一：spark运行原理

Spark 是一个分布式（很多机器，每个机器负责一部部分数据），基于内存（内存不够可以放在磁盘中），特别适合于迭代计算的计算框架。

基于内存（在一些情况下也会基于磁盘），优先考虑放入内存中，有更好的数据本地性。如果内存中放不完的话，会考虑将数据 或者部分数据放入磁盘中。

擅长迭代式计算是spark 的真正精髓。基于磁盘的迭代计算比hadoop快 10x倍，基于内存的迭代计算比hadoop 快100x倍。

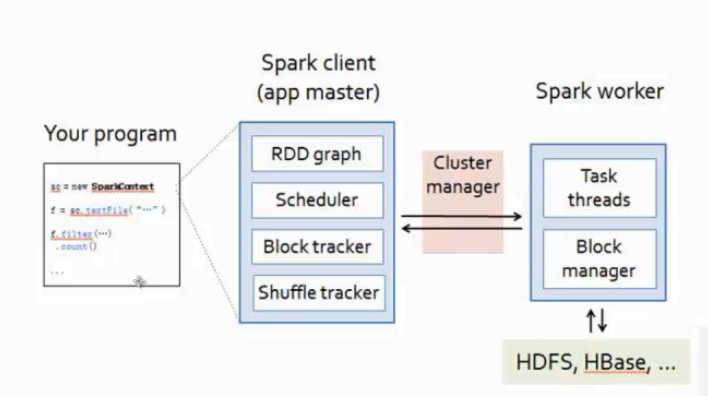
Driver 端， 就是写好的本地程序提交到特定的机器上。

Spark开发语言说明

国内开发程序spark程序 有些使用java开发，

1. 人才问题 java开发人员很多（scala开发人员较少）
2. 整合更加容易
3. 维护更加容易
4. 但是要更好的掌握spark还是需要用scala写spark，因为java写起来太繁琐了而且有些功能实现起来很困难。

Spark组件



处理数据来源： hdfs 、hbase、hive 、db。

Hivee 包括数据仓库和计算引擎，sparksql只能取代hive的计算引擎。

处理数据输出： hdfs 、hbase、hive 、db、s3（云）。还可以直接输出给客户端（dirver端）。

RDD 解密

1. Spark中一切基于RDD，RDD 是弹性分布式的数据集。例如1P的数据的处理，处理时是分布式的，分成了很多分片分布在几百或者上千台的机器上。存储时默认存储在内存中，如果内存中放不下 会放到磁盘上。

RDD 本身有一系列的数据分片，一个RDD逻辑上抽象的代表了底层的一个输入文件（或者文件夹），但是实际是按照分区（patition）分为多个分区，分区会放到spark 集群不同的机器的节点上。假设有1亿数据，每台机器放100万条，需要100万台机器。而且每台机器的100万条数据按照patition（特定规模的一个数据集合）来管理。

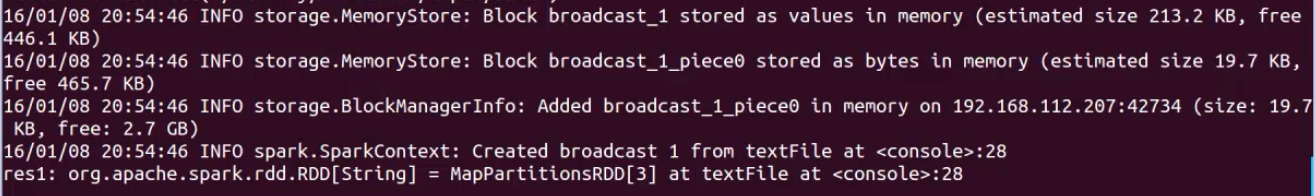
1. RDD 的弹性特点
2. 自动的进行内存和磁盘的存储的切换
3. 基于Lineage 的高效容错。例如作业链条油1000个步骤，在901个步骤出错，由于具有血统关系会在第900 个步骤重新计算，而不是从第一个步骤计算。提高了错误恢复速度。
4. Task如果失败会自动进行特定次数的重试。默认4次
5. Stage如果失败会自动进行特定次数的重试， task 底层尝试了好几次都失败，这时候整个阶段就会失败，整个阶段肯定有很多并行的数据分片，数据分片的计算逻辑是一样的，只是处理的数据不同。再次提交这个stage 的时候，除了再次自动提交体现弹性表现以外，更重要的是再次提交这个stage 的时候，如果说这个stage 中（假设有100万数据分片），有5个失败，再次提交stage 时会查看其它任务有没有输出，如果有就不在提交这些任务，只会提交那失败的5个任务，这是会非常的高效（只计算运算失败的分片）。 默认3次。

问题：spark的中间数据都在内存中，不在硬盘中，如何得到失败前一步的结果？

回答：数据优先考虑存在内存中，如果内存不够用，会存在磁盘中。不是每个步骤都做缓存。缓存的条件：任务特别耗时、任务计算链条很长、Shuffle之后，checkpoint之前。

RDD 实战

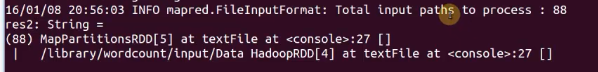
1. 从hdfs 上读取数据 val data = sc.textfile(“/library/wordcount/input/Data”)



返回的内容为RDD 类型，RDD泛型为String 因为sc.textFile读取的文本。读取的一行一行的数据，一行一行的数据是数据分片，RDD 是一系列的数据分片的。数据分片中每个分片中每一行是String类型的。

查看依赖关系

data.toDebugString



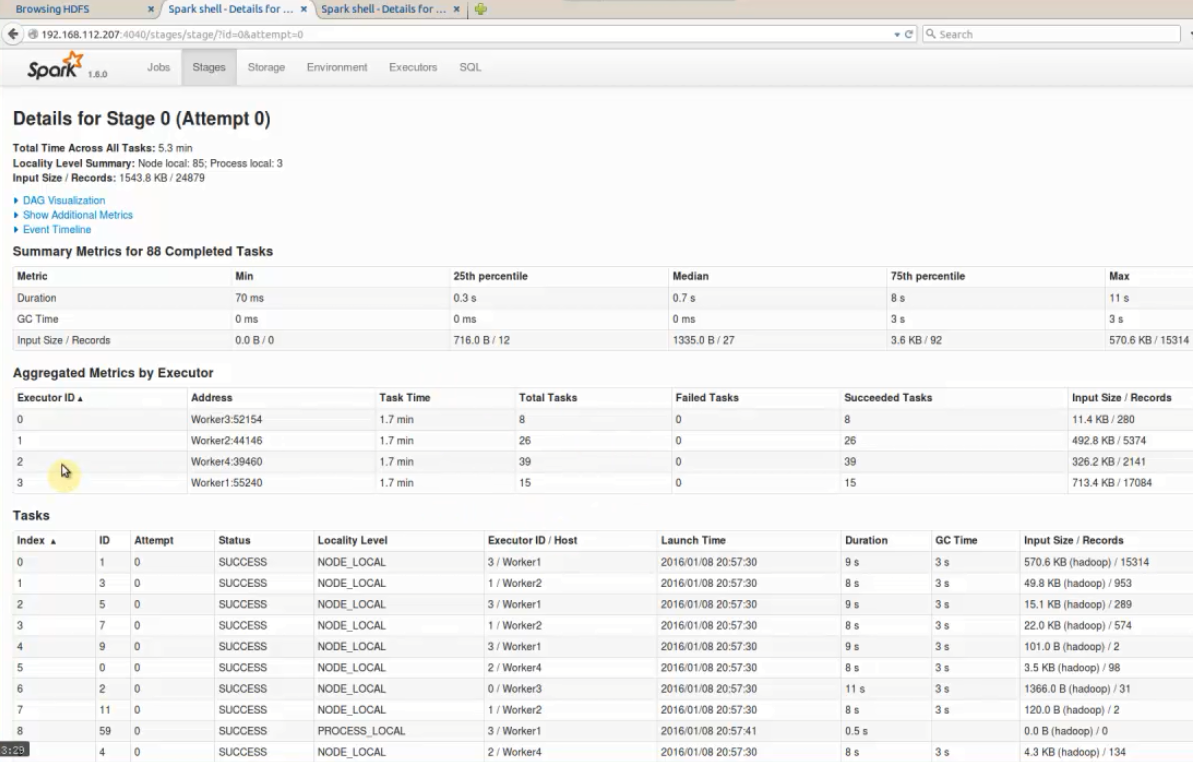
textFile 从 Hadoop 读取数据 所以产生了 HadoopRdd，然后进行map，为什么进行map呢？ 因为获取了每个的数据分片，我们只对每个数据分片的每一行内容感兴趣，不会对数据分片的没一行key（行的索引） 感兴趣。

所以进行map 操作，产生 mappartitionRDD 。 这次产生了2个RDD。有时候会产生1个或者更多RDD。

sc.textfile 从hdfs 分布式文件系统中，读取我们需要的具体的数据，而这个数据是一系列分片的方式 分布在不同的机器上。怎么证明分布在不同的机器上呢？

执行 data.count(action 级别的操作)

运行的结果会搜集给driver 端。



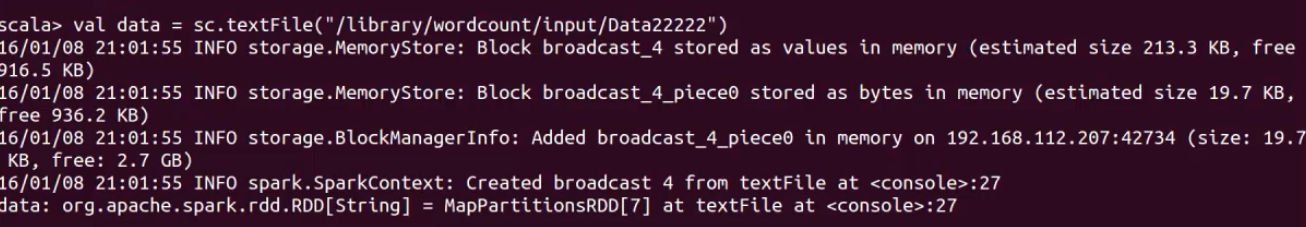
4个exector，（node\_local 代表数据在本地磁盘上，process\_local 数据在本地的内存中）

真正的计算分布在各个机器上，散落在集群上不同的机器上，数据要符合数据本地性，所以数据也分布在集群中不同的机器上。 这个证明了RDD 在逻辑上代表了 hdfs 上的文件，实际上是很多数据分片，这些数据分片散落在spark集群 中不同的节点上。计算的时候要符合数据本地性，所以就是数据不动，代码动。所以也是在做并行的计算的。

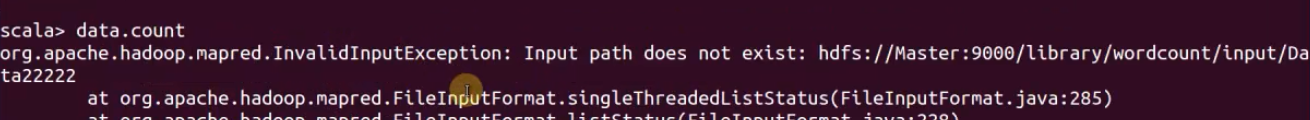
问题 ：hdfs 的分片和spark rDD 的分区的关系？

Spark 在读取数据的时候 一个partition 默认对应hdfs中第一个block，默认大小是128M。（ 由于数据可能会跨block 存储，所以一个partition 可能是128M+10个字节，下一个 partition 是128M-10个字节。）

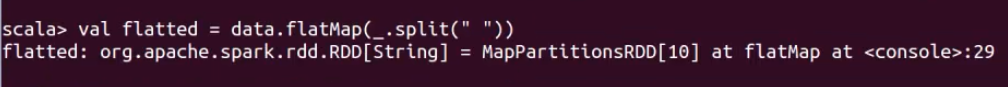
如果 hdfs 中路径不存在，在textfile 时时不会报错的因为是transformation级别的操作（是lazy操作）。如图



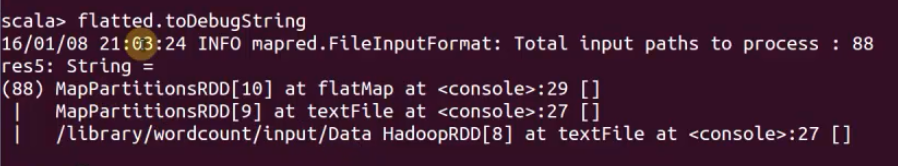
但是如果进行 data.count 这个时候要发生计算，是action级别的。找不到相应的hdfs目录就会报错。如下图



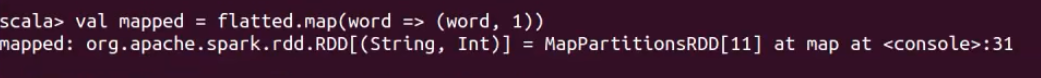
1. 进行单词切分 val flatted= data.flatMap(\_.split(“ ”)) 对每行数据 以空格进行切分，又会产生一个MapPartitionsRdd 执行结果如下



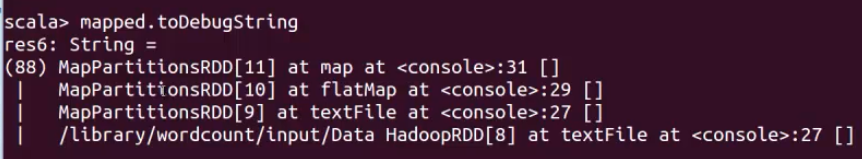
查看 依赖关系 faltted.toDebugString .自己有个MappatitionRDD 把之前依赖的RDD 也都列出来了。说明RDD 之间是由依赖关系的。



1. 对每个单词计数为1 val mapped= flatted.map(word =>(word,1)) 写匿名函数，执行后又产生一个RDD 里面的key 是字符串， value 是整数所以产生的是 RDD[(String,Int)]

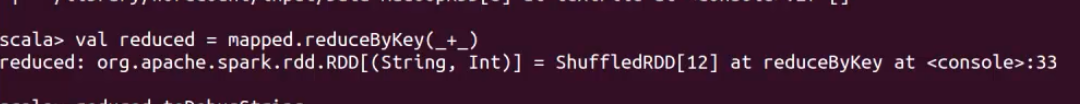


查看RDD 的依赖 mapped.toDebugString ,产生如下结果。map 产生MapPartitionsRDD 。MapPartitionsRDD 和HadoopRDD 是RDD 的子类。RDD 是抽象的，需要具体的子类来实现（实现数据存储在哪里，具体怎么计算）。map 产生MapPartitionsRDD 依赖于上一步flatM安排 产生的MapPartitionsRDD，而 flatMap产生的MapPartitionsRDD依赖于上一步textFlie 产生的MapPartitionsRDD，而 textFile 产的MapPartitionsRDD依赖于textFile产生的HadoopRDD，通过HadoopRDD 来读取具体的数据。

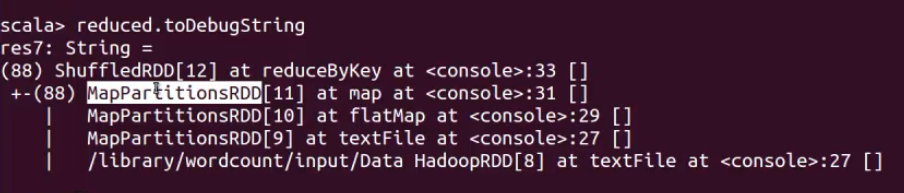


1. 进行reduce 操作，就是key 相同，value进行相加 。相当于图书馆有1000个书架，每个书架一个负责人对书的数量进行统计，然后这1000个人将自己统计的数据交给你来汇总。这个过程就叫做shuffle。 Shuffle 会产生ShuffledRDD 。就是每个书架的书交个一个节点去处理（这个处理过程是并行的）， 另外一个节点会从这1000个节点上抓取所有的书的数量的信息进行汇总。

val reduced=mapped.reduceByKey(\_+\_) 产生的结果如下图所示

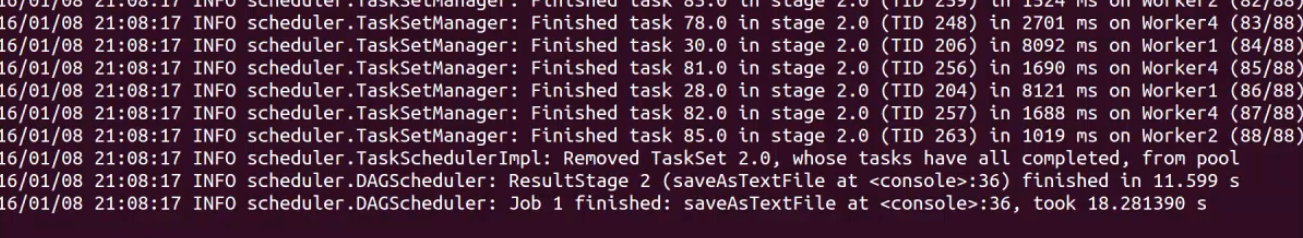


查看依赖关系 reduced.toDebugString 。reduceByKey 产生的ShuffledRDD 依赖上一步 map 产生的MapPartitionsRDD。执行结果如下图



1. 保存到hdfs 中。（前面所有的操作都没有触发操作,现在才开始真正的计算并将数据写入磁盘）。Spark在计算时每个步骤都是RDD ，RDD 的本质是提供的容错性，自动从失败的节点恢复。如果某个节点的RDD 丢失了，会根据RDD 的血统关系，重新生成。

Reduced.saveAsTextFile(“/library/wordcount/output/dt\_spark\_clicked4”) 。执行结果如下图



在浏览器上查看

