84 个回答 默认排序

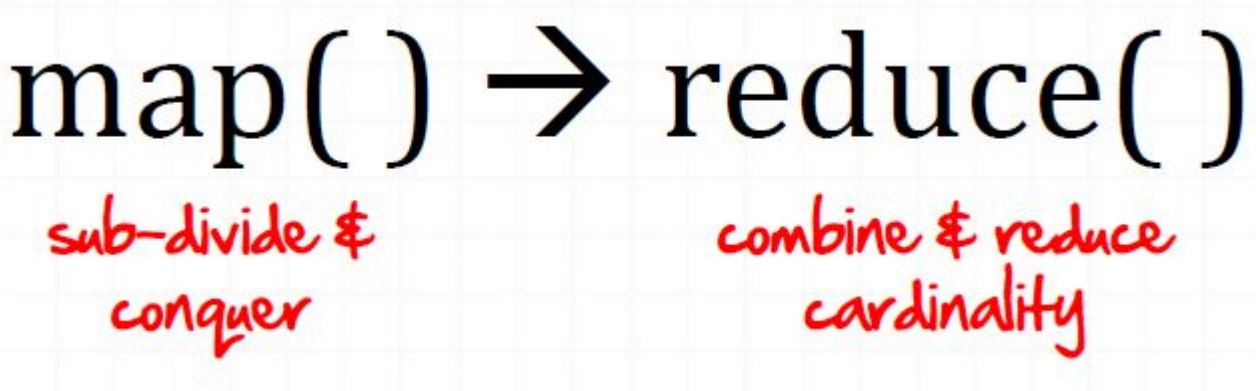
用心阁
软件工程师

3,752 人赞同了该回答

Hadoop

首先看一下Hadoop解决了什么问题，Hadoop就是解决了大数据（大到一台计算机无法进行存储，一台计算机无法在要求的时间内进行处理）的可靠存储和处理。

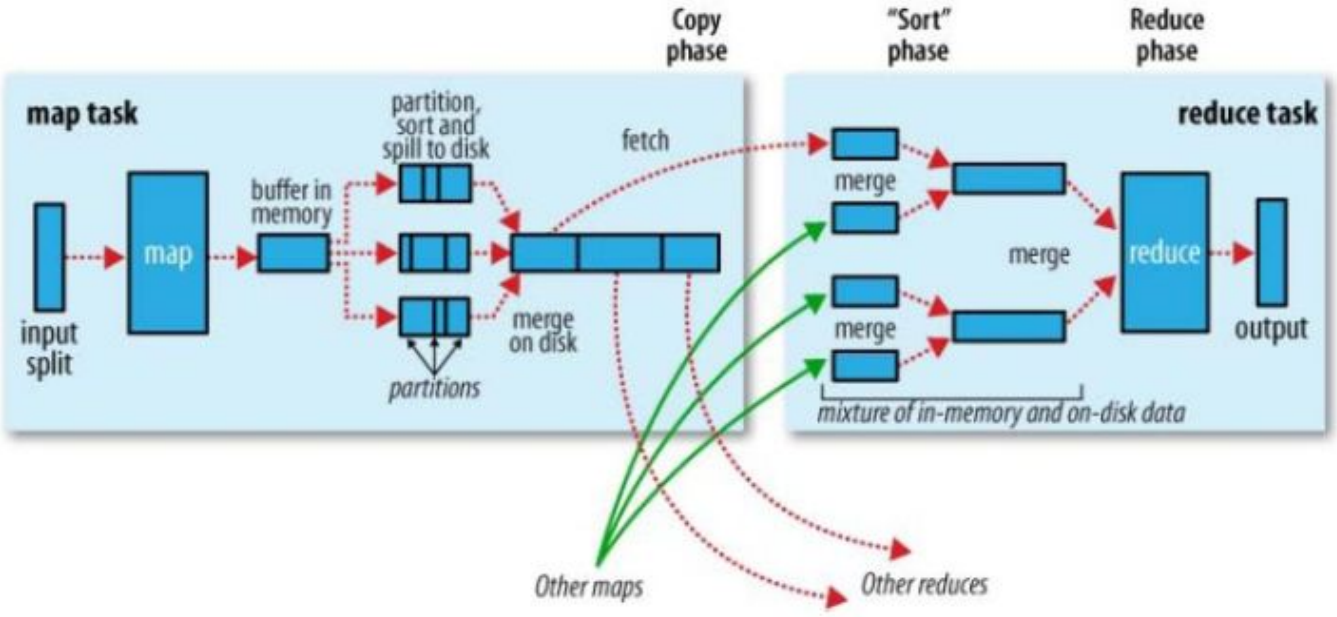
HDFS，在由普通PC组成的集群上提供高可靠的文件存储，通过将块保存多个副本的办法解决服务器或硬盘坏掉的问题。
MapReduce，通过简单的Mapper和Reducer的抽象提供一个编程模型，可以在一个由几十台上百台的PC组成的不可靠集群上并发地，分布式地处理大量的数据集，而把并发、分布式（如机器间通信）和故障恢复等计算细节隐藏起来。而Mapper和Reducer的抽象，又是各种各样的复杂数据处理都可以分解为的基本元素。这样，复杂的数据处理可以分解为由多个Job（包含一个Mapper和一个Reducer）组成的有向无环图（DAG）,然后每个Mapper和Reducer放到Hadoop集群上执行，就可以得出结果。



（图片来源：slideshare.net/davideng）

用MapReduce统计一个文本文件中单词出现的频率的示例WordCount请参见：WordCount - Hadoop Wiki，如果对MapReduce不恨熟悉，通过该示例对MapReduce进行一些了解对理解下文有帮助。

在MapReduce中，Shuffle是一个非常重要的过程，正是有了看不见的Shuffle过程，才可以使在MapReduce之上写数据处理的开发者完全感知不到分布式和并发的存在。

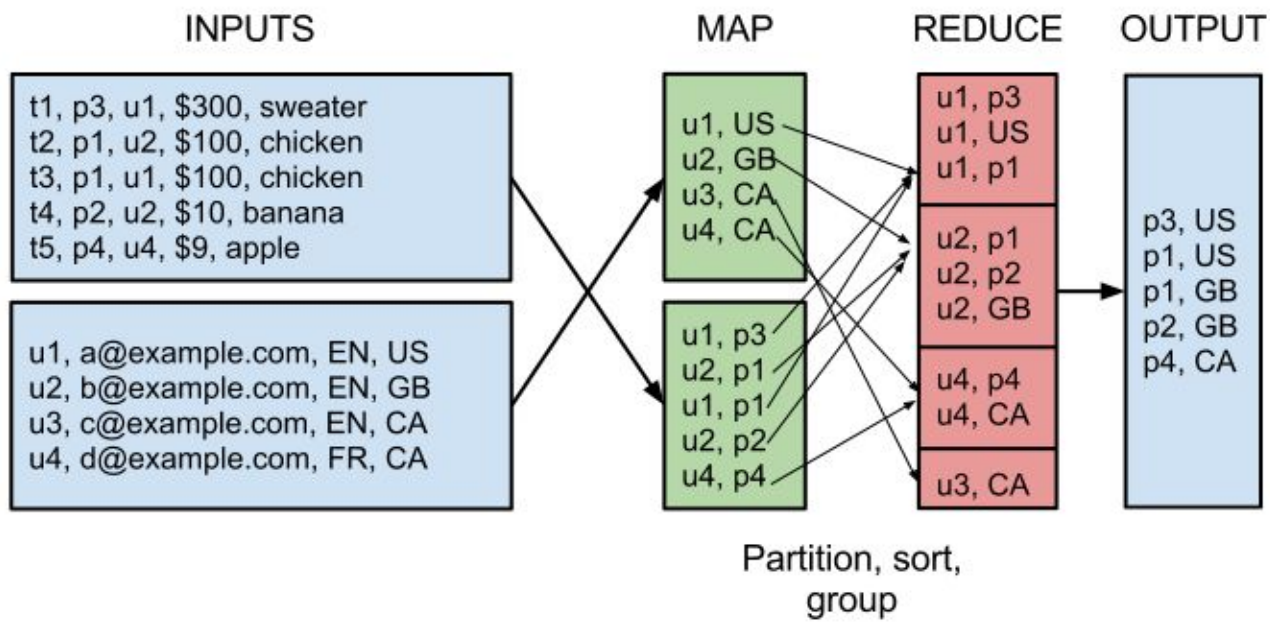


（图片来源： Hadoop Definitive Guide By Tom White）
广义的Shuffle是指图中在Map和Reuce之间的一系列过程。

Hadoop的局限和不足
但是，MapRecue存在以下局限，使用起来比较困难。

抽象层次低，需要手工编写代码来完成，使用上难以上手。
只提供两个操作，Map和Reduce，表达力欠缺。
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。
处理逻辑隐藏在代码细节中，没有整体逻辑
中间结果也放在HDFS文件系统中
ReduceTask需要等待所有MapTask都完成后才可以开始
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
对于迭代式数据处理性能比较差

比如说，用MapReduce实现两个表的Join都是一个很有技巧性的过程，如下图所示：



（图片来源：Real World Hadoop）

因此，在Hadoop推出之后，出现了很多相关的技术对其中的局限进行改进，如Pig，Cascading，JAQL，OOzie，Tez，Spark等。

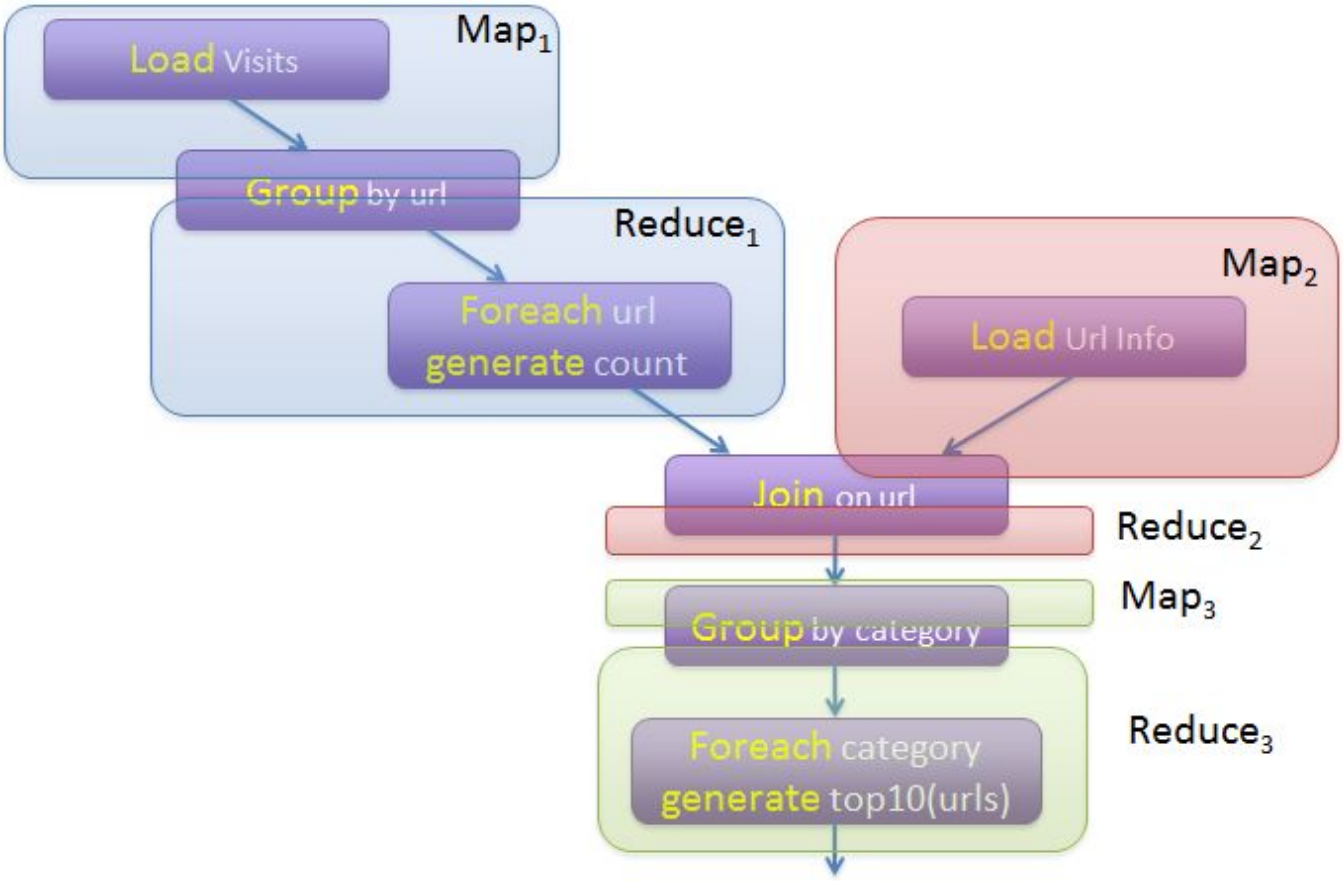
Apache Pig

Apache Pig也是Hadoop框架中的一部分，Pig提供类SQL语言（Pig Latin）通过MapReduce来处理大规模半结构化数据。而Pig Latin是更高级的过程语言，通过将MapReduce中的设计模式抽象为操作，如Filter，GroupBy，Join，OrderBy，由这些操作组成**有向无环图（DAG）**。例如如下程序：

```
visits             = load ‘/data/visits’ as (user, url, time);
gVisits            = group visits by url;
visitCounts  = foreach gVisits generate url, count(visits);

urlInfo            = load ‘/data/urlInfo’ as (url, category, pRank);
visitCounts  = join visitCounts by url, urlInfo by url;

gCategories = group visitCounts by category;
topUrls = foreach gCategories generate top(visitCounts,10);

store topUrls into ‘/data/topUrls’;
```

描述了数据处理的整个过程。

而Pig Latin又是通过编译为MapReduce，在Hadoop集群上执行的。上述程序被编译成MapReduce时，会产生如下图所示的Map和Reduce：



（图片来源：cs.nyu.edu/courses/Fall）

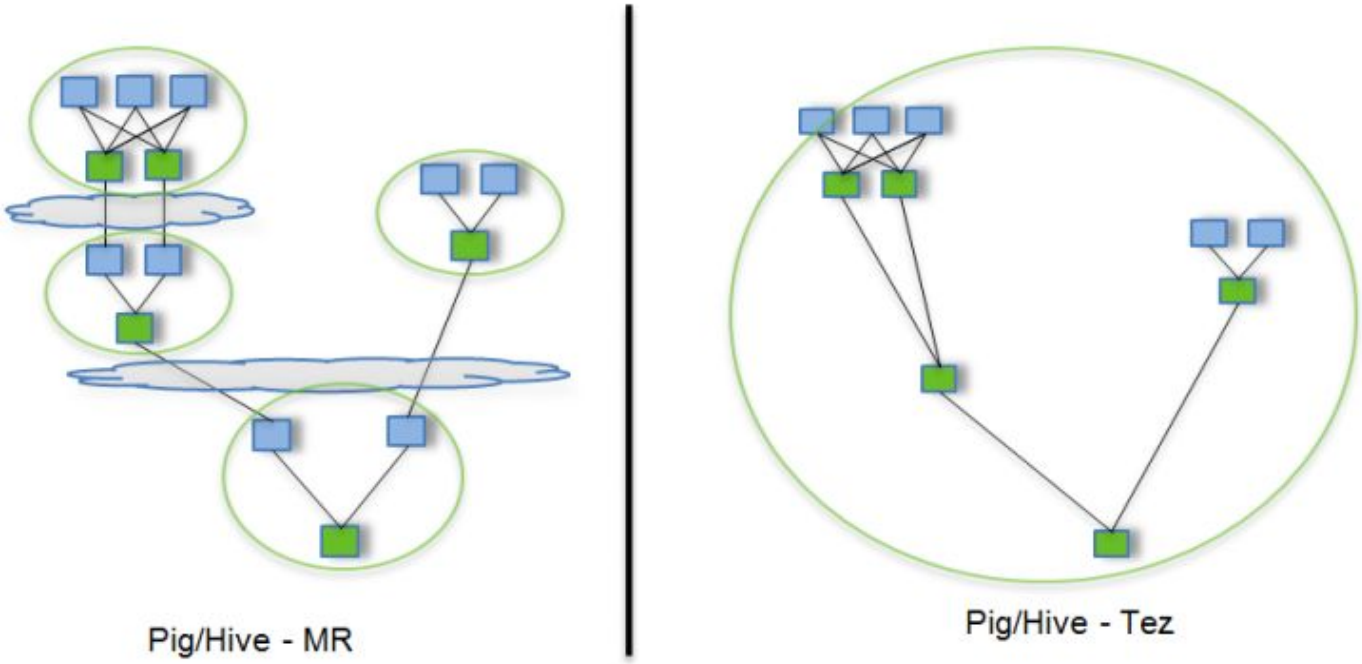
Apache Pig解决了MapReduce存在的大量手写代码，语义隐藏，提供操作种类少的问题。类似的项目还有Cascading，JAQL等。

Apache Tez

Apache Tez，Tez是HortonWorks的Stinger Initiative的的一部分。作为执行引擎，Tez也提供了**有向无环图（DAG）**，DAG由顶点（Vertex）和边（Edge）组成，Edge是对数据的移动的抽象，提供了One-To-One，BroadCast，和Scatter-Gather三种类型，只有Scatter-Gather才需要进行Shuffle。

以如下SQL为例：

```
SELECT a.state, COUNT(*),
AVERAGE(c.price)
FROM a
JOIN b ON (a.id = b.id)
JOIN c ON (a.itemId = c.itemId)
GROUP BY a.state
```



（图片来源：slideshare.net/hortonwo）
途中蓝色方块表示Map，绿色方块表示Reduce，云状表示写屏障（write barrier，一种内核机制，可以理解为持久的写），Tez的优化主要体现在：

去除了连续两个作业之间的写屏障
去除了每个工作流中多余的Map阶段（Stage）

通过提供DAG语义和操作，提供了整体的逻辑，通过减少不必要的操作，Tez提升了数据处理的执行性能。

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://...")
```

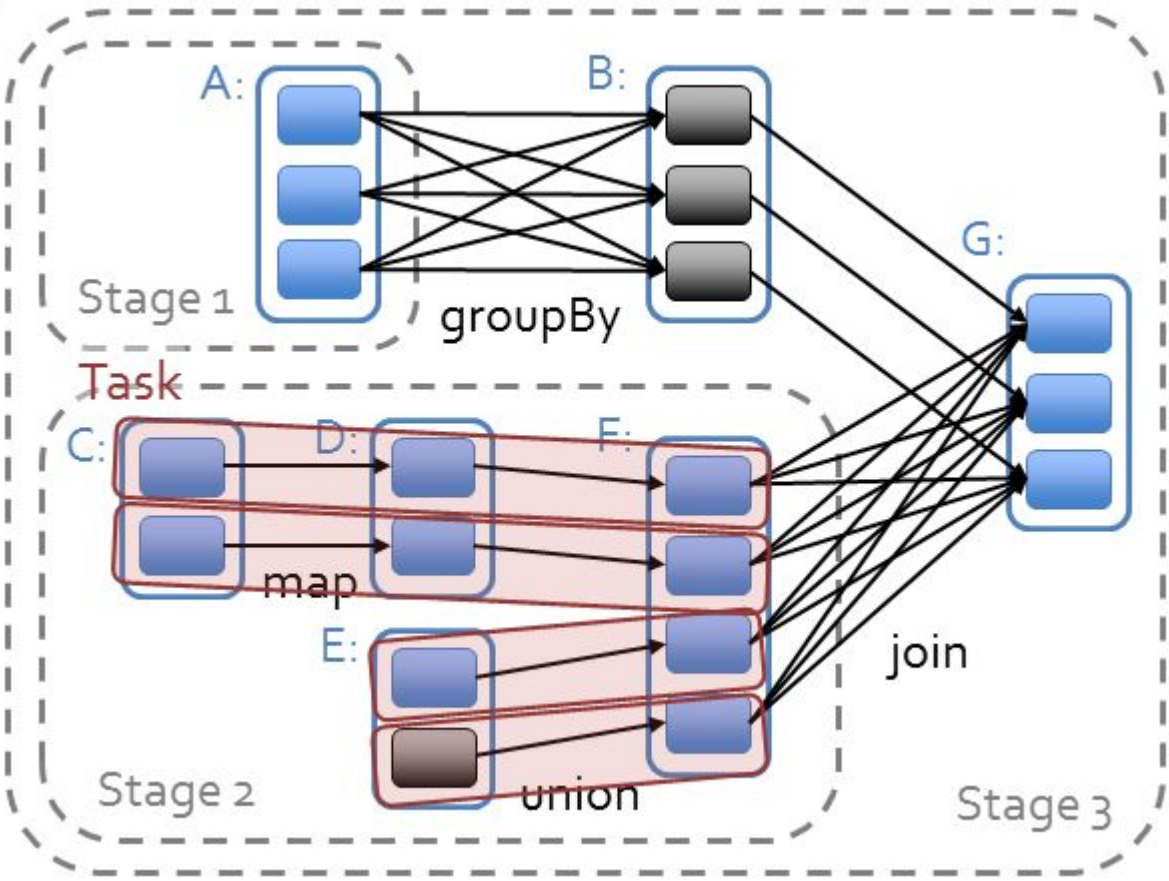
其中的file是根据HDFS上的文件创建的RDD，后面的flatMap，map，reduceByKe都创建出一个新的RDD，一个简短的程序就能够执行很多个转换和动作。

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：databricks-training.s3.amazonaws.com）

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。

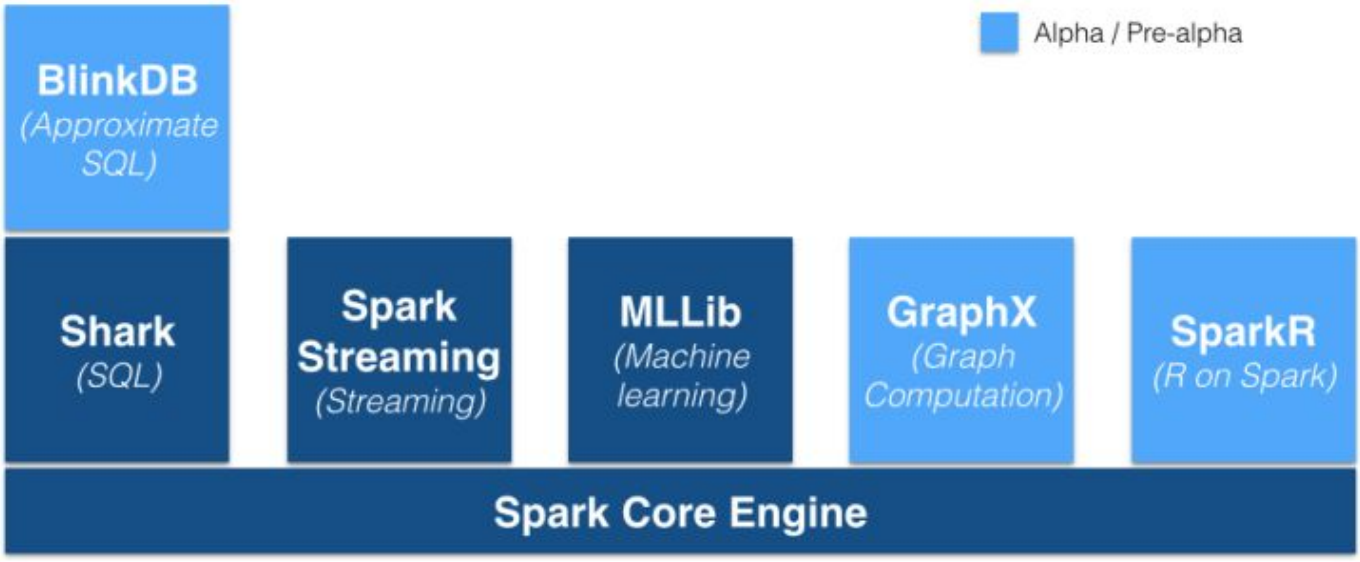
Spark为**迭代式数据处理**提供更好的支持。每次迭代的数据可以保存在内存中，而不是写入文件。

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：

	Hadoop MR Record	Spark Record	Spark 1 PB
Data Size	102.5 TB	100 TB	1000 TB
Elapsed Time	72 mins	23 mins	234 mins
# Nodes	2100	206	190
# Cores	50400 physical	6592 virtualized	6080 virtualized
Cluster disk throughput	3150 GB/s (est.)	618 GB/s	570 GB/s
Sort Benchmark Daytona Rules	Yes	Yes	No
Network	dedicated data center, 10Gbps	virtualized (EC2) 10Gbps network	virtualized (EC2) 10Gbps network
Sort rate	**1.42 TB/min**	**4.27 TB/min**	**4.27 TB/min**
Sort rate/node	**0.67 GB/min**	**20.7 GB/min**	**22.5 GB/min**

（表格来源： Spark officially sets a new record in large-scale sorting ）
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：gigaom.com/2014/06/28/4）

按照Databricks的连城的说法是**One Stack To Rule Them All**

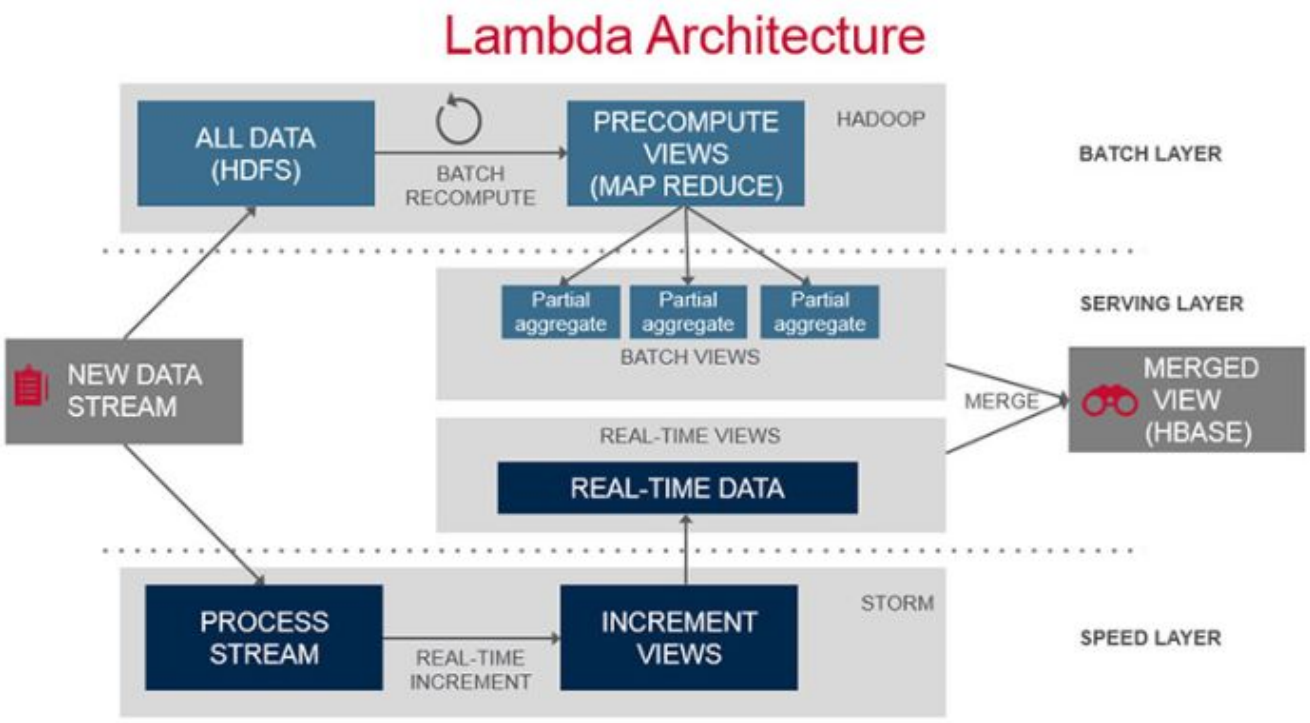
特别是在有些情况下，你需要进行一些ETL工作，然后训练一个机器学习的模型，最后进行一些查询，如果是使用Spark，你可以在一段程序中将这三部分的逻辑完成形成一个大的**有向无环图（DAG）**，而且Spark会对大的有向无环图进行整体优化。

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(“
```

（示例来源：slideshare.net/Hadoop_S）
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： 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。

编辑于 2015-03-24

Xiaoyu Ma
大数据 话题的优秀回答者

我本人是类似Hive平台的系统工程师，我对MapReduce的熟悉程度是一般，它是我的底层框架。我隔壁组在实验Spark，想将一部分计算迁移到Spark上。
年初的时候，看Spark的评价，几乎一致表示，Spark是小数据集上处理复杂迭代的交互系统，并不擅长大数据集，也没有稳定性。但是最近的风评已经变化，尤其是14年10月他们完成了Peta sort的实验，这标志着Spark越来越接近替代Hadoop MapReduce了。
Spark the fastest open source engine for sorting a petabyte
Sort和Shuffle是MapReduce上最核心的操作之一，比如上千个Mapper之后，按照Key将数据集分发到对应的Reducer上，要走一个复杂的过程，要平衡各种因素。Spark能处理Peta sort的话，本质上已经没有什么能阻止它处理Peta级别的数据了。这差不多远超大多数公司单次Job所需要处理的数据上限了。

回到本题，来说说Hadoop和Spark。Hadoop包括Yarn和HDFS以及MapReduce，说Spark代替Hadoop应该说是代替MapReduce。
MapReduce的缺陷很多，最大的缺陷之一是Map + Reduce的模型。这个模型并不适合描述复杂的数据处理过程。很多公司（包括我们）把各种奇怪的Machine Learning计算用MR模型描述，不断挖（lan）掘（yong）MR潜力，对系统工程师和Ops也是极大挑战了。很多计算，本质上并不是一个Map，Shuffle再Reduce的结构，比如我编译一个SubQuery的SQL，每个Query都做一次Group By，我可能需要Map，Reduce + Reduce，中间不希望有无用的Map；又或者我需要Join，这对MapReduce来说简直是噩梦，什么给左右表加标签，小表用Distributed Cache分发，各种不同Join的Hack，都是因为MapReduce本身是不直接支持Join的，其实我需要的是，两组不同的计算节点扫描了数据之后按照Key分发数据到下一个阶段再计算，就这么简单的规则而已；再或者我要表示一组复杂的

数据Pipeline，数据在一个无数节点组成的图上流动，而因为MapReduce的呆板模型，我必须一次一次在一个Map/Reduce步骤完成之后不必要地把数据写到磁盘上再读出，才能继续下一个节点，因为Map Reduce2个阶段完成之后，就算是一个独立计算步骤完成，必定会写到磁盘上等待下一个Map Reduce计算。
上面这些问题，算是每个号称下一代平台都尝试解决的。
现在号称次世代平台现在做的相对有前景的是Hortonworks的Tez和Databricks的Spark。他们都尝试解决了上面说的那些问题。Tez和Spark都可以很自由地描述一个Job里执行流（所谓DAG，有向无环图）。他们相对现在的MapReduce模型来说，极大的提升了对各种复杂处理的直接支持，不需要再绞尽脑汁“挖掘”MR模型的潜力。

有兴趣的童鞋可以看看这个PPT
slideshare.net/Hadoop_S
这是Hadoop峰会上Tez的材料，第九页开始有描述Hive on Tez和传统MR Hive的区别，这些区别应该也适用于MR Hive和Spark SQL，也很清楚的体现了为何MR模型很笨重。

相比Tez，Spark加入了更多内存Cache操作，但据了解它也是可以不Cache直接处理的，只是效率就会下降。

再说Programming Interface，Tez的Interface更像MapReduce，但是允许你定义各种Edge来连接不同逻辑节点。Spark则利用了Functional Programming的理念，API十分简洁，相比MR和Tez简单到令人发指。我不清楚Spark如果要表现复杂的DAG会不会也变得很麻烦，但是至少wordcount的例子看起来是这样的，大家可以比较感受下：
incubator-tez/WordCount.java at master · apache/incubator-tez · GitHub
Examples | Apache Spark

处理大规模数据而言，他们都需要更多proven cases。至少Hadoop MapReduce是被证明可行的。

作为Data Pipeline引擎来说，MapReduce每个步骤都会存盘，而Spark和Tez可以直接网络发送到下一个步骤，速度上是相差很多的，但是存盘的好处是允许继续在失败的数据上继续跑，所以直观上说MapReduce作为pipeline引擎更稳健。但理论上来说，如果选择在每个完成的小步骤上加CheckPoint，那Tez和Spark完全能和现在的MapReduce达到一样的稳健。

总结来说，即便现在不成熟，但是并没有什么阻碍他们代替现有的MapReduce Batch Process。
对Tez而言，似乎商业上宣传不如Spark成功。Databricks头顶Berkley的光环，商业宣传又十分老道，阵营增长极快。光就系统设计理念，没有太大的优劣，但是商业上可能会拉开差距。Cloudera也加入了Spark阵营，以及很多其他大小公司，可以预见的是，Spark会成熟的很快，相比Tez。
但Tez对于Hortonworks来说是赢取白富美的关键，相信为了幸福他们也必须努力打磨推广tez。
所以就算现在各家试用会有种种问题，但是毕竟现在也就出现了2个看起来有戏的“次世代”平台，那慢慢试用，不断观望，逐步替换，会是大多数公司的策略。

编辑于 2014-11-12