1. 前言

Apache Spark是一个围绕速度、易用性和复杂分析构建的大数据处理框架,Spark是用Scala程序设计语言编写而成，运行于Java虚拟机（JVM）环境之上

Spark运行在现有的Hadoop分布式文件系统基础之上（HDFS）提供额外的增强功能。它支持将Spark应用部署到现存的Hadoop v1集群（with SIMR – Spark-Inside-MapReduce）或Hadoop v2 YARN集群甚至是Apache Mesos之中。也有自己的资源管理器(Standalone),可以脱离Hadoop生态圈独立存在

Spark通过在数据处理过程中成本更低的洗牌（Shuffle）方式，将MapReduce提升到一个更高的层次。

Spark将中间结果保存在内存中而不是将其写入磁盘，当内存放了足够多的数据时,会放在磁盘上(有存储策略),所以Spark可以用于处理大于集群内存容量总和的数据集

Spark允许程序开发者使用有向无环图（DAG）开发复杂的多步数据管道。而且还支持跨有向无环图的内存数据共享，以便不同的作业可以共同处理同一个数据。

2. 生态系统

Spark生态圈以Spark Core为核心，从HDFS、Amazon S3和HBase等持久层读取数据，以MESS、YARN和自身携带的Standalone为资源管理器调度Job完成Spark应用程序的计算。 这些应用程序可以来自于不同的组件，如Spark Shell/Spark Submit的批处理、Spark Streaming的实时处理应用、Spark SQL的即时查询、BlinkDB的权衡查询、MLlib/MLbase的机器学习、GraphX的图处理和SparkR的数学计算等等。

- **Spark Core:**

实现了Spark的基本功能，包含任务调度、内存管理、错误恢复、与存储系统交互等模块。Spark Core中还包含了对弹性分布式数据集（resilient distributed dataset，简称RDD）的API定义。 Spark Core提供了创建和操作这些集合的多个API。

来自〈https://www.douban.com/note/536766108/?from=tag〉

- **Spark Streaming:**

Spark Streaming基于微批量方式的计算和处理，可以用于处理实时的流数据。它使用DStream，简单来说就是一个弹性分布式数据集（RDD）系列，处理实时数据。

- **Spark SQL:**

Spark SQL可以通过JDBC API将Spark数据集暴露出去，而且还可以用传统的BI(商业智能:提供报表展示分析帮助企业作出决策)和可视化工具在Spark数据上执行类似SQL的查询。用户还可以用Spark SQL对不同格式的数据（如JSON，Parquet以及数据库等）执行ETL，将其转化，然后暴露给特定的查询。

- **Spark MLlib:**

MLlib是一个可扩展的Spark机器学习库，由通用的学习算法和工具组成，包括二元分类、线性回归、聚类、协同过滤、梯度下降以及底层优化原语。

- **Spark GraphX:**

GraphX是用于图计算和并行图计算的新的（alpha）Spark API。通过引入弹性分布式属性图（Resilient Distributed Property Graph），一种顶点和边都带有属性的有向多重图，扩展了Spark RDD。为了支持图计算，GraphX暴露了一个基础操作符集合（如subgraph，joinVertices和aggregateMessages）和一个经过优化的Pregel API变体。此外，GraphX还包括一个持续增长的用于简化图分析任务的图算法和构建器集合。

除了这些库以外，还有一些其他的库，如BlinkDB和Tachyon。

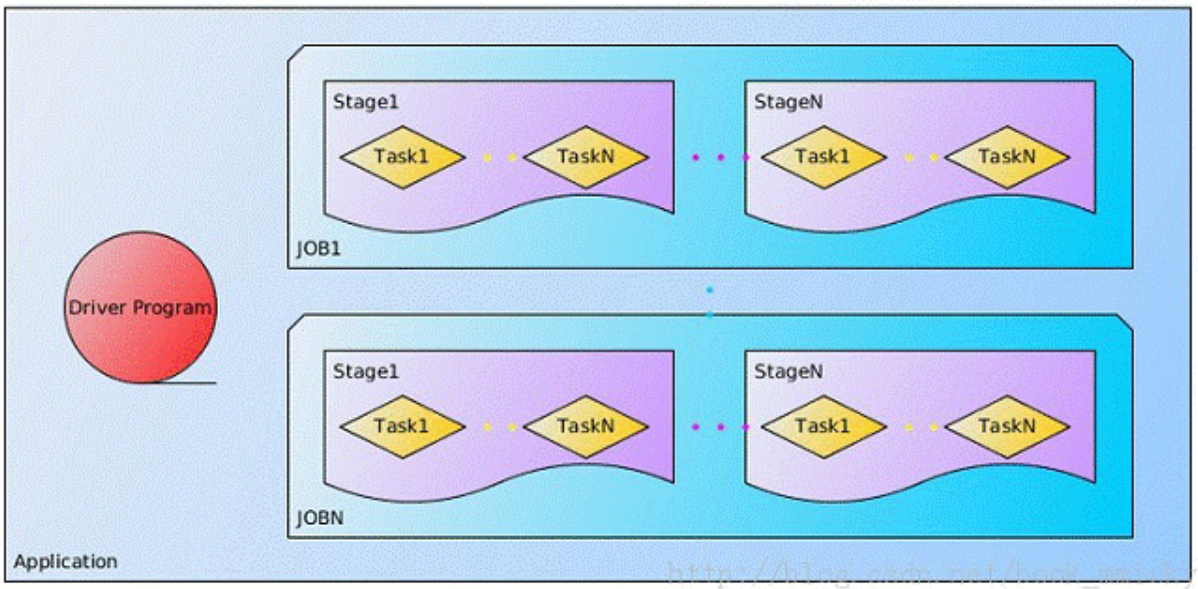
Spark常用术语

术语	描述
Application	Spark的应用程序，包含一个Driver program和若干Executor
SparkContext	Spark应用程序的入口，负责调度各个运算资源，协调各个Worker Node上的E
Driver Program	运行Application的main()函数并且创建SparkContext

Executor	是为Application运行在Worker node上的一个进程，该进程负责运行Task，并 每个Application都会申请各自的Executor来处理任务
Cluster Manager	在集群上获取资源的外部服务 (例如：Standalone、Mesos、Yarn)
Worker Node	集群中任何可以运行Application代码的节点，运行一个或多个Executor进程
Task	运行在Executor上的工作单元(rdd的转换过程)
Job	SparkContext提交的具体Action操作，常和Action对应,一个JOB包含多个RD 解为一个action表示一个job)
Stage	每个Job会被拆分很多组task，每组任务被称为Stage，也称TaskSet(窄依赖是-
RDD	是Resilient distributed datasets的简称，中文为弹性分布式数据集;是Spark最
DAGScheduler	根据Job构建基于Stage的DAG，并提交Stage给TaskScheduler
TaskScheduler	将Taskset提交给Worker node集群运行并返回结果
Transformations	是Spark API的一种类型，Transformation返回值还是一个RDD， 所有的Transformation采用的都是懒策略，如果只是将Transformation提交是
Action	是Spark API的一种类型，Action返回值不是一个RDD，而是一个scala集合；i (懒计算)。
worker	集群中任何可以运行Application代码的节点，类似于YARN中的NodeMana 通过Slave文件配置的Worker节点，在Spark on Yarn模式中指的就是Node

来自〈http://www.cnblogs.com/shishanyuan/p/4700615.html〉

运行架构图：



3.弹性分布式数据集

弹性分布式数据集或RDD（Resilient Distributed Datasets）是Spark框架中的核心概念。可以将RDD视作数据库中的一张表。其中可以保存任何类型的数据。Spark将数据存储在不同分区上的RDD之中,一个RDD中有多个分区,这些分区可以分布在不同节点上(也就是说,RDD是分布存储的),分区的多少涉及对这个RDD进行并行计算的粒度,每个RDD分区计算操作都在一个单独的任务中被执行,分区个数可以自行指定和改变

RDD可以帮助重新安排计算并优化数据处理过程。

此外，它还具有容错性，因为RDD知道如何重新创建和重新计算数据集。

RDD是不可变的,是只可读的。你可以用变换（Transformation）操作修改RDD,但是这个变换所返回的是一个全新的RDD，而原有的RDD仍然保持不变(有点像String的不变性),也就是说,在丢失或者操作失败后都是可以重建的,具有容错。

spark五大特性(源自rdd类注释)

1. 是分区(可以存储在不同节点)的集合,因为一个rdd包含了多个分区的数据,把block块的东西整合到rdd(字段名：dependencies 存储在seq数据集中,类型是dependency,便利和取第一个)
2. 对每个分片并行计算(一般情况下分片大小等于分区大小)(名为computer函数,可重写)
3. 是其他rdd的依赖集合,就是知道该rdd从哪来,便于回溯(若宕机,存于内存中的rdd会丢失,spark会借由此重算)(字段名：partitions 存储Array中,类型是Partition ,便于通过下标获取)
4. (可选)可以重新分区(调节并行度)(一个分区对应一个并行度) (主要是k-v形式的rdd有)
5. (可选)给每个分片找到优先数据位置(找最近的数据处理最快嘛)(主要是来源有多个备份的rdd,例如HDFS文件,因为重写了getPreferredLocaltions方法)

> 可选：只有部分类型rdd才有的特性,

RDD支持两种类型的操作：

- 变换（Transformation）

- 行动（Action）

变换：变换的返回值是一个新的RDD集合，而不是单个值。调用一个变换方法，不会有任何求值计算，它只获取一个RDD作为参数，然后返回一个新的RDD。
变换函数包括：map，filter，flatMap，groupByKey，reduceByKey，aggregateByKey，pipe和coalesce等。
行动：行动操作计算并返回一个新的值。当在一个RDD对象上调用行动函数时，会在这一时刻计算全部的数据处理查询并返回结果值(Driver会接收到)。
行动操作包括：reduce，collect，count，first，take，countByKey以及foreach等。

注：

1.只有在行动(action)时才会触发运算,也就是说变换是不会运行的计算的;
2.RDD是粗粒度计算的.粗粒度:一个转化或者行动会把整个RDD里面的东西都进行操作

3.1 RDD依赖关系

由于RDD是粗粒度的操作数据集，每个Transformation操作都会生成一个新的RDD，所以RDD之间就会形成类似流水线的前后依赖关系,进而形成了有向无环图（DAG）；

在spark中，RDD之间存在两种类型的依赖关系：窄依赖(Narrow Dependency)和宽依赖(Wide Dependency)

窄依赖 :是指每个父RDD的一个Partition最多被子RDD的一个Partition所使用，例如map、filter、union等操作都会产生窄依赖；(父RDD的分区都在相同的节点)
宽依赖 :是指每个父RDD的一个Partition会被多个子RDD的Partition所使用，例如groupByKey、reduceByKey、sortByKey等操作都会产生宽依赖；(可能跨越多个节点妈的)

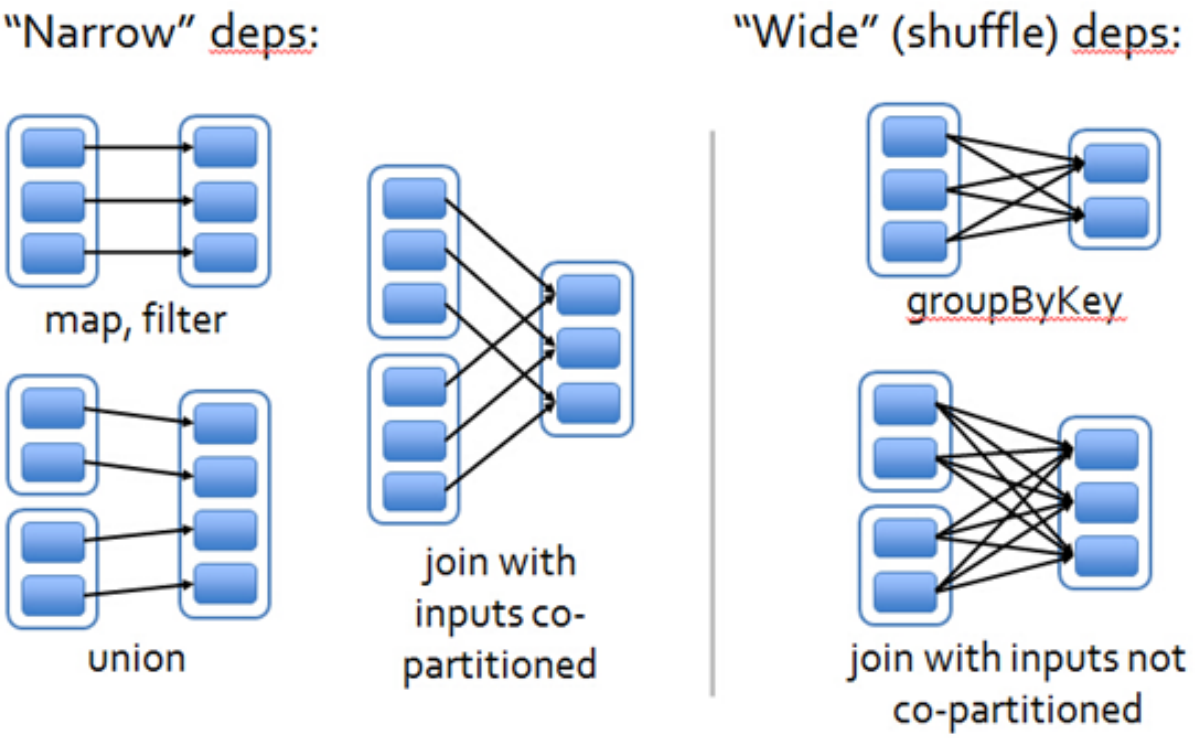
需要特别说明的是对join操作有两种情况：

- 如果两个RDD在进行join操作时，一个RDD的partition仅仅和另一个RDD中已知个数的Partition进行join，那么这种类型的join操作就是窄依赖，例如图1中左半部分的join操作(join with inputs co-partitioned)；
- 其它情况的join操作就是宽依赖,例如图1中右半部分的join操作(join with inputs not co-partitioned)，由于是需要父RDD的所有partition进行join的转换，这就涉及到了shuffle，因此这种类型的join操作也是宽依赖。

> 简单的说,就是元素被用几次,只用一次就是窄依赖,超过一次就是宽依赖

之所以这么区分依赖关系，是因为它们有本质的区别。使用窄依赖时，可以精确知道依赖的上级RDD分区。这样便于回溯。而宽依赖则开销会大。RDD仔细维护者这种依赖关系和计算方法,使得通过重新计算来恢复RDD成为可能。如果链条太长，则恢复代价太大，所以spark又提出一种检查点的机制。

来自〈http://bbs.pinggu.org/thread-4637506-1-1.html〉



3.2 RDD运行原理

那么 RDD在Spark架构中是如何运行的呢？总高层次来看，主要分为三步：

1. 创建 RDD 对象
2. DAGScheduler模块介入运算，计算RDD之间的依赖关系。RDD之间的依赖关系就形成了DAG
3. 每一个JOB被分为多个Stage，划分Stage的一个主要依据是当前计算因子的输入是否是确定的，如果是则将其分在同一个Stage，避免多个Stage之间的消息传递开销。

来自〈http://www.cnblogs.com/shishanyuan/p/4721326.html〉

3.2.1 DGA调度

sparkcontext在初始化时,创建了DAG调度与task调度来负责RDD action操作的调度执行。

3.2.1.1 DAGScheduler

DAGSchedule负责spark的最高级别的任务调度调度的力度是stage，它为每个job的所有stage计算一个有向无环图控制他们的并发，便找到一个最佳的路径来执行他们。具体的执行过程是将stage下的task集提交给Taskschedule对象，由它来提交到集群上去申请资源并最终完成执行。

1.runjob过程

所有需要执行的RDD action都会调用sparkCcontext.runJob来提交任务，而SparkContest.runjob调用的是DAGScheduler.runjob， sunjob调用submitjob提交任务,并等待任务结束.提交任务是不是按job的先后顺序提交的,而是倒序,每个job的最后一个操作是action操作，DAG把这最后的action操作操作当做一个stage首先提交，然后逆向逐级递规填补缺少的上级stage，从而生成一颗实现最后action操作的最短有效无环图，然后从头开始计算。

任务提交后的处理过程大致如下:

1.submitJob生成新的job id, 发送消息jobsubmitted。

2.DAG收到jobSubmitted消息,调用handleJobSubmitted来处理

3.handleJobSubmitted创建一个ResultStage,,并使用submitStage来提交这个ResultStage

3.2.1.2 TaskSched

相对DAG schedule而言tasked sketches低级别的调度接口。允许实现不同的task调度器。每个task sketched对象只服务于一个sparkleContext的task调度。taskScheduler从DAGScheduler的每个 stage 接受一组task，并负责将他们送到集群上运行他们。

4.Spark on YARN运行过程

spark运行在不同的资源管理器上有不同的运行过程,这里解释在YARN平台的运行,详情见

<http://www.cnblogs.com/shishanyuan/p/4721326.html>

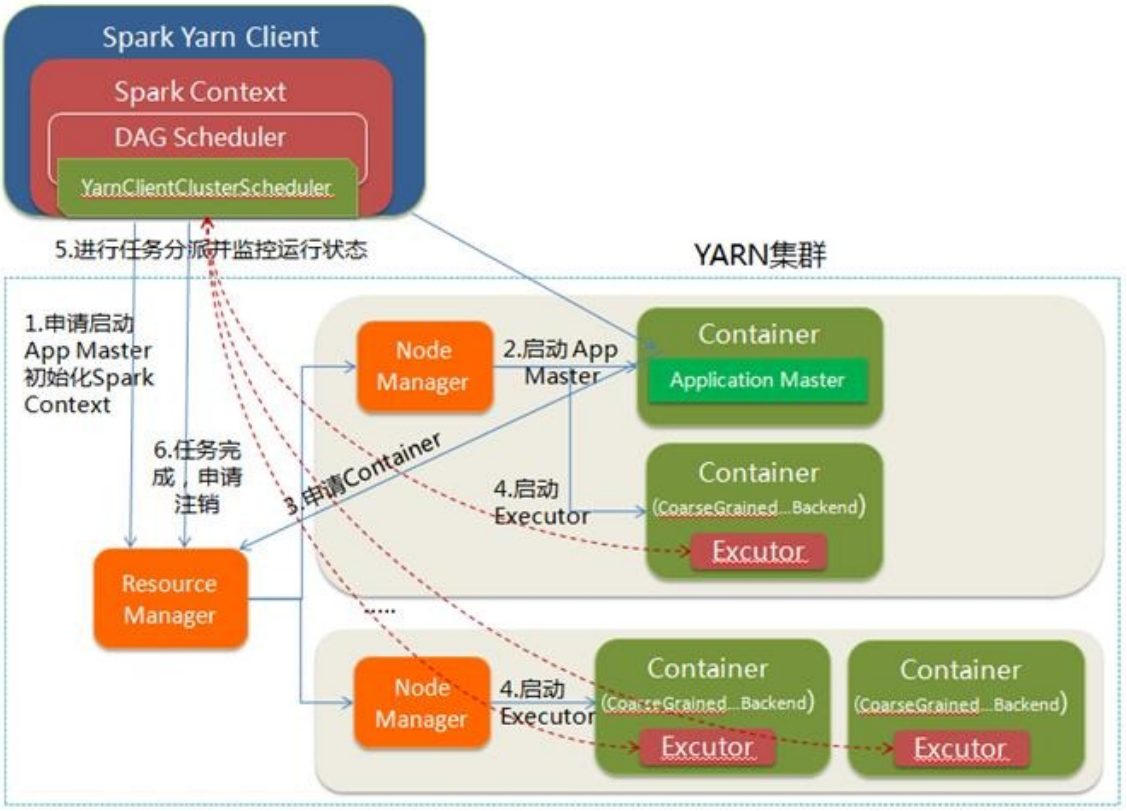
Spark运行模式

运行环境	模式	描述
Local	本地模式	常用于本地开发测试，本地还分为local单线程和local-cluster多线程
Standalone	集群模式	典型的Mater/slave模式，不过也能看出Master是有单点故障的；S 现HA
On yarn	集群模式	运行在yarn资源管理器框架之上，由yarn负责资源管理，Spark负责
On mesos	集群模式	运行在mesos资源管理器框架之上，由mesos负责资源管理，Spark
On cloud	集群模式	比如AWS的EC2，使用这个模式能很方便的访问Amazon的S3; Spark支持多种分布式存储系统：HDFS和S3

Spark on YARN模式根据Driver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（或称为YARN-Standalone模式）。

4.1 YARN-Client

Yarn-Client模式中，Driver在客户端本地运行，这种模式可以使得Spark Application和客户端进行交互，因为Driver在客户端，所以可以通过webUI访问Driver的状态，默认是http://hadoop1:4040访问，而YARN通过http:// hadoop1:8088访问。

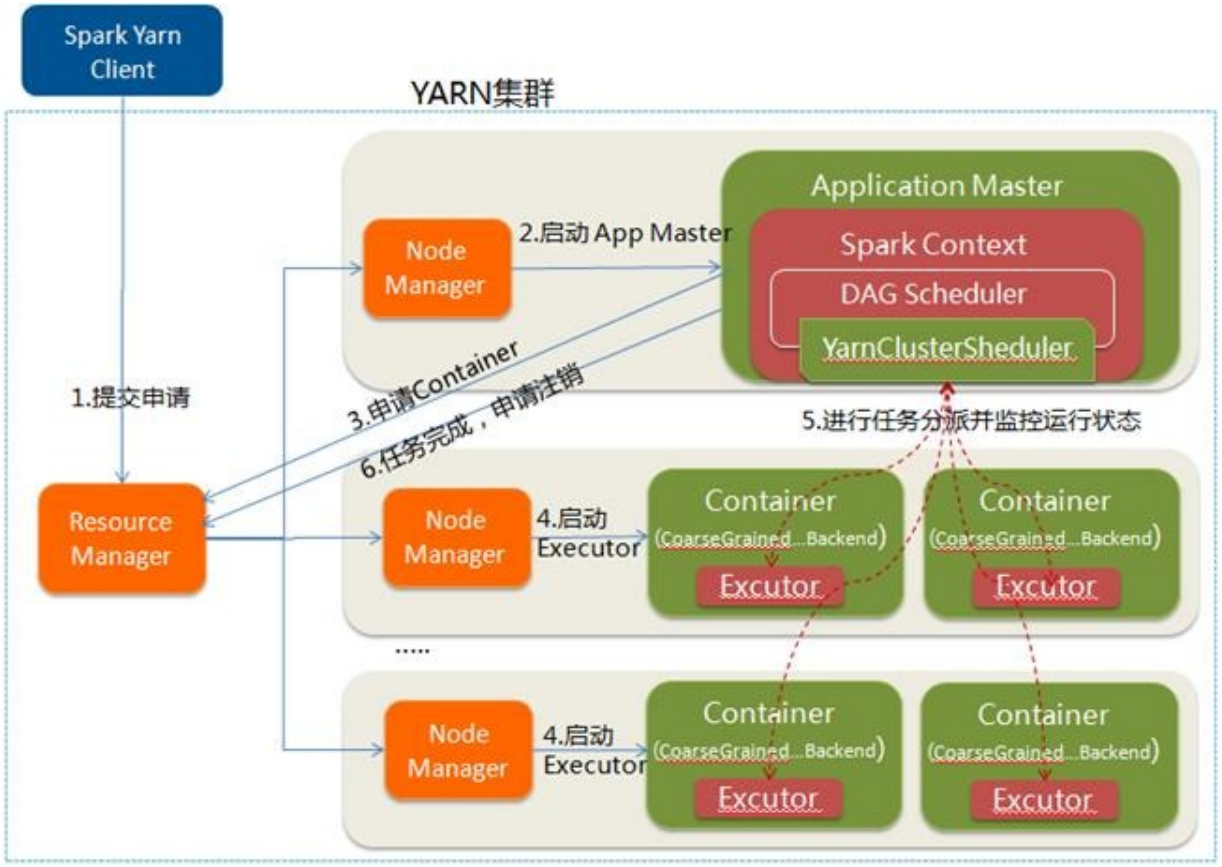


4.2 YARN-Cluster

在YARN-Cluster模式中，当用户向YARN中提交一个应用程序后，YARN将分两个阶段运行该应用程序：

第一个阶段是把Spark的Driver作为一个ApplicationMaster在YARN集群中先启动；

第二个阶段是由ApplicationMaster创建应用程序，然后为它向ResourceManager申请资源，并启动Executor来运行Task，同时监控它的整个运行过程，直到运行完成。



YARN-Client 与 YARN-Cluster 区别

理解YARN-Client和YARN-Cluster深层次的区别之前先清楚一个概念：Application Master。在YARN中，每个Application实例都有一个ApplicationMaster进程，它是Application启动的第一个容器。它负责和ResourceManager打交道并请求资源，获取资源之后告诉NodeManager为其启动Container。从深层次的含义讲YARN-Cluster和YARN-Client模式的区别其实就是ApplicationMaster进程的区别。

- 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，因而YARN-Cluster模式不适合运行交互类型的作业；
- 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。

来自〈http://www.cnblogs.com/shishanyuan/p/4721326.html〉