文章编号:1007-1423(2015)08-0055-06 DOI:10.3969/j.issn.1007-1423.2015.08.013

# 大数据处理模型 Apache Spark 研究

黎文阳

(四川大学计算机学院,成都 610065)

**摘要:**

Apache Spark 是当前流行的大数据处理模型,具有快速、通用、简单等特点。Spark 是针对 MapReduce 在迭代式机器学习算法和交互式数据挖掘等应用方面的低效率,而提出的新的内存计算框架,既保留了 MapReduce 的可扩展性、容错性、兼容性,又弥补了 MapReduce 在这些应用上的不足。由于采用基于内存的集群计算,所以 Spark 在这些应用上比 MapReduce 快 100 倍。介绍 Spark 的基本概念、组成部分、部署模式,分析 Spark 的核心内容与编程模型,给出相关的编程示例。

**关键词:**

Spark; Hadoop; MapReduce; 大数据; 数据分析

## 0 引言

MapReduce 计算模型在大规模数据分析领域已取得很大成绩,并被很多公司广泛采用。这些系统都是基于非循环的数据流模型,有很好的容错性,同时为开发人员提供了高级接口以便于编写并行程序。目前这些系统能很容易地访问集群中的计算资源,但是不能充分地利用分布式内存,导致了对那些重用中间结果的应用不是很有效。这些应用的特点是在多个并行操作之间重用数据,例如机器学习中的 PageRank 算法、K-means 聚类算法、逻辑回归算法等迭代式算法。交互式的数据挖掘算法中也经常重用数据。Spark 计算模型刚好解决了这些问题,并且能在 Hadoop 集群下部署,访问 HDFS 文件系统。Spark 将分布式内存抽象成弹性分布式数据集 (Resilient Distributed Datasets,RDD)[1]。RDD 支持基于工作集的应用,同时具有数据流模型的特点:自动容错、位置感知调度和可伸缩性。RDD 允许用户在执行多个查询时显式地将工作集缓存在内存中,以便后续的查询能够重用,这极大地提升了查询速度。

## 1 Spark 简介

Spark 是 UC Berkeley AMPLab 于 2009 年发起的,然后被 Apache 软件基金会接管的类 Hadoop MapReduce 通用性并行计算框架,是当前大数据领域最活跃的开源项目之一。Spark 是基于 MapReduce 计算框架实现的分布式计算,拥有 Hadoop MapReduce 所具有的优点;但不同于 MapReduce 的是中间输出和结果可以保存在内存中,从而不再需要读写 HDFS,因此 Spark 更适用于数据挖掘与机器学习等需要迭代的算法。如图 1 所示,逻辑回归算法在 Hadoop 和 Spark 上的运行时间对比图,可以看出 Spark 的效率有很大的提升[3]。

Spark 由 Scala[4]语言实现的,Scala 是一种基于 JVM 的函数式编程语言,提供了类似 DryadLINQ[5]的编程接口。而且 Spark 还提供了一个修改的 Scala 语言解释器,能方便地用于交互式编程,用户可以定义变量、函数、类以及 RDD。

Spark 集成了丰富的编程工具,其中 Spark SQL 用于 SQL 语言和结构化数据处理,Spark Streaming 用于流处理,MLlib 用于机器学习算法,GraphX 用于图处理。Spark 不但能够访问多种数据源,例如 HDFS、Cassandra、HBase、Amazon S3,还提供了 Scala、Java 和 Python 三种语言的 API 接口,以便于编写并行程序。而且 Spark 还能部署在已有的 Hadoop 系统上,由 YARN 进行集群调度,极大地利用了 Hadoop 系统。

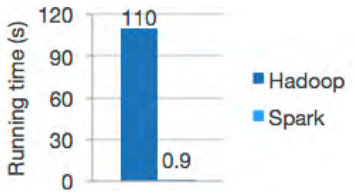


图 1 逻辑回归算法在 Hadoop 和 Spark 上的运行时间

### 1.1 适用场景

Spark 适用于那些在多个并行操作之间重用数据的应用，而 MapReduce 在这方面效率并不高，因为 MapReduce 和 DAG 引擎是基于非循环数据流的，即一个应用被分成一些不同的作业(job)，每个作业从磁盘中读数据，然后再写到磁盘[2]。Spark 不太适合那些异步更新共享状态的应用，例如并行 Web 爬行器。Spark 的适用场景有：

●迭代式算法：许多机器学习算法都用一个函数对相同的数据进行重复的计算，从而得到最优解。MapReduce 计算框架把每次迭代看成是一个 MapReduce 作业 (job)，而每个作业都要从磁盘重新加载数据，这就导致了效率不高，而 Spark 可以把中间数据缓存到内存中加快计算效率。

●交互式数据分析：用户经常会用 SQL 对大数据集合做临时查询(Ad-Hoc Query)。Hive 把每次查询都当作一个独立的 MapReduce 作业，并且从磁盘加载数据，有很大的延迟，而 Spark 可以把数据加载到内存中，然后重复的查询。

●流应用：即需要实时处理的应用，这类应用往往需要低延迟，高效率。

### 1.2 组成部分

目前 Spark 由四部分构成：Spark SQL、MLlib、GraphX、Spark Streaming，如图 2 所示。

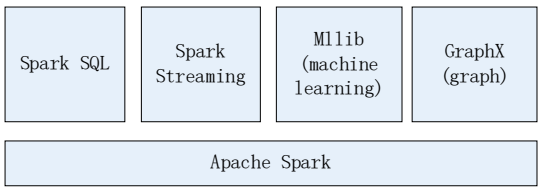


图 2 Spark 组成部分

(1)Spark SQL：是 Spark 处理 SQL 和结构化数据工具，Spark 引入了 SchemaRDD 的数据抽象，使其能以统一地、高效地访问和查询各种不同的数据源，例如 Apache Hive 表、parquet 文件、JSON 文件。Spark SQL 兼容 Apache Hive，能重用 Hive 的前端和元存储。Spark SQL API 能像查询 RDD 一样查询结构化的数据，并且 Spark SQL 还提供了 JDBC/ODBC 的服务端模式，以便建立 JDBC/ODBC 数据连接。

(2)MLlib (Machine Learning)：是 Spark 提供的机器学习库，包含了常见的机器学习算法。Spark 擅长于迭代式计算，所以与 MapReduce 相比，MLlib 中的算法效率更高，性能更好。常见的算法有：

①SVM、逻辑回归(logistic regression)、线性回归(linear regression)、朴素贝叶斯(naive Bayes)

②K 均值算法(K-means)

③奇异值分解(singular value decomposition)

④特征提取与转换(feature extraction and transformation)

(3)GraphX (Graph Processing)：是 Spark 处理图(graph)的框架，利用 Pregel API 可以用 RDD 有效地转换(transform)和连接(join)图，实现图算法。GraphX 在速度上可与最快的专用图处理系统相媲美。

(4)Spark Streaming：是 Spark 处理流应用的库。其结合了批处理查询与交互式查询，方便重用批处理的代码和历史数据。基本原理是 Spark 将流数据分成小的时间片断(几秒)，以类似批处理的方式来处理这小部分数据。Spark Streaming API 能像编写批处理作业一样构建可扩展的流应用。Spark Streaming 也能访问各种不同的数据源，例如能够从 HDFS、Flume、Kafka、Twitter 和 ZeroMQ 中读取数据。

### 1.3 部署模式

Spark 集群如图 3 划分，主要有驱动程序、集群管理程序，以及各 worker 节点上的执行程序[3]。

Spark 应用的主程序称为驱动程序 (driver program)，其中包含 SparkContext 对象，用于连接集群管理程序(cluster manager)。集群管理程序用于分配集群中的资源，目前有三种：Spark 独自的集群管理程序、Mesos[6]和 YARN[7]。Spark 应用的运行过程是，SparkContext 对象首先连接集群管理程序，然后 Spark 获取集群中各个节点上的执行程序(executor)，执行程序是

用于计算和存储数据的进程，然后 Spark 把代码发送到执行程序，最后运行执行程序上的任务。所以 Spark 目前支持 3 种集群部署模式：Standalone 模式、Apache Mesos 模式、Hadoop YARN 模式。

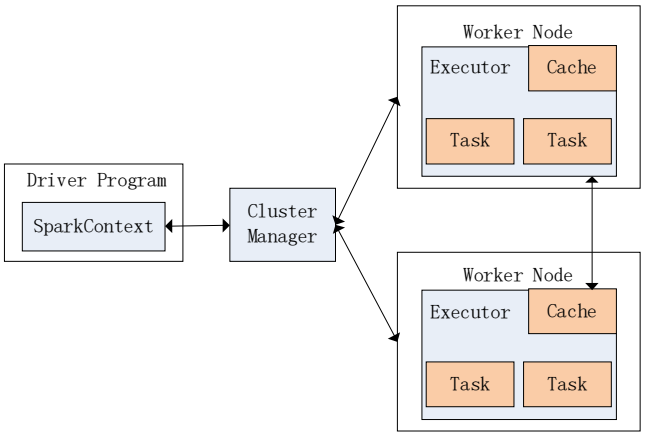


图 3 Spark 集群组成部分

(1)Standalone 模式：即独立模式，使用 Spark 自带的集群管理程序，好处是不需要额外的软件就能运行，配置简单。

(2)Apache Mesos 模式：需要安装 Mesos 资源管理程序，Spark 的集群管理就交给 Mesos 处理了。使用 Mesos 的好处是，可以在 Spark 与其他框架之间或多个 Spark 实例之间动态地划分。

(3)Hadoop YARN 模式：利用已有的 Hadoop 集群，让 Spark 在 Hadoop 集群中运行，访问 HDFS 文件系统，使用 YARN 资源调度程序。Spark on YARN 是 Spark 0.6.0 版本加入的，该模式需要额外的配置参数。

Spark 既可以在单机上运行，也可以在 Amazon EC2 上运行。Spark 提供了相应的配置文件、启动脚本、结束脚本用于配置、启动、结束 Spark 集群中的 master 和 slave。也提供了相应的 Web 监控系统与日志系统，方便地监控与调试程序。而且还可以配置 Zookeeper 以保证高可用性。

## 2 Spark 编程模型

Spark 最主要的抽象就是弹性分布式数据集(Resilient Distributed Datasets，RDD)以及对 RDD 的并行操作(例如 map、filter、groupBy、join)。而且，Spark 还支持两种受限的共享变量 (shared variables)：广播变量(broadcast variables)和累加变量(accumulators)。

### 2.1 RDD

RDD 是只读的对象集合，RDD 分区分布在集群的节点中。如果某个节点失效，或者某部分数据丢失，RDD 都能重新构建。Spark 将创建 RDD 的一系列转换记录下来，以便恢复丢失的分区，这称为血系(lineage)。每次对 RDD 数据集的操作之后的结果，都可以缓存到内存中，下一个操作可以直接从内存中输入，省去了 MapReduce 大量的磁盘操作。RDD 只支持粗粒度转换，即在大量记录上执行的单个操作。虽然只支持粗粒度转换限制了编程模型，但 RDD 仍然可以很好地适用于很多应用，特别是支持数据并行的批量分析应用，包括数据挖掘、机器学习、图算法等，因为这些程序通常都会在很多记录上执行相同的操作。

使用 RDD 的好处有：

●RDD 只能从持久存储或通过转换(transformation)操作产生，相比于分布式共享内存(DSM)可以更高效地实现容错，对于丢失部分数据分区只需根据它的血系就可重新计算出来，而不需要做特定的检查点(checkpoint)。

●RDD 的不变性，可以实现类 MapReduce 的预测式执行。

●RDD 的数据分区特性，可以通过数据的本地性来提高性能，这与 MapReduce 是一样的。

●RDD 是可序列化的，当内存不足时可自动改为磁盘存储，把 RDD 存储于磁盘上，此时性能会有大的下降但不会差于现有的 MapReduce。

在 Spark 中，RDD 是一个 Scala 对象，对 RDD 的并行操作即是调用对象上的方法。有四种方法创建一个 RDD：

(1)通过一个文件系统中的文件创建，例如常见的 HDFS 文件。

(2)通过并行化 Scala 集合，即把一个集合切分成很多片，然后发送到各种节点。

(3)通过对已有的 RDD 执行转换操作，可以得到一个新的 RDD。例如通过 flatMap 可以把类型 1 的 RDD 转换成类型 2 的 RDD。

(4)通过把 RDD 持久化。RDD 默认是惰性的，即

只有当 RDD 在执行并行操作时,RDD 才被物化，执行完后即被释放。用户可以通过显式的 cache 或 save 操作使 RDD 持久化。

### 2.2 并行操作

作用在 RDD 上的并行操作有两种:转换(transformation)和动作(action),转换返回一个新的 RDD,动作返回一个值或把 RDD 写到文件系统中。转换是惰性的，即从一个 RDD 转换成另一个 RDD 不是马上执行的,Spark 只是记录这样的操作,并不执行,等到有动作操作时才会启动计算过程。常见的转换(transformation)如表 1 所示,常见的动作(action)如表 2 所示。

表 1 常见的转换(transformation)操作

| 函数名 | 含义 |
|---|---|
| map(func) | 对源 RDD 的每个元素执行 func 函数，得到一个新的 RDD |
| filter(func) | 选择满足条件函数 func 的源 RDD 元素，组成一个新的 RDD |
| flatMap(func) | 与 map 类似，但是对每个源 RDD 元素可以产生多个新的 RDD 元素 |
| sample(withReplacement, fraction, seed) | 根据随机数种子 seed，取 RDD 中的某一部分组成新的 RDD |
| groupByKey([numTasks]) | 根据 key 值，对<K, V>形式的 RDD 执行聚集操作，得到新的 RDD |
| reduceByKey(func, [numTasks]) | 根据 key 值，先对<K, V>形式的 RDD 执行聚集操作，然后对相同的 key 执行规约操作得到一个值，结果是值的 RDD |
| sortByKey([ascending], [numTasks]) | 根据 key 值排序 |
| join(otherDataset, [numTasks]) | 对<K, V1>RDD 和<K, V2>RDD 执行 join 操作，得到<K, <V1, V2>>RDD |
| union(otherDataset) | 对两个 RDD 执行并操作，即取两个 RDD 共有的元素 |
| distinct([numTasks])) | 去除 RDD 的重复元素 |

注:有些操作只对 key 有效,例如 join、groupByKey,reduceByKey。除了这些操作以外,用户还可以请求将 RDD 缓存起来。而且,用户还可以通过 Partitioner 类获取 RDD 的分区顺序，然后将另一个 RDD 按照同样的方式分区。

### 2.3 共享变量

通常情况下,Spark 中的 map、filter、reduce 等函数的参数是一个函数(闭包),运行时这些函数参数被复制到各个 worker 节点上,互不干扰。Spark 还提供了共享变量用于其他用途,常见的有两种:

(1)广播变量(broadcast variables):对于大量的只读数据,当有多个并行操作时,最好只复制一次而不是每执行一次函数就复制一次到各个 worker 节点。广播变量就是用于这种情况，它只是包装了一下原有的数据,然后只复制一次到各 worker 节点。

(2)累加变量(accumulators):累加变量只能用于关联操作,并且只有驱动程序才能读取。只要某个类型有“add”操作和“0”值都可以是累加变量。累加变量经常用于实现 MapReduce 的计数器,而且由于是只加性的,所以很容易实现容错性。

表 2 常见的动作(action)操作

| 函数名 | 含义 |
|---|---|
| reduce(func) | 对 RDD 的元素执行规约操作 func，得到一个值，要求 RDD 的元素可计算 |
| collect() | 把 RDD 作为数组返回 |
| count() | 返回 RDD 中元素个数 |
| takeSample(withReplacement, num, [seed]) | 返回 num 个随机 RDD 元素组成的数组 |
| takeOrdered(n, [ordering]) | 返回排好序的 RDD 的前 n 个元素 |
| saveAsTextFile(path) | 把 RDD 元素写到文本文件中，path 既可以是本地文件，也可以是 HDFS 文件 |
| saveAsObjectFile(path) | 把 RDD 元素序列化后写到文件中，path 既可以是本地文件，也可以是 HDFS 文件 |
| foreach(func) | 对每个 RDD 元素执行 func 函数 |

## 3 编程示例

Spark 提供了 Scala、Java 和 Python 三种语言的 API。而且 Spark 程序既可以通过交互式 Spark shell 运行 (Scala 或 Python 语言)，又能以独立的程序运行(Scala、Java 或 Python 语言)。下面这些编程示例使用 Scala 语言,在 Spark shell 下执行[8]。Scala 是一种基于 JVM 的函数式编程语言。

Spark 的首要抽象便是弹性分布式数据集 RDD,所以编程的主要任务就是编写驱动程序,创建 RDD,然后对 RDD 执行并行操作。RDD 既能从外部文件中创建(例如 HDFS 文件),又能从对其他 RDD 执行转换操作(transformation)得到。对 RDD 有两种操作:一种是转换操作,产生新的 RDD;另一种是动作操作(action),开始一个作业并返回值。首先配置好 Spark 运行环境,然后启动 Spark 集群,在此不再细述。

### 3.1 文本搜索

本示例搜索某个 HDFS 日志文件的 ERROR 信息:

```
// Create a new RDD from the HDFS file
var logFile = sc.textFile("hdfs://user/hduser/test.log")
// Use the filter transformation to return a new RDD with a subset of the items in the file
var errors = file.filter(line => line.contains("ERROR"))
```

```
// Count all the errors
errors.count()
// Count errors containing Spark
errors.filter(line => line.contains("Spark")).count()
// Fetch the Spark errors as an array of strings
errors.filter(line => line.contains("Spark")).collect()
```

Spark 还能显示的缓存 RDD，只需执行 cache 操作：

```
errors.cache()
```

### 3.2 单词统计

本示例统计某个 HDFS 文件的各个单词出现的次数，并将结果保存到 HDFS 文件：

```
// Create a new RDD
val wordFile = sc.textFile("hdfs://user/hduser/test.txt")
// Split the string with " " and count the key
val wordCount = wordFile.flatMap(line => line.split(" ")).
map(word => (word, 1)).reduceByKey(_ + _)
// Save the result to HDFS
wordCount.saveAsTextFile("hdfs://user/hduser/result.txt")
```

### 3.3 估算 PI 值

Spark 也用于计算密集型任务，本示例使用“扔飞镖法”估算 PI 值，在(0, 0) – (1, 1)的正方形中，随机生成坐标(x, y)，统计落在圆内的点数，那么落在圆内的点数/总点数等于 PI/4：

```
val count = sc.parallelize(1 to 999999).map{i =>
val x = Math.random()
val y = Math.random()
if (x*x + y*y < 1) 1 else 0
}.reduce(_ + _)
println("PI is roughly" + 4.0 * count/999999)
```

## 4 结语

本文大致介绍了 Spark 系统的基本概念与核心思想，并给出了编程示例。Spark 最重要的抽象就是 RDD，一种有效的、通用的分布式内存抽象，它解决了集群环境下并行处理大数据的效率问题，比 Hadoop MapReduce 的效率高，特别适用于机器学习中的迭代式算法和交互式数据分析等特殊的应用场景。目前，Spark 是非常流行的内存计算框架，一直在发布新版本，还处于比较活跃的开发阶段。当前遇到的技术挑战有：①资源调度程序如何为 Spark 作业确定合适的资源需求；②Spark 如何更好地与 YARN 集群管理程序配合，使系统最优；③提供更多的编程 API 供用户使用。


**参考文献：**

[1]Zaharia M, Chowdhury M, Das T, et al. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for in-Memory Cluster Computing [C]. Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation. USENIX Association, 2012: 2-2

[2]Zaharia M, Chowdhury M, Franklin M J, et al. Spark: Cluster Computing with Working Sets[C]. Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing, 2010: 10~10

[3]Spark[EB/OL]. http://spark.apache.org

[4]Scala[EB/OL]. https://www.scala-lang.org

[5]Yu Y, Isard M, Fetterly D, et al. DryadLINQ: A System for General-Purpose Distributed Data-Parallel Computing Using a High-Level Language[C]. OSDI. 2008, 8: 1·14

[6]Hadoop MapReduce Tutorial[EB/OL]. http://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html

[7]Apache Mesos. http://mesos.apache.org

[8]Spark Programming Guides[EB/OL]. http://spark.apache.org/docs/1.1.0/quick-start.html



**作者简介：**

**黎文阳**(1990-)，**男**，**河南信阳人**，**硕士研究生**，**研究方向为分布式与数据库**



**收稿日期**：2015-02-10　　**修稿日期**：2015-02-28

# Research on Apache Spark for Big Data Processing

LI Wen-yang

(College of Computer Science, Sichuan University, Chengdu 610065)

Abstract:
Apache Spark is a popular model for large scale data processing at present, which is fast, general and easy. Compared with the MapReduce computing framework, Spark is efficient in iterative machine learning algorithms and interactive data mining applications while retaining the compatibility, scalability and fault-tolerance of MapReduce. With its in-memory computing, Spark is up to 100x faster than Hadoop MapReduce in memory. Presents the basic conception, component and the deploying mode of Spark, introduces the internal abstraction and the programming model, gives the programming examples.

Keywords:
Spark; Hadoop; MapReduce; Big Data; Data Analysis

(上接第 54 页)

# Basic Concept, Technology and Challenge of Big Data

ZHAO Su-yang, LI Yan-jun, QIAN Xiao-yan, CAO Yu-yuan, XU Zhen-teng, QIAO Lei, WANG Lei

(Nanjing University of Aeronautics & Astronautics, Nanjing 210000)

Abstract:
With the development of cloud computing, Internet of things, social networks, the number and types of data showing explosive growth, the big data era is coming. People found that the data can be used as a fundamental resource and not just a simple processing object. The complexity of data complexity, large data calculation complexity and data processing systems are a great challenge for calculation and application of large data. Analysing the basic concept, features of the data processing mode and technical difficulties is helpful to better tap the potential and advantages of large data.

Keywords:
Big Data; Basic Concept; Processing Mode; Problems and Challenges