Spark

#### Spark 概述

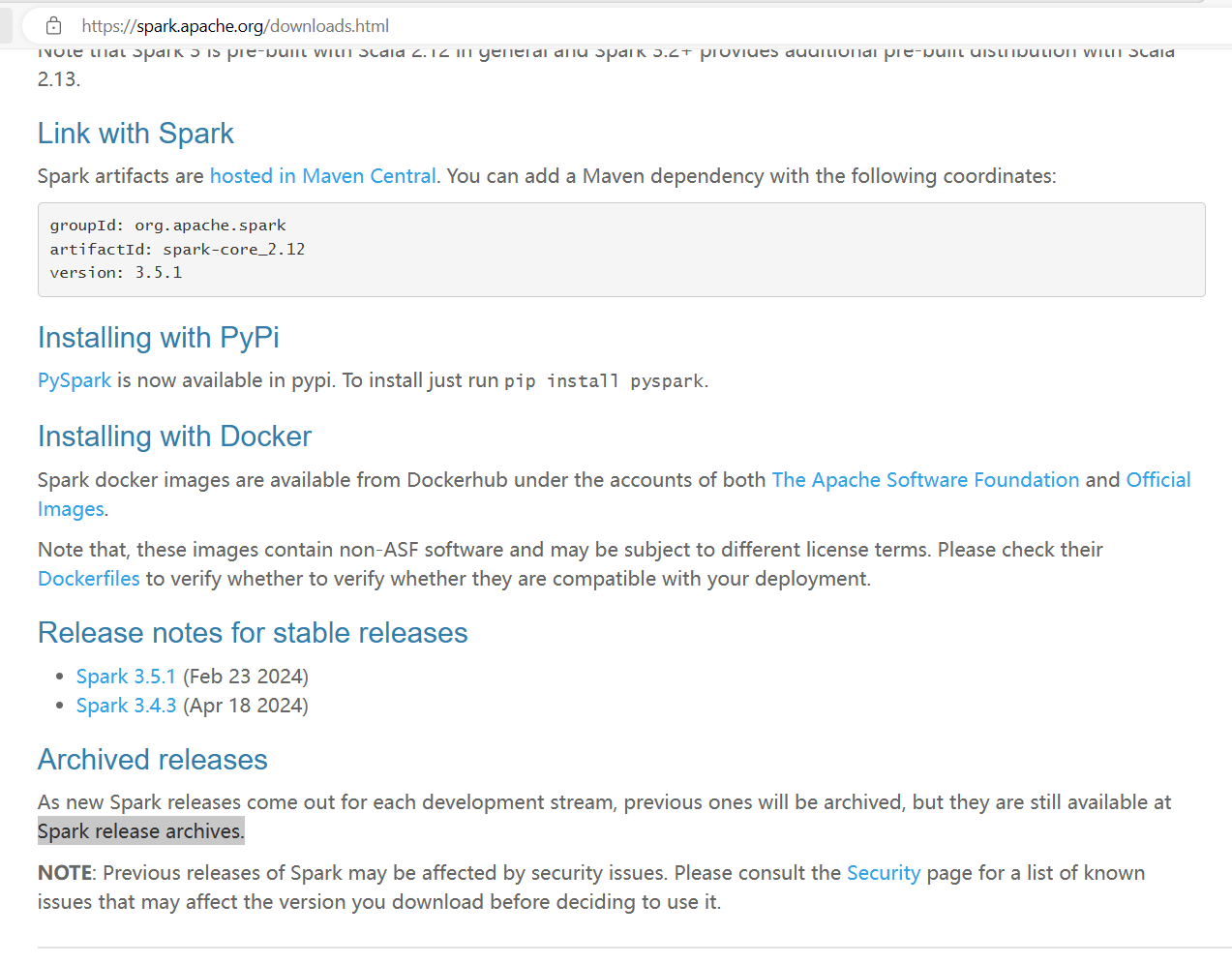
Spark 是专为大规模数据处理而设计的快速通用计算引擎，它是借鉴hadoop map reduce 优点技术上又一个大数据分析工具，

Spark 是scala写的，运行在jvm上，所以运行环境是java7+

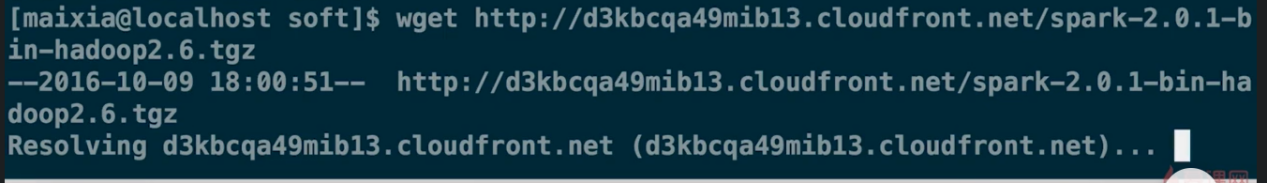
Spark 1.6.2 -Scala 2.10 Spark 2.0.0 -Scala 2.11

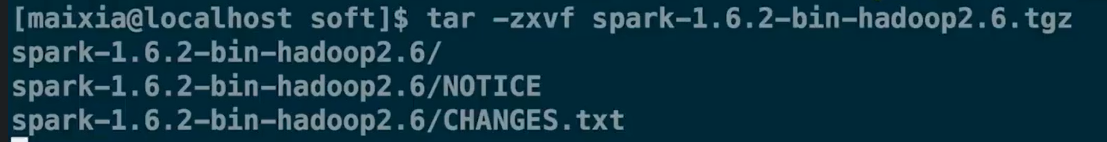
<https://spark.apache.org>

搭建spark ，不需要hadoop,如果有则，可下载相应的版本。



Linux 版本下载spark





本质上是基于MapReduce算法模型实现的分布式计算框架。

spark不仅拥有了Hadoop MapReduce的优点，将job中间输出结果保存在内存中，从而不再需要读写Hdfs,能够更好的适用于数据挖掘和机器学习。

Spark 4 个功能模块：

Spark sql

批处理流式（实时）计算框架（streaming）

分布式图处理框架(GraphX)

机器学习库 MLIib

技术框架和工作流

主要是将job中间输出结果保存在内存中，从而不再需要读写hdfs ,能够更好适用于数据挖掘与机器学习。

Spark Sql : 可以通过spark sql直接查询hive 或impala中的数据

Spark streaming:是个批处理的流式（实时）计算框架。其基本原理是把输入数据以某一个时间间隔批量的处理，当批量处理间隔缩短到秒级时，便可以用于处理实时数据流。

Spark Graph X 分布式图处理框架

Spark MLIib 是一个可扩展Spark机器学习库，在Spark中可以使用许多常见的机器学习算法，简化大规模机器学习时间。

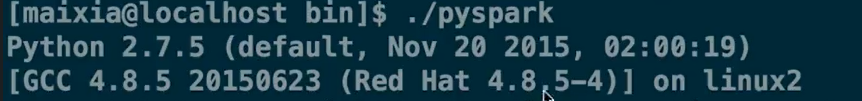
Spark MLIib 算法包括：分类与回归、聚类和协同过滤、决策树、朴素贝叶斯

Spark 的shell

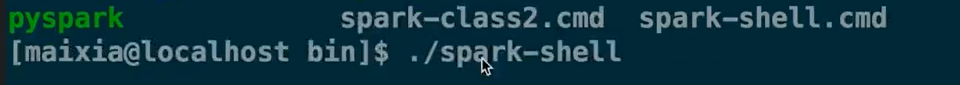
Spark 的shell 使你能够处理分布在集群上的数据，spark把数据加载到节点的内存中，因此分布式处理可在秒级完成。快速迭代式计算，实时查询，分析一般能够在shell中完成。

**Spark 总共提供了 Python shells 和 Scala shells**

Pyspark: /bin/pyspark

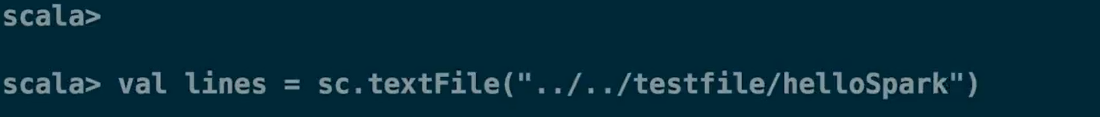


Scala shell : /bin/spark-shell

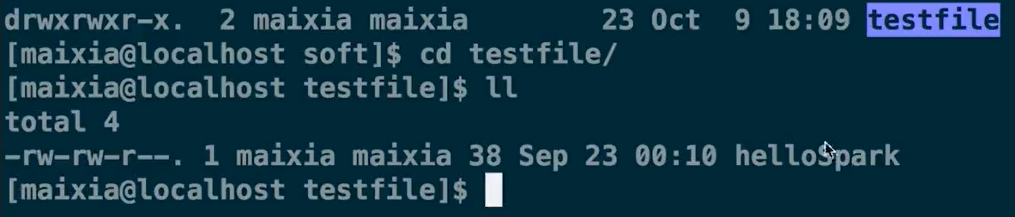


用scala 语句定义一个文本文件。通过sc.textFile去加载一个文本文件

Scala >val lines=sc.textFile(“../../helloSpark.txt”)



首先本地需要建立一个helloSpark文件



Cat helloSpark 查看文件 ---总共3行



Lines.count()

Lines.first()

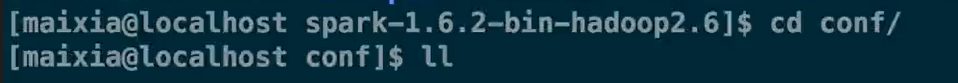
修改日志级别：

Log4j.rootCategory=WARN,console ---将info级别日志修改为warn级别

日志级别定义文件在 spark/conf/log4j.properties/

vi log4j.properties

log4j.rootCategory=INFO,console



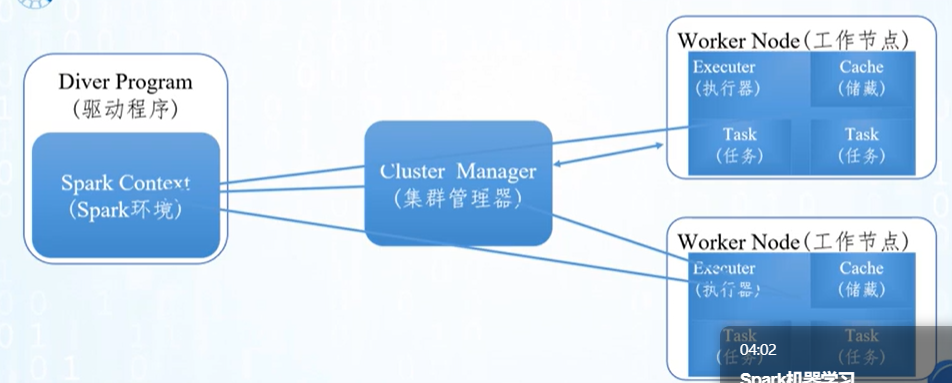
##### Spark 工作流

集群资源管理器cluster manager

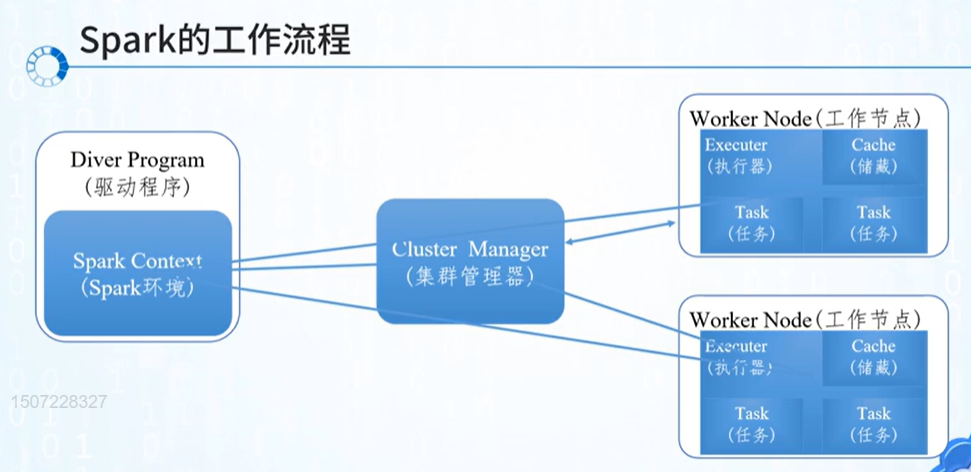
运行作业任务节点 worker node

任务控制节点 driver

进程执行 executor



集群资源管理器 cluster Manager ，是Spark 自带的资源管理器，可以是Yarn或 Mesos 等资源管理框架。



运行作业任务工作节点 worker node

每个应用的任务控制节点 driver

每个工作节点上负责具体任务的执行进程 executor

一个应用application 由一个任务控制节点driver和若干个job 构成，一个作业由多个阶段stage 构成，一个阶段由多个任务task组成。

当执行具体应用时，任务控制节点会向集群管理器 cluster Manager 申请资源，启动executor

并向 executor 发送应用程度代码和文件。

然后在executor上执行任务，运行结束后，执行结果会返回给任务控制节点，或者写到hdfs其他数据库中。

##### Spark 计算引擎

Spark 需要运行在集群管理器（Cluster Manager）之上，常见的有 Hadoop YARN、Apache Mesos 或 Spark 自带的 Standalone 集群管理器

供高级 API（Scala, Java, Python, R, SQL），简化开发.

**执行流程：**

* + **Driver Program：** 用户编写 Spark 应用程序的入口点，负责将应用程序代码分发给 Executor，并跟踪集群的状态。
  + **SparkContext：** Driver Program 中的核心对象，负责与集群管理器（如 YARN, Standalone）通信，获取资源，创建 RDD，调度作业（Job）。
  + **Cluster Manager：** 负责管理集群资源，分配 Executor 给应用程序。
  + **Executor：** 运行在 Worker 节点上，负责执行实际的任务（Task），处理数据，并将结果返回给 Driver 或写入存储。每个 Executor 包含多个核心（Core）和一定的内存（Memory）。
  + **Job：** 每个 Action 操作会触发一个 Job。一个 Job 包含了为计算该 Action 所需的所有 Transformation 操作。
  + **Stage：** Spark 会根据 RDD 的血缘关系和 Shuffle 操作将每个 Job 划分为多个 Stage。Stage 是任务调度和执行的基本单位。通常，发生 Shuffle 的地方会划分 Stage 的边界。一个 Stage 包含一组不涉及 Shuffle 的 Task。
  + **Task：** Stage 被进一步分解为 Task。每个 Task 负责处理 RDD 中的一个分区（Partition）的数据。有多少个分区，通常就有多少个 Task 在一个 Stage 中并行执行（假设 Executor 核心数足够）。

##### Spark 和hadoop的关系

Spark 的RDD之间的血缘关系 lineage .一旦某个rdd失败了，则可以由父rdd重建

虽然lineage可用于错误后rdd的恢复，但对于很长的lineage来说，恢复过程非常耗时。

如果启用了检查点，那么在state中的task 都执行成功后，spark contex将把rdd 计算的结果保存到检查点。

目前spark 支持java 、scala 、python 等多种语言编写应用程序，开发者可以根据应用需求选择不同开发应用程序，大大降低使用门槛

Spark还自带80 多个更等级操作符，运行scala python R 的shell 中进行交互式查询。

Spark 使用scala 开发，并借助于scala 类库中的Iloop实现交互式shell ,提供了对RePL （read eval print loop ）的实现。

Spark 支持sql 及hive sql 方便了传统sql 开发和数据仓库的使用者。

Spark 支持实时流计算，依赖Spark Streaming 对数据进行实时的处理，其流式处理能力还要强于storm. 而MapReduce只能处理离线数据，

高可用性，自身实现了standalone部署模式，此模式下的Master可以有多个，解决了单点故障问题。

Spark 提供了Hadoop Storage API ,支持Hadoop的hdfs系统，并且支持hadoop Yarn ，可共享存储资源，与hive几乎完全兼容

Spark 也完全支持使用外部部署模式，比如Yarn Mesos EC2 等。

Spark 支持文件格式、csv文件格式 json文件格式 orc文件格式 parquet文件格式 Libsvm文件格式，有利于spark 与其他数据处理平台的对接。

Spark 提供了Hadoop Storage Api ,支持hadoop的hdfs系统，并且支持hadoop Yarn ,可共享存储资源，与Hive 几乎完全兼容。

Spark 与hadoop的关系

Spark 框架与hadoop mapredce计算框架相比，有2个优点

利用多线程来执行具体任务，减少任务启动开销；而hadoop Mapreduce 采用的是进程模型，磁盘I/O开销大，延迟性高。

Executor中有一个BlockManager存储模块，会将内存和磁盘共同作为存储设备，当需要多轮迭代计算时，可以将中间结果存储到这个存储模型里，下次需要时，可直接读取该存储模块里面的数据，而不需要读写hdfs等文件系统里。有效减少了I/O开销。在交互查询场景下，预先将表缓存到该存储系统上，从而提高读写i/o 性能。

Spark 将应用程序上传的资源文件缓存Driver 本地文件服务内存中，当Executor执行任务时直接从Driver的内存中读取，从而大量减少了磁盘i/o

而hadoop具体任务需要利用NodeManager从Hdfs 的不同节点下载任务所需的资源(jar 包) 增加了磁盘i/o .

Spark 把不同的环节抽象为Stage,允许多个stage既可以串行执行，又可以并行执行。Stage中某个分区的task执行失败后，spark会重新对此stage调度，但在重新调度的时候会过滤已经执行成功的分区任务，所以不会造成重复计算和资源浪费。

Hadoop 将中间结果写到磁盘，再从磁盘读取中间结果，不同的环节操作，Hadoop通过简单的串行执行。

Spark 可以根据不同场景选择在map端排序还是reduce端排序

而hadoop mapreduce将中间结果按key 的hash 值和key 值大小进行2层排序。

Spark 将内存分为堆上的存储内存，堆外的存储内存，堆上的执行内存，堆外的执行内存4个部分。

Spark 既提供了自行内存和存储内存之间固定边界的实现，又提供了执行内存和存储内存之间 ”软“ 边界的实现。

而 hadoop MapReduce 任务之间的衔接涉及 i/o 开销较高，在前一个任务执行完成之前，其他任务就无法开始，难以胜任复杂 多阶段的计算任务。

##### RDD

RDD resilient Distributed datasets 弹性分布式数据库，是分布式内存的一个抽象概念，RDD

并行分布在整个集群中的；一个RDD是一个不可改变的分布式集合对象。

scala> val lines=sc.textFile(“/home/maixia/soft/helloSpark.txt”)

Lines:org.apache.spark.rdd.RDD[String]=/home/maixia/soft/helloSpark.txt MapPartitionsRDD[11] at textFile at <console>:27

RDDs是spark 分发数据和计算的基础抽象类。

Spark中，所有的计算都是通过RDDs的创建、转换，操作完成。一个RDD内部由多个partition(分片)组成。每个分片包括一部分数据，partition可在集群不同节点上计算。

是只读的记录分区的集合，只能通过其他RDD执行确定的转换操作（map join 和group by）而创建，然而这些限制使得实现容错的开销很低。

Rdd 充分体现了spark 并发操作和灵活容错性能。允许用户在执行多个查询时显示的将工作集缓存在内存中，后续查询能够重用工作集。极大的提升了查询速度。

5个属性

每个分片被一个计算任务处理，并决定并行计算力度，用户可以创建rdd 时指定rd 的分片个数，如果没有指定，就会采用默认值。默认值就是程序所分配到cpu core的数目。

Spark 中Rdd 的计算是以分片为单位的，每个rdd都会实现 Compute函数以达到这个目的。

Compute函数会对迭代器进行复合，不需要保存每次计算的结果。

RDD是只读记录分区的集合，只能通过在其他RDD执行确定的转换操作，如 map join 和group by 而创建，

Rdd的分片函数。 当前spark中实现了2种类型的分片函数，一个是基于哈希的hashPartitioner ,另外一个是基于范围的RangePartitioner.只有对于key-value 的Rdd ,才会有partitioner , 非key value 的rdd的 partitioner的值是None.

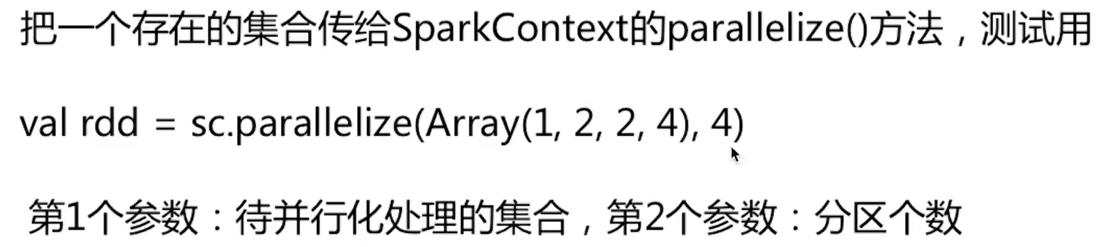
Partitioner函数不但决定了rdd本身的分片数量，也决定了parentRdd Shuffle输出时的分片数量。

RDD 最后一个属性是，一个列表。 存储存取每个Partition的优先位置preferred location

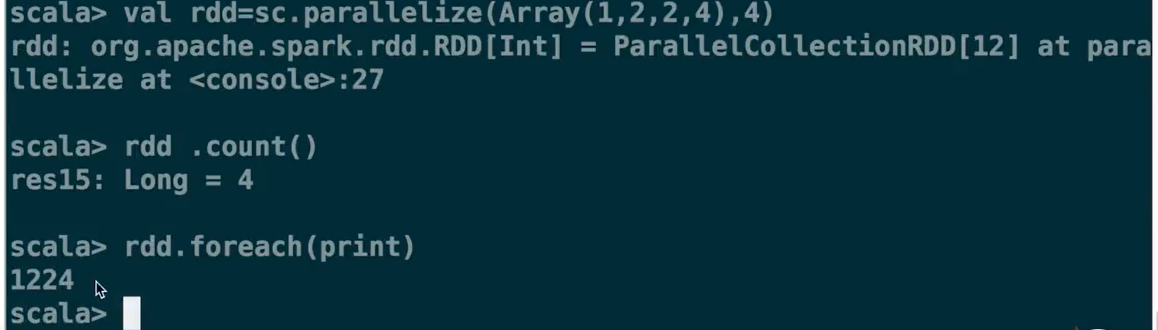
对于hdfs文件来说，这个列表保存的就是每个Partition 所在的块的位置。

按照‘移动数据不如移动计算’ 的理念。 Spark在进行任务调度的时候，会尽可能将计算任务分配到其所要处理数据块的存储位置。

#### RDDs创建方法：



Array(1,2,2,4) 第一个参数：待并行化处理的集合；第2各参数：分区个数



rdd.foreach 遍历每一个变量，然后打印出来

RDDs 加载外部数据集：

var rddText=sc.textFile(“helloSpark.txt”)

Scala 的基础知识：

创建变量使用var（分配后，可以指向类型相同的值） 或

val (变量值不可修改，一旦分配不能重新指向别的值)

val lines=sc.textFile(“helloSpark.txt”)

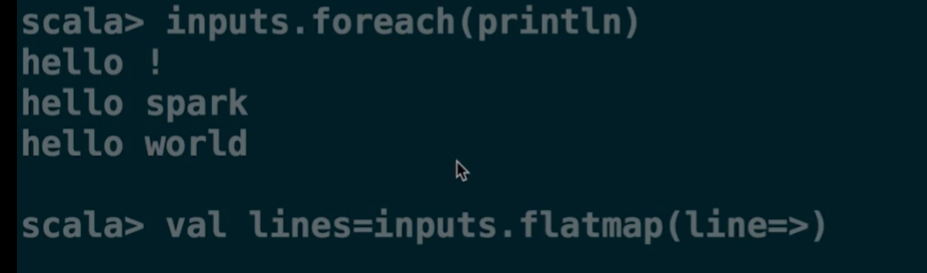
Lines=sc.textFile(“helloSpark2.txt”)

var lines2=sc.textFile(“helloSpark.txt”)

lines2=sc.textFile(“helloSpark2.txt”)

Scala> inputs.foreach(println)

val lines=input.flatMap(line=> line.split(“ ”)) ----使用空格分隔符



scala> val lines=sc.parallelize(Array(“hello”,”spark”,”hello”,”world”,”!”))

scala> lines.foreach(println)

Scala> val lines2=lines.map(word=>(word,1))