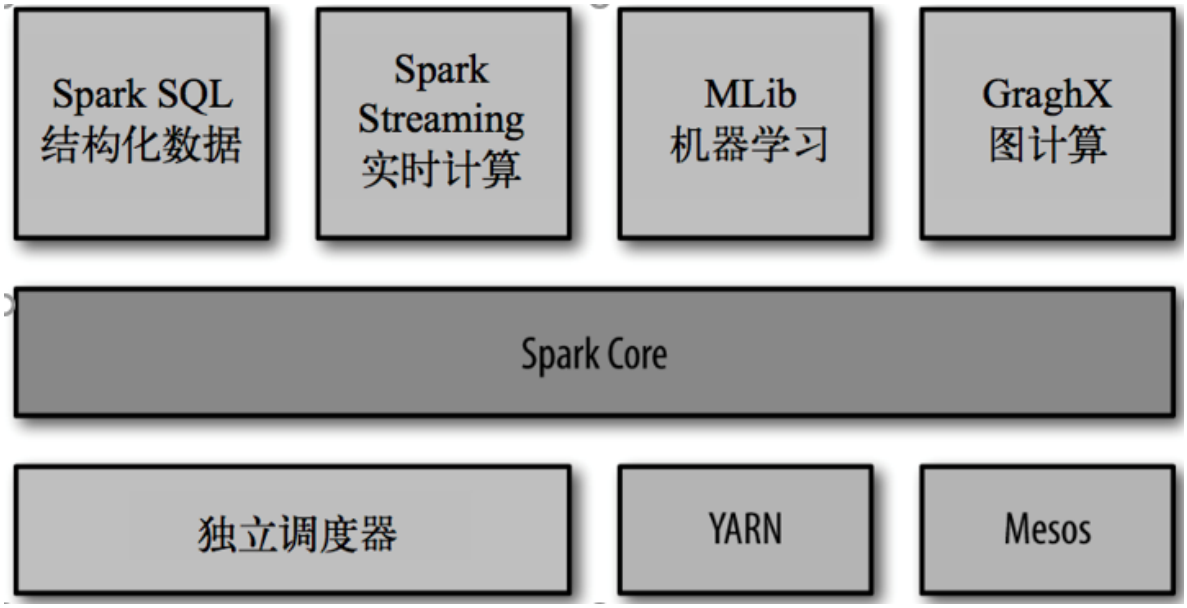
spark架构和原理

参考博客1

第一章 生态及架构

1、spark生态

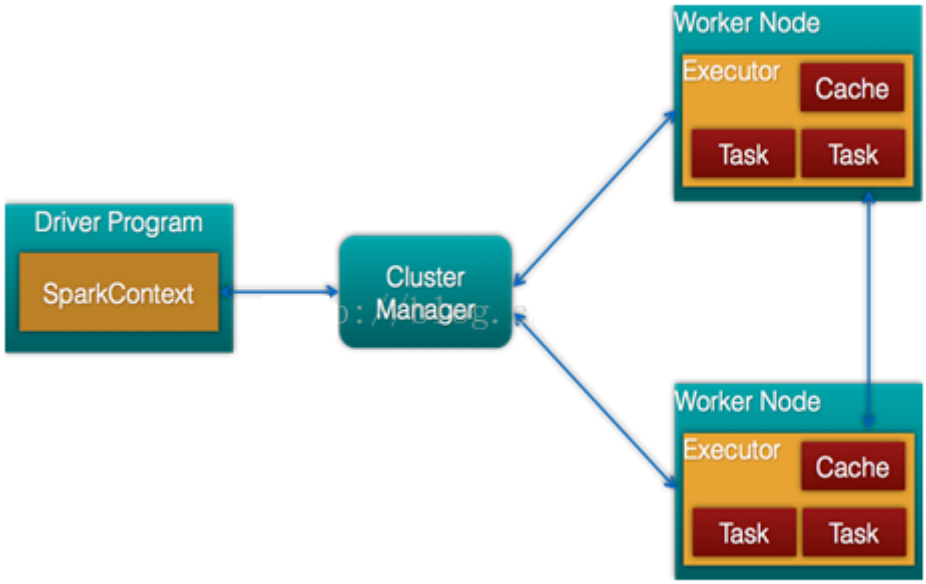


- Spark Core：包含Spark的基本功能；尤其是定义RDD的API、操作以及这两者上的动作。其他Spark的库都是构建在RDD和Spark Core之上的
- Spark SQL：提供通过Apache Hive的SQL变体Hive查询语言（HiveQL）与Spark进行交互的API。每个数据库表被当做一个RDD，Spark SQL查询被转换为Spark操作。
- Spark Streaming：对实时数据流进行处理和控制。Spark Streaming允许程序能够像普通RDD一样处理实时数据
- MLlib：一个常用机器学习算法库，算法被实现为对RDD的Spark操作。这个库包含可扩展的学习算法，比如分类、回归等需要对大量数据集进行迭代的操作。
- GraphX：控制图、并行图操作和计算的一组算法和工具的集合。GraphX扩展了RDD API，包含控制图、创建子图、访问路径上所有顶点的操作

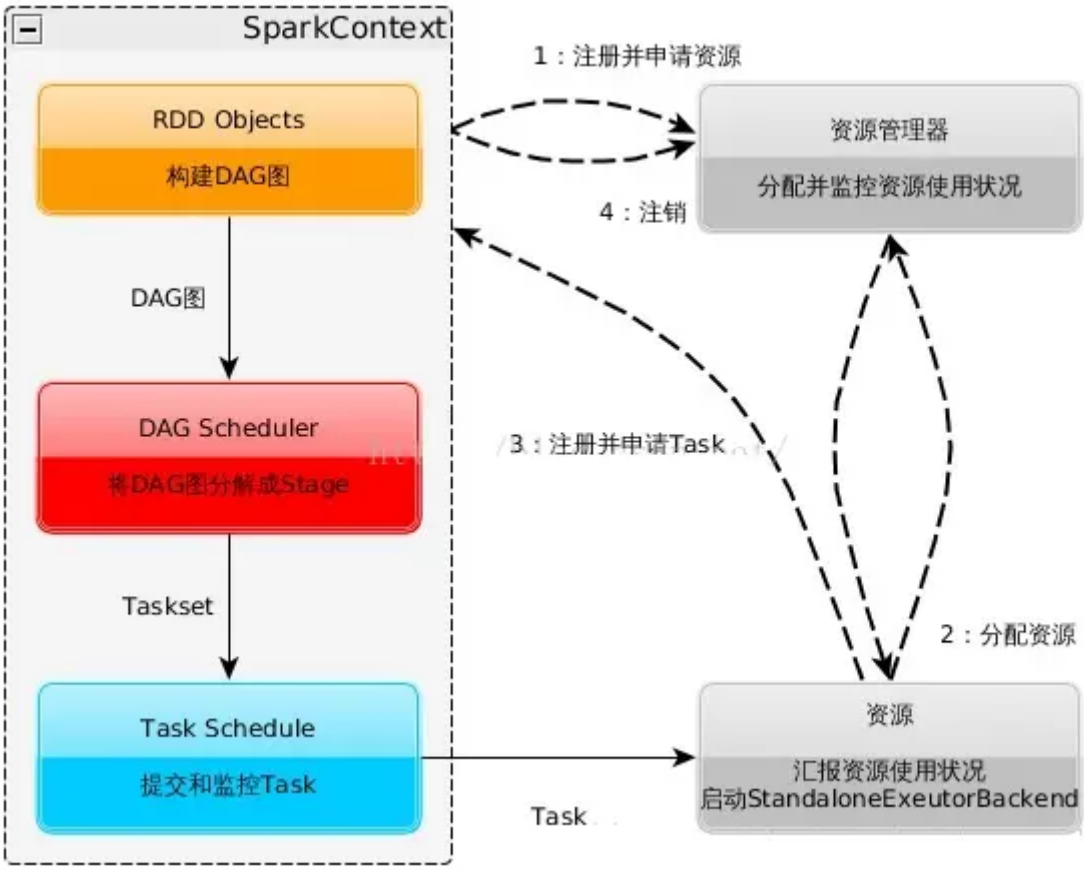
2、spark架构

参考博客2

spark 基本架构图



spark 流程图



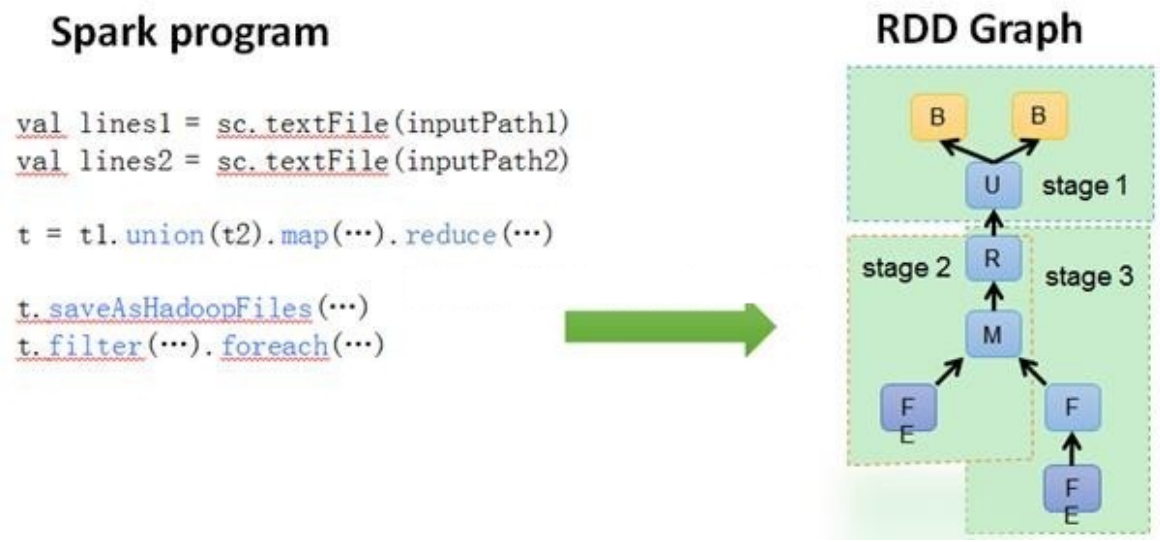
从物理部署层面上来看，Spark主要分为两种类型的节点，Master节点和Worker节点：Master节点主要运行集群管理器的中心化部分，所承载的作用是分配Application到Worker节点，维护Worker节点，Driver，Application的状态。Worker节点负责具体的业务运行。

从Spark程序运行的层面来看，Spark主要分为驱动器节点和执行器节点。

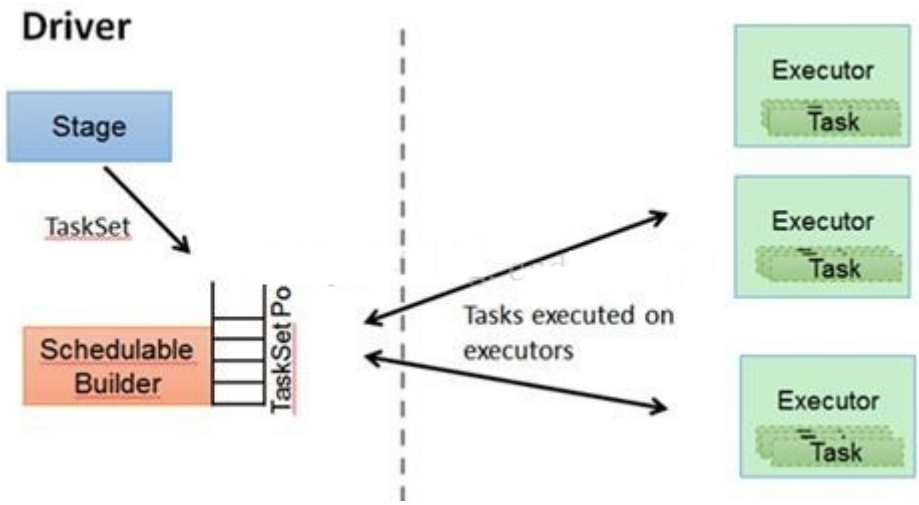
1. Application: Appliction都是指用户编写的Spark应用程序，其中包括一个Driver功能的代码和分布在集群中多个节点上运行的Executor代码
2. Driver: Spark中的Driver即运行上述Application的main函数并创建SparkContext

3. SparkContext：创建SparkContext的目的是为了准备Spark应用程序的运行环境，在Spark中有SparkContext负责与ClusterManager通信，进行资源申请、任务的分配和监控等，当Executor部分运行完毕后，Driver同时负责将SparkContext关闭，通常用SparkContext代表Driver（SparkContext也是我们对Spark编程的核心入口）
4. Executor: 某个Application运行在worker节点上的一个进程， 该进程负责运行某些Task， 并且负责将数据存到内存或磁盘上，每个Application都有各自独立的一批Executor， 在Spark on Yarn模式下，其进程名称为CoarseGrainedExecutor Backend。一个CoarseGrainedExecutor Backend有且仅有一个Executor对象， 负责将Task包装成taskRunner,并从线程池中抽取一个空闲线程运行Task， 这个每一个CoarseGrainedExecutor Backend能并行运行Task的数量取决与分配给它的cpu个数
5. Cluster Manager：指的是在集群上获取资源的外部服务。目前有三种类型
   Standalone : spark原生的资源管理，由Master负责资源的分配
   Apache Mesos:与hadoop MR兼容性良好的一种资源调度框架
   Hadoop Yarn: 主要是指Yarn中的ResourceManager
6. Worker: 集群中任何可以运行Application代码的节点，在Standalone模式中指的是通过slave文件配置的Worker节点，在Spark on Yarn模式下就是NodeManager节点
7. Task: 被送到某个Executor上的工作单元，但hadoopMR中的MapTask和ReduceTask概念一样，是运行Application的基本单位，多个Task组成一个Stage，而Task的调度和管理等是由TaskScheduler负责
8. Job: 包含多个Task组成的并行计算，往往由Spark Action触发生成， 一个Application中往往会产生多个Job
9. Stage: 每个Job会被拆分成多组Task， 作为一个TaskSet， 其名称为Stage，Stage的划分和调度是有DAGScheduler来负责的，Stage有非最终的Stage（Shuffle Map Stage）和最终的Stage（Result Stage）两种，Stage的边界就是发生shuffle的地方
10. DAGScheduler: 根据Job构建基于Stage的DAG（Directed Acyclic Graph有向无环图)，并提交Stage给TASkScheduler。 其划分Stage的依据是RDD之间的依赖的关系找出开销最小的调度方法

    其划分Stage的依据是RDD之间的依赖的关系找出开销最小的调度方法，如下图



11. TASKSedulter: 将TaskSET提交给worker运行，每个Executor运行什么Task就是在此处分配的. TaskScheduler维护所有TaskSet，当Executor向Driver发生心跳时，TaskScheduler会根据资源剩余情况分配相应的Task。另外TaskScheduler还维护着所有Task的运行标签，重试失败的Task。下图展示了TaskScheduler的作用



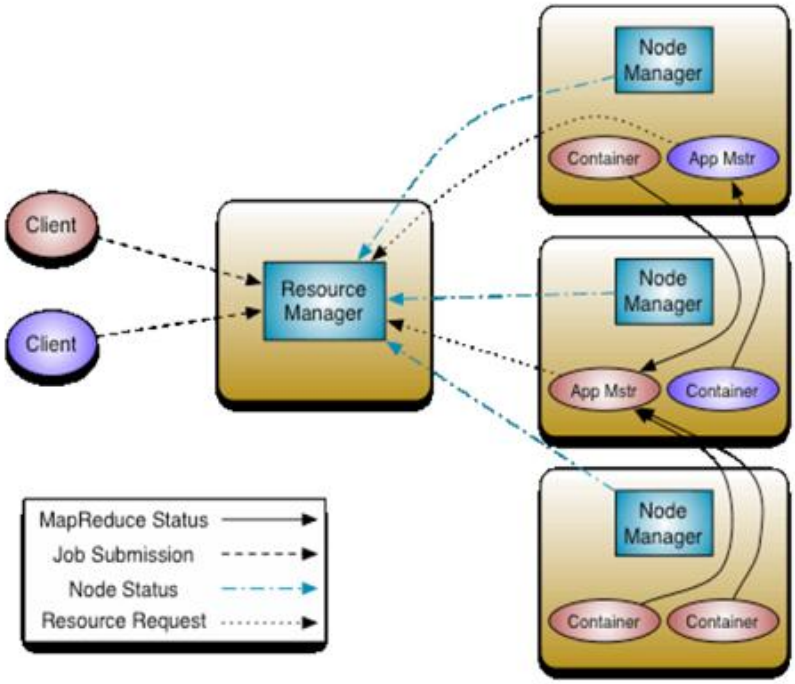
第二章 spark yarn运行模式的两种区别

参考博客3

yarn是一种统一的资源管理机制，可以通过队列的方式，管理运行多套计算框架。Spark on Yarn模式根据Dirver在集群中的位置分为两种模式

一种是Yarn-Client模式，另一种是Yarn-Cluster模式

yarn框架的基本运行流程图



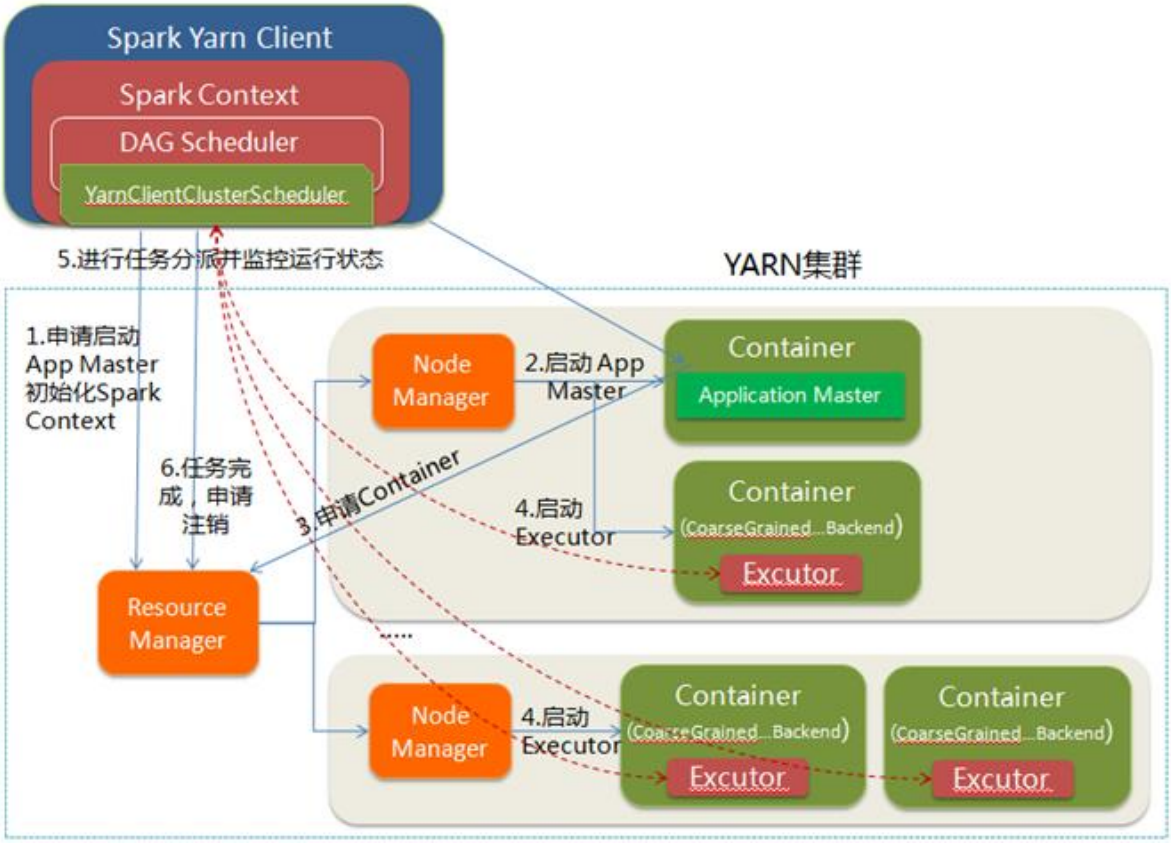
ResourceManager：负责将集群的资源分配给各个应用使用，而资源分配和调度的基本单位是Container，其中封装了集群资源（CPU、内存、磁盘等），每个任务只能在Container中运行，并且只使用Container中的资源；

NodeManager：是一个个计算节点，负责启动Application所需的Container，并监控资源的使用情况汇报给ResourceManager

ApplicationMaster：主要负责向ResourceManager申请Application的资源，获取Container并跟踪这些Container的运行状态和执行进度，执行完后通知ResourceManager注销ApplicationMaster，ApplicationMaster也是运行在Container中；

(1)client

yarn-client模式，Dirver运行在本地的客户端上。



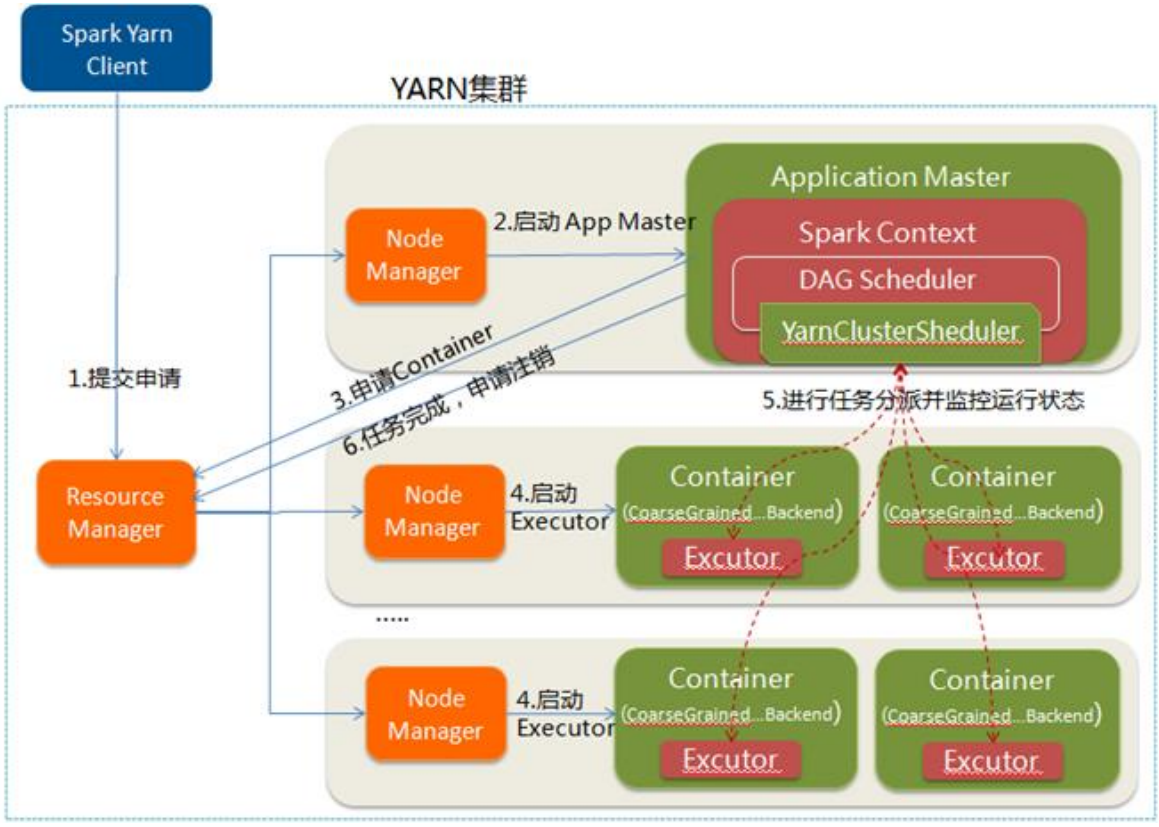
1. client向ResouceManager申请启动ApplicationMaster，同时在SparkContext初始化中创建DAGScheduler和TaskScheduler
2. ResouceManager收到请求后，在一台NodeManager中启动第一个Container运行ApplicationMaster
3. Dirver中的SparkContext初始化完成后与ApplicationMaster建立通讯，ApplicationMaster向ResourceManager申请Application的资源
4. 一旦ApplicationMaster申请到资源，便与之对应的NodeManager通讯，启动Executor，并把Executor信息反向注册给Dirver
5. Dirver分发task，并监控Executor的运行状态，负责重试失败的task
6. 运行完成后，Client的SparkContext向ResourceManager申请注销并关闭自己

2)cluster

yarn-cluster模式中，当用户向yarn提交应用程序后，yarn将分为两阶段运行该应用程序：

第一个阶段是把Spark的Dirver作为一个ApplicationMaster在yarn中启动；

第二个阶段是ApplicationMaster向ResourceManager申请资源，并启动Executor来运行task，同时监控task整个运行流程并重试失败的task；

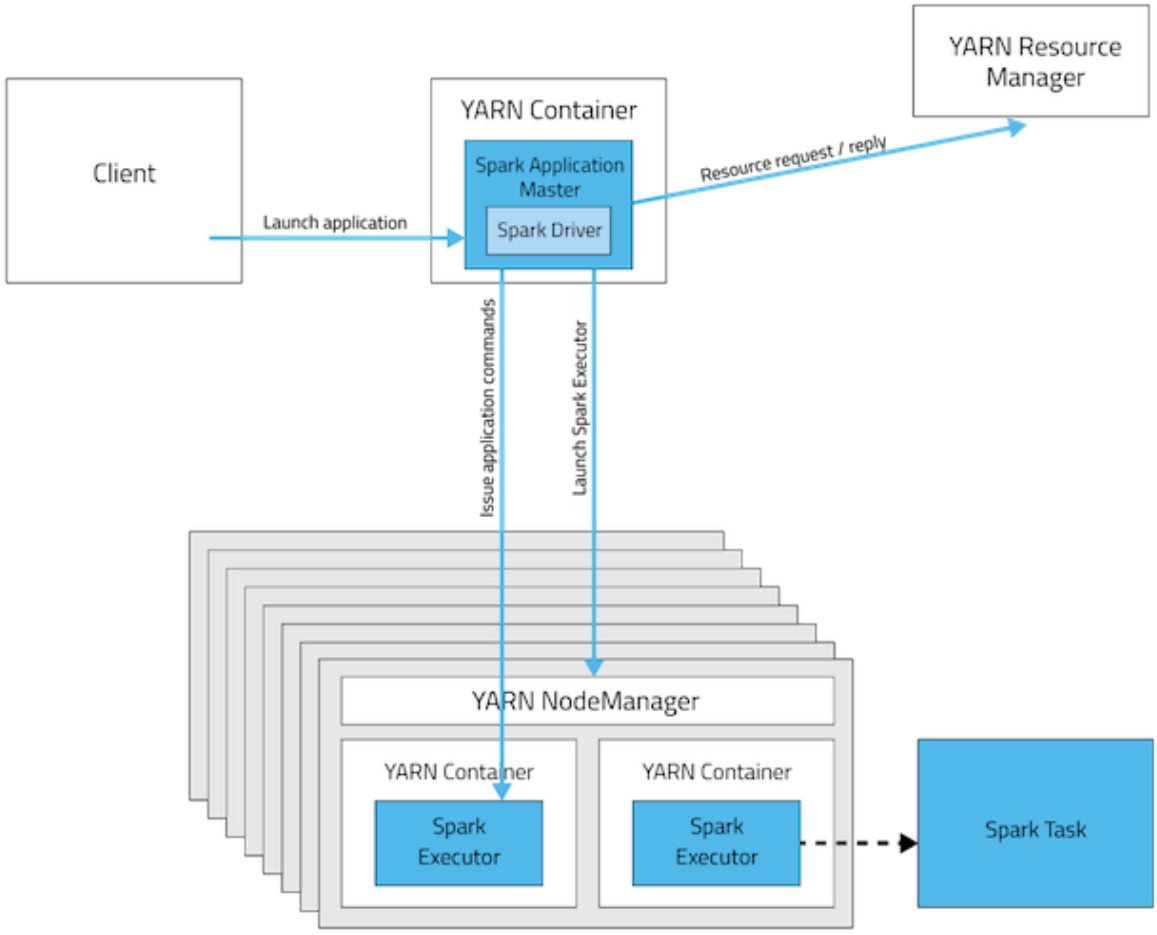


Yarn-client和Yarn-cluster的区别：

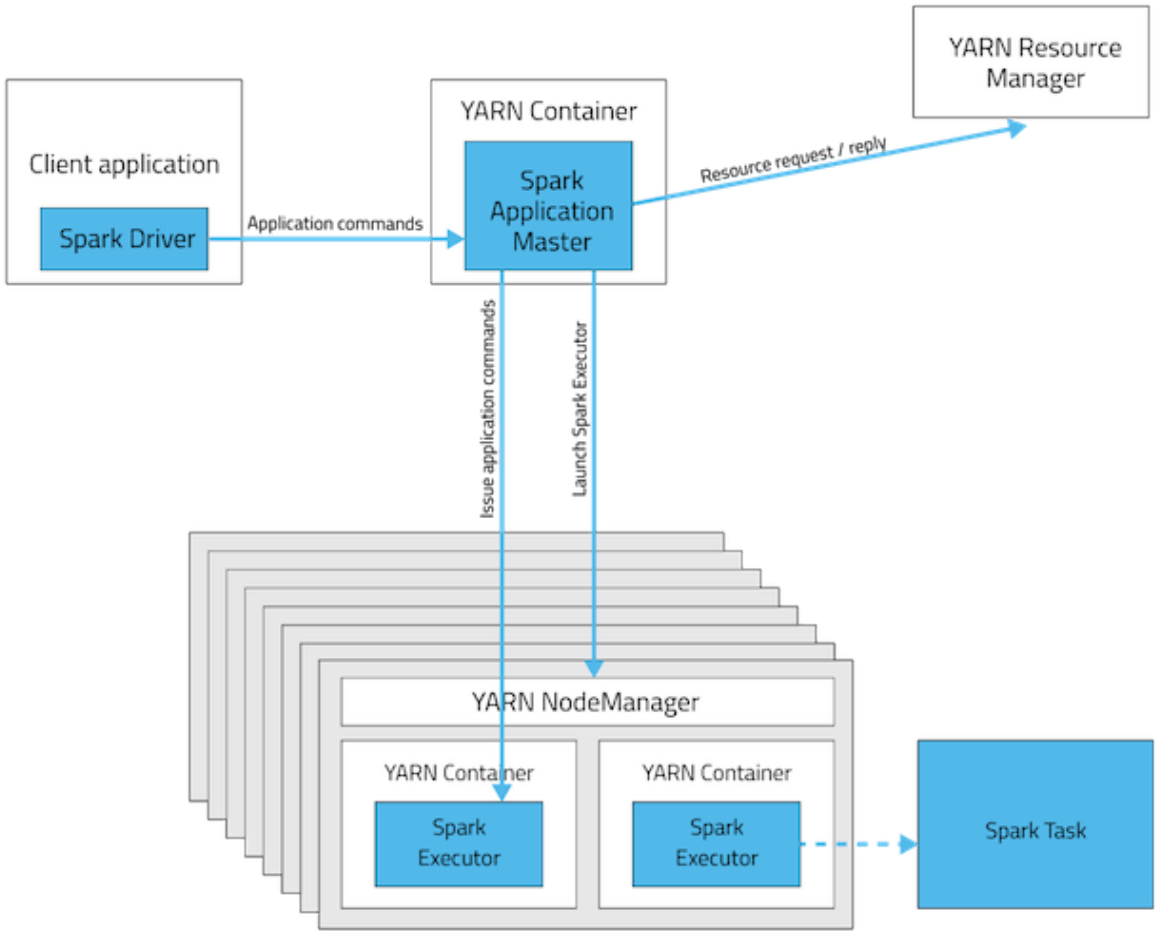
参考博客

`yarn-cluster` 和 `yarn-client` 模式的区别其实就是 `Application Master` 进程的区别，在 `yarn-cluster` 模式下，`driver` 运行在 `AM(Application Master)` 中，它负责向 `YARN` 申请资源，并监督作业的运行状况。当用户提交了作业之后，就可以关掉 `Client`，作业会继续在 `YARN` 上运行。然而 `yarn-cluster` 模式不适合运行交互类型的作业。 在 `yarn-client` 模式下，`Application Master` 仅仅向 `YARN` 请求 `executor`，`client` 会和请求的 `container` 通信来调度他们工作，也就是说 `Client` 不能离开。下面的图形象表示了两者的区别。

yarn-cluster



Yarn-client



第三章 spark运行参数的含义

参考博客1

参考博客2

参数名	格式	参数说明

参数名	格式	参数说明
--master	MASTER_URL	如spark://host:port
--deploy-mode	DEPLOY_MODE	Client或者master，默认是client
--class	CLASS_NAME	应用程序的主类
--name	NAME	应用程序的名称
--jars	JARS	逗号分隔的本地jar包，包含在driver和executor的classpath下
--packages		包含在driver和executor的classpath下的jar包逗号分隔的”groupId:artifactId：version”列表
--exclude-packages		用逗号分隔的”groupId:artifactId”列表
--repositories		逗号分隔的远程仓库
--py-files	PY_FILES	逗号分隔的”.zip”,”.egg”或者“.py”文件，这些文件放在python app的PYTHONPATH下面
--files	FILES	逗号分隔的文件，这些文件放在每个executor的工作目录下面
--conf	PROP=VALUE	固定的spark配置属性
--properties-file	FILE	加载额外属性的文件
--driver-memory	MEM	Driver内存，默认1G
--driver-java-options		传给driver的额外的Java选项
--driver-library-path		传给driver的额外的库路径
--driver-class-path		传给driver的额外的类路径
--executor-memory	MEM	每个executor的内存，默认是1G
--proxy-user	NAME	模拟提交应用程序的用户
--driver-cores	NUM	Driver的核数，默认是1。这个参数**仅仅在standalone集群deploy模式下使用**
--supervise		Driver失败时，重启driver。**在mesos或者standalone下使用**
--verbose		打印debug信息

参数名	格式	参数说明
--total-executor-cores	NUM	所有executor总共的核数。**仅仅在mesos或者standalone下使用**
--executor-core	NUM	每个executor的核数。**在yarn或者standalone下使用**
--driver-cores	NUM	Driver的核数，默认是1。**在yarn集群模式下使用**
--queue	QUEUE_NAME	队列名称。**在yarn下使用**
--num-executors	NUM	启动的executor数量。默认为2。**在yarn下使用**

可以通过`spark-submit --help`或者`spark-shell --help`来查看这些参数

yarn cluster模式

指定固定的executor数

```
spark-submit \
    --master yarn-cluster \
    --deploy-mode cluster \                    #集群运行模式
    --name wordcount_${date} \                 #作业名
    --conf spark.default.parallelism=1000 \  #并行度，shuffle后的默认partition数
    --conf spark.network.timeout=1800s \
    --conf spark.yarn.executor.memoryOverhead=1024 \   #堆外内存
    --conf spark.scheduler.executorTaskBlacklistTime=30000 \
    --conf spark.core.connection.ack.wait.timeout=300s \
    --num-executors 200 \                     #executor数目
    --executor-memory 4G \                    #executor中堆的内存
    --executor-cores 2 \                      #executor执行core的数目，设置大于1
    --driver-memory 2G \                      #driver内存，不用过大
    --class ${main_class} \                   #主类
    ${jar_path} \                             #jar包位置
    param_list \                              #mainClass接收的参数列表
```

动态调整executor数目

```
spark-submit \
    --master yarn-cluster \
    --deploy-mode cluster \
    --name wordcount_${date} \
    --conf spark.dynamicAllocation.enabled=true \     #开启动态分配
    --conf spark.shuffle.service.enabled=true \       #shuffle service，可以保证
executor被删除时，shuffle file被保留
    --conf spark.dynamicAllocation.minExecutors=200 \ #最小的executor数目
    --conf spark.dynamicAllocation.maxExecutors=500 \ #最大的executor数目
    --class ${main_class} \
    ${jar_path} \
    param_list
```

yarn client模式

```
spark-shell \
    --master yarn-client \
    --queue production.group.yanghao \      #指定队列
    --num-executors 200 \                   #executor数目
    --executor-memory 4G \                  #executor中堆的内存
    --executor-cores 2 \                    #executor执行core的数目，设置大于1
    --driver-memory 2G \                    #driver内存，不用过大
    --jars ${jar_path}                      #jar包位置
```