Spark文档整理

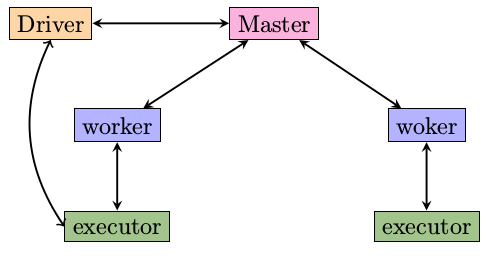
# 一、关于Spark

### 简介

在**[Hadoop](http://lib.csdn.net/base/hadoop" \o "Hadoop知识库" \t "_blank)**的整个生态系统中，Spark和MapReduce在同一个层级，即主要解决分布式计算框架的问题。

### 架构

Spark的[架构](http://lib.csdn.net/base/architecture" \o "大型网站架构知识库" \t "_blank)如下图所示，主要包含四大组件：Driver、Master、Worker和Executor。



### Spark特点

Spark可以部署在YARN上

Spark原生支持对HDFS文件系统的访问

使用Scala语言编写

### 部署模型

单机模型：主要用来开发测试。特点：Driver、Master、Worker和Executor都运行在同一个JVM进程之中。

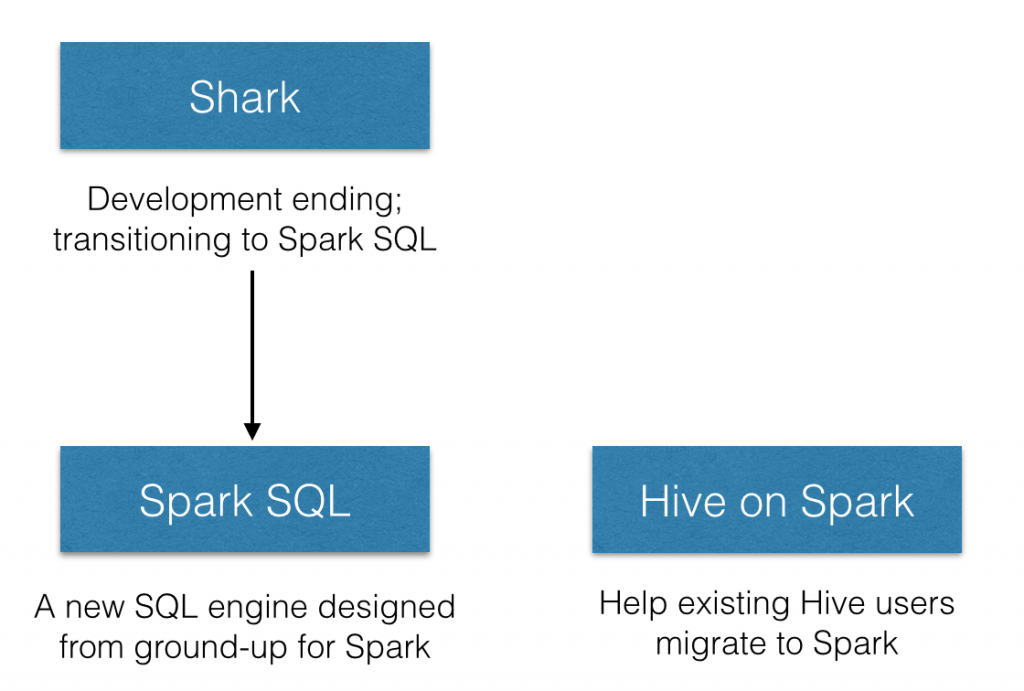
伪集群模型：主要用来开发测试。特点：Master、Worker都运行在同一个JVM进程之中；Master、Worker和Executor都运行于同一台机器，无法跨机器运行；

独立集群（又叫做原生集群模式）：在集群规模不是非常大的情况下，可用于生产环境。特点：Master、Worker和Executor都运行于独立的JVM进程。

**YARN集群**：YARN生态中的ApplicationMaster角色使用Apache开发好的Spark ApplicationMaster代替，每一个YARN生态中的NodeManager角色相当于一个Spark生态中的Worker角色，由NodeManger负责Executor的启动。

Mesos集群：暂无详细调研。

# 二、关于shark



Shark演变为Spark SQL(主要)和 Hive on Spark

### Spark SQL

##### ****简介****

它主要用于结构化数据处理和对Spark数据执行类SQL的查询。通过Spark SQL，可以针对不同格式的数据执行ETL操作（如JSON，Parquet，数据库）然后完成特定的查询操作。一般来说，Spark每支持一种新的应用开发，都会引入一个新的Context及相应的RDD，对于SQL这一特性来说，引入的就是SQLContext和SchemaRDD。注意：在Spark1.3之后，SchemaRDD已经更名为DataFrame，但它本质就类似一个RDD，因为可以将DataFrame无缝的转换成一个RDD

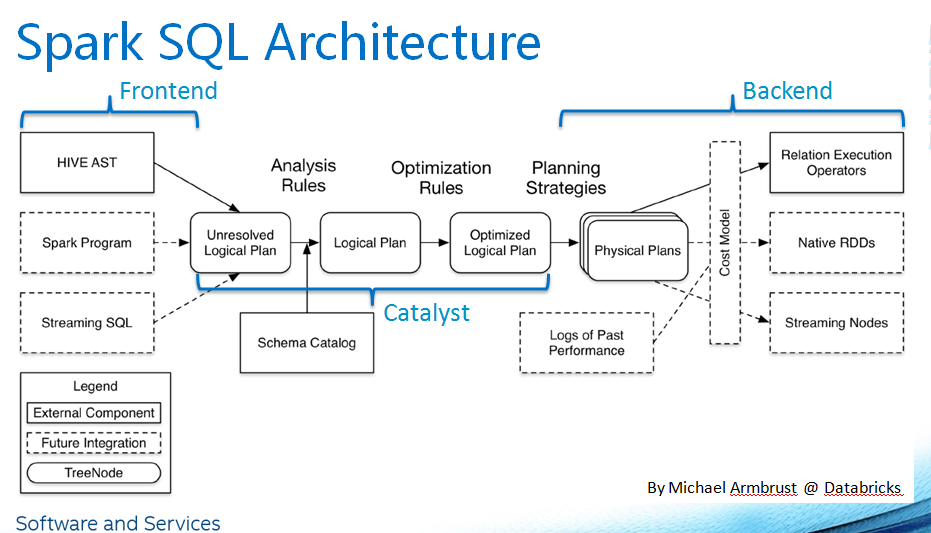
RDD，全称为Resilient Distributed Datasets，是一个容错的、并行的数据结构，可以让用户显式地将数据存储到磁盘和内存中，并能控制数据的分区。同时，RDD还提供了一组丰富的操作来操作这些数据。在这些操作中，诸如map、flatMap、filter等转换操作实现了monad模式，很好地契合了Scala的集合操作。除此之外，RDD还提供了诸如join、groupBy、reduceByKey等更为方便的操作（注意，reduceByKey是action，而非transformation），以支持常见的数据运算。

##### ****架构****

Spark要很好的支持SQL，要完成解析(parser)、优化(optimizer)、执行(execution)三大过程。

处理顺序大致如下：

1. SQlParser生成LogicPlan Tree；
2. Analyzer和Optimizer将各种Rule作用于LogicalPlan Tree；
3. 最终优化生成的LogicalPlan生成SparkRDD；
4. 最后将生成的RDD交由Spark执行；



##### ****Spark SQL的两个组件****

1. SQLContext：Spark SQL提供SQLContext封装Spark中的所有关系型功能。可以用之前的示例中的现有SparkContext创建SQLContext。
2. DataFrame：DataFrame是一个分布式的，按照命名列的形式组织的数据集合。DataFrame基于R语言中的data frame概念，与关系型数据库中的数据库表类似。通过调用将DataFrame的内容作为行RDD（RDD of Rows）返回的rdd方法，可以将DataFrame转换成RDD。可以通过如下数据源创建DataFrame：已有的RDD、结构化数据文件、JSON数据集、Hive表、外部数据库。

### Hive on Spark

##### ****背景****

Hive on Spark是由Cloudera发起，由Intel、MapR等公司共同参与的开源项目，其目的是把Spark作为Hive的一个计算引擎，将Hive的查询作为Spark的任务提交到Spark集群上进行计算。通过该项目，可以提高Hive查询的性能，同时为已经部署了Hive或者Spark的用户提供了更加灵活的选择，从而进一步提高Hive和Spark的普及率。

##### ****简介****

Hive on Spark是从Hive on MapReduce演进而来，Hive的整体解决方案很不错，但是从查询提交到结果返回需要相当长的时间，查询耗时太长，这个主要原因就是由于Hive原生是基于MapReduce的，那么如果我们不生成MapReduce Job，而是生成Spark Job，就可以充分利用Spark的快速执行能力来缩短HiveQL的响应时间。

Hive on Spark现在是Hive组件(从Hive1.1 release之后)的一部分。

##### spark计算引擎

以 Hive 的表作为RDD

Spark 以分布式可靠数据集（Resilient Distributed Dataset，RDD）作为其数据抽象，因此我们需要将Hive 的表转化为RDD 以便Spark 处理。本质上，Hive的表和Spark Hadoop RDD 都是HDFS 上的一组文件，通过InputFormat 和RecordReader 读取其中的数据，因此这个转化是自然而然的。

使用Hive 原语

这 里主要是指使用Hive 的操作符对数据进行处理。Spark 为RDD 提供了一系列的转换（Transformation），其中有些转换也是面向SQL 的，如groupByKey、join 等。但如果使用这些转换（就如Shark 所做的那样），就意味着我们要重新实现一些Hive 已有的功能；而且当 Hive 增加新的功能时，我们需要相应地修改Hive on Spark模式。有鉴于此，我们选择将Hive的操作符包装为Function，然后应用到RDD 上。这样，我们只需要依赖较少的几种RDD 的转换，而主要的计算逻辑仍由Hive 提供。由于使用了Hive 的原语，因此我们需要显式地调用一些Transformation 来实现Shuffle 的功能。下表中列举了Hive on Spark 使用的所有转换。



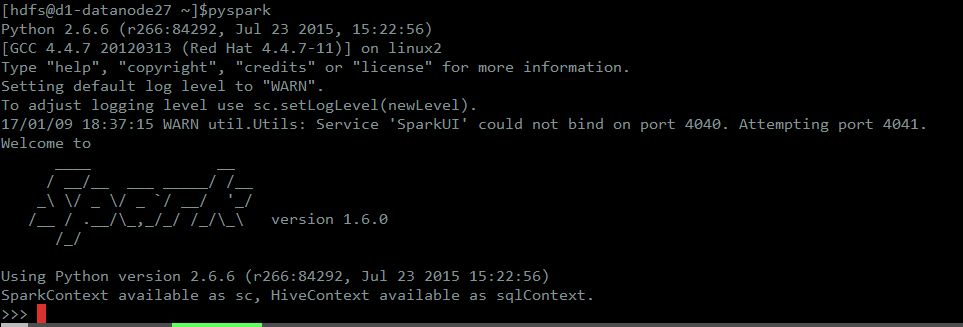
对repartitionAndSortWithinPartitions 简单说明一下，这个功能由SPARK-2978 引入，目的是提供一种MapReduce 风格的Shuffle。虽sortByKey也提供了排序的功能，但某些情况下我们并不需要全局有序，另外其使用的Range Partitioner 对于某些Hive 的查询并不适用。

# 三、SparkSql使用示例

目前测试环境中已安装spark on yarn：

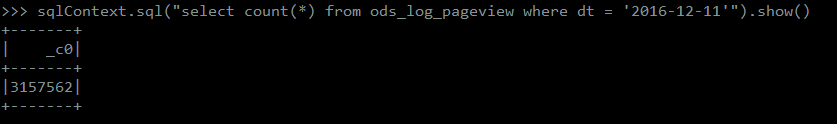
### pyspark启动SparkContext实例测试

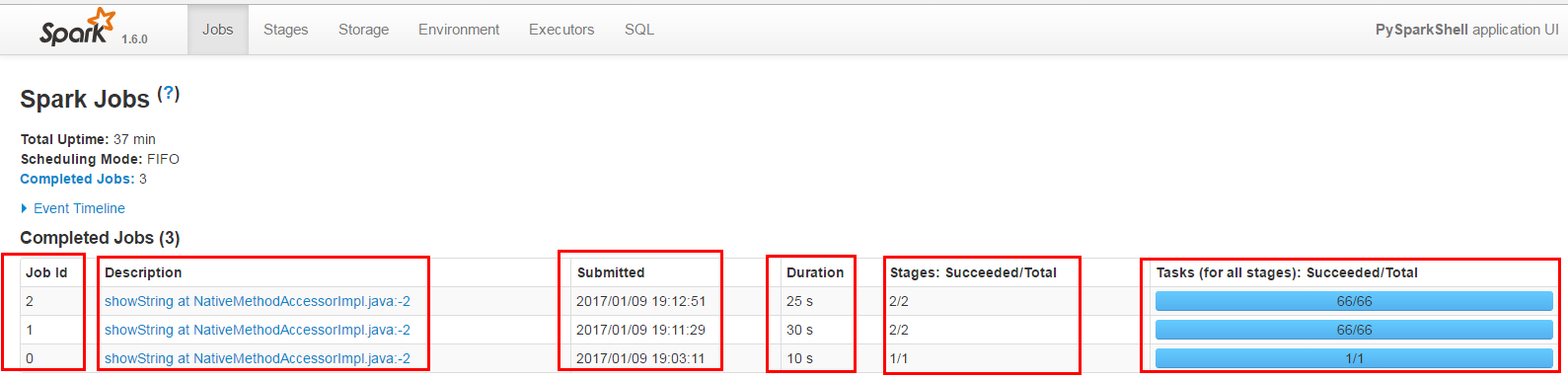
pyspark可以启动一个SparkContext实例



sqlContext = HiveContext(sc)

sqlContext.sql("select count(\*) from ods\_log\_pageview where dt = '2016-12-11'").show()





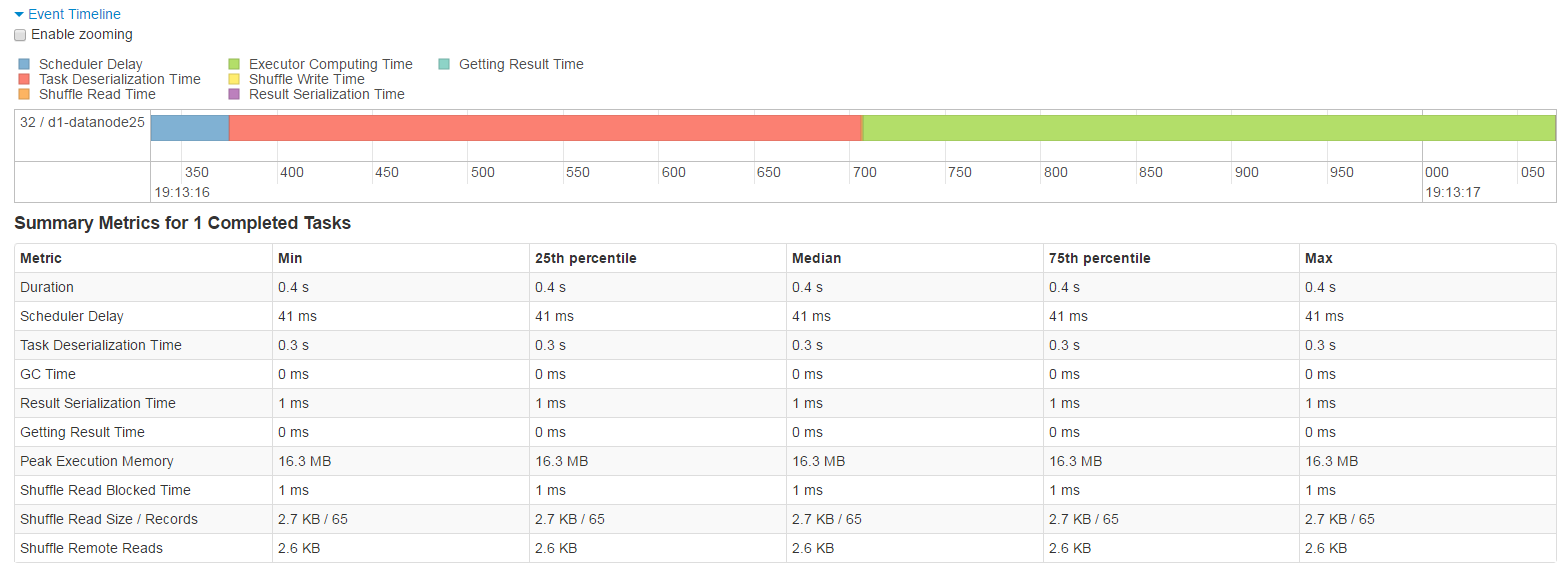
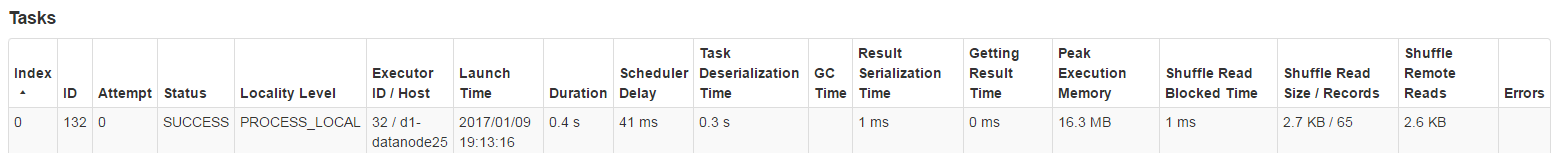
Job 一次完整的计算称为一个

job Submitted 提交任务的时间

Duration 耗时：这个10s是很慢的。

Stages 执行的阶段，现在的任务比较简单，执行很快，所以就一个阶段，复杂任务会有 多个。

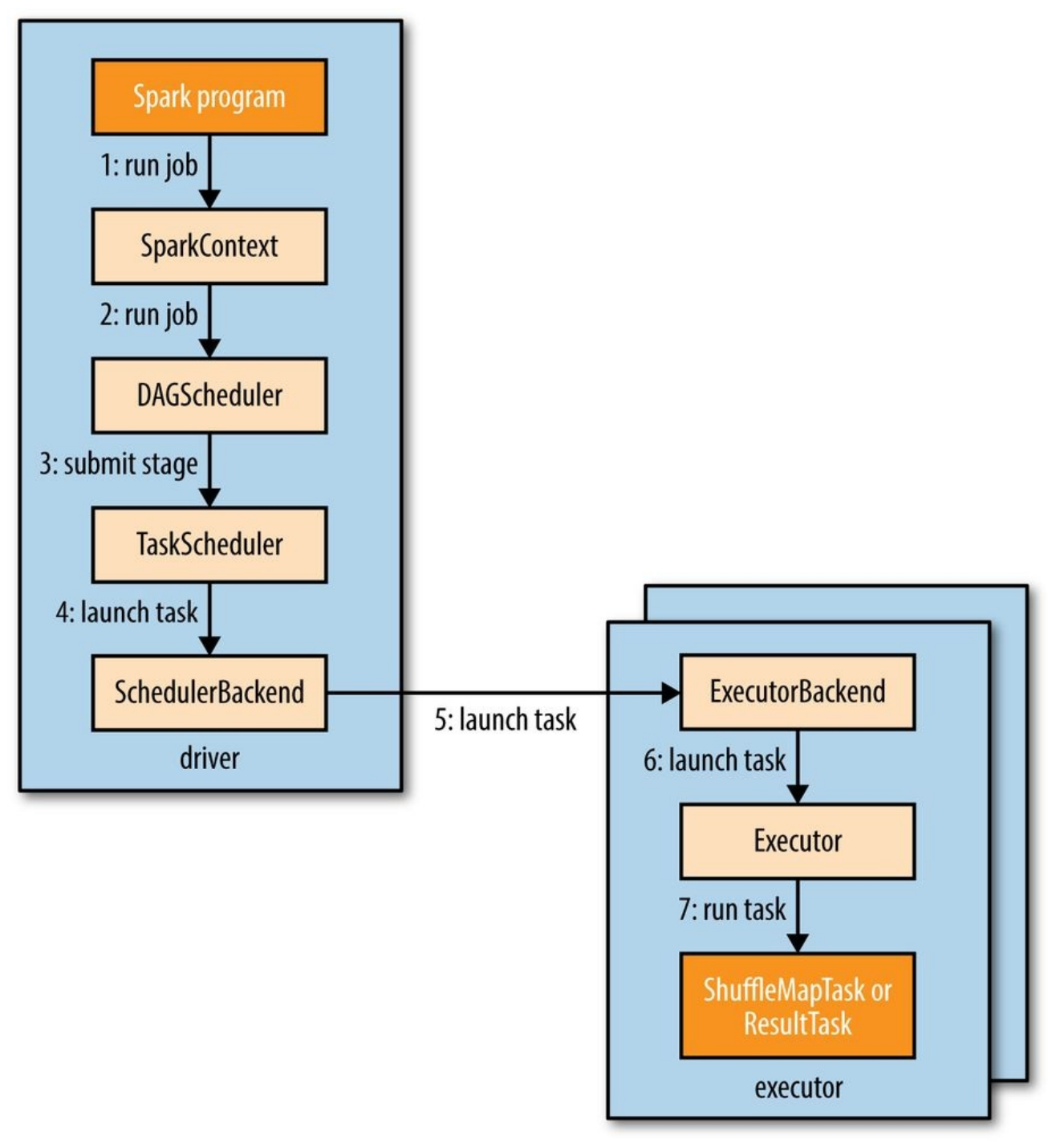
Tasks 任务：成功执行的任务/总共的任务数





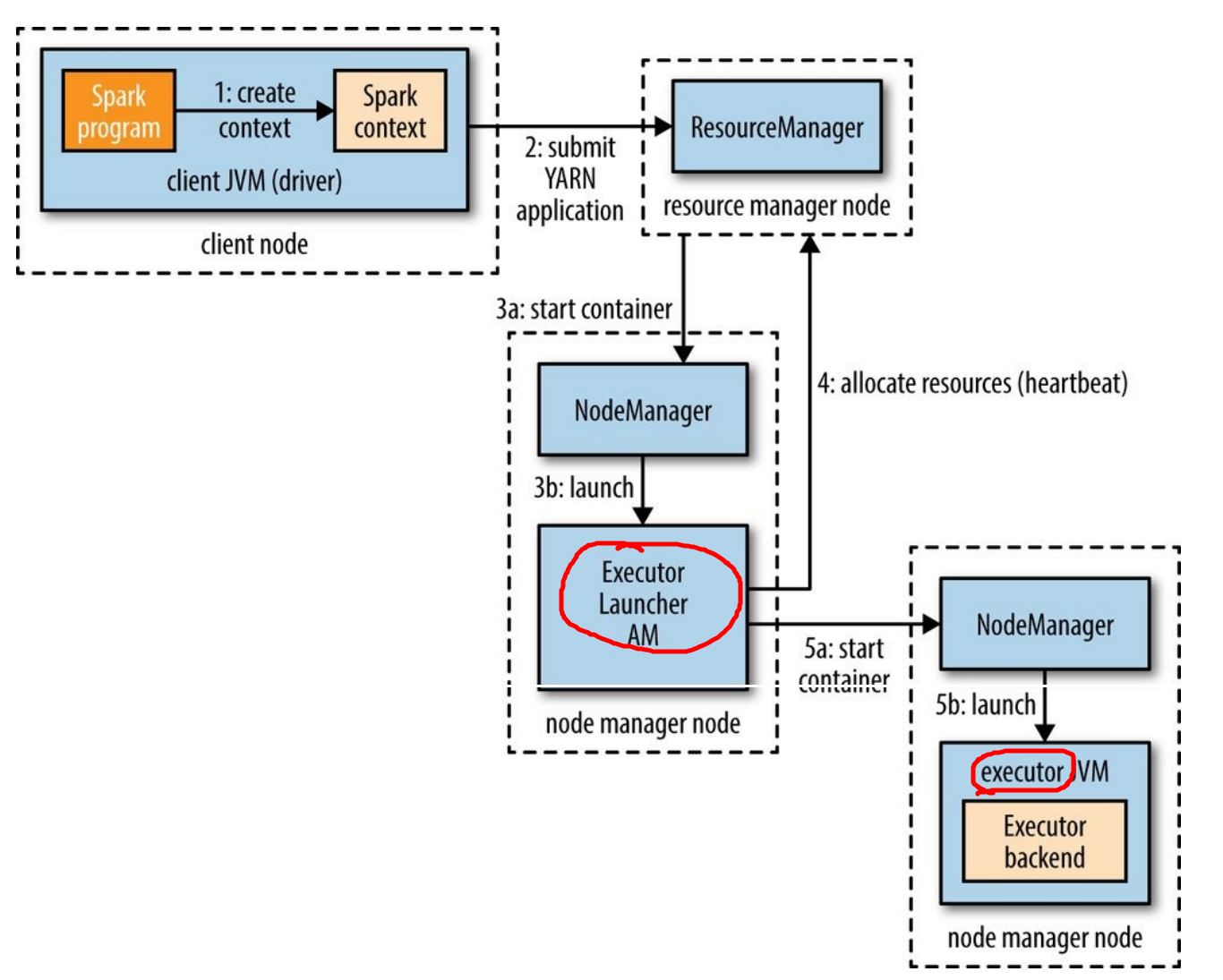
### Spark作业运行过程

Spark作业的运行从宏观上看，只要由driver和executors组成。driver运行application（SparkContext）、调度作业（schedule tasks）。executor负责具体任务的执行。通常情况下，driver运行在client机器上（client一般不受整个集群管理），但是在YARN的cluster模式上，driver运行在Application Master上。

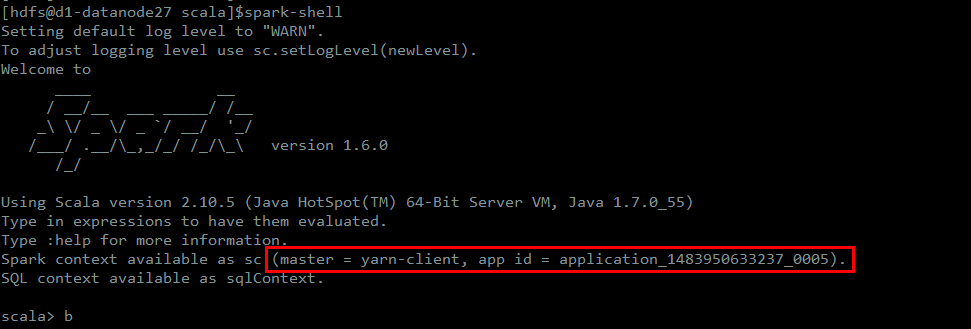


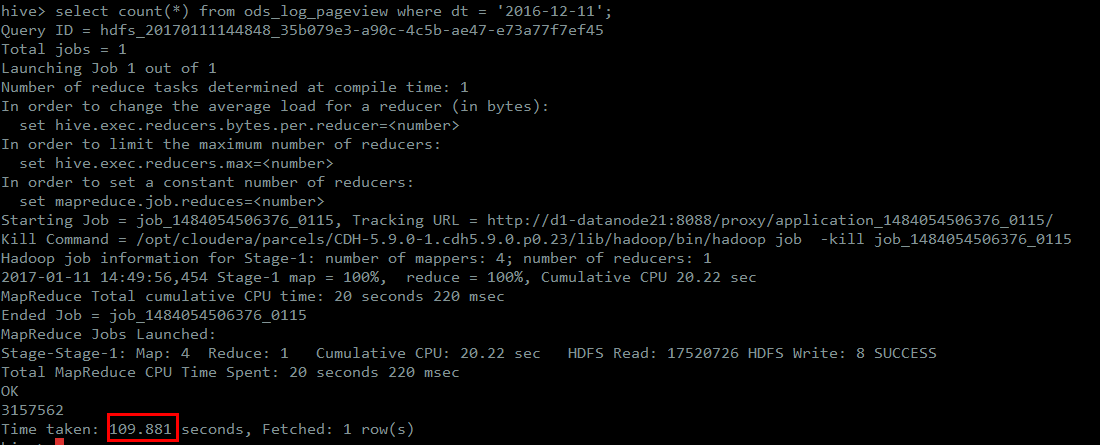
当有Action在RDD上执行时，作业被自动提交，提交将调用SparkContext的run\_job被调用，进而把作业提交给Scheduler。Scheduler运行在driver上，由两部分组成：

* DAG Scheduler：负责将作业分解为stage组成的DAG
* Task Scheduler：负责提交每个stage的Task到集群。
* 在YARN client模式上，在driver构造出SparkContext实例(下图step1)的时候，就开始于YARN进行交互。Context向YARN的资源管理器（RM）提交一个应用（step2），RM在集群的NodeManager中启动一个Container，并在Container中运行Spark ExecutorLauncher（step3）。ExecutorLauncher的任务是向RM申请资源（step4），并在申请到资源后将Executor Backend作为容器在相应的NodeManager中启动（step5)



### Spark-shell启动一个sqlContext实例



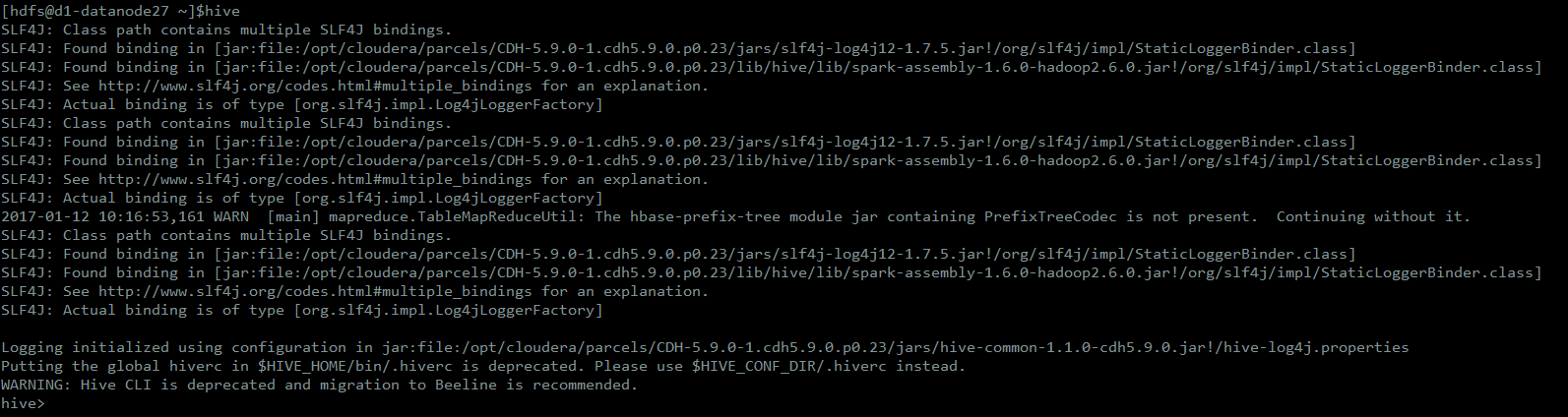


# 四、Hive on spark使用示例

### 加载spark-assembly\_jar包

要想在Hive中使用Spark执行引擎，第一步当前就是环境设置，我们需要在Hive启动的时候加载spark-assembly-1.6.0-hadoop2.6.0.jar，最简单的方法是把spark-assembly-1.6.0-hadoop2.6.0.jar包直接拷贝到$HIVE\_HOME/lib目录下。

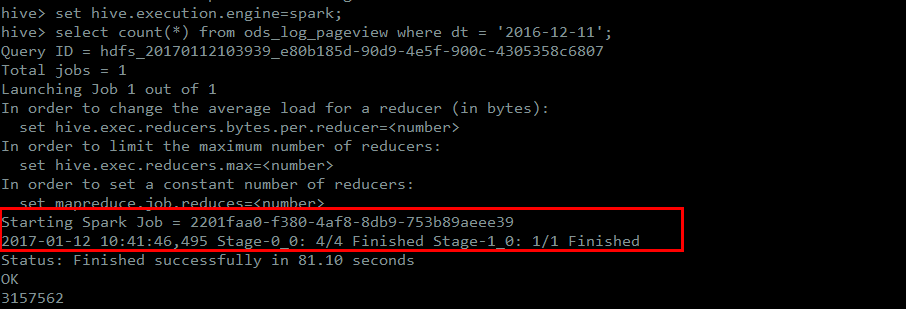
启动hive会自动加载spark所运行的spark-assembly\_jar包



### 设置hive执行引擎

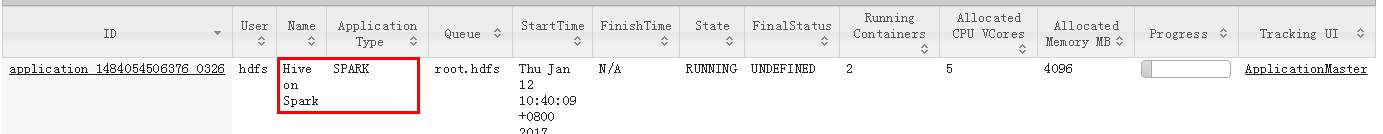
需要在Hive中设置任务的执行引擎，可以通过hive.execution.engine参数进行设置。目前Hive支持三种执行引擎：mr、tez、spark。因为我们需要使用Spark执行引擎，所以需要将hive.execution.engine设置为spark，具体如下：

**hive>**set hive.execution.engine=spark;（默认的执行引擎还是mapreduce）

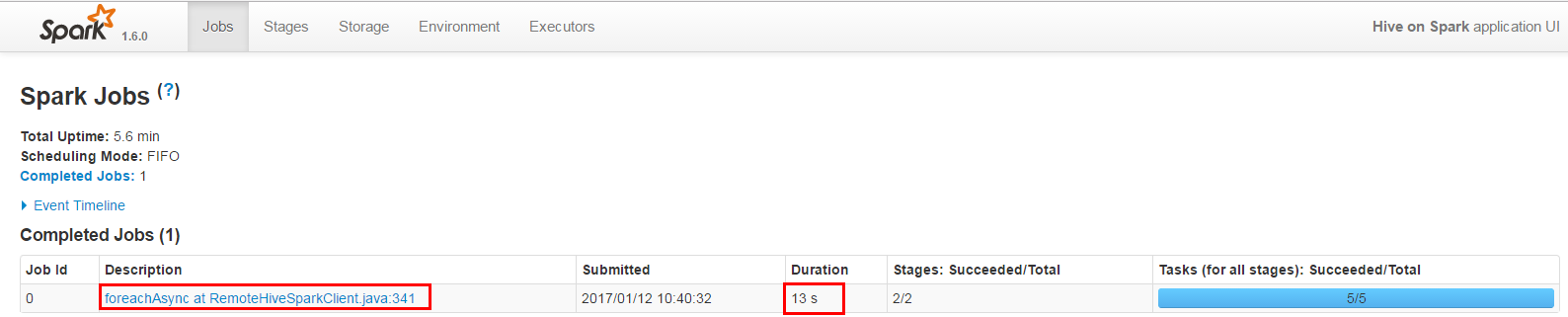


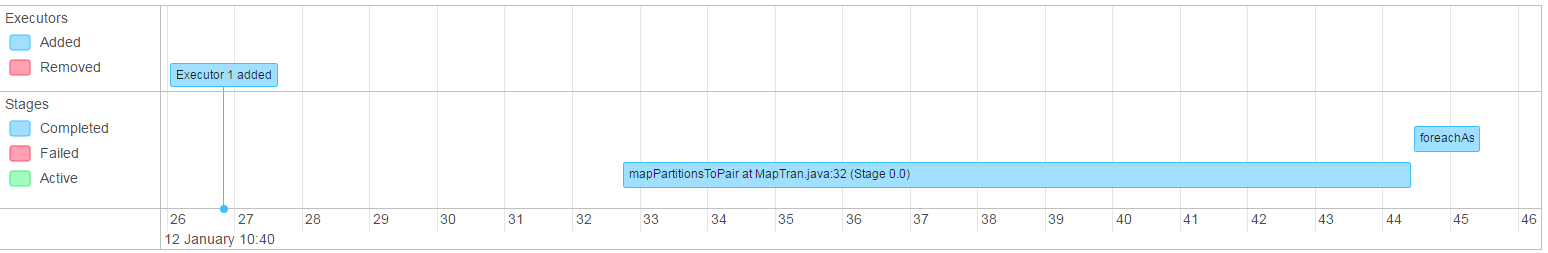
### 执行过程

再来看看YARN Web UI界面：



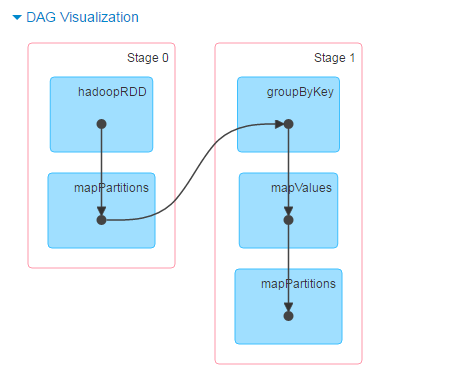
注意看里面的作业名称（Name）变成了Hive on Spark，而且应用类型（Application Type）变成了SPARK。点进去ApplicationMaster会跳转到spark的UI界面观察执行情况：





Hive DAG图：





### 性能对比

特别说明：目前测试环境hive很慢，测试的hive对结果比较会有影响。故采取beeline直接访问hiveserver来进行对比：

Select count(\*) from ods\_log\_pageview where dt = ‘2016-12-11’;

数据量大小：

|  |  |  |
| --- | --- | --- |
|  | mapreduce(s) | spark(s) |
| hive | 226.176 | 135.366 |
| beeline | 35.574 | 13.124 |

可以看出来spark的确比mapreduce执行引擎快很多！

# 五、Troubleshooting Hive on Spark

### 1. Problem: Delayed result from the first query after starting a new Hive on Spark session

The first query after starting a new Hive on Spark session might be delayed due to the start-up time for the Spark on YARN cluster. The query waits for YARN containers to initialize. Subsequent queries will be faster.

### 2. Problem: Out-of-memory error

This error indicates that the Spark driver does not have enough off-heap memory.Increase the off-heap memory by setting spark.yarn.driver.memoryOverhead or spark.driver.memory.

### 3. Problem: Spark applications stay alive forever and occupy cluster resources

Open a terminal and run:  
yarn application -kill <applicationID>  
applicationID is each YARN application ID