mb=3072MB，表示分配给reduce Container的容器大小为3072MB，而map Container的大小分配的是1536MB，从这也看出，reduce Container容器的大小最好是map Container大小的两倍。
- NM参数yarn.nodemanager.resource.mem.mb=24576MB，这个值是表示节点分配给NodeManager的可用内存，也就是节点用来执行yarn任务的内存大小。这个值要根据实际服务器内存大小来配置，比如hadoop集群机器的内存是128GB，就可以分配其中的80%给yarn，也就是102GB。
- RM的两个参数分别是1024MB和8192MB，分别表示分配给AM Container的最大值和最小值

结束

谢谢！