1. 大数据平台！
2. 数据分析：使用工具，流程。
3. 数据挖掘：使用工具，流程。
4. 数据整合，mysql数据导出。

数据分析REPL：

Sublime工具：

PySpark:

Impala:

SparkSQL:

MySql/Oracle/MongoDB

Sqoop导入、导出

训练算法模型，编写python代码：

1、Sublime编写、调试，保存代码

2、调教spark任务：sparkSubmit --master

3、训练数据

输入、模型输出

4、预测数据输入、结果输出

注意：数据放在HDFS上进行读取计算，与其他几种方法相比，速度是否加快？移动计算而非数据！

HDFS: 文件目录

Hive：分区、桶

HBase：？

ml VS mllib: 选择哪个？

1. CDH5.10.1支持spark 1.6.0, 部分算法有 ml api 的python的实现。（分类与回归）
2. CDH5.10.1无法升级到spark 2.0.1 ，无相关资料。
3. Mllib API 会继续使用，ml的api 功能没有Mllib 的完善。
4. 基于Spark 1.6.0 ，使用：有Ml的python实现的算法 和 MLlib的python实现的算法。

data = sc.textFile("/data/t1.txt",3).cache() #这里加上cache，m1的操作就是PROCESS\_LOCAL数据本地性，Stages界面可以看到，就不会有/tmp问题，不cache()就会有/tmp问题。

data.glom().collect()

m1 = data.groupBy(lambda x: x.split(" ")[0],2).cache()

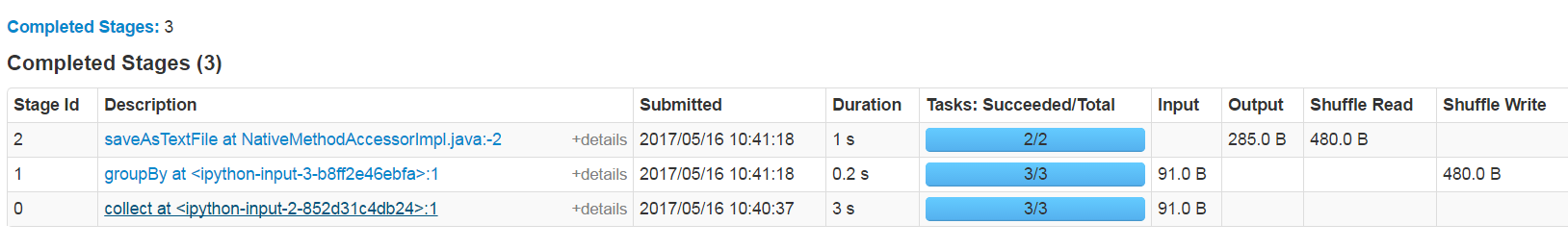
m1.saveAsTextFile("/data/sort004")

当前stage的task数是指调用了shuffle操作的那个RDD的分区。比如stage1的task数是data的分区数，而非groupBy()中的分区数，groupBy()是在具体是在下个stage指定分区执行的，m1的分区数是groupBy()中指定的分区数。

当前Stage的Description无非是Action和shuffle算子，shuffle算子有返回RDD，这个RDD在shuffle算子中可以指定分区，shuffle算子的操作具体在第二个Stage运行，首先看shuffle指定了几个分区，就启动几个task，去进行shuffle read，读取数据后进行，groupBy等等shuffle操作。

Action：

first(),take(),count(),takeSample(),takeOrdered(n, [ordering]),collect(),reduce(),countByKey(),foreach(func),saveAsTextFile(),saveAsNewAPIHadoopDataset().



适当的cache()会避免/tmp问题！

GC时，Executor、task会停止，因此调整spark.shuffle.io.maxRetries和spark.shuffle.io.retryWait是很有必要的！

解决/tmp问题：

1. 尽量避免groupBy()等操作，拆分操作。
2. 合理进行cache()
3. 设置Executors,在一个节点上。
4. 调整三个参数：（参数不起作用？）

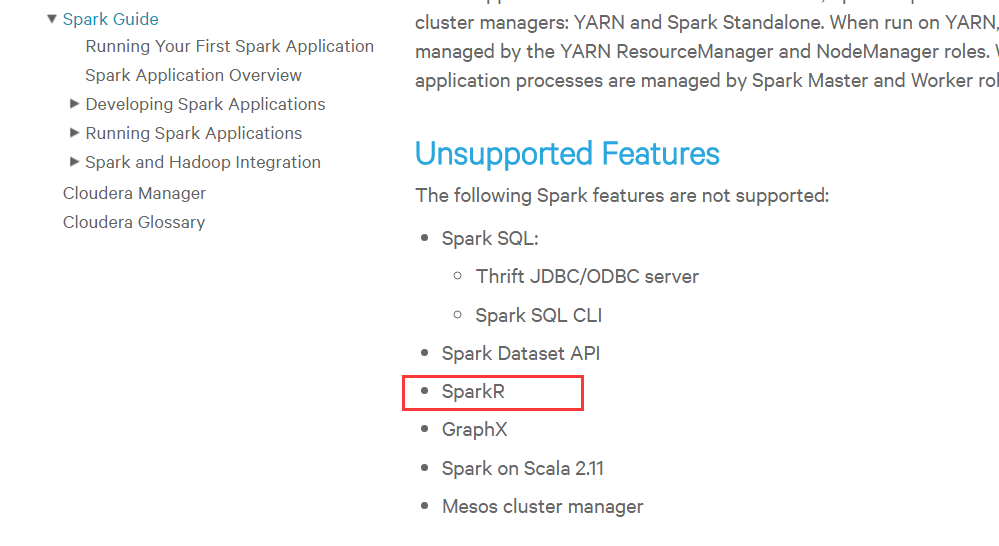
.set("spark.shuffle.file.buffer","32k")

.set("spark.shuffle.io.maxRetries","10")

.set("spark.shuffle.io.retryWait","4s")

GC会影响/tmp问题：<http://blog.csdn.net/duan_zhihua/article/details/51901207>

Cdh5不支持sparkR：https://www.cloudera.com/documentation/enterprise/latest/topics/spark.html



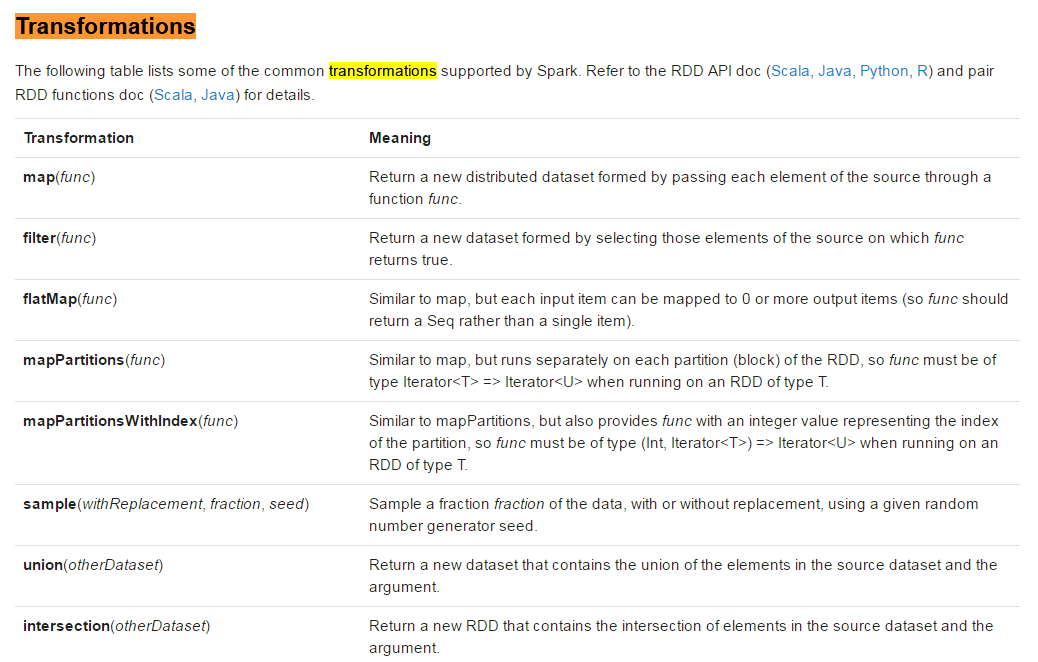
围绕这四点学习Spark：

1、Spark编程指南：<http://spark.apache.org/docs/1.6.0/programming-guide.html>

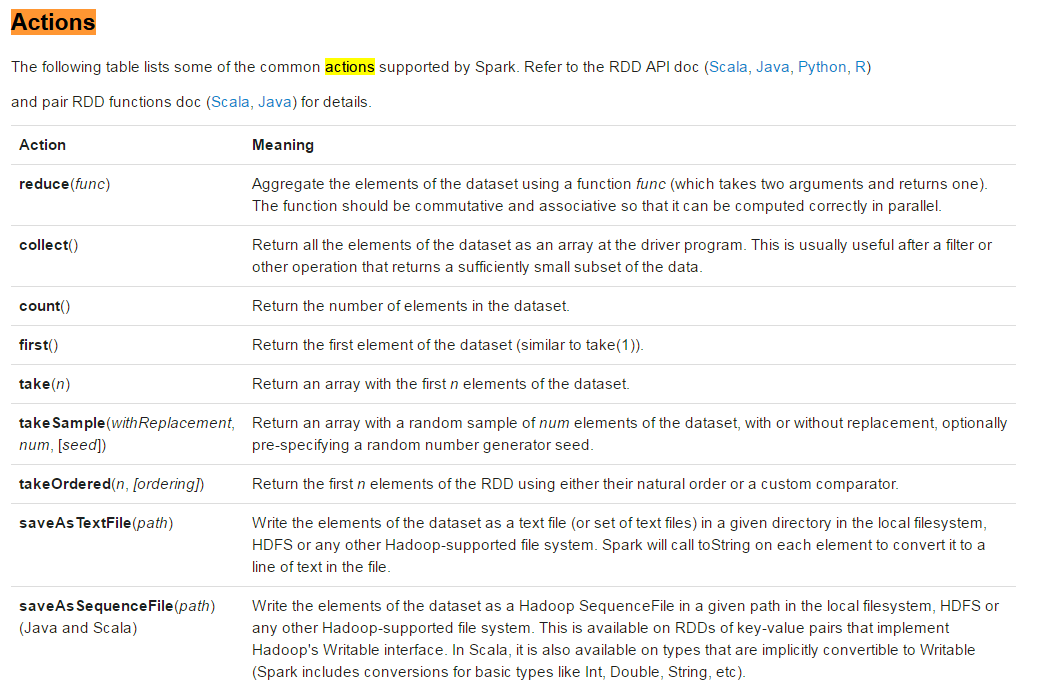
以上链接是Spark官网编程指南，里边有使用Python实现的小Demo。

Spark编程，核心就是RDD和它的算子，其中RDD是一个弹性数据集，算子有Transformations和Actions 两种。

1）Transformations算子：在页面中搜索：Transformations ，找到第6条，如下图：



2）Actions算子：页面搜索Actions，找到第3条，如下图：

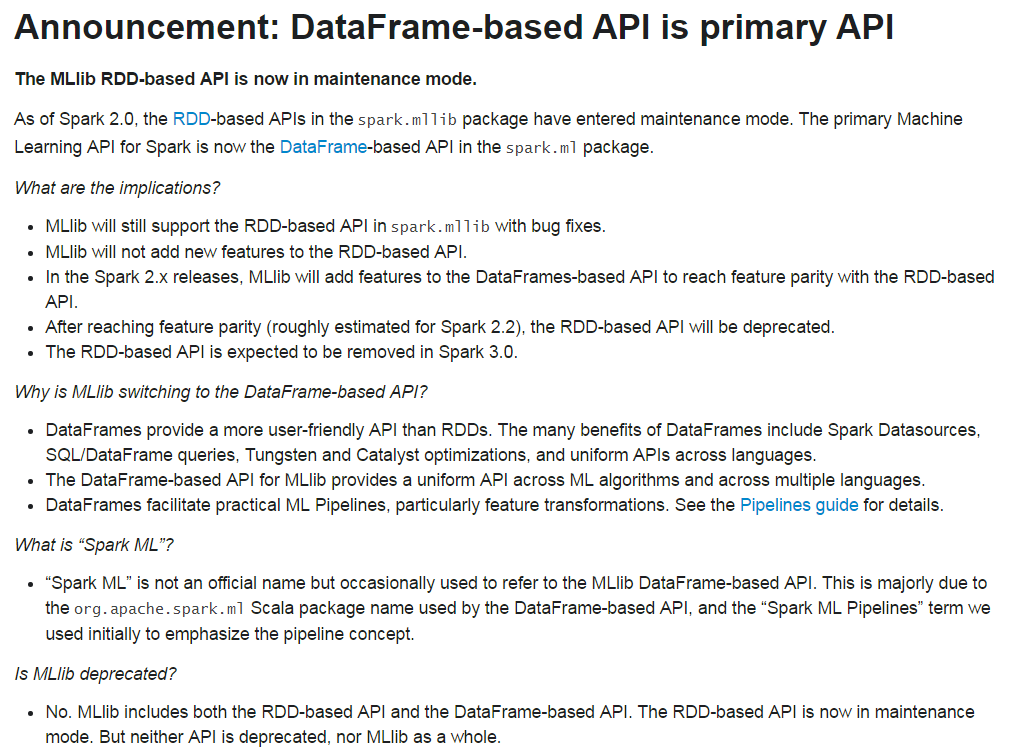


2、SparkSQL Python Demo : <http://spark.apache.org/docs/1.6.0/sql-programming-guide.html>

SparkSQL是Spark中的一个组件，SparkSQL可以读取多种数据来源，且读进的数据可以使用SQL进行分析，运行在Spark之上（系统将SQL语句转化成了Spark任务）。

1. 机器学习有两个包：the RDD-based APIs in the spark.mllib 和 the DataFrame-based API in the spark.ml package.

两者区别可见Spark在2.0中的公告：<http://spark.apache.org/docs/latest/ml-guide.html>



Spark机器学习Python Demo: <http://spark.apache.org/docs/1.6.0/mllib-guide.html>

4、Spark Python API: <http://spark.apache.org/docs/1.6.0/api/python/index.html>

1. ：每个Application获取专属的executor进程，该进程在Application期间一直驻留，并以多线程方式运行Task。这种Application隔离机制是有优势的，无论是从调度角度看（每个Driver调度他自己的任务），还是从运行角度看（来自不同Application的Task运行在不同JVM中），当然这样意味着Spark Application不能跨应用程序共享数据，除非将数据写入外部存储系统,每个任务都会占有自己的EXECUTOR，每个Executor只会为一个Application程序服务，程序结束，Executor退出。

Spark为何划分DAG？

保证执行顺序！每个stage有相同逻辑的task。

在Spark作业调度系统中，调度的前提是判断多个作业任务的依赖关系，这些作业任务之间可能存在因果的依赖关系，也就是说有些任务必须先获得执行，然后相关的依赖任务才能执行，但是任务之间显然不应出现任何直接或间接的循环依赖关系，所以本质上这种关系适合用DAG表示。

DAGscheduler简单来说就是负责任务的逻辑调度，负责将作业拆分成不同阶段的具有依赖关系的多批任务。  
DAGscheduler最重要的任务之一就是计算作业和任务的依赖关系，制定调度逻辑。

spark中rdd经过若干次transform操作，由于transform操作是lazy的，因此，当rdd进行action操作时，rdd间的转换关系也会被提交上去，得到rdd内部的依赖关系，进而根据依赖，划分出不同的stage。DAG是有向无环图，一般用来描述任务之间的先后关系，spark中的DAG就是rdd内部的转换关系，这些转换关系会被转换成依赖关系，进而被划分成不同阶段，从而描绘出任务的先后顺序。

根据不同的transformation操作，RDD的依赖可以分为窄依赖（Narrow Dependency）和宽依赖（Wide Dependency，在代码中为ShuffleDependency）两种类型。窄依赖指的是生成的RDD中每个partition只依赖于父RDD(s) 固定的partition。宽依赖指的是生成的RDD的每一个partition都依赖于父 RDD(s) 所有partition。窄依赖典型的操作有map, filter, union等，宽依赖典型的操作有groupByKey, sortByKey等。

DAGScheduler将Stage划分完成后，提交实际上是通过把Stage转换为TaskSet，然后通过TaskScheduler将计算任务最终提交到集群。其所在的位置如下图所示。

区分这两种依赖很有用。首先，窄依赖允许在一个集群节点上以流水线的方式（pipeline）计算所有父分区。例如，逐个元素地执行map、然后filter操作；而宽依赖则需要首先计算好所有父分区数据，然后在节点之间进行Shuffle，这与MapReduce类似。第二，窄依赖能够更有效地进行失效节点的恢复，即只需重新计算丢失RDD分区的父分区，而且不同节点之间可以并行计算；而对于一个宽依赖关系的Lineage图，单个节点失效可能导致这个RDD的所有祖先丢失部分分区，因而需要整体重新计算。

Spark 读取数据：HDFS、集合、Hbase、Hive

本地文件需要确保所有机器相同路径有相同的文件！

If using a path on the local filesystem, the file must also be accessible at the same path on worker nodes. Either copy the file to all workers or use a network-mounted shared file system.

The textFile method also takes an optional second argument for controlling the number of partitions of the file. By default, Spark creates one partition for each block of the file (blocks being 64MB by default in HDFS), but you can also ask for a higher number of partitions by passing a larger value. Note that you cannot have fewer partitions than blocks.

spark task 位置

Spark 数据本地性

<https://www.zhihu.com/question/36996853?sort=created>

<https://www.zhihu.com/question/58292385>

<http://blog.csdn.net/lxhandlbb/article/details/52987909>

<http://blog.csdn.net/lxhandlbb/article/details/52987928>

<https://baijiahao.baidu.com/po/feed/share?wfr=spider&for=pc&context=%7B%22sourceFrom%22%3A%22bjh%22%2C%22nid%22%3A%22news_3462036130046936536%22%7D>

数据本地化理解：<https://baijiahao.baidu.com/po/feed/share?wfr=spider&for=pc&context=%7B%22sourceFrom%22%3A%22bjh%22%2C%22nid%22%3A%22news_3462036130046936536%22%7D>

Spark从本地读取文件、HDFS、HBase、数据库等，对应分区，task,分区调整，shuffle过程数据。

常用action算子，对比监控页面。

广播变量、累加器使用！

Shuffle中间结果：

数据本地化，NO\_PREF：对于task来说，数据从哪里获取都一样，没有好坏之分,比如从数据库中获取数据。

Standalone client模式：

Sc.parallelize()

1）方法中设置分区，按方法的来。

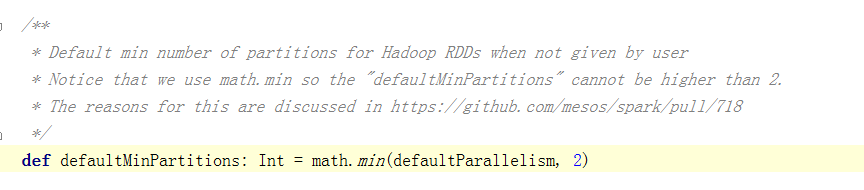
2）方法中没有，按照spark-defaults.conf的：spark.default.parallelism，不管是什么，也不用跟谁比较。

3） 1和2都没有：就是2.

sc.textFile(file/HDFS)（读取本地文件时，block大小大约30M，其他参数规则跟读取HDFS一样，以下是读取hdfs时的参数设置）

1. HDFS，方法中不设置，minPartitions取spark-defaults.conf的：spark.default.parallelism=3 和 2的较小值，然后max(min(2,parallelism),Blocks),小于128M，Blocks为1 。
2. HDFS，方法中不设置，spark.default.parallelism=3 也不设置，minPartitions取max(2,Blocks)。
3. HDFS，读取的文件为54Byte，在方法中设置，设置为1,2,3就是1,2,3.但minPartitions只要大于3时，实际的分区=设置的分区+1 ，不知什么原因（可能因为数据太小了）。这时，spark.default.parallelism=3设置了也不起作用，优先级最高的是程序的方法中设置的，可能因为数据太小。
4. HDFS，读取的文件为8.39M 和 419.51KB，听从方法中设置的分区数，方法中没设置按照spark.default.parallelism 和 2的较小值，再取max(min(2,parallelism),Blocks)，spark.default.parallelism 没设置，就按max(2,Blocks)。

方法中设置分区数小于Blocks数时，不起作用，必须遵循：分区数>= Blocks数



对于\*ByKey等需要shuffle而生成的RDD，其Partition数量依如下顺序确定：1. 方法的第二个参数 > 2. spark.default.parallelism参数 > 3. 所有依赖的RDD中，Partition最多的RDD的Partition的数量。

对于其他的RDD则其依赖于父RDD的分区个数。对于读取Hadoop文件的RDD，其默认的分区个数是HDFS块个数。同时textFile函数也接受参数，指定分区个数，但不能小于HDFS块个数——不使用HDFS块个数。

理解：分区个数不小于Block个数：也是分区个数>= Block个数，一个分区对应一个task，一个task最多处理一个块的数据大小。

Sc.mysql

分区的设置以及数据本地化。

Spark数据倾斜：<http://blog.csdn.net/lw_ghy/article/details/51419877>

使用Demo进行实验说明：

coalesce repartition的区别：

<http://blog.csdn.net/dax1n/article/details/53431373>

<http://www.cnblogs.com/fillPv/p/5392186.html>

<http://blog.csdn.net/wwyxg/article/details/51035324>

coalesce(numPartitions: Int, shuffle: Boolean = false)

repartition(numPartitions: Int)

Repartition可增大减小分区，是调用了coalesce，shuffle=true，增大分区固然会有宽依赖（父RDD的一个分区会对应子RDD的多个分区），自然会有shuffle过程。但减小分区可以不使用Repartition，使用coalesce，shuffle=false，这样减小分区是窄依赖，避免shuffle。想使用coalesce增大分区必须shuffle=true才有效。

repartition()方法就是coalesce()方法shuffle为true的情况。

窄依赖：是指父Rdd的分区最多只能被一个子Rdd的分区所引用，即一个父Rdd的分区对应一个子Rdd的分区，或者多个父Rdd的分区对应一个子Rdd的分区。

宽依赖：就是宽依赖是指子RDD的分区依赖于父RDD的多个分区或所有分区，即存在一个父RDD的一个分区对应一个子RDD的多个分区。1个父RDD分区对应多个子RDD分区，这其中又分两种情况：1个父RDD对应所有子RDD分区（未经协同划分的Join）或者1个父RDD对应非全部的多个RDD分区（如groupByKey）。



Secondary NameNode合并一些NameNode的日志文件，帮助HDFS快速启动！

只有这样a = sc.parallelize(data,3).cache()将a.cache才可以正常执行。

广播变量是发送到Worker还是Executor，一个应用不同Job，如果不cache，不能共享数据?

提交两个程序，验证广播变量。

设置spark.memory.fraction等参数。

Wordcount1：三种数据来源：sc.parall(),sc.textFile(本地、HDFS),sparkSQL(读取多种数据源)

1、编写wordCount程序：

2、使用sparkSQL做简单分析

Ml：使用SparkSQL读取，用ml包进行计算。

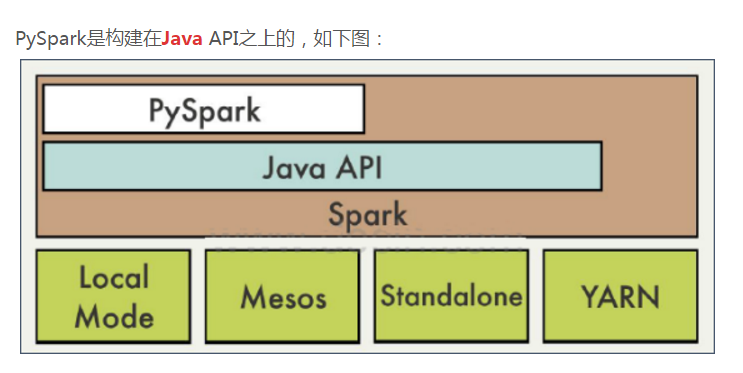
pyspark补全命令。

所有算子的执行过程：对应一个个实例或场景。

熟悉每个算子的实例或场景方便开发。

Scala/python都是讲代码翻译为JVM字节码。Python主要是依赖了py4J这个库,即 python for JVM，py4j让python可以使用JVM的RDD等对象。

Spark使用py4j来实现python与java的互操作，从而实现使用python编写Spark程序。Spark也同样提供了pyspark，一个Spark的python shell，可以以交互式的方式使用Python编写Spark程序。



Lambda方法传递实例！分许多种情况。

pySpark是一个应用程序，在其中执行一个action就会产生一个job。

Cache一个rdd，会增快速度。

Pyspark看rdd类型，日志级别。

读取本地文件时，理论上是：确保每个Worker上都有该文件，且目录一致。

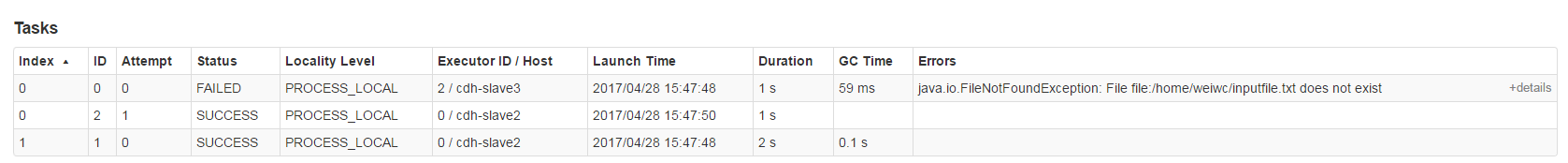
动态资源分配开启或关闭时，资源的分配规则一样，都是先试图使用所有的Core。在当前节点的spark-defaults.conf设置true或false时，即可生效。也可在程序中设置。

1、 开启动态资源分配时，指定参数为--executor-cores 4，pyspark读取本地文件，有三个分区，刚启动pyspark时，会启动3个Executor，并且在三个不同节点，每个Executor分配到一个分区，启动一个task，执行代码：

rdd2 = sc.textFile("file:///home/weiwc/inputfile.txt").cache()

rdd2.count()

当且仅当：执行action即count()时会报错，执行textFile时还不会报错，虽然报错但也有结果，因为把其他Executor的task移到了slave2上。



上图的Index、 ID、 Launch Time三项可知，先启动Index0/ID=0和Index1/ID=1,接着Index0/ID=0失败，因此又在cdh-slave2上重新启动了Index0/ID=2。因此虽然报错，结果还是返回来的。**上图为什么失败后又在cdh-slave2上重新启动了Index0/ID=2 ?** 难道跟**数据本地性**有关，上图可看到

Locality Level级别为：PROCESS\_LOCAL,代表进程本地化！

因为inputfile.txt文件只在slave2上有，其他两个节点本地目录没有，当其他两个节点的分区读取inputfile.txt时会找不到文件，但由于开启了资源动态分配，过一会儿，其他两个Executor都被Kill掉了，只剩下slave2上的Executor，并且该Executor会有三个task，因此这时再执行rdd2.count()时就不会报错了，三个分区都在slave2的Executor中读取本地文件。

1. 当关闭动态资源分配时，设置spark.dynamicAllocation.enabled=false，rdd2一直会保持最开始的Executor个数，即一直会占用资源，因此读取本地文件时，需要保证inputfile.txt在每个Worker节点都要有。

开启动态分配时，指定参数启动Pyspark，就算当时分配的资源Core为0，也可进入pyspark，可以进行Transformations操作，但不能进行Collect时，会卡在那里，没有足够的资源，当其他资源释放，当前程序获取到资源时，就会执行。

当把其他Executor进程Kill掉后，会把Kill掉的Executor中的task放到最后剩下的那个Executor中，是否存在最后一个Core，就一个核，但task有多个，串行执行。

资源回收，如果一个Executor空闲时间超过60S，将会被Kill掉：spark.dynamicAllocation.executorIdleTimeout=60

在pyspark中使用hiveSql能用下面语法吗？

create table salaries2 as select gender,salary from salaries limit 2

数据的来源以及去向，常用的数据处理思路？

代码没有提示？能否使用eclipse+python去开发程序。

课件做完，提出一些问题，需要讨论的？提出相关问题？数据存储、应用场景？

Spark官方支持的算法，python实现，数据输入类型，参数，结果输出，预测类的算法需要进行实时处理，结合SparkStreaming？

否则这些算法如何进行应用？

Avro是一个数据序列化系统，设计用于支持大批量数据交换的应用。它的主要特点有：支持二进制序列化方式，可以便捷，快速地处理大量数据；动态语言友好，Avro提供的机制使动态语言可以方便地处理Avro数据。

Source: dir/exec/memory

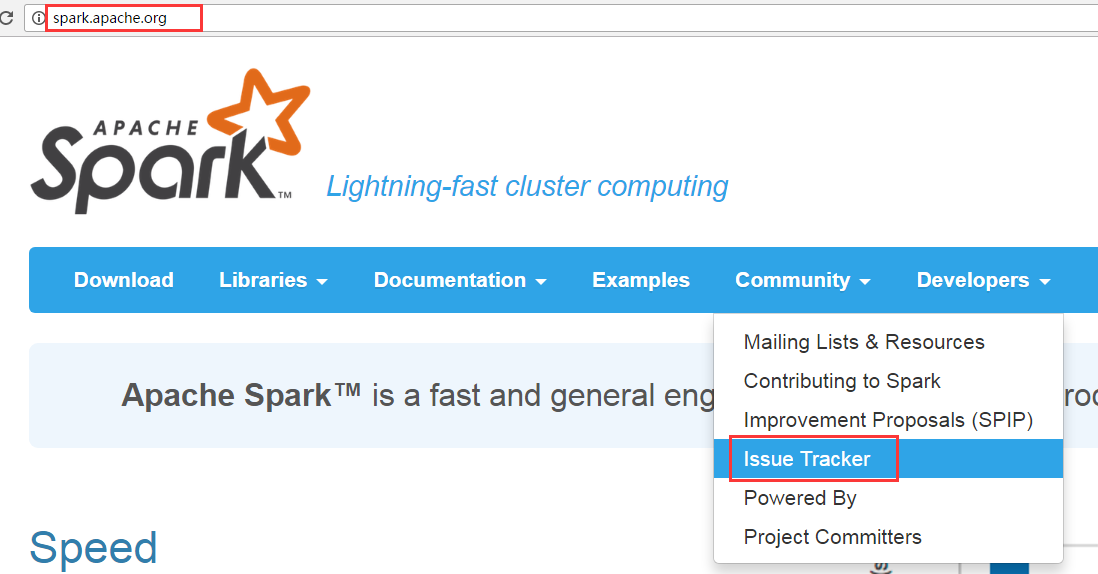
Channel：file/memory

Sink:hive/hbase/hdfs

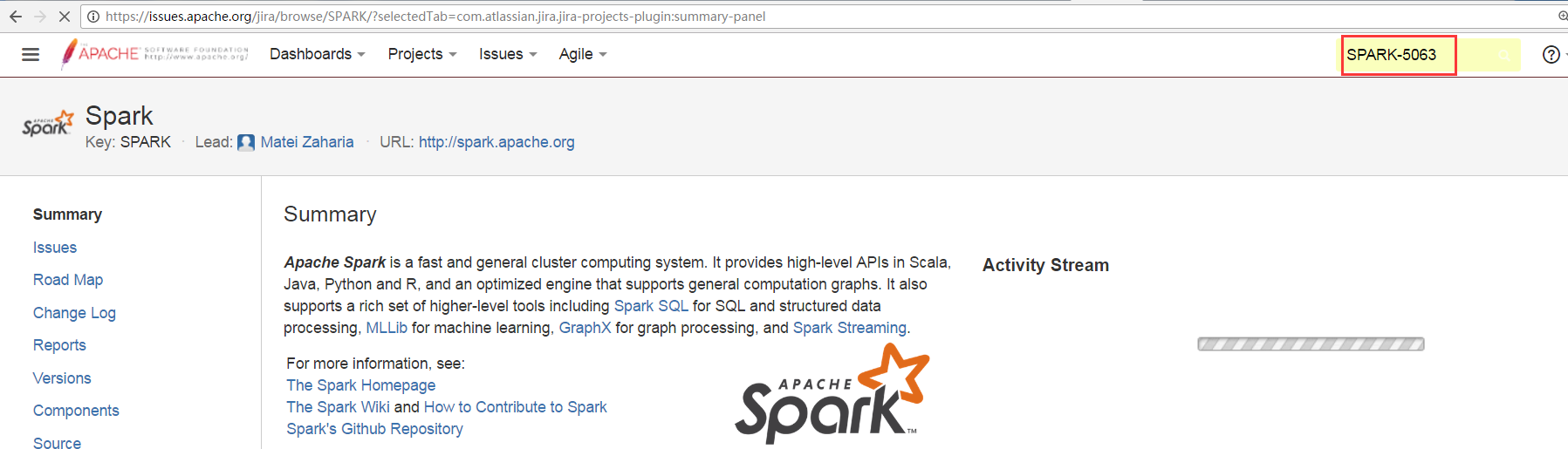
广播：

比如说：SPARK-5063 版本解决了一些问题，可通过官网去查看。

1、进入官网：Issue Tracker



2、在页面搜索：SPARK-5063



3、跳转到具体页面

