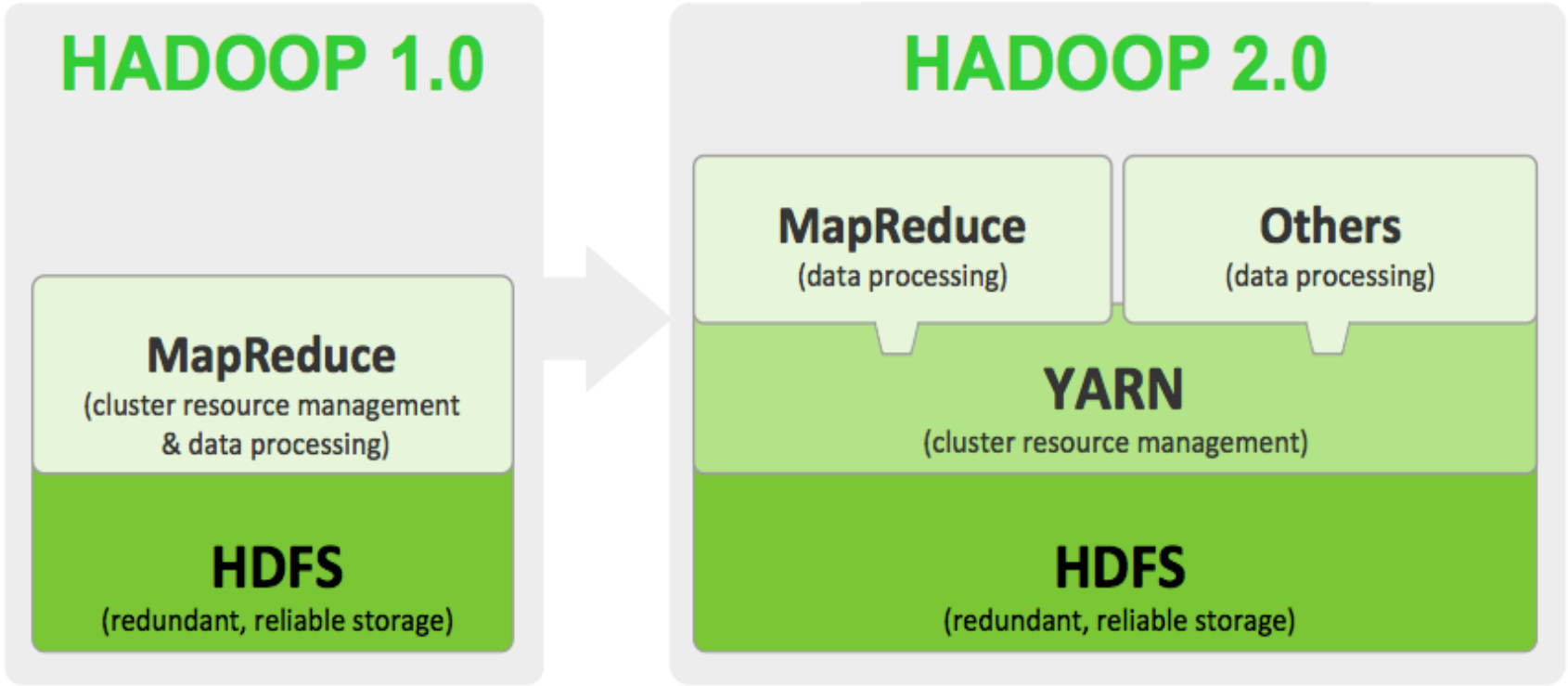
Hadoop资源管理器YARN

目录

YARN产生的背景

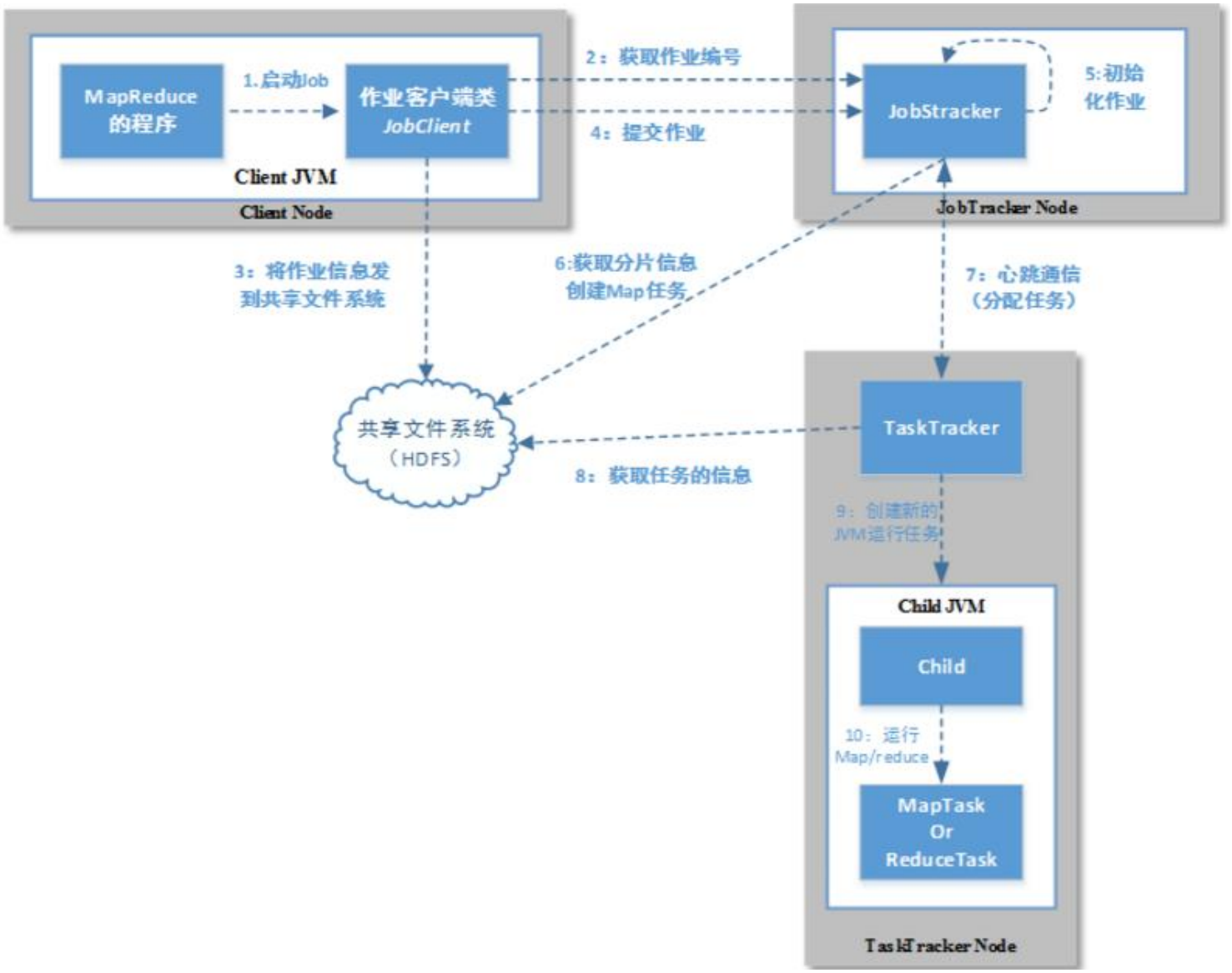
为了克服MRv1的缺点。Apache开始尝试对Hadoop进行升级改造，进而诞生了更加先进的下一代MRv2。MRv2将资源管理统一抽象为独立的通用系统YARN。

在搜索引擎公司中，一种可能的方案：网页建立索引采用MapReduce框架，自然语言处理/数据挖掘采用Spark（如网页PageRank计算，聚类分类算法等），对性能要求很高的数据挖掘算法用MPI等。公司一般系统将所有的框架都部署到一个公共的集群中，让他们共享集群中的资源，对资源进行统一的使用，同时采取资源隔离方案对各个任务进行隔离。YARN便是弹性计算平台的一个典型代表。

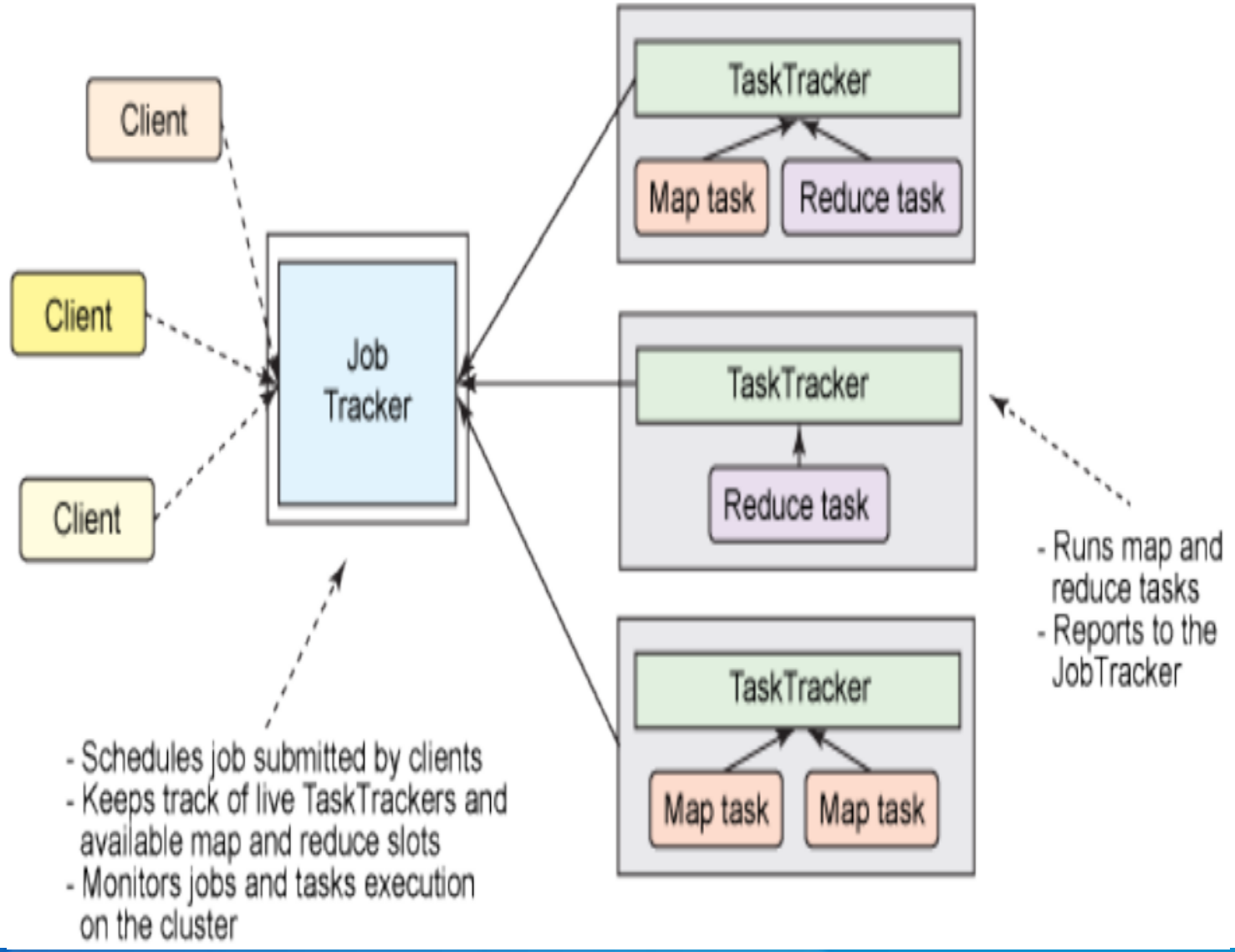


YARN产生的背景

- ✓ 创建job,获取jobID。
- ✓ 检查作业的输出说明并计算作业的输入分片，然后将运行作业所需要的资源都复制到以作业ID命名的目录下。
- ✓ 提交作业，告知jobtracker作业准备执行。（submitJob()方法）
- ✓ 初始化作业。创建一个表示正在运行作业的对象，用来封装任务和记录信息。
- ✓ 获取客户端计算好的输入分片，然后为每个分片创建一个map任务。在此步骤的时候还会创建reduce任务、作业创建任务、作业清理任务。
- ✓ taskTraker发送心跳给JobTraker。
- ✓ 从共享文件系统把作业的JAR文件复制到tasktracker所在的文件系统。
- ✓ tasktracker创建一个TaskRunner实例。
- ✓ 启动一个新的JVM来运行map/reduce任务。



YARN产生的背景

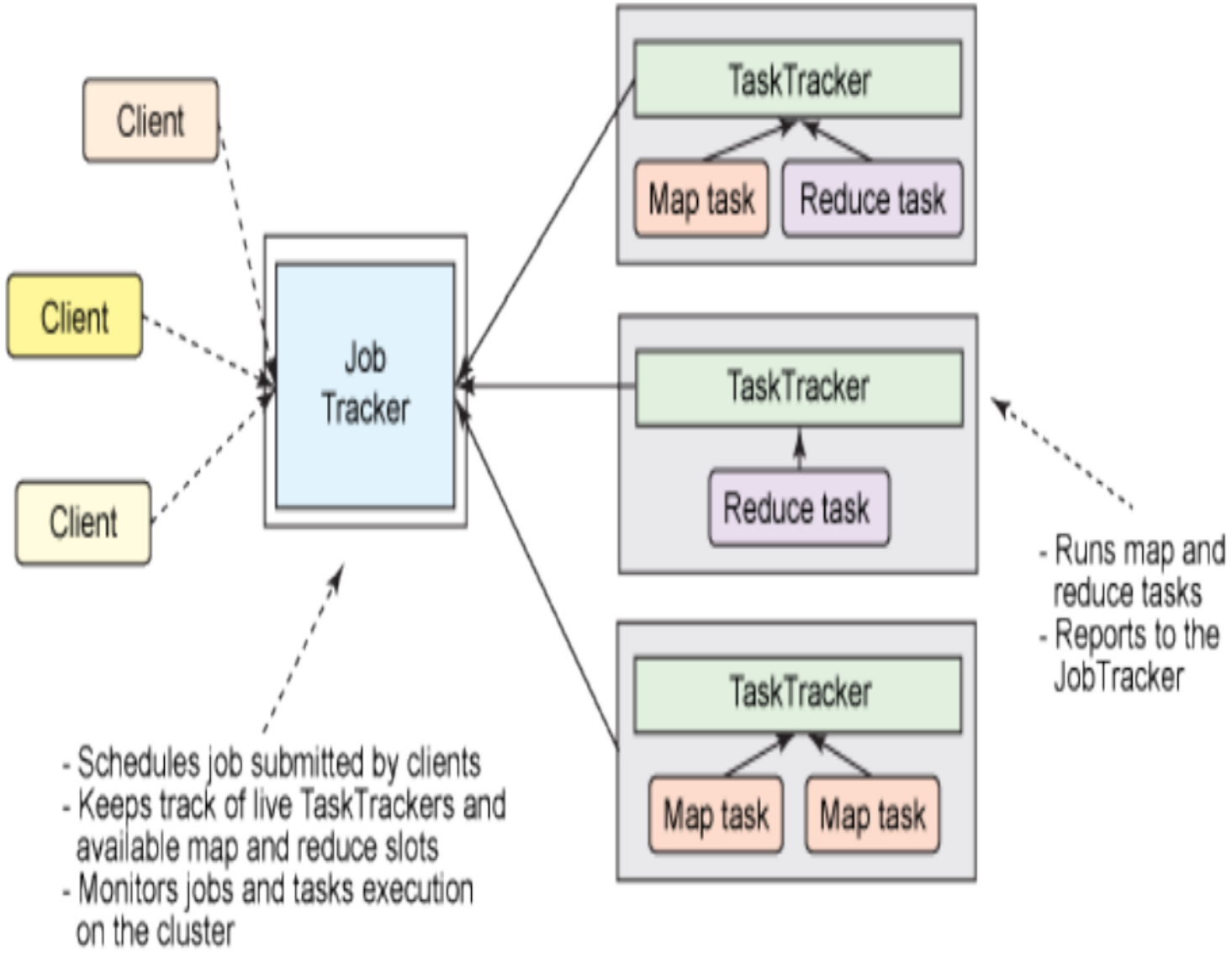


YARN是在MRv1基础上演化而来的，它克服了MRv1中的各种局限性。

扩展性差：在 MRv1 中，JobTracker 同时兼备了资源管理和作业控制两个功能，这成为系统的一个最大瓶颈，严重制约了 Hadoop 集群扩展性。

可靠性差：MRv1采用了master/slave结构，其中master存在单点故障问题，一旦它出现故障将导致整个集群不可用。

YARN产生的背景



资源利用率低：MRv1采用了基于槽位的资源分配模型，槽位是一种粗粒度的资源划分单位，通常一个任务不会用完槽位对应的资源，其他任务也无法使用这些空闲资源。此外，Hadoop将槽位分为Map Slot和Reduce Slot两种，且不允许它们之间共享， 常常会导致一种槽位资源紧张而另外一种闲置（比如一个作业刚刚提交时，只会运行Map Task，此时Reduce Slot闲置）。

无法支持多种计算框架：随着互联网高速发展，MapReduce这种基于磁盘的离线计算框架已经不能满足应用要求，从而出现了一些新的计算框架，包括内存计算框架、流式计算框架和迭代式计算框架等，而MRv1不能支持多种计算框架并存。

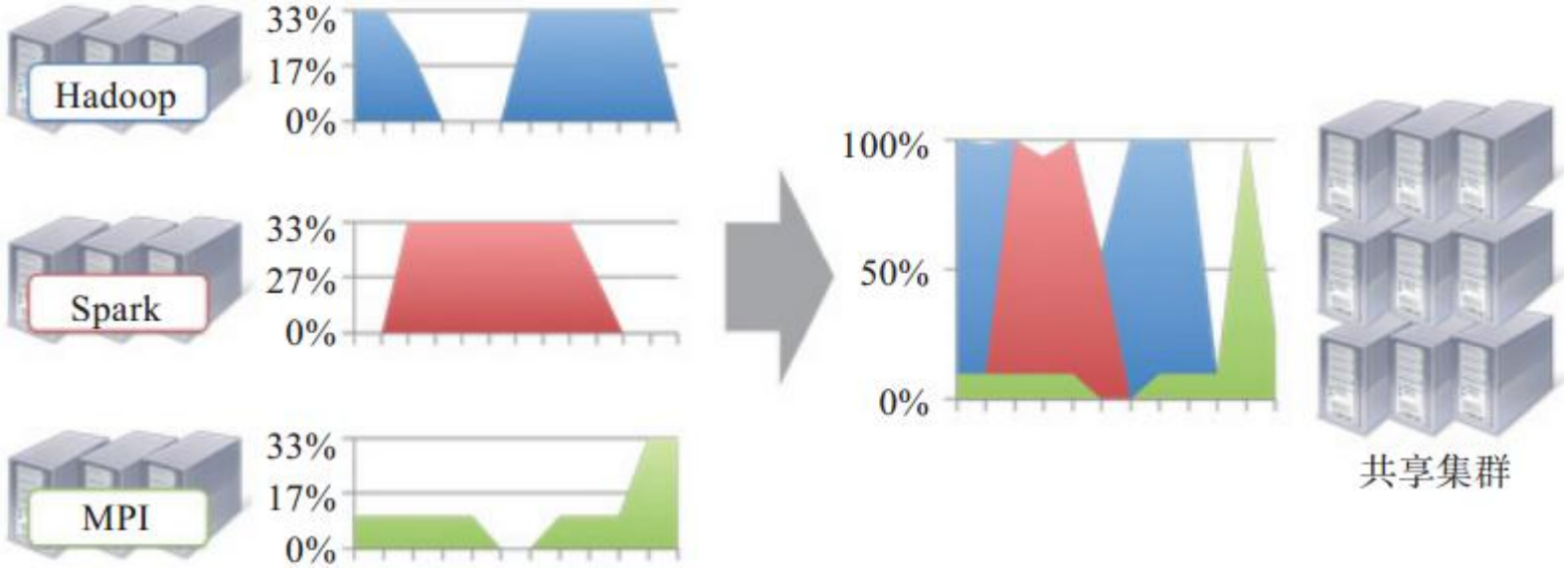
YARN产生的背景

相比于“一种计算框架一个集群”模式，共享模式存在如下好处。

资源利用率高：一种计算框架一个集群模式，往往会出现某段时间，有些计算框架的集群资源紧张而另外一些集群资源却空闲。共享模式通过多种框架共享资源，使得集群中的资源得到更加充分的利用。

运维成本低：共享模式只需要少数管理员统一管理一个集群即可。

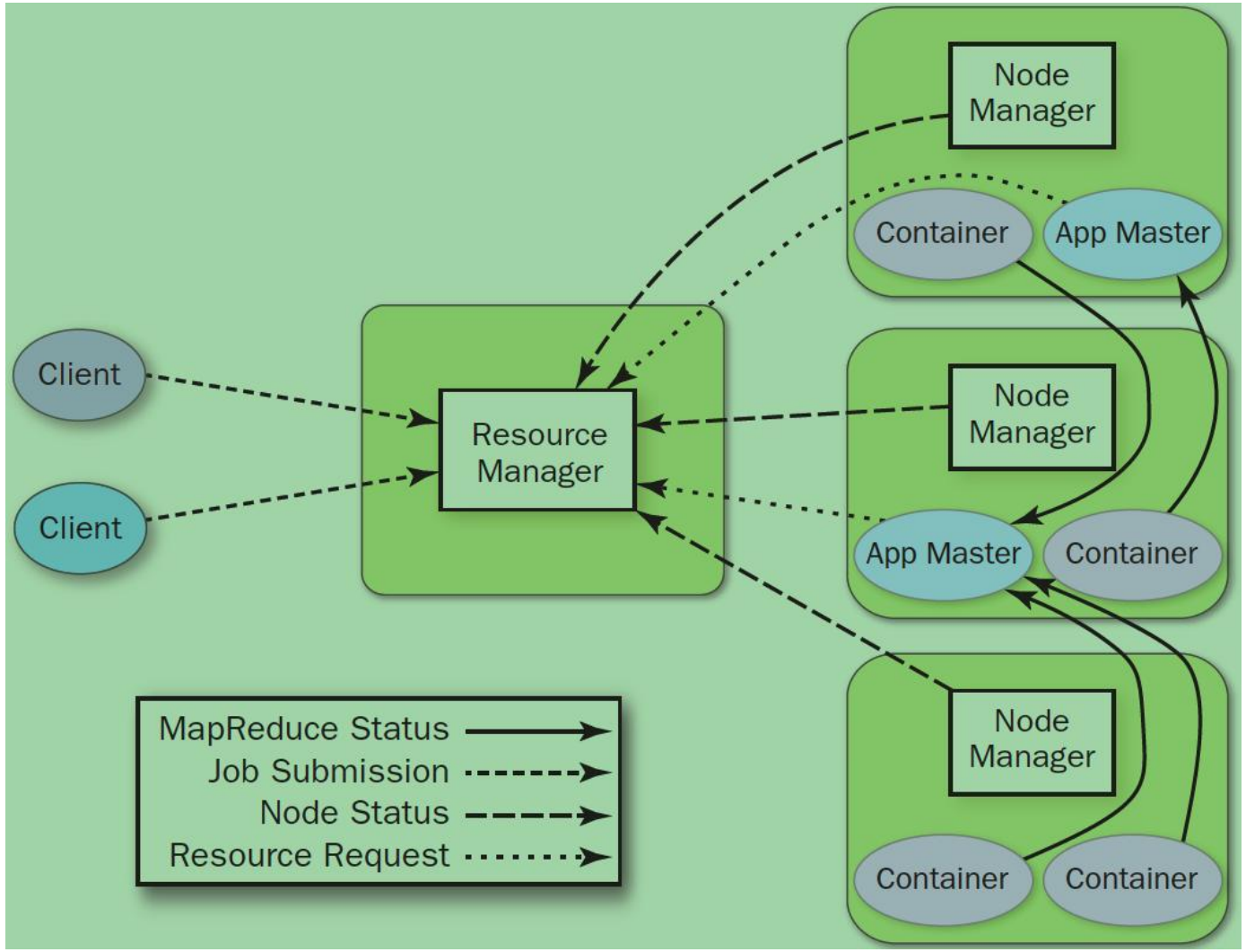
数据共享：夸集群数据移动耗时且硬件成本也高。共享模式使得多种计算框架共享数据和硬件，大大减少成本。



YARN产生的背景-总结

- 直接源于MRv1的几个缺点。
  - ✓扩展性差
  - ✓单点故障
  - ✓难以支持MR之外的计算。
- 多计算框架各自为战，数据共享困难。
  - ✓MR：离线计算框架
  - ✓Storm:实时计算框架
  - ✓Spark:内存计算框架

YARN的基本架构



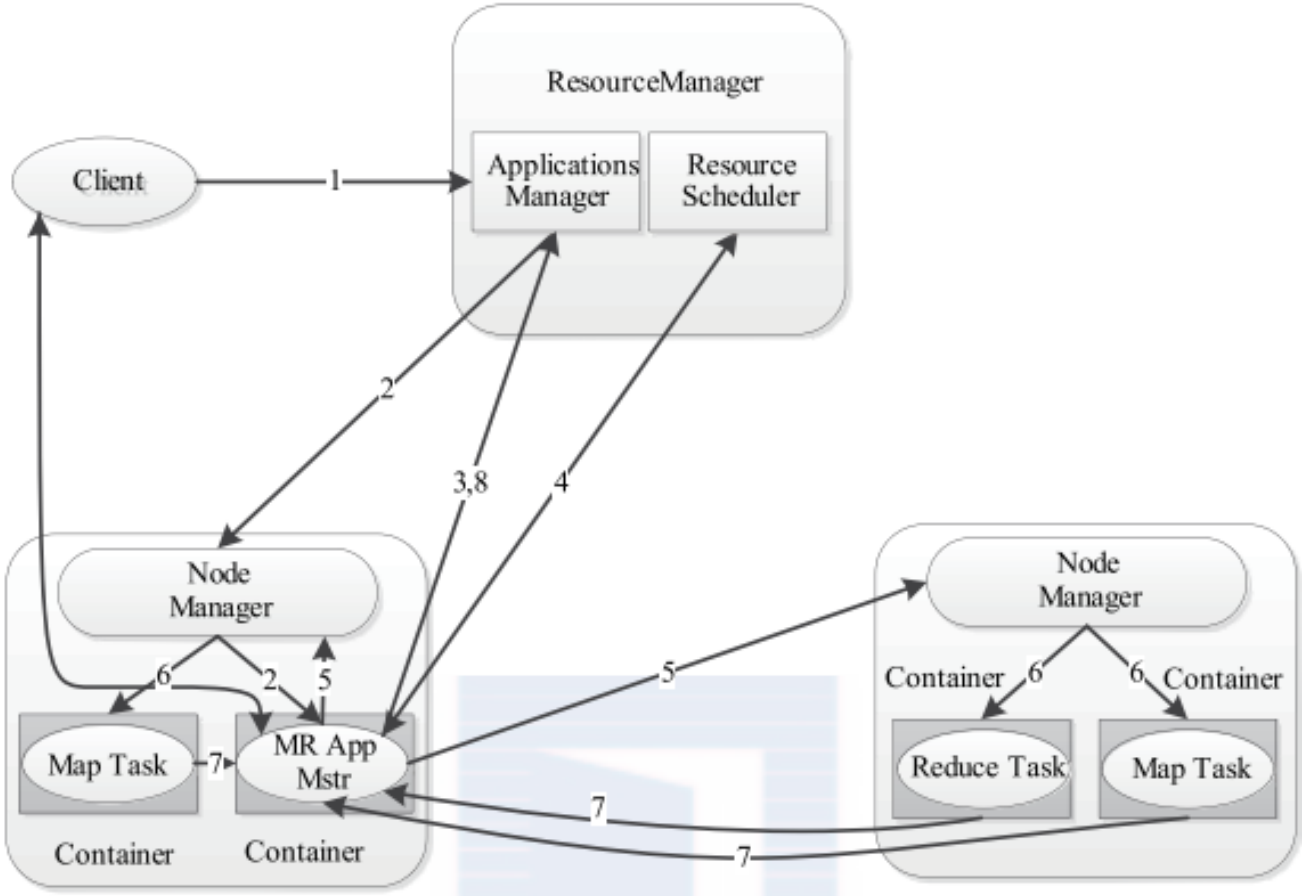
ResourceManager(RM):全局的资源管理器，负责整个系统资源的管理和分配。由调度器（Scheduler）和应用程序管理器（Applications Manager ASM）组成。调度器是纯调度器，只负责资源分配。资源分配单位抽象为Container。ASM用于应用提交，启动ApplicationMaster、监控AM运行状态并在失败时重启它。

NodeManager(NM):NM是每个节点上的资源和任务管理器，一方面定时向RM汇报本节点的资源使用情况和各个Container的运行状态，另一方面，接受来自AM的Container启停请求。

ApplictionMaster(AM):每个应用程序对应一个AM。主要负责向RM请求资源、与NM通信来启停任务、监控所有任务的运行状态，并在任务运行失败时重新为任务申请资源并重启。

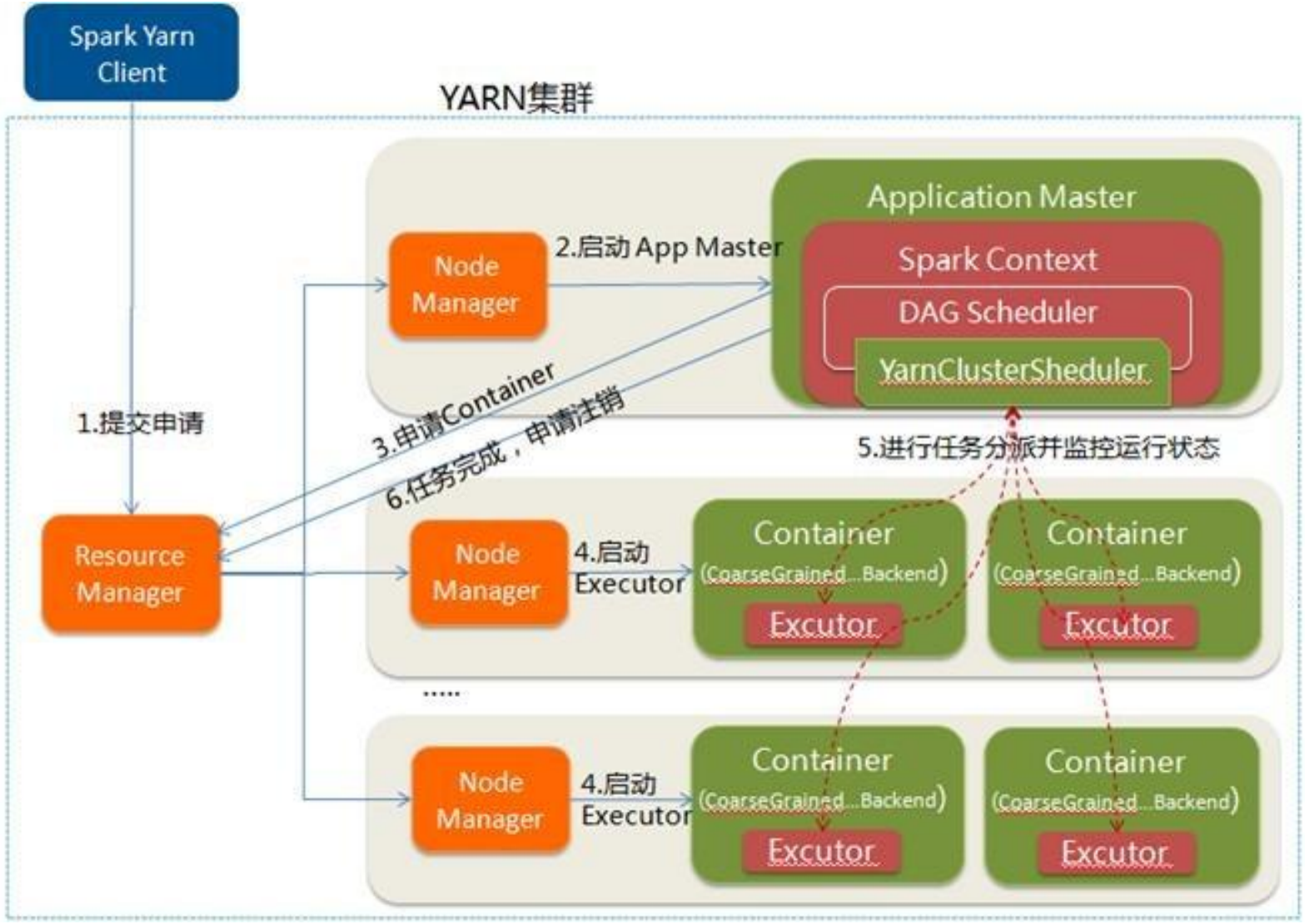
Container:是YARN的资源抽象，封装了某节点的多维度资源，例如内存、CPU、网络、磁盘等(目前只支持CPU和内存)。

YARN的工作流程



1.用户向YARN提交应用程序，其中包括ApplicationMaster程序，启动AM的命令，用户程序。
2.RM为该应用程序分配第一个Container,并与对应的NM通信，要求它在这个Container中启动应用程序对应的AM。
3.AM启动后向RM注册，用户可以直接通过RM查看应用程序的运行状态。重复4~7。
4.AM采用轮询的方式通过RPC协议向RM申请和领取资源。
5.一旦AM申请到资源后，与对应的NM通信，要求它启动任务。
6.NM为任务设置好运行环境（包括环境变量、JAR包、二级制程序等）后，将任务启动命令写入一个脚本中，通过该脚本启动任务。
7.各个任务通过RPC协议向AM汇报自己的状态和进度，以让AM随时掌握任务的运行状态，从而可以在任务失败时重启任务。
8.任务运行完成后，AM向RM注销并关闭自己。

YARN的工作流程



Spark on YARN

YARN的容错

- **ResourceManager的单点故障**
  - ✓ ResourceManager有备份节点，当主节点出现故障，将切换到从节点继续工作。
- **NodeManager**
  - ✓ 失败之后，ResourceManager将失败任务告诉对应的ApplicationMaster
  - ✓ ApplicationMaster 决定如何去处理失败任务。
- **ApplicationMaster**
  - ✓ 失败后，由ResourceManager负责重启。
  - ✓ ApplicationMaster需要处理内部任务的容错问题。
  - ✓ ResourceManager会保存已经运行的task，重启后无需重新运行。

YARN的调度器-背景

随着Hadoop的普及，单个Hadoop集群的用户量越来越大，不同用户提交的应用往往具有不同的质量要求。典型的应用有以下几种。

- ✓ **批处理作业：**作业耗时较长，对时间完成一般没有严格的要求，如数据挖掘、机器学习等方面应用。
- ✓ **交互式作业：**作业要求实时性高，例如SQL查询(Hive)。
- ✓ **生产作业：**这种作业要求有一定的资源保证，如统计值计算、垃圾数据分析等。

此外，这些应用程序对硬件资源需求量也是不同的，如过滤、统计类作业一般为**CPU密集型作业**，而数据挖掘、机器学习作业一般为**I/O密集型作业**。因此，简单的FIFO调度策略不仅不能满足多样化需求，也不能充分利用硬件资源。为了防止单个用户或者应用程序独占资源，进而能够满足各种计算需求，典型的代表是Yahoo!的CapacityScheduler和Facebook的Fair Scheduler。

资源调度器是YARN中最核心的组件之一，且是插拔式的，它定义了一整套接口规范以便用户可按照需要实现自己的调度器。YARN自带了FIFO、Capacity Scheduler和Fair Scheduler三种常用资源调度器，当然，用户可按照接口规范编写一个新的资源调度器，并通过简单的配置使它运行起来。

YARN的调度器-资源调度模型

双层资源调度模型

YARN采用了双层资源调度模型：在第一层中，ResourceManager中的资源调度器将资源分配给各个ApplicationMaster；在第二层中，ApplicationMaster再进一步将资源分配给它内部的各个任务。这里资源调度器主要关注的是第一层的调度问题，至于第二层的调度策略，完全由用户应用程序自己决定。

YARN采用的是pull-base通信模型，而不是push-base通信模型。资源调度器将资源分配给一个应用程序后，它不会立刻push给对应的ApplicationMaster，而是暂时放到一个缓冲区中，等待ApplicationMaster通过周期性的心跳主动来取。

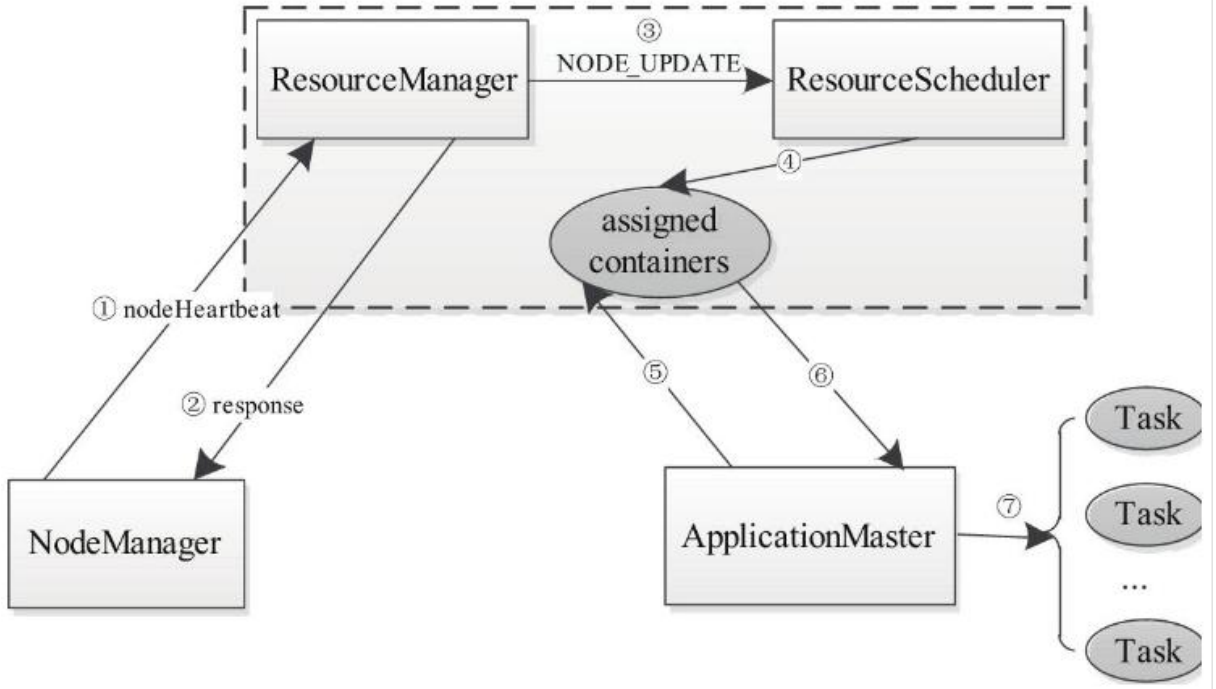
YARN的调度器-资源调度模型

1.NodeManager通过周期性心跳汇报节点信息。

2.ResourceManager为NodeManager返回一个心跳应答，包括需释放的Container列表等信息。

3.ResourceManager收到来自NM的信息后，会触发一个NODE_UPDATE事件。

4.ResourceScheduler收到NODE_UPDATE事件后，会按照一定的策略将该节点上的资源（步骤2中有释放的资源）分配给应用程序，并将分配结果放到一个内存数据结构中。



5.应用程序的ApplicationMaster向ResourceManager发送周期性的心跳，以领取最新分配的Container。

6.ResourceManager收到来自ApplicationMaster心跳信息后，为它分配的container将以心跳应答的形式返回给ApplicationMaster。

7.ApplicationMaster收到新分配的container列表后，会将这些Container进一步分配给它内部的各个任务。

YARN的调度器-资源保证机制

增量资源分配

当应用程序申请的资源暂时无法保证时，是优先为应用程序预留一个节点上的资源直到累计释放的空闲资源满足应用程序的需求。这种资源分配方式，预留资源会造成资源的浪费，降低集群资源利用率。

一次性资源分配

当应用程序申请的资源暂时无法保证时，会放弃当前资源，直到出现一个节点剩余资源一次性满足应用程序需求。这种资源分配方式会出现饿死现象。即应用程序可能永远也等不到满足资源需求的节点出现。

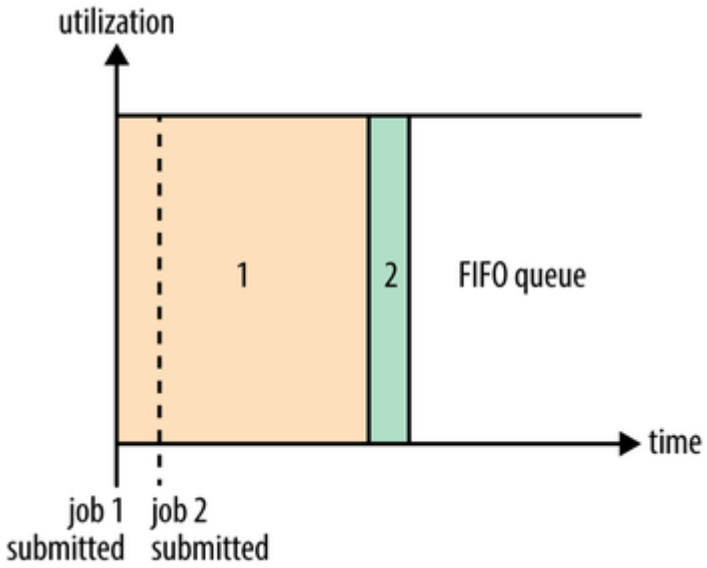
YARN采用增量资源分配机制，尽管这种机制会造成浪费，但不会造成饿死现象。（假设应用程序不会永久占用某个资源，它会在一定时间内释放占用的资源）。

YARN的调度器-资源抢占模型

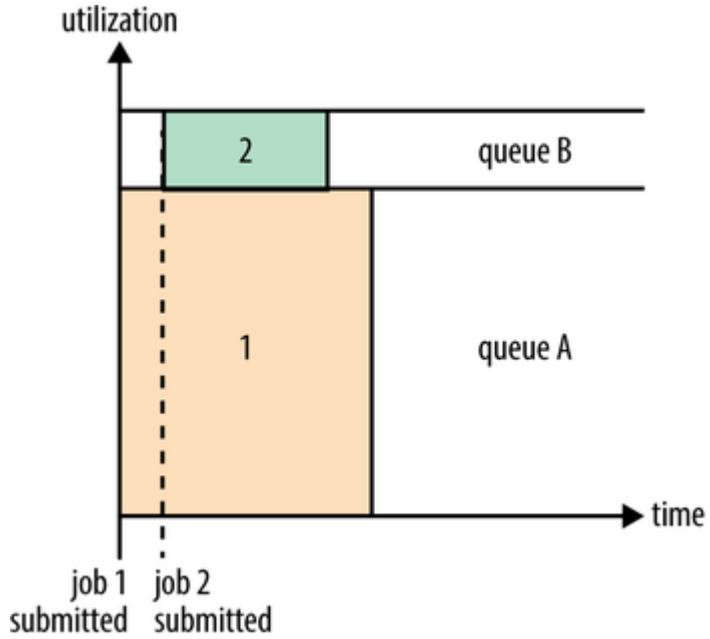
在资源调度器中，每个队列可设置一个最小资源量和最大资源量，其中，最小资源量是资源紧缺情况下每个队列需保证的资源量，而最大资源量则是极端情况下队列也不能超过的资源使用量。

资源抢占发生的原因则完全是由于“最小资源量”这一概念。通常而言，为了提高资源利用率，资源调度器（包括Capacity Scheduler和Fair Scheduler）**会将负载较轻的队列的资源暂时分配给负载重的队列**（即最小资源量并不是硬资源保证，当队列不需要任何资源时，并不会满足它的最小资源量，而是暂时将空闲资源分配给其他需要资源的队列），仅当负载较轻队列突然收到新提交的应用程序时，调度器才进一步将本属于该队列的资源分配给它。但由于此时资源可能正被其他队列使用，因此调度器必须等待其他队列释放资源后，才能将这些资源“物归原主”，这通常需要一段不确定的等待时间。为了防止应用程序等待时间过长，**调度器等待一段时间后若发现资源并未得到释放，则进行资源抢占。**

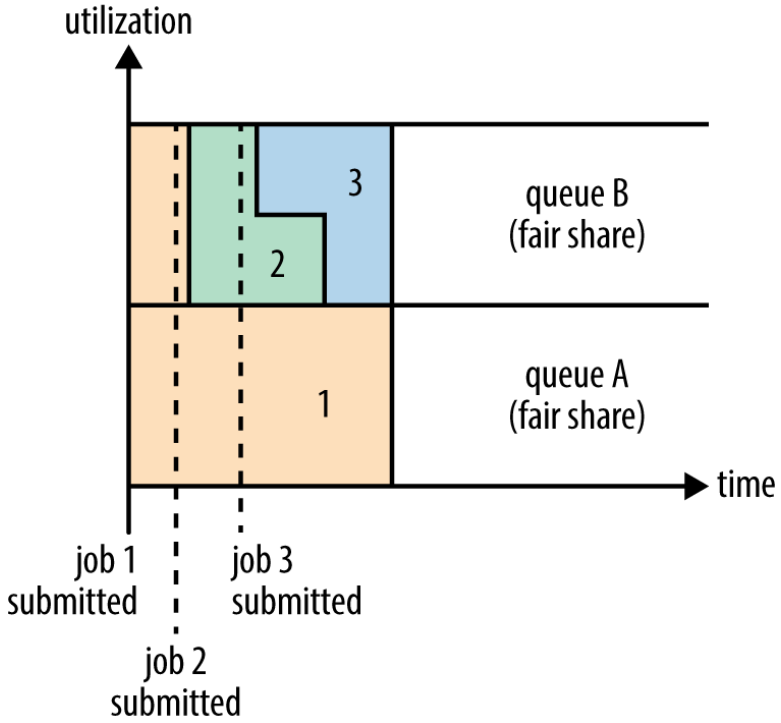
YARN的调度器



FIFO Schedule



Capacity Schedule



FAIR Schedule

YARN的调度器-Capacity Shedule

Capacity Scheduler是Yahoo!开发的多用户调度器，它以队列为单位划分资源，每个队列可设定一定比例的资源最低保证和使用上限，同时，每个用户也可设定一定的资源使用上限以防止资源滥用。而当一个队列的资源有剩余时，可暂时将剩余资源共享给其他队列。总之，Capacity Scheduler主要有以下几个特点。

容量保证： 每个队列可以设定最低资源保证和最高资源使用上限，而提交到该队列的任务则共享该队列的资源。

灵活性： 如果一个队列资源有剩余，而其他队列资源紧张，可以先借用资源给其他队列。而一旦该队列有应用程序提交，则其他队列释放资源后会归还给该队列。

多种租赁： 支持多用户共享集群和多应用程序同时运行。管理员可以为之增加多种约束（某队列最大运行的任务数等）。

安全保证： 每个队列都有严格的ACL列表规定它的访问用户，每个用户可以指定哪些用户可以查看自己应用程序的状态或者控制应用程序（例如杀死应用程序）。管理员可以指定队列管理员和系统管理员。

动态配置更新： 管理员可以根据需要动态修改配置参数。

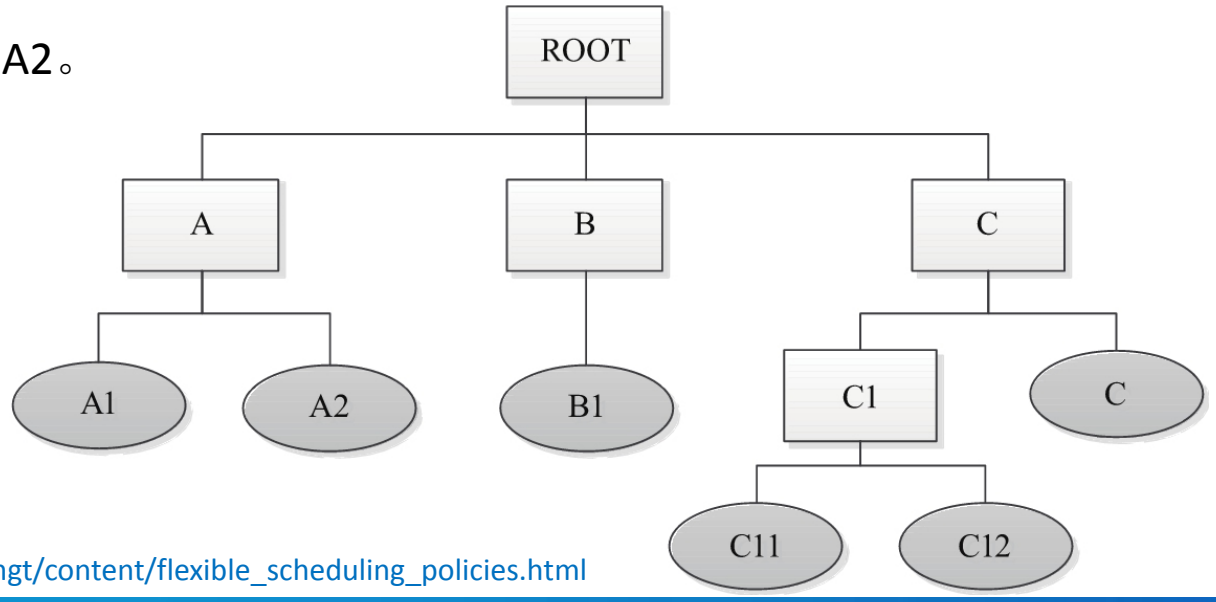
YARN的调度器-Capacity Shedule

子对列

- 队列可以嵌套，每个队列可以有子队列。
- 用户只能讲应用程序提交到最底层队列，即叶子队列。

最少最大容量

- 每个队列配置最小和最大容量。
- 调度器总会选择当前资源使用率最低的队列，并为之分配资源。例如同级的队列A1,A2。它们最小容量均为30。而A1使用了12，A2使用了10，则调度器会优先将资源分给A2。
- 最小容量不是“总会保证最低容量”。
- 最小容量>=0，且不能大于最大容量。
- 最大容量限制了队列使用容量的极限值。



叶子节点内部使用多种策略

- FIFO,user limit ,application limit ,etc.

参照：https://docs.hortonworks.com/HDPDocuments/HDP2/HDP-2.4.2/bk_yarn_resource_mgt/content/flexible_scheduling_policies.html

YARN的调度器-Capacity Shedule

yarn.resourcemanager.scheduler.class	org.apache.hadoop.yarn.server.resourcemanager.scheduler.capacity.CapacitySchedule
Capacity Scheduler	yarn.scheduler.capacity.root.queues=default,spark yarn.scheduler.capacity.root.default.user-limit-factor=1 yarn.scheduler.capacity.root.default.state=RUNNING yarn.scheduler.capacity.root.default.maximum-capacity=70 yarn.scheduler.capacity.root.default.capacity=50 yarn.scheduler.capacity.root.default.acl_submit_applications=* yarn.scheduler.capacity.root.capacity=100 yarn.scheduler.capacity.root.acl_administer_queue=* yarn.scheduler.capacity.root.accessible-node-labels=* yarn.scheduler.capacity.node-locality-delay=40 yarn.scheduler.capacity.maximum-applications=10000 yarn.scheduler.capacity.maximum-am-resource-percent=0.2 yarn.scheduler.capacity.root.spark.user-limit-factor=1 yarn.scheduler.capacity.queue-mappings-override.enable=false yarn.scheduler.capacity.root.spark.acl_administer_queue=* yarn.scheduler.capacity.root.spark.acl_submit_applications=*

参考：https://hadoop.apache.org/docs/r2.7.2/hadoop-yarn/hadoop-yarn-site/CapacityScheduler.html

YARN shell命令

使用方法： yarn [--config confdir] COMMAND

说明：--config confdir，覆盖缺省配置目录。 默认是 $HADOOP_PREFIX/conf。

COMMAND分为用户命令和管理命令。下面分别介绍。

[jar]

使用方法：yarn jar <jar> [mainClass] args...

说明：运行一个jar文件，用户可以捆绑yarn代码在一个jar文件，然后使用这个命令执行。

Example:

yarn jar org.eosftdata.WordCout /data/test/wordcout/input/words.txt /data/test/wordcout/output

[application]

使用方法： yarn application <options>

说明：打印应用程序的报告和kill掉的应用程序。

COMMAND_OPTIONS	Description
-list	Lists applications from the RM. Supports optional use of -appTypes to filter applications based on application type, and -appStates to filter applications based on application state.
-appStates States	Works with -list to filter applications based on input comma-separated list of application states. The valid application state can be one of the following: ALL, NEW, NEW_SAVING, SUBMITTED, ACCEPTED, RUNNING, FINISHED, FAILED, KILLED
-appTypes Types	Works with -list to filter applications based on input comma-separated list of application types.
-status ApplicationId	Prints the status of the application.
-kill ApplicationId	Kills the application.

YARN shell命令

[node]

使用方法：yarn node <options>

说明：打印节点报告。

COMMAND_OPTIONS	Description
-list	Lists all running nodes. Supports optional use of -states to filter nodes based on node state, and -all to list all nodes.
-states States	Works with -list to filter nodes based on input comma-separated list of node states.
-all	Works with -list to list all nodes.
-status NodeId	Prints the status report of the node.

[logs]

使用方法：yarn logs -applicationId <application ID> <options>

说明：转储容器日志。

COMMAND_OPTIONS	Description
-applicationId <application ID>	Specifies an application id
-appOwner AppOwner	AppOwner (assumed to be current user if not specified)
-containerId ContainerId	ContainerId (must be specified if node address is specified)
-nodeAddress NodeAddress	NodeAddress in the format nodename:port (must be specified if container id is specified)

[classpath]

使用方法：yarn classpath

说明：打印需要得到Hadoop的jar和所需要的库的类路径

YARN shell命令

[version]

使用方法：yarn version

说明：打印版本信息

[resourcemanager]

使用方法：yarn resourcemanager

说明：启动resourcemanager。

[nodemanager]

使用方法：yarn nodemanager

说明：启动nodemanager。

[proxyserver]

使用方法：yarn proxyserver

说明：启动proxyserver。

YARN shell命令

[rmadmin]

使用方法：yarn rmadmin [-refreshQueues][-refreshNodes][-refreshUserToGroupsMapping] [-refreshSuperUserGroupsConfiguration][-refreshAdminAcls][-refreshServiceAcl][-getGroups [username]] [-help [cmd]][-transitionToActive <serviceId>][-transitionToStandby <serviceId>][-getServiceState <serviceId>] [-checkHealth <serviceId>]

说明：运行ResourceManager管理客户端

COMMAND_OPTIONS	Description
-refreshQueues	刷新队列的ACL，状态和调度程序特定的属性。ResourceManager将刷新 mapred-queues的配置文件。
-refreshNodes	刷新在ResourceManager中的主机信息。
-refreshUserToGroupsMappings	刷新用户到组的映射。
-refreshSuperUserGroupsConfiguration	刷新超级用户代理组的映射。
-refreshAdminAcls	刷新ACL的ResourceManager的管理
-refreshServiceAcl	刷新服务级授权策略文件。ResoureceManager将重新加载授权策略文件。
-getGroups [username]	获取指定用户所属的组
-help [cmd]	显示帮助对于给定的命令或如果没有指定显示所有命令。
-transitionToActive <serviceId>	将应用转换为active状态。
-transitionToStandby <serviceId>	将应用程序转换为standby状态。
-getServiceState <serviceId>	获取service状态。
-checkHealth <serviceId>	Requests that the service perform a health check. The RMAdmin tool will exit with a non-zero exit code if the check fails.

YARN shell命令

[daemonlog]

使用方法：

yarn daemonlog -getlevel <host:port> <name>

yarn daemonlog -setlevel <host:port> <name> <level>

说明：获取/设置每个守护进程的日志级别。

Example:

yarn daemonlog -getlevel idh104:8088 org.apache.hadoop.yarn.server.resourcemanager.rmapp.RMAppImpl

yarn daemonlog -setlevel 127.0.0.1:8088 org.apache.hadoop.yarn.server.resourcemanager.rmapp.RMAppImpl DEBUG

COMMAND_OPTIONS	Description
-getlevel <host:port> <name>	打印运行守护进程的日志级别 host:port. 该命令在内部连接到 http://host:port/logLevel?log=name
-setlevel <host:port> <name> <level>	设置守护进程运行时的日志级别 host:port. 该命令在内部连接到http://host:port/logLevel?log=name

更多shell命令请参考：http://hadoop.apache.org/docs/r2.7.3/hadoop-yarn/hadoop-yarn-site/YarnCommands.html

YARN 监控界面

All Applicatio

- Cluster
  - About
  - Nodes
  - Node Labels
  - Applications
    - NEW
    - NEW_SAVING
    - SUBMITTED
    - ACCEPTED
    - RUNNING
    - FINISHED
    - FAILED
    - KILLED
  - Scheduler
- Tools

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Use
109	12	4	93	51	114 GB	176 GB	0 B	58

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	
Capacity Scheduler	[MEMORY, CPU]	<memory:2048, vCore

Show 20 entries

ID	User	Name	Application Type	Queue	Application Priority	StartT
application_1489626712521_0107	root	Kylin_Fact_Distinct_Columns_DATAANALYSIS_V3_16_2016_CUBE_ENERGYYEAR_Step	MAPREDUCE	default	0	Thu Ma 16 20:22:1 +0800 2017
application_1489626712521_0106	root	Kylin_Fact_Distinct_Columns_DATAANALYSIS_V3_16_2016_CUBE_FARMNORMMONTH_Step	MAPREDUCE	default	0	Thu Ma 16 20:22:0 +0800 2017
application_1489626712521_0105	root	Kylin_Fact_Distinct_Columns_DATAANALYSIS_V3_16_2016_CUBE_PCBIN_YEAR_Step	MAPREDUCE	default	0	Thu Ma 16 20:22:0 +0800 2017
application_1489626712521_0104	root	Kylin_Fact_Distinct_Columns_DATAANALYSIS_V3_16_2016_CUBE_ENERGYMONTH_Step	MAPREDUCE	default	0	Thu Ma 16 20:21:4 +0800

作业

1.YARN是什么，为什么会产生YARN，它解决了什么问题？
2.应用程序提交到YARN之后的执行流程是什么，请简要描述？
3.提交之前练习的Mapreduce程序到YARN上运行。
4.学习FairSchedule并在集群中配置FairSchedule，并对FairSchedule和CapacitySchedule做一个对比。

总结

- YARN的产生背景
  - ✓源于MRv1的缺点：扩展性差，单点故障，只支持MR，无法共享数据等。
- YARN的基本架构
  - ✓ ResourceManager
  - ✓ NodeManager
  - ✓ ApplicationMaster
  - ✓ Container
- YARN的工作流程
- YARN的调度模型
  - ✓ FIFO
  - ✓ Capacity Schedule
  - ✓ Fair Schedule
- YARN的shell命令
- YARN的监控界面

参考资料&预习

- 《 hadoop技术内幕：深入解析YARN架构设计与实现原理》
- 《Yarn框架代码详细分析V0.5 》
- http://hadoop.apache.org/docs/r2.7.3/hadoop-yarn/hadoop-yarn-site
- https://hadoop.apache.org/docs/r2.7.2/hadoop-yarn/hadoop-yarn-site/CapacityScheduler.html
- https://hortonworks.com/blog/apache-hadoop-yarn-resourcemanager/
- 董西成Hadoop第3讲。

预习：

《 Hive编程指南》第2,3,4章。

THANKS