**用Apache Spark进行大数据处理——第一部分：入门介绍**

**什么是Spark**

[Apache Spark](https://spark.apache.org/)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。最初在2009年由加州大学伯克利分校的AMPLab开发，并于2010年成为Apache的开源项目之一。

与Hadoop和Storm等其他大数据和MapReduce技术相比，Spark有如下优势。

首先，Spark为我们提供了一个全面、统一的框架用于管理各种有着不同性质（文本数据、图表数据等）的数据集和数据源（批量数据或实时的流数据）的大数据处理的需求。

Spark可以将Hadoop集群中的应用在内存中的运行速度提升100倍，甚至能够将应用在磁盘上的运行速度提升10倍。

Spark让开发者可以快速的用Java、Scala或Python编写程序。它本身自带了一个超过80个高阶操作符集合。而且还可以用它在shell中以交互式地查询数据。

除了Map和Reduce操作之外，它还支持SQL查询，流数据，机器学习和图表数据处理。开发者可以在一个数据管道用例中单独使用某一能力或者将这些能力结合在一起使用。

在这个Apache Spark文章系列的第一部分中，我们将了解到什么是Spark，它与典型的MapReduce解决方案的比较以及它如何为大数据处理提供了一套完整的工具。

## Hadoop和Spark

Hadoop这项大数据处理技术大概已有十年历史，而且被看做是首选的大数据集合处理的解决方案。MapReduce是一路计算的优秀解决方案，不过对于需要多路计算和算法的用例来说，并非十分高效。数据处理流程中的每一步都需要一个Map阶段和一个Reduce阶段，而且如果要利用这一解决方案，需要将所有用例都转换成MapReduce模式。

在下一步开始之前，上一步的作业输出数据必须要存储到分布式文件系统中。因此，复制和磁盘存储会导致这种方式速度变慢。另外Hadoop解决方案中通常会包含难以安装和管理的集群。而且为了处理不同的大数据用例，还需要集成多种不同的工具（如用于机器学习的Mahout和流数据处理的Storm）。

如果想要完成比较复杂的工作，就必须将一系列的MapReduce作业串联起来然后顺序执行这些作业。每一个作业都是高时延的，而且只有在前一个作业完成之后下一个作业才能开始启动。

而Spark则允许程序开发者使用有向无环图（[DAG](http://en.wikipedia.org/wiki/Directed_acyclic_graph)）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。

Spark运行在现有的Hadoop分布式文件系统基础之上（[HDFS](http://wiki.apache.org/hadoop/HDFS)）提供额外的增强功能。它支持[将Spark应用部署到](http://databricks.com/blog/2014/01/21/Spark-and-Hadoop.html)现存的Hadoop v1集群（with SIMR – Spark-Inside-MapReduce）或Hadoop v2 YARN集群甚至是[Apache Mesos](http://mesos.apache.org/)之中。

我们应该将Spark看作是Hadoop MapReduce的一个替代品而不是Hadoop的替代品。其意图并非是替代Hadoop，而是为了提供一个管理不同的大数据用例和需求的全面且统一的解决方案。

## Spark特性

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。利用内存数据存储和接近实时的处理能力，Spark比其他的大数据处理技术的性能要快很多倍。

Spark还支持大数据查询的延迟计算，这可以帮助优化大数据处理流程中的处理步骤。Spark还提供高级的API以提升开发者的生产力，除此之外还为大数据解决方案提供一致的体系架构模型。

Spark将中间结果保存在内存中而不是将其写入磁盘，当需要多次处理同一数据集时，这一点特别实用。Spark的设计初衷就是既可以在内存中又可以在磁盘上工作的执行引擎。当内存中的数据不适用时，Spark操作符就会执行外部操作。Spark可以用于处理大于集群内存容量总和的数据集。

Spark会尝试在内存中存储尽可能多的数据然后将其写入磁盘。它可以将某个数据集的一部分存入内存而剩余部分存入磁盘。开发者需要根据数据和用例评估对内存的需求。Spark的性能优势得益于这种内存中的数据存储。

Spark的其他特性包括：

* 支持比Map和Reduce更多的函数。
* 优化任意操作算子图（operator graphs）。
* 可以帮助优化整体数据处理流程的大数据查询的延迟计算。
* 提供简明、一致的Scala，Java和Python API。
* 提供交互式Scala和Python Shell。目前暂不支持Java。

Spark是用[Scala程序设计语言](http://www.scala-lang.org/)编写而成，运行于Java虚拟机（JVM）环境之上。目前支持如下程序设计语言编写Spark应用：

* Scala
* Java
* Python
* Clojure
* R

## Spark生态系统

除了Spark核心API之外，Spark生态系统中还包括其他附加库，可以在大数据分析和机器学习领域提供更多的能力。

这些库包括：

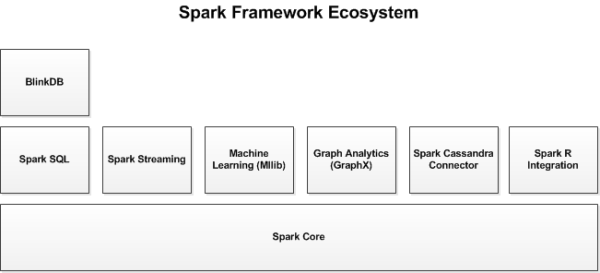
* **Spark Streaming:**
  + [Spark Streaming](https://spark.apache.org/streaming/)基于微批量方式的计算和处理，可以用于处理实时的流数据。它使用DStream，简单来说就是一个弹性分布式数据集（RDD）系列，处理实时数据。
* **Spark SQL:**
  + [Spark SQL](https://spark.apache.org/sql/)可以通过JDBC API将Spark数据集暴露出去，而且还可以用传统的BI和可视化工具在Spark数据上执行类似SQL的查询。用户还可以用Spark SQL对不同格式的数据（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](https://spark.apache.org/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](https://spark.apache.org/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。

[BlinkDB](http://blinkdb.org/)是一个近似查询引擎，用于在海量数据上执行交互式SQL查询。BlinkDB可以通过牺牲数据精度来提升查询响应时间。通过在数据样本上执行查询并展示包含有意义的错误线注解的结果，操作大数据集合。

[Tachyon](http://tachyon-project.org/index.html)是一个以内存为中心的分布式文件系统，能够提供内存级别速度的跨集群框架（如Spark和MapReduce）的可信文件共享。它将工作集文件缓存在内存中，从而避免到磁盘中加载需要经常读取的数据集。通过这一机制，不同的作业/查询和框架可以以内存级的速度访问缓存的文件。  
此外，还有一些用于与其他产品集成的适配器，如Cassandra（[Spark Cassandra 连接器](http://www.datastax.com/dev/blog/accessing-cassandra-from-spark-in-java)）和R（SparkR）。Cassandra Connector可用于访问存储在Cassandra数据库中的数据并在这些数据上执行数据分析。

下图展示了在Spark生态系统中，这些不同的库之间的相互关联。



**图1. Spark框架中的库**

我们将在这一系列文章中逐步探索这些Spark库

## Spark体系架构

Spark体系架构包括如下三个主要组件：

* 数据存储
* API
* 管理框架

接下来让我们详细了解一下这些组件。

**数据存储：**

Spark用HDFS文件系统存储数据。它可用于存储任何兼容于Hadoop的数据源，包括HDFS，HBase，Cassandra等。

**API：**

利用API，应用开发者可以用标准的API接口创建基于Spark的应用。Spark提供Scala，Java和Python三种程序设计语言的API。

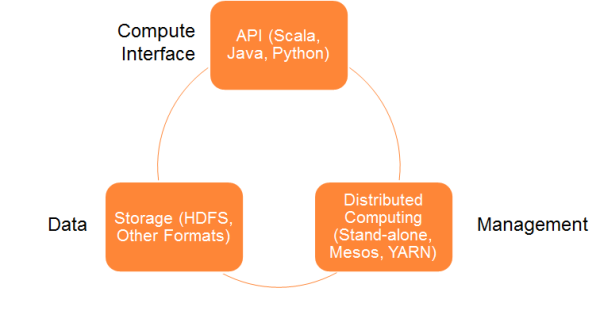
下面是三种语言Spark API的网站链接。

* [**Scala API**](http://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.package)
* [**Java**](http://spark.apache.org/docs/latest/api/java/index.html)
* [**Python**](http://spark.apache.org/docs/latest/api/python/index.html)

**资源管理：**

Spark既可以部署在一个单独的服务器也可以部署在像Mesos或YARN这样的分布式计算框架之上。

下图2展示了Spark体系架构模型中的各个组件。



**图2 Spark体系架构**

## 弹性分布式数据集

[弹性分布式数据集](https://spark.apache.org/docs/latest/programming-guide.html#resilient-distributed-datasets-rdds)（基于Matei的[研究论文](https://www.cs.berkeley.edu/%7Ematei/papers/2012/nsdi_spark.pdf)）或RDD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。

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。你可以用变换（Transformation）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。

RDD支持两种类型的操作：

* 变换（Transformation）
* 行动（Action）

**变换：**[变换](https://spark.apache.org/docs/latest/programming-guide.html" \l "transformations)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。

**行动：**[行动](https://spark.apache.org/docs/latest/programming-guide.html#actions)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。

行动操作包括：reduce，collect，count，first，take，countByKey以及foreach。

## 如何安装Spark

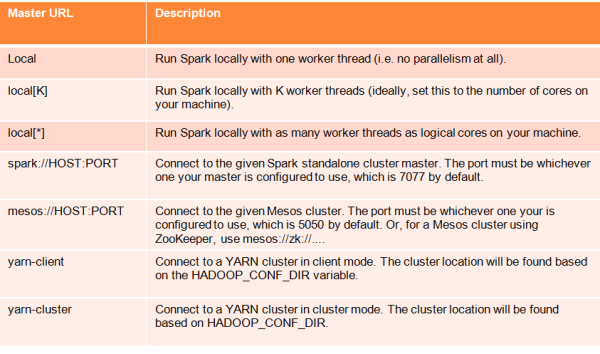
安装和使用Spark有几种不同方式。你可以在自己的电脑上将Spark作为一个独立的框架安装或者从诸如[Cloudera](http://www.cloudera.com/content/support/en/downloads/quickstart_vms/cdh-5-1-x.html)，HortonWorks或MapR之类的供应商处获取一个Spark虚拟机镜像直接使用。或者你也可以使用在云端环境（如[Databricks Cloud](http://databricks.com/product)）安装并配置好的Spark。

在本文中，我们将把Spark作为一个独立的框架安装并在本地启动它。最近Spark刚刚发布了1.2.0版本。我们将用这一版本完成示例应用的代码展示。

## 如何运行Spark

当你在本地机器安装了Spark或使用了基于云端的Spark后，有几种不同的方式可以连接到Spark引擎。

下表展示了不同的Spark运行模式所需的Master URL参数。



## 如何与Spark交互

Spark启动并运行后，可以用Spark shell连接到Spark引擎进行交互式数据分析。Spark shell支持Scala和Python两种语言。Java不支持交互式的Shell，因此这一功能暂未在Java语言中实现。

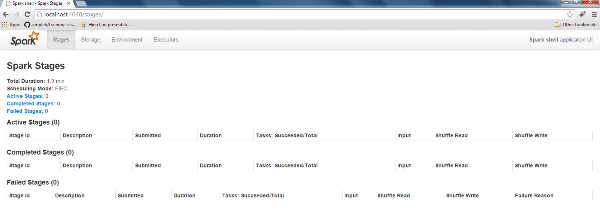
可以用spark-shell.cmd和pyspark.cmd命令分别运行Scala版本和Python版本的Spark Shell。

## Spark网页控制台

不论Spark运行在哪一种模式下，都可以通过访问Spark网页控制台查看Spark的作业结果和其他的统计数据，控制台的URL地址如下：

http://localhost:4040

Spark控制台如下图3所示，包括Stages，Storage，Environment和Executors四个标签页



### 共享变量

Spark提供两种类型的共享变量可以提升集群环境中的Spark程序运行效率。分别是广播变量和累加器。

**广播变量：**广播变量可以在每台机器上缓存只读变量而不需要为各个任务发送该变量的拷贝。他们可以让大的输入数据集的集群拷贝中的节点更加高效。

下面的代码片段展示了如何使用广播变量。

//

// Broadcast Variables

//

val broadcastVar = sc.broadcast(Array(1, 2, 3))

broadcastVar.value

**累加器：**只有在使用相关操作时才会添加累加器，因此它可以很好地支持并行。累加器可用于实现计数（就像在MapReduce中那样）或求和。可以用add方法将运行在集群上的任务添加到一个累加器变量中。不过这些任务无法读取变量的值。只有驱动程序才能够读取累加器的值。

下面的代码片段展示了如何使用累加器共享变量：

//

// Accumulators

//

val accum = sc.accumulator(0, "My Accumulator")

sc.parallelize(Array(1, 2, 3, 4)).foreach(x => accum += x)

accum.value

## Spark应用示例

本篇文章中所涉及的示例应用是一个简单的字数统计应用。这与学习用Hadoop进行大数据处理时的示例应用相同。我们将在一个文本文件上执行一些数据分析查询。本示例中的文本文件和数据集都很小，不过无须修改任何代码，示例中所用到的Spark查询同样可以用到大容量数据集之上。

为了让讨论尽量简单，我们将使用Spark Scala Shell。

首先让我们看一下如何在你自己的电脑上安装Spark。

**前提条件：**

* 为了让Spark能够在本机正常工作，你需要安装Java开发工具包（JDK）。这将包含在下面的第一步中。
* 同样还需要在电脑上安装Spark软件。下面的第二步将介绍如何完成这项工作。

**注：**下面这些指令都是以Windows环境为例。如果你使用不同的操作系统环境，需要相应的修改系统变量和目录路径已匹配你的环境。

**I. 安装JDK**

1）从Oracle网站上下载JDK。推荐使用[JDK 1.7版本](http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html)。

将JDK安装到一个没有空格的目录下。对于Windows用户，需要将JDK安装到像c:\dev这样的文件夹下，而不能安装到“c:\Program Files”文件夹下。“c:\Program Files”文件夹的名字中包含空格，如果软件安装到这个文件夹下会导致一些问题。

**注：不要**在“c:\Program Files”文件夹中安装JDK或（第二步中所描述的）Spark软件。

2）完成JDK安装后，切换至JDK 1.7目录下的”bin“文件夹，然后键入如下命令，验证JDK是否正确安装：

java -version

如果JDK安装正确，上述命令将显示Java版本。

**II. 安装Spark软件：**

从[Spark网站](https://spark.apache.org/downloads.html)上下载最新版本的Spark。在本文发表时，最新的Spark版本是1.2。你可以根据Hadoop的版本选择一个特定的Spark版本安装。我下载了与Hadoop 2.4或更高版本匹配的Spark，文件名是spark-1.2.0-bin-hadoop2.4.tgz。

将安装文件解压到本地文件夹中（如：c:\dev）。

为了验证Spark安装的正确性，切换至Spark文件夹然后用如下命令启动Spark Shell。这是Windows环境下的命令。如果使用Linux或Mac OS，请相应地编辑命令以便能够在相应的平台上正确运行。

c:

cd c:\dev\spark-1.2.0-bin-hadoop2.4

bin\spark-shell

如果Spark安装正确，就能够在控制台的输出中看到如下信息。

….

15/01/17 23:17:46 INFO HttpServer: Starting HTTP Server

15/01/17 23:17:46 INFO Utils: Successfully started service 'HTTP class server' on port 58132.

Welcome to

\_\_\_\_ \_\_

/ \_\_/\_\_ \_\_\_ \_\_\_\_\_/ /\_\_

\_\ \/ \_ \/ \_ `/ \_\_/ '\_/

/\_\_\_/ .\_\_/\\_,\_/\_/ /\_/\\_\ version 1.2.0

/\_/

Using Scala version 2.10.4 (Java HotSpot(TM) 64-Bit Server VM, Java 1.7.0\_71)

Type in expressions to have them evaluated.

Type :help for more information.

….

15/01/17 23:17:53 INFO BlockManagerMaster: Registered BlockManager

15/01/17 23:17:53 INFO SparkILoop: Created spark context..

Spark context available as sc.

可以键入如下命令检查Spark Shell是否工作正常。

sc.version

（或）

sc.appName

完成上述步骤之后，可以键入如下命令退出Spark Shell窗口：

:quit

如果想启动Spark Python Shell，需要先在电脑上安装Python。你可以下载并安装[Anaconda](http://continuum.io/downloads)，这是一个免费的Python发行版本，其中包括了一些比较流行的科学、数学、工程和数据分析方面的Python包。

然后可以运行如下命令启动Spark Python Shell：

c:

cd c:\dev\spark-1.2.0-bin-hadoop2.4

bin\pyspark

## Spark示例应用

完成Spark安装并启动后，就可以用Spark API执行数据分析查询了。

这些从文本文件中读取并处理数据的命令都很简单。我们将在这一系列文章的后续文章中向大家介绍更高级的Spark框架使用的用例。

首先让我们用Spark API运行流行的Word Count示例。如果还没有运行Spark Scala Shell，首先打开一个Scala Shell窗口。这个示例的相关命令如下所示：

import org.apache.spark.SparkContext

import org.apache.spark.SparkContext.\_

val txtFile = "README.md"

val txtData = sc.textFile(txtFile)

txtData.cache()

我们可以调用cache函数将上一步生成的RDD对象保存到缓存中，在此之后Spark就不需要在每次数据查询时都重新计算。需要注意的是，cache()是一个延迟操作。在我们调用cache时，Spark并不会马上将数据存储到内存中。只有当在某个RDD上调用一个行动时，才会真正执行这个操作。

现在，我们可以调用count函数，看一下在文本文件中有多少行数据。

txtData.count()

然后，我们可以执行如下命令进行字数统计。在文本文件中统计数据会显示在每个单词的后面。

val wcData = txtData.flatMap(l => l.split(" ")).map(word => (word, 1)).reduceByKey(\_ + \_)

wcData.collect().foreach(println)

如果想查看更多关于如何使用Spark核心API的代码示例，请参考网站上的[Spark文档](http://spark.apache.org/docs/latest/programming-guide.html)。

## 后续计划

在后续的系列文章中，我们将从Spark SQL开始，学习更多关于Spark生态系统的其他部分。之后，我们将继续了解Spark Streaming，Spark MLlib和Spark GraphX。我们也会有机会学习像Tachyon和BlinkDB等框架。

作者：用心阁  
链接：https://www.zhihu.com/question/26568496/answer/41608400  
来源：知乎  
著作权归作者所有，转载请联系作者获得授权。  
  
**Apache Spark**  
Apache Spark是一个新兴的大数据处理的引擎，主要特点是提供了一个集群的分布式内存抽象，以支持需要工作集的应用。  
  
这个抽象就是RDD（Resilient Distributed Dataset），RDD就是一个不可变的带分区的记录集合，RDD也是Spark中的编程模型。Spark提供了RDD上的两类操作，转换和动作。转换是用来定义一个新的RDD，包括map, flatMap, filter, union, sample, join, groupByKey, cogroup, ReduceByKey, cros, sortByKey, mapValues等，动作是返回一个结果，包括collect, reduce, count, save, lookupKey。  
  
Spark的API非常简单易用，Spark的WordCount的示例如下所示：

val spark = new SparkContext(master, appName, [sparkHome], [jars])

val file = spark.textFile("hdfs://...")

val counts = file.flatMap(line => line.split(" "))

.map(word => (word, 1))

.reduceByKey(\_ + \_)

counts.saveAsTextFile("hdfs://...")

其中的file是根据HDFS上的文件创建的RDD，后面的flatMap，map，reduceByKe都创建出一个新的RDD，一个简短的程序就能够执行很多个转换和动作。  
  
在Spark中，所有RDD的转换都是是**惰性求值**的。RDD的转换操作会生成新的RDD，新的RDD的数据依赖于原来的RDD的数据，每个RDD又包含多个分区。那么一段程序实际上就构造了一个由相互依赖的多个RDD组成的**有向无环图（DAG）。**并通过在RDD上执行动作将这个有向无环图作为一个Job提交给Spark执行。  
  
例如，上面的WordCount程序就会生成如下的DAG

scala> counts.toDebugString

res0: String =

MapPartitionsRDD[7] at reduceByKey at <console>:14 (1 partitions)

ShuffledRDD[6] at reduceByKey at <console>:14 (1 partitions)

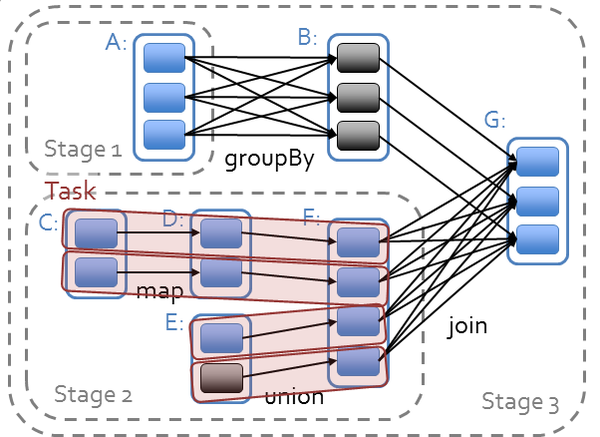
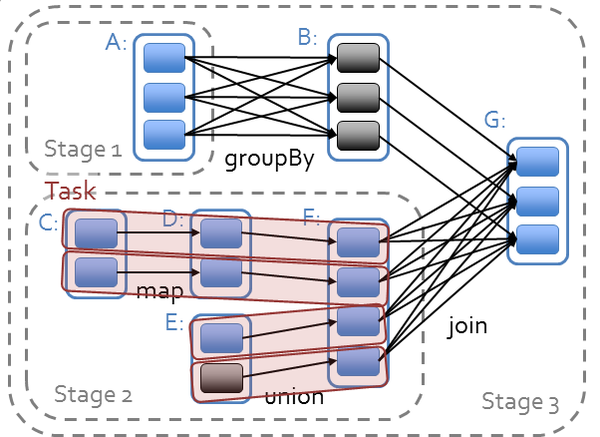
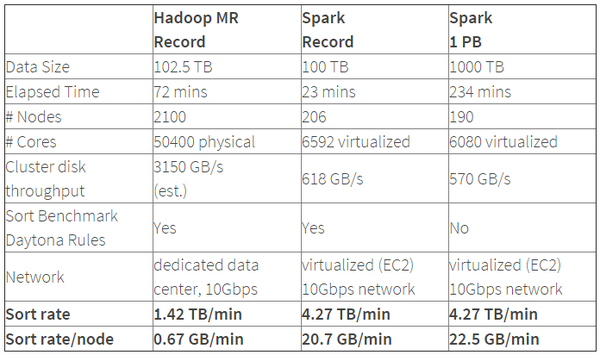
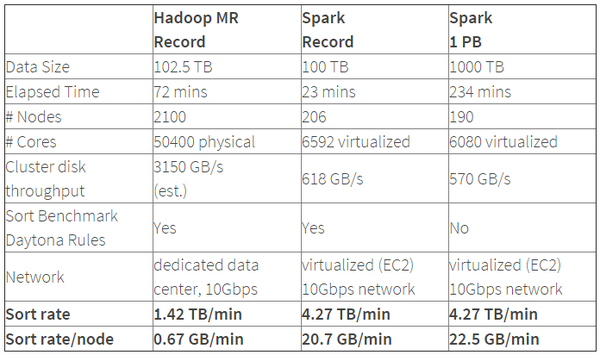
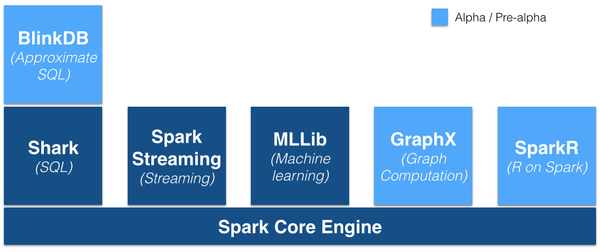
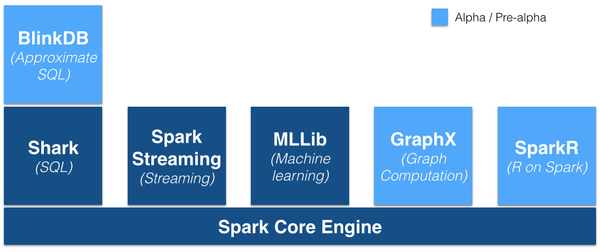
MapPartitionsRDD[5] at reduceByKey at <console>:14 (1 partitions)

MappedRDD[4] at map at <console>:14 (1 partitions)

FlatMappedRDD[3] at flatMap at <console>:14 (1 partitions)

MappedRDD[1] at textFile at <console>:12 (1 partitions)

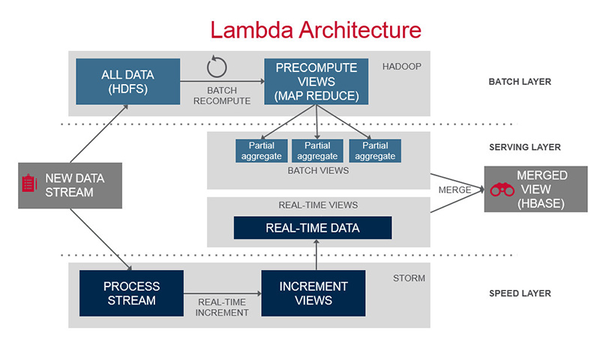
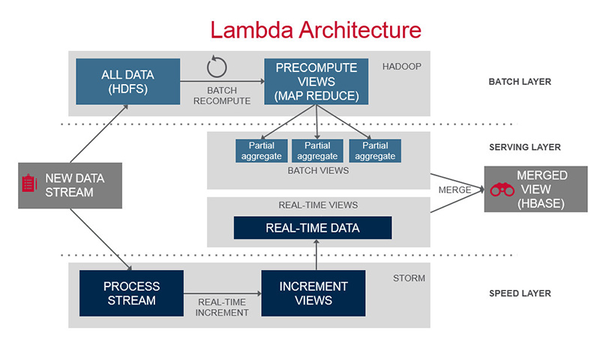
HadoopRDD[0] at textFile at <console>:12 (1 partitions)

Spark对于有向无环图Job进行调度，确定**阶段（Stage）**，**分区（Partition）**，**流水线（Pipeline）**，**任务（Task）**和**缓存（Cache）**，进行优化，并在Spark集群上运行Job。RDD之间的依赖分为**宽依赖**（依赖多个分区）和**窄依赖**（只依赖一个分区），在确定阶段时，需要根据宽依赖划分阶段。根据分区划分任务。  
  
（图片来源：（图片来源：[https://databricks-training.s3.amazonaws.com/slides/advanced-spark-training.pdf](//link.zhihu.com/?target=https%3A//databricks-training.s3.amazonaws.com/slides/advanced-spark-training.pdf" \t "_blank)）  
  
Spark支持故障恢复的方式也不同，提供两种方式，**Linage**，通过数据的血缘关系，再执行一遍前面的处理，**Checkpoint**，将数据集存储到持久存储中。  
  
Spark为**迭代式数据处理**提供更好的支持。每次迭代的数据可以保存在内存中，而不是写入文件。  
  
Spark的性能相比Hadoop有很大提升，2014年10月，Spark完成了一个Daytona Gray类别的Sort Benchmark测试，排序完全是在磁盘上进行的，与Hadoop之前的测试的对比结果如表格所示：  
（表格来源： （表格来源： [Spark officially sets a new record in large-scale sorting](//link.zhihu.com/?target=http%3A//databricks.com/blog/2014/11/05/spark-officially-sets-a-new-record-in-large-scale-sorting.html)）  
从表格中可以看出排序100TB的数据（1万亿条数据），**Spark只用了Hadoop所用1/10的计算资源，耗时只有Hadoop的1/3**。  
  
  
Spark的优势不仅体现在性能提升上的，Spark框架为批处理（Spark Core），交互式（Spark SQL），流式（Spark Streaming），机器学习（MLlib），图计算（GraphX）提供一个统一的数据处理平台，这相对于使用Hadoop有很大优势。  
  
（图片来源：（图片来源：[https://gigaom.com/2014/06/28/4-reasons-why-spark-could-jolt-hadoop-into-hyperdrive/](//link.zhihu.com/?target=https%3A//gigaom.com/2014/06/28/4-reasons-why-spark-could-jolt-hadoop-into-hyperdrive/" \t "_blank)）  
  
按照Databricks的连城的说法是**One Stack To Rule Them All**  
  
特别是在有些情况下，你需要进行一些ETL工作，然后训练一个机器学习的模型，最后进行一些查询，如果是使用Spark，你可以在一段程序中将这三部分的逻辑完成形成一个大的**有向无环图（DAG）**，而且Spark会对大的有向无环图进行整体优化。  
  
例如下面的程序：

val points = sqlContext.sql( “SELECT latitude, longitude FROM historic\_tweets”)

val model = KMeans.train(points, 10)

sc.twitterStream(...) .map(t => (model.closestCenter(t.location), 1)) .reduceByWindow(“5s”, \_ + \_)

（示例来源：[http://www.slideshare.net/Hadoop\_Summit/building-a-unified-data-pipeline-in-apache-spark](//link.zhihu.com/?target=http%3A//www.slideshare.net/Hadoop_Summit/building-a-unified-data-pipeline-in-apache-spark" \t "_blank)）  
这段程序的第一行是用Spark SQL 查寻出了一些点，第二行是用MLlib中的K-means算法使用这些点训练了一个模型，第三行是用Spark Streaming处理流中的消息，使用了训练好的模型。  
  
**Lambda Architecture**  
Lambda Architecture是一个大数据处理平台的参考模型，如下图所示：  
（图片来源： （图片来源： [Lambda Architecture](//link.zhihu.com/?target=https%3A//www.mapr.com/developercentral/lambda-architecture)）  
其中包含3层，Batch Layer，Speed Layer和Serving Layer，由于Batch Layer和Speed Layer的数据处理逻辑是一致的，如果用Hadoop作为Batch Layer，而用Storm作为Speed Layer，你需要维护两份使用不同技术的代码。  
  
而Spark可以作为Lambda Architecture一体化的解决方案,大致如下：

* Batch Layer，HDFS+Spark Core，将实时的增量数据追加到HDFS中，使用Spark Core批量处理全量数据，生成全量数据的视图。，
* Speed Layer，Spark Streaming来处理实时的增量数据，以较低的时延生成实时数据的视图。
* Serving Layer，HDFS+Spark SQL（也许还有BlinkDB），存储Batch Layer和Speed Layer输出的视图，提供低时延的即席查询功能，将批量数据的视图与实时数据的视图合并。

**总结**  
如果说，MapReduce是公认的分布式数据处理的低层次抽象，类似逻辑门电路中的与门，或门和非门，那么Spark的RDD就是分布式大数据处理的高层次抽象，类似逻辑电路中的编码器或译码器等。  
  
RDD就是一个分布式的数据集合（Collection），对这个集合的任何操作都可以像函数式编程中操作内存中的集合一样直观、简便，但集合操作的实现确是在后台分解成一系列Task发送到几十台上百台服务器组成的集群上完成的。最近新推出的大数据处理框架Apache Flink也使用数据集（Data Set）和其上的操作作为编程模型的。  
  
由RDD组成的有向无环图（DAG）的执行是调度程序将其生成物理计划并进行优化，然后在Spark集群上执行的。Spark还提供了一个类似于MapReduce的执行引擎，该引擎更多地使用内存，而不是磁盘，得到了更好的执行性能。  
  
那么Spark解决了Hadoop的哪些问题呢？

* 抽象层次低，需要手工编写代码来完成，使用上难以上手。
  + =>基于RDD的抽象，实数据处理逻辑的代码非常简短。。
* 只提供两个操作，Map和Reduce，表达力欠缺。
  + =>提供很多转换和动作，很多基本操作如Join，GroupBy已经在RDD转换和动作中实现。
* 一个Job只有Map和Reduce两个阶段（Phase），复杂的计算需要大量的Job完成，Job之间的依赖关系是由开发者自己管理的。
  + =>一个Job可以包含RDD的多个转换操作，在调度时可以生成多个阶段（Stage），而且如果多个map操作的RDD的分区不变，是可以放在同一个Task中进行。
* 处理逻辑隐藏在代码细节中，没有整体逻辑
  + =>在Scala中，通过匿名函数和高阶函数，RDD的转换支持流式API，可以提供处理逻辑的整体视图。代码不包含具体操作的实现细节，逻辑更清晰。
* 中间结果也放在HDFS文件系统中
  + =>中间结果放在内存中，内存放不下了会写入本地磁盘，而不是HDFS。
* ReduceTask需要等待所有MapTask都完成后才可以开始
  + => 分区相同的转换构成流水线放在一个Task中运行，分区不同的转换需要Shuffle，被划分到不同的Stage中，需要等待前面的Stage完成后才可以开始。
* 时延高，只适用Batch数据处理，对于交互式数据处理，实时数据处理的支持不够
  + =>通过将流拆成小的batch提供Discretized Stream处理流数据。
* 对于迭代式数据处理性能比较差
  + =>通过在内存中缓存数据，提高迭代式计算的性能。

因此，Hadoop MapReduce会被新一代的大数据处理平台替代是技术发展的趋势，而在新一代的大数据处理平台中，Spark目前得到了最广泛的认可和支持，从参加Spark Summit 2014的厂商的各种基于Spark平台进行的开发就可以看出一二。