第一篇　基础篇

第 1 章

Spark 及其生态圈概述

1.1　Spark 简介

1.1.1　什么是 Spark

Spark 是加州大学伯克利分校 AMP 实验室（Algorithms、Machines and People Lab）开发的通用大数据处理框架。Spark 生态系统也称为 BDAS，是伯克利 APM 实验室所开发的，力图在算法（Algorithms）、机器（Machines）和人（People）三者之间通过大规模集成来展现大数据应用的一个开源平台。AMP 实验室运用大数据、云计算等各种资源以及各种灵活的技术方案，对海量的数据进行分析并转化为有用的信息，让人们更好地了解世界。

Spark 在 2013 年 6 月进入 Apache 成为孵化项目，8 个月后成为 Apache 顶级项目，速度之快足见过人之处。Spark 以其先进的设计理念，迅速成为社区的热门项目，围绕着 Spark 推出了 Spark SQL、Spark Streaming、MLlib、GraphX 和 SparkR 等组件，这些组件逐渐形成大数据处理一站式解决平台。Spark 并非池鱼，它的志向不是作为 Hadoop 的绿叶，而是期望替代 Hadoop 在大数据中的地位，成为大数据处理的主流标准。

Spark 使用 Scala 语言进行实现，它是一种面向对象、函数式编程语言，能够像操作本地集合对象一样轻松地操作分布式数据集。Spark 具有运行速度快、易用性好、通用性强和随处运行等特点。

1．运行速度快

Spark 的中文意思是“电光火石”，Spark 确实如此！官方提供的数据表明，如果数据由磁盘读取，速度是 Hadoop MapReduce 的 10 倍以上；如果数据从内存中读取，速度可以高达 100

多倍。图 1-1 是在逻辑回归算法中 Hadoop 与 Spark 处理时间的比较，左边是 Hadoop，耗时 110 秒，而 Spark 耗时 0.9 秒。

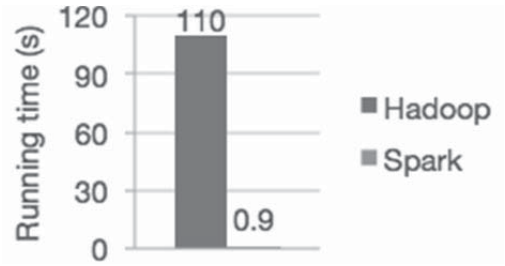

图 1-1　逻辑回归算法在 Hadoop 和 Spark 上处理时间的比较

Spark 相对于 Hadoop 有如此快的计算速度有数据本地性、调度优化和传输优化等原因，其中最主要的是基于内存计算和引入 DAG 执行引擎。

（1）Spark 默认情况下迭代过程的数据保存到内存中，后续的运行作业利用这些结果进行计算，而 Hadoop 每次计算结果都直接存储到磁盘中，在随后的计算中需要从磁盘中读取上次计算的结果。由于从内存读取数据时间比磁盘读取时间低两个数量级，这就造成了 Hadoop 的运行速度较慢，这种情况在迭代计算中尤为明显。

（2）由于较复杂的数据计算任务需要多个步骤才能实现，且步骤之间具有依赖性。对于这些步骤之间，Hadoop 需要借助 Oozie 等工具进行处理。而 Spark 在执行任务前，可以将这些步骤根据依赖关系形成 DAG 图（有向无环图），任务执行可以按图索骥，不需要人工干预，从而优化了计算路径，大大减少了 I/O 读取操作。

2．易用性好

Spark 不仅支持 Scala 编写应用程序，而且支持 Java 和 Python 等语言进行编写。Scala 是一种高效、可拓展的语言，能够用简洁的代码处理较为复杂的处理工作。比如经典的 WordCount 例子，使用 Scala 编写，仅用简单两条语句就能够实现。具体代码如下：

```
scala> val textFile = sc.textFile("file:///home/hadoop/README.md")
scala> val counts = textFile.flatMap(line => line.split(" ")).map(word => (word,
  1)).reduceByKey(_ + _)
```

3．通用性强

Spark 生态圈即 BDAS（伯克利数据分析栈）所包含的组件：Spark Core 提供内存计算框架、SparkStreaming 的实时处理应用、Spark SQL 的即席查询、MLlib 的机器学习和 GraphX 的图处理，它们都是由 AMP 实验室提供，能够无缝地集成，并提供一站式解决平台，如图 1-2 所示。

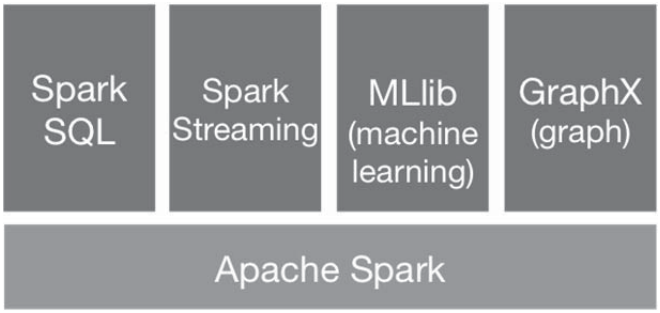

图 1-2　Spark 技术堆栈

4．随处运行

Spark 具有很强的适应性，能够读取 HDFS、Cassandra、HBase、S3 和 Tachyon，为持久层读写原生数据，能够以 Mesos、YARN 和自身携带的 Standalone 作为资源管理器调度作业来完成 Spark 应用程序的计算。图 1-3 是 Spark 支持的技术框架。

图 1-3　Spark 支持的技术框架

1.1.2　Spark 与 MapReduce 比较

Spark 是通过借鉴 Hadoop MapReduce 发展而来的，继承了其分布式并行计算的优点，并改进了 MapReduce 明显的缺陷，具体体现在以下几个方面。

（1）Spark 把中间数据放在内存中，迭代运算效率高。MapReduce 中的计算结果是保存在磁盘上，这样势必会影响整体的运行速度，而 Spark 支持 DAG 图的分布式并行计算的编程框架，

减少了迭代过程中数据的落地，提高了处理效率。

（2）Spark 的容错性高。Spark 引进了弹性分布式数据集（Resilient Distributed Dataset，RDD）的概念，它是分布在一组节点中的只读对象集合，这些集合是弹性的，如果数据集一部分丢失，则可以根据“血统”（即允许基于数据衍生过程）对它们进行重建。另外，在 RDD 计算时可以通过 CheckPoint 来实现容错，而 CheckPoint 有两种方式，即 CheckPoint Data 和 Logging The Updates，用户可以控制采用哪种方式来实现容错。

（3）Spark 更加通用。不像 Hadoop 只提供了 Map 和 Reduce 两种操作，Spark 提供的数据集操作类型有很多种，大致分为转换操作和行动操作两大类。转换操作包括 Map、Filter、FlatMap、Sample、GroupByKey、ReduceByKey、Union、Join、Cogroup、MapValues、Sort 和 PartionBy 等多种操作类型，行动操作包括 Collect、Reduce、Lookup 和 Save 等操作类型。另外，各个处理节点之间的通信模型不再像 Hadoop 只有 Shuffle 一种模式，用户可以命名、物化，控制中间结果的存储、分区等。

1.1.3 Spark 的演进路线图

Spark 由 Lester 和 Matei 在 2009 年算法比赛的思想碰撞中诞生，随后 4 年中，Spark 在 Berkeley's AMPLab 逐渐形成了现有的 Spark 雏形。Spark 在 2013 年 6 月进入 Apache 成为孵化项目，8 个月后成为 Apache 顶级项目，从此 Spark 的发展进入了快车道。2014 年 5 月底发布了第一个正式版本 Spark 1.0.0，在随后的时间里大致以 3 个月为周期发布一个小版本，并经过两年沉淀在 2016 年 7 月推出了 Spark 2.0 正式版本，其具体演进时间如下：

- 2009 年由 Berkeley's AMPLab 开始编写最初的源代码
- 2010 年开放源代码
- 2012 年 2 月发布 0.6.0 版本
- 2013 年 6 月进入 Apache 孵化器项目
- 2013 年年中 Spark 主要成员创立 Databricks 公司
- 2014 年 2 月成为 Apache 的顶级项目（8 个月的时间）
- 2014 年 5 月底 Spark 1.0.0 发布
- 2014 年 9 月 Spark 1.1.0 发布
- 2014 年 12 月 Spark 1.2.0 发布
- 2015 年 3 月 Spark 1.3.0 发布
- 2015 年 6 月 Spark 1.4.0 发布
- 2015 年 9 月 Spark 1.5.0 发布

- 2016 年 1 月 Spark 1.6.0 发布
- 2016 年 5 月 Spark 2.0.0 Preview 版本发布
- 2016 年 7 月 Spark 2.0.0 正式版本发布

图 1-4 为 Spark 的演进时间轴。

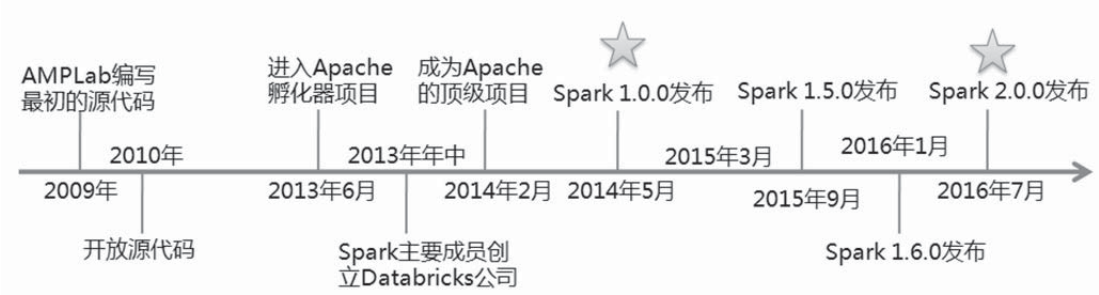

图 1-4　Spark 演进时间轴

Spark 进入 Apache 后以其代码开源、内存计算和一栈式解决方案风靡大数据生态圈，成为该生态圈和 Apache 基金会内最活跃的项目，得到了大数据研究人员、机构和众多厂商的支持。

- Spark 成为整个大数据生态圈和 Apache 基金会内最活跃的项目。
- Hadoop 最大的厂商 Cloudera 宣称加大 Spark 框架的投入来取代 MapReduce。
- Hortonworks 加大 Hadoop 与 Spark 整合。
- Hadoop 厂商 MapR 投入 Spark 阵营。
- Apache Mahout 放弃 MapReduce，将使用 Spark 作为后续算子的计算平台。

……

1.2　Spark 生态系统

Spark 生态系统以 Spark Core 为核心，能够读取传统文件（如文本文件）、HDFS、Amazon S3、Alluxio 和 NoSQL 等数据源，利用 Standalone、YARN 和 Mesos 等资源调度管理，完成应用程序分析与处理。这些应用程序来自 Spark 的不同组件，如 Spark Shell 或 Spark Submit 交互式批处理方式、Spark Streaming 的实时流处理应用、Spark SQL 的即席查询、采样近似查询引擎 BlinkDB 的权衡查询、MLbase/MLlib 的机器学习、GraphX 的图处理和 SparkR 的数学计算等，如图 1-5 所示，正是这个生态系统实现了“One Stack to Rule Them All”目标。

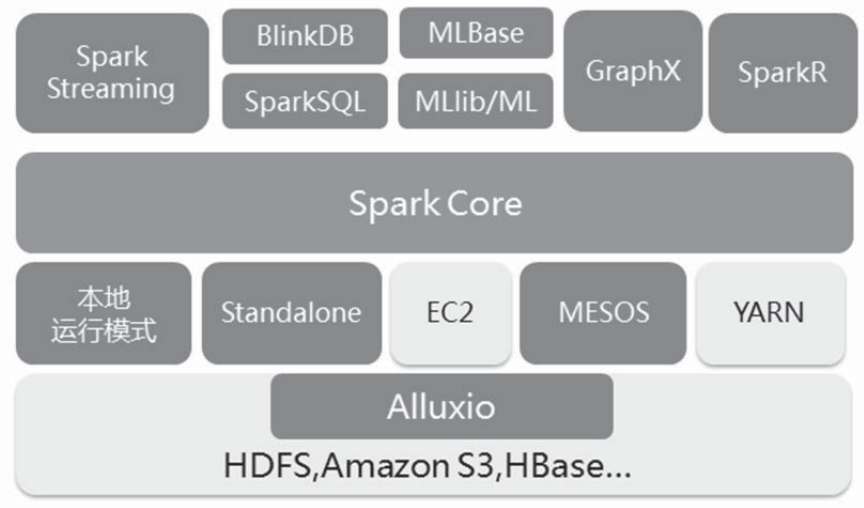

图 1-5　Spark 生态系统

1.2.1　Spark Core

Spark Core 是整个 BDAS 生态系统的核心组件，是一个分布式大数据处理框架。Spark Core 提供了多种资源调度管理，通过内存计算、有向无环图（DAG）等机制保证分布式计算的快速，并引入了 RDD 的抽象保证数据的高容错性，其重要特性描述如下。

- Spark Core 提供了多种运行模式，不仅可以使用自身运行模式处理任务，如本地模式、Standalone，而且可以使用第三方资源调度框架来处理任务，如 YARN、MESOS 等。相比较而言，第三方资源调度框架能够更细粒度管理资源。
- Spark Core 提供了有向无环图（DAG）的分布式并行计算框架，并提供内存机制来支持多次迭代计算或者数据共享，大大减少迭代计算之间读取数据的开销，这对于需要进行多次迭代的数据挖掘和分析性能有极大提升。另外，在任务处理过程中移动计算而非移动数据，RDD Partition 可以就近读取分布式文件系统中的数据块到各个节点内存中进行计算。
- 在 Spark 中引入了 RDD 的抽象，它是分布在一组节点中的只读对象集合，这些集合是弹性的，如果数据集一部分丢失，则可以根据“血统”对它们进行重建，保证了数据的高容错性。

1.2.2　Spark Streaming

Spark Streaming 是一个对实时数据流进行高吞吐、高容错的流式处理系统，可以对多种数据源（如 Kafka、Flume、Twitter 和 ZeroMQ 等）进行类似 Map、Reduce 和 Join 等复杂操作，并将结果保存到外部文件系统、数据库或应用到实时仪表盘，如图 1-6 所示。相比其他的处理引擎要么只专注于流处理，要么只负责批处理（仅提供需要外部实现的流处理 API 接口），而 Spark Streaming 最大的优势是提供的处理引擎和 RDD 编程模型可以同时进行批处理与流处理。

图 1-6　Spark Streaming 的输入/输出类型

对于传统流处理中一次处理一条记录的方式而言，Spark Streaming 使用的是将流数据离散化处理（Discretized Streams），通过该处理方式能够进行秒级以下的数据批处理。在 Spark Streaming 处理过程中，Receiver 并行接收数据，并将数据缓存至 Spark 工作节点的内存中。经过延迟优化后，Spark 引擎对短任务（几十毫秒）能够进行批处理，并且可将结果输出至其他系统中。与传统连续算子模型不同，其模型是静态分配给一个节点进行计算，而 Spark 可基于数据的来源以及可用资源情况动态分配给工作节点，如图 1-7 所示。

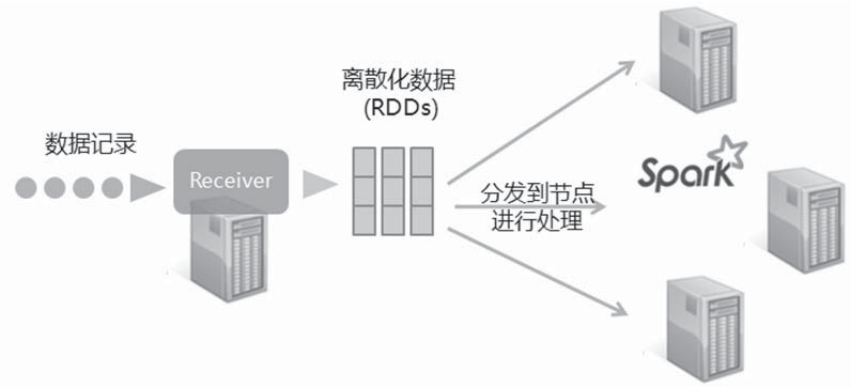

图 1-7　Spark Streaming 处理架构

使用离散化流数据（DStreaming），Spark Streaming 将具有如下特性。

- **动态负载均衡**：Spark Streaming 将数据划分为小批量，通过这种方式可以实现对资源更细粒度的分配。例如，传统实时流记录处理系统在输入数据流以键值进行分区处理情况下，如果一个节点计算压力较大超出了负荷，该节点将成为瓶颈，进而拖慢整个系统的处理速度。而在 Spark Streaming 中，作业任务将会动态地平衡分配给各个节点，如图 1-8 所示，即如果任务处理时间较长，分配的任务数量将少些；如果任务处理时间较短，则分配的任务数据将更多些。

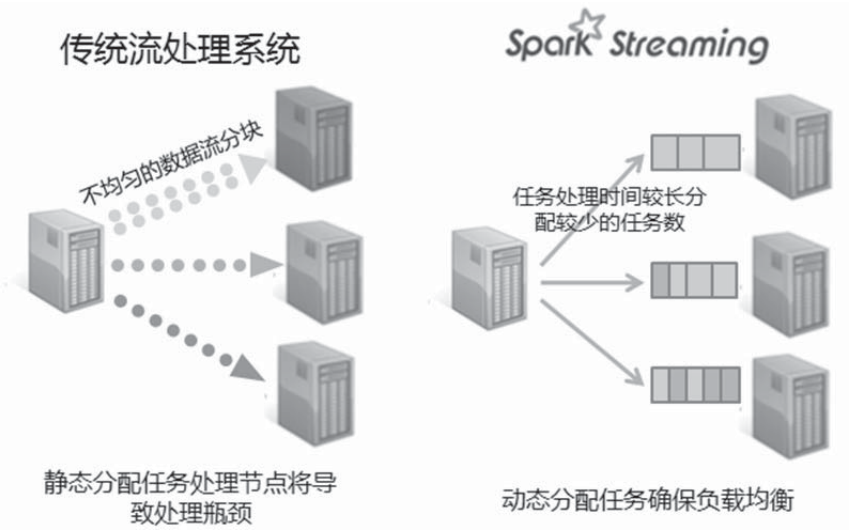

图 1-8　动态负载均衡

- **快速故障恢复机制**：在节点出现故障的情况下，传统流处理系统会在其他的节点上重启失败的连续算子，并可能重新运行先前数据流处理操作获取部分丢失数据。在此过程中只有该节点重新处理失败的过程，只有在新节点完成故障前所有计算后，整个系统才能够处理其他任务。在 Spark 中，计算将分成许多小的任务，保证能在任何节点运行后能够正确进行合并。因此，在某节点出现的故障的情况，这个节点的任务将均匀地分散到集群中的节点进行计算，相对于传递故障恢复机制能够更快地恢复，如图 1-9 所示。

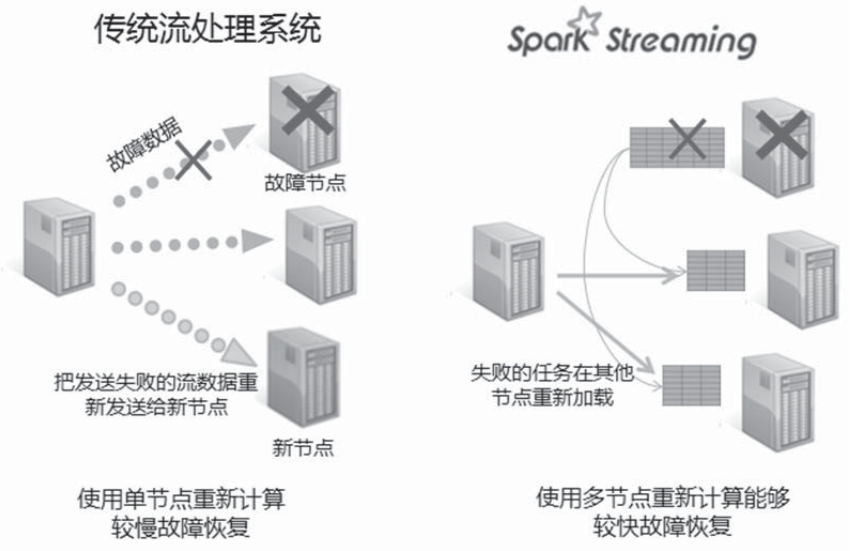

图 1-9　快速故障恢复机制

- **批处理、流处理与交互式分析的一体化**：Spark Streaming 是将流式计算分解成一系列短小的批处理作业，也就是把 Spark Streaming 的输入数据按照批处理大小（如几秒）分成一段一段的离散数据流（DStream），每一段数据都转换成 Spark 中的 RDD，然后将 Spark Streaming 中对 DStream 流处理操作变为针对 Spark 中对 RDD 的批处理操作。另外，流数据都储存在 Spark 节点的内存里，用户便能根据所需进行交互查询。正是利用了 Spark 这种工作机制将批处理、流处理与交互式工作结合在一起。

1.2.3　Spark SQL

Spark SQL 的前身是 Shark，它发布时 Hive 可以说是 SQL on Hadoop 的唯一选择（Hive 负责将 SQL 编译成可扩展的 MapReduce 作业），鉴于 Hive 的性能以及与 Spark 的兼容，Shark 由此而生。

Shark 即 Hive on Spark，本质上是通过 Hive 的 HQL 进行解析，把 HQL 翻译成 Spark 上对应的 RDD 操作，然后通过 Hive 的 Metadata 获取数据库里的表信息，实际为 HDFS 上的数据和文件，最后由 Shark 获取并放到 Spark 上运算。Shark 的最大特性就是速度快，能与 Hive 的完全兼容，并且可以在 Shell 模式下使用 rdd2sql 这样的 API，把 HQL 得到的结果集继续在 Scala 环境下运算，支持用户编写简单的机器学习或简单分析处理函数，对 HQL 结果进一步分析计算。

在 2014 年 7 月 1 日的 Spark Summit 上，Databricks 宣布终止对 Shark 的开发，将重点放到

Spark SQL 上。在此次会议上，Databricks 表示，Shark 更多是对 Hive 的改造，替换了 Hive 的物理执行引擎，使之有一个较快的处理速度。然而，不容忽视的是，Shark 继承了大量的 Hive 代码，因此给优化和维护带来大量的麻烦。随着性能优化和先进分析整合的进一步加深，基于 MapReduce 设计的部分无疑成为了整个项目的瓶颈。因此，为了更好的发展，给用户提供一个更好的体验，Databricks 宣布终止 Shark 项目，从而将更多的精力放到 Spark SQL 上。

Spark SQL 允许开发人员直接处理 RDD，同时也可查询在 Hive 上存在的外部数据。Spark SQL 的一个重要特点是能够统一处理关系表和 RDD，使得开发人员可以轻松地使用 SQL 命令进行外部查询，同时进行更复杂的数据分析。

Spark SQL 的特点如下。

- 引入了新的 RDD 类型 SchemaRDD，可以像传统数据库定义表一样来定义 SchemaRDD。SchemaRDD 由定义了列数据类型的行对象构成。SchemaRDD 既可以从 RDD 转换过来，也可以从 Parquet 文件读入，还可以使用 HiveQL 从 Hive 中获取。
- 内嵌了 Catalyst 查询优化框架，在把 SQL 解析成逻辑执行计划之后，利用 Catalyst 包里的一些类和接口，执行了一些简单的执行计划优化，最后变成 RDD 的计算。
- 在应用程序中可以混合使用不同来源的数据，如可以将来自 HiveQL 的数据和来自 SQL 的数据进行 Join 操作。

Shark 的出现使得 SQL-on-Hadoop 的性能比 Hive 有了 10～100 倍的提高，那么，摆脱了 Hive 的限制，Spark SQL 的性能又有怎么样的表现呢？虽然没有 Shark 相对于 Hive 那样瞩目的性能提升，但也表现得优异，如图 1-10 所示（其中，右侧数据为 Spark SQL）。

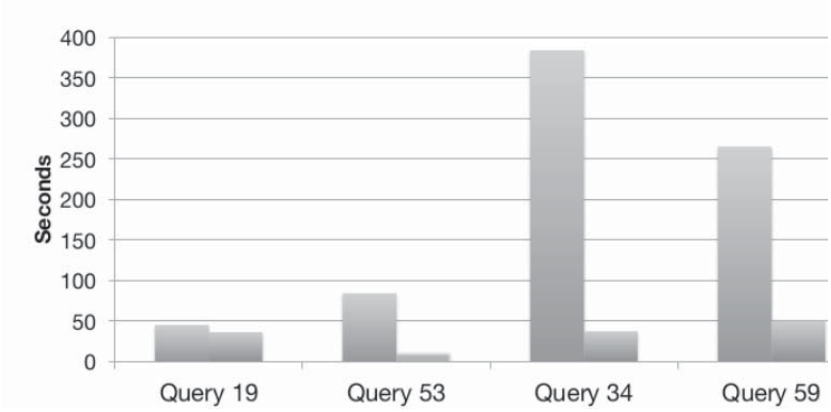

图 1-10　Shark 与 Spark SQL 处理速度的比较

为什么 Spark SQL 的性能会得到这么大的提升呢？主要是 Spark SQL 在以下几点做了优化。

- 内存列存储（In-Memory Columnar Storage）：Spark SQL 的表数据在内存中存储不是采用原生态的 JVM 对象存储方式，而是采用内存列存储。
- 字节码生成技术（Bytecode Generation）：Spark 1.1.0 在 Catalyst 模块的 Expressions 增加了 Codegen 模块，使用动态字节码生成技术，对匹配的表达式采用特定的代码动态编译。另外对 SQL 表达式都做了 CG 优化。CG 优化的实现主要还是依靠 Scala 2.10 运行时的反射机制（Runtime Reflection）。
- Scala 代码优化：Spark SQL 在使用 Scala 编写代码的时候，尽量避免低效的、容易 GC 的代码；尽管增加了编写代码的难度，但对于用户来说接口统一。

1.2.4　BlinkDB

BlinkDB 是一个用于在海量数据上运行交互式 SQL 查询的大规模并行查询引擎，它允许用户通过权衡数据精度来提升查询响应时间，其数据的精度被控制在允许的误差范围内。为了达到这个目标，BlinkDB 使用如下核心思想：

- 自适应优化框架，从原始数据随着时间的推移建立并维护一组多维样本。
- 动态样本选择策略，选择一个适当大小的示例，该示例基于查询的准确性和响应时间的紧迫性。

和传统关系型数据库不同，BlinkDB 是一个交互式查询系统，就像一个跷跷板，用户需要在查询精度和查询时间上做权衡；如果用户想更快地获取查询结果，那么将牺牲查询结果的精度；反之，用户如果想获取更高精度的查询结果，就需要牺牲查询响应时间。图 1-11 为 BlinkDB 架构。

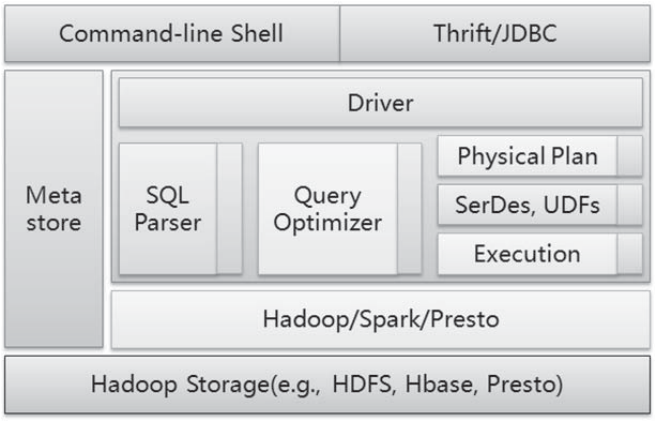

图 1-11　BlinkDB 架构

1.2.5 MLBase/MLlib

MLBase 是 Spark 生态系统中专注于机器学习的组件，它的目标是让机器学习的门槛更低，让一些可能并不了解机器学习的用户能够方便地使用 MLBase。MLBase 分为 4 个部分：MLRuntime、MLlib、MLI 和 ML Optimizer。

- MLRuntime：是由 Spark Core 提供的分布式内存计算框架，运行由 Optimizer 优化过的算法进行数据的计算并输出分析结果。
- MLlib：是 Spark 实现一些常见的机器学习算法和实用程序，包括分类、回归、聚类、协同过滤、降维以及底层优化。该算法可以进行可扩充。
- MLI：是一个进行特征抽取和高级 ML 编程抽象算法实现的 API 或平台。
- ML Optimizer：会选择它认为最适合的已经在内部实现好了的机器学习算法和相关参数，来处理用户输入的数据，并返回模型或其他帮助分析的结果。

图 1-12 为 MLBase/MLlib 结构。

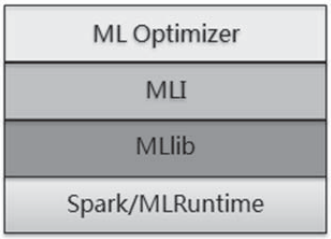

图 1-12 MLBase/MLlib 结构

MLBase 的核心是其优化器（ML Optimizer），它可以把声明式的任务转化成复杂的学习计划，最终产出最优的模型和计算结果。MLBase 与其他机器学习 Weka 和 Mahout 不同，三者各有特色，具体内容如下。

- MLBase 基于 Spark，它是使用的是分布式内存计算的；Weka 是一个单机的系统，而 Mahout 是使用 MapReduce 进行处理数据（Mahout 正向使用 Spark 处理数据转变）。
- MLBase 是自动化处理的；Weka 和 Mahout 都需要使用者具备机器学习技能，来选择自己想要的算法和参数来做处理。
- MLBase 提供了不同抽象程度的接口，可以由用户通过该接口实现算法的扩展。

1.2.6 GraphX

GraphX 最初是伯克利 AMP 实验室的一个分布式图计算框架项目，后来整合到 Spark 中成

为一个核心组件。它是 Spark 中用于图和图并行计算的 API，可以认为是 GraphLab 和 Pregel 在 Spark 上的重写及优化。跟其他分布式图计算框架相比，GraphX 最大的优势是：在 Spark 基础上提供了一栈式数据解决方案，可以高效地完成图计算的完整的流水作业。

GraphX 的核心抽象是 Resilient Distributed Property Graph，一种点和边都带属性的有向多重图。GraphX 扩展了 Spark RDD 的抽象，它有 Table 和 Graph 两种视图，但只需要一份物理存储，两种视图都有自己独有的操作符，从而获得了灵活操作和执行效率。GraphX 的整体架构如图 10-4 所示，其中大部分的实现都是围绕 Partition 的优化进行的，这在某种程度上说明了，点分割的存储和相应的计算优化的确是图计算框架的重点和难点。

GraphX 的底层设计有以下几个关键点。

（1）对 Graph 视图的所有操作，最终都会转换成其关联的 Table 视图的 RDD 操作来完成。这样对一个图的计算，最终在逻辑上，等价于一系列 RDD 的转换过程。因此，Graph 最终具备了 RDD 的 3 个关键特性：Immutable、Distributed 和 Fault-Tolerant。其中最关键的是 Immutable（不变性）。逻辑上，所有图的转换和操作都产生了一个新图；物理上，GraphX 会有一定程度的不变顶点和边的复用优化，对用户透明。

（2）两种视图底层共用的物理数据，由 RDD[Vertex-Partition]和 RDD[EdgePartition]这两个 RDD 组成。点和边实际都不是以表 Collection[tuple]的形式存储的，而是由 VertexPartition/EdgePartition 在内部存储一个带索引结构的分片数据块，以加速不同视图下的遍历速度。不变的索引结构在 RDD 转换过程中是共用的，降低了计算和存储开销。

（3）图的分布式存储采用点分割模式，而且使用 partitionBy 方法，由用户指定不同的划分策略（PartitionStrategy）。划分策略会将边分配到各个 EdgePartition，顶点 Master 分配到各个 VertexPartition，EdgePartition 也会缓存本地边关联点的 Ghost 副本。划分策略的不同会影响到所需要缓存的 Ghost 副本数量，以及每个 EdgePartition 分配的边的均衡程度，需要根据图的结构特征选取最佳策略。

1.2.7　SparkR

R 是遵循 GNU 协议的一款开源、免费的软件，广泛应用于统计计算和统计制图，但是它只能单机运行。为了能够使用 R 语言分析大规模分布式的数据，伯克利分校 AMP 实验室开发了 SparkR，并在 Spark 1.4 版本中加入了该组件。通过 SparkR 可以分析大规模的数据集，并通过 R Shell 交互式地在 SparkR 上运行作业。SparkR 特性如下：

- 提供了 Spark 中弹性分布式数据集（RDDs）的 API，用户可以在集群上通过 R Shell 交互性地运行 Spark 任务。

- 支持序化闭包功能，可以将用户定义函数中所引用到的变量自动序化发送到集群中其他的机器上。
- SparkR 还可以很容易地调用 R 开发包，只需要在集群上执行操作前用 includePackage 读取 R 开发包就可以了。

图 1-13 为 SparkR 的处理流程示意图。

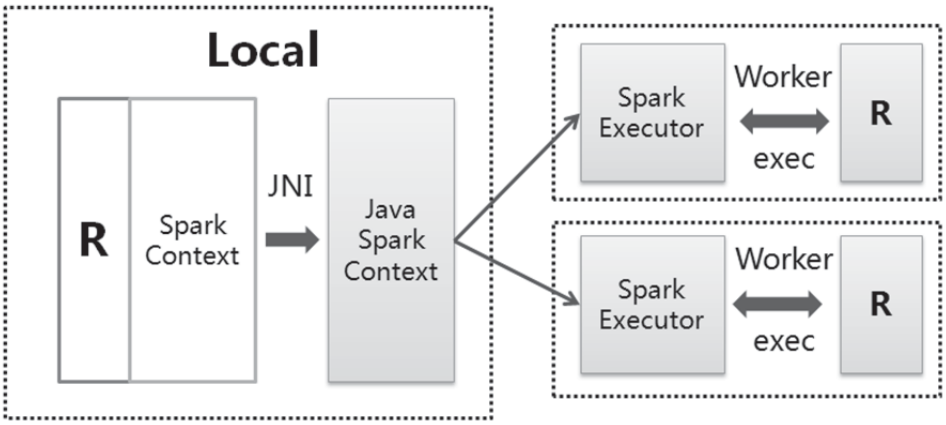

图 1-13　SparkR 处理流程

1.2.8　Alluxio

Alluxio 是一个分布式内存文件系统，它是一个高容错的分布式文件系统，允许文件以内存的速度在集群框架中进行可靠的共享，就像 Spark 和 MapReduce 那样。Alluxio 是架构在最底层的分布式文件存储和上层的各种计算框架之间的一种中间件。其主要职责是将那些不需要落地到 DFS 里的文件，落地到分布式内存文件系统中，来达到共享内存，从而提高效率。同时可以减少内存冗余、GC 时间等。

和 Hadoop 类似，Alluxio 的架构是传统的 Master-Slave 架构，所有的 Alluxio Worker 都被 Alluxio Master 所管理，Alluxio Master 通过 Alluxio Worker 定时发出的心跳来判断 Worker 是否已经崩溃以及每个 Worker 剩余的内存空间量，为了防止单点问题使用了 ZooKeeper 做了 HA，具体参见图 12-2 Alluxio 高可用架构图。

Alluxio 具有如下特性。

- JAVA-Like File API：Alluxio 提供类似 Java File 类的 API。
- 兼容性：Alluxio 实现了 HDFS 接口，所以 Spark 和 MapReduce 程序不需要任何修改即

可运行。

- 可插拔的底层文件系统：Alluxio 是一个可插拔的底层文件系统，提供容错功能，它将内存数据记录在底层文件系统。它有一个通用的接口，可以很容易地插入到不同的底层文件系统。目前支持 HDFS、S3、GlusterFS 和单节点的本地文件系统，以后将支持更多的文件系统。Alluxio 所支持的应用如图 1-14 所示。

图 1-14　Alluxio 支持的应用

1.3　小结

本章先介绍了 Spark 诞生的背景，它继承了分布式并行计算的优点并改进了 MapReduce 明显的缺陷，然后介绍了 Spark 的演进路线，最后对 Spark 生态系统进行了介绍，由于这些组件的存在实现了“One Stack to Rule Them All”目标。