厦门大学林子雨编著

《大数据技术原理与应用》

第 16 章 Spark

（版本号：2016 年 4 月 20 日版本）

（备注：2015 年 8 月 1 日第一版教材中没有本章，本章为 2016 年新增内容，将被放入第二版教材中）

主讲教师：林子雨
厦门大学数据库实验室
二零一六年四月

中国高校大数据课程
公 共 服 务 平 台

中国高校大数据课程公共服务平台，由中国高校首个“数字教师”的提出者和建设者——林子雨老师发起，由厦门大学数据库实验室全力打造，由厦门大学云计算与大数据研究中心、海峡云计算与大数据应用研究中心携手共建。这是国内第一个服务于高校大数据课程建设的公共服务平台，旨在促进国内高校大数据课程体系建设，提高大数据课程教学水平，降低大数据课程学习门槛，提升学生课程学习效果。

平台为教师开展大数据教学和学生学习大数据课程，提供全方位、一站式**免费**服务，包括**讲义 PPT**、教学大纲、备课指南、**学习指南**、上机习题、**授课视频**、技术资料等。

百度搜索“厦门大学数据库实验室”，访问平台主页，或直接访问平台地址：
http://dblab.xmu.edu.cn/post/bigdata-teaching-platform/

扫一扫访问平台主页

《大数据技术原理与应用——概念、存储、处理、分析与应用》，由厦门大学计算机科学系教师林子雨博士编著，是中国高校第一本系统介绍大数据知识的专业教材。本书定位为大数据技术入门教材，为读者搭建起通向“大数据知识空间”的桥梁和纽带，以“构建知识体系、阐明基本原理、引导初级实践、了解相关应用”为原则，为读者在大数据领域“深耕细作”奠定基础、指明方向。

全书共有 13 章，系统地论述了大数据的基本概念、大数据处理架构 Hadoop、分布式文件系统 HDFS、分布式数据 库 HBase、NoSQL 数据库、云数据库、分布式并行编程模型 MapReduce、流计算、图计算、数据可视化以及大数据在互联网、生物医学和物流等各个领域的应用。在 Hadoop、HDFS、HBase 和 MapReduce 等重要章节，安排了入门级的实践操作，让读者更好地学习和掌握大数据关键技术。

本书可以作为高等院校计算机专业、信息管理等相关专业的大数据课程教材，也可供相关技术人员参考、学习、培训之用。

欢迎访问《大数据技术原理与应用——概念、存储、处理、分析与应用》教材官方网站：
http://dblab.xmu.edu.cn/post/bigdata

扫一扫访问教材官网

目录

第 16 章 Spark

Spark 最初诞生于伯克利大学的 APM 实验室，是一个可应用于大规模数据处理的快速、通用引擎，如今是 Apache 软件基金会下的顶级开源项目之一。Spark 正如其名，是如闪电般快速的集群计算平台（Spark 官网的标题为：Apache Spark - Lightning-fast cluster computing）。Spark 最初的设计目标是使数据分析更快——不仅运行速度快，也要能快速、容易地编写程序。为了使程序运行更快，Spark 提供了内存计算，减少了迭代计算时的 IO 开销；而为了使编写程序更为容易，Spark 使用简练、优雅的 Scala 语言编写，基于 Scala 提供了交互式的编程体验。虽然，Hadoop 已成为大数据的事实标准，但其 MapReduce 分布式计算模型仍存在诸多缺陷，而 Spark 不仅具备 Hadoop MapReduce 所具有的优点，且解决了 Hadoop MapReduce 的缺陷。Spark 正以其结构一体化、功能多元化的优势逐渐成为当今大数据领域最热门的大数据计算平台。

本章首先简单介绍 Spark 与 Scala 编程语言，接着分析 Spark 与 Hadoop 的区别，认识 Hadoop MapReduce 计算模型的缺陷与 Spark 的优势；然后讲解了 Spark 的生态系统和架构设计，并介绍了 Spark SQL 和 Spark Streaming，以及 Spark 的部署和应用方式；最后介绍 Spark 的安装与基本的编程实践。

16.1 概述

16.1.1Spark 简介

Spark 最初由美国加州伯克利大学（UCBerkeley）的 AMP（Algorithms, Machines and People）实验室于 2009 年开发，是基于内存计算的大数据并行计算框架，可用于构建大型的、低延迟的数据分析应用程序。Spark 在诞生之初属于研究性项目，其诸多核心理念均源自学术研究论文。2013 年，Spark 加入 Apache 孵化器项目后，开始获得迅猛的发展，如今已成为 Apache 软件基金会最重要的三大分布式计算系统开源项目之一（即 Hadoop、Spark、Storm）。

Spark 作为大数据计算平台的后起之秀，在 2014 年打破了 Hadoop 保持的基准排序（Sort Benchmark）纪录，使用 206 个节点在 23 分钟的时间里完成了 100TB 数据的排序，而 Hadoop 则是使用 2000 个节点在 72 分钟的时间里完成同样数据的排序。也就是说，Spark 仅使用了十分之一的计算资源，获得了比 Hadoop 快 3 倍的速度。新纪录的诞生，使得 Spark 获得多方追捧，也表明了 Spark 可以作为一个更加快速、高效的大数据计算平台。

Spark 具有如下几个主要特点：

- **运行速度快**：Spark 使用先进的 DAG（Directed Acyclic Graph，有向无环图）执行引擎，以支持循环数据流与内存计算，基于内存的执行速度可比 Hadoop MapReduce 快上百倍，基于磁盘的执行速度也能快十倍；
- **容易使用**：Spark 支持使用 Scala、Java、Python 和 R 语言进行编程，简洁的 API 设计有助于用户轻松构建并行程序，并且可以通过 Spark Shell 进行交互式编程；
- **通用性**：Spark 提供了完整而强大的技术栈，包括 SQL 查询、流式计算、机器学习和图算法组件，这些组件可以无缝整合在同一个应用中，足以应对复杂的计算；
- **运行模式多样**：Spark 可运行于独立的集群模式中，或者运行于 Hadoop 中，也可运

行于 Amazon EC2 等云环境中，并且可以访问 HDFS、Cassandra、HBase、Hive 等多种数据源。

Spark 源码托管在 Github 中，截至 2016 年 3 月，共有超过 800 名来自 200 多家不同公司的开发人员贡献了 15000 次代码提交，可见 Spark 的受欢迎程度。从图 16-1 中也可以看出，从 2013 年至 2016 年，Spark 搜索趋势逐渐增加，Hadoop 则相对变化不大。

图 16-1 谷歌趋势：Spark 与 Hadoop 对比

此外，每年举办的全球 Spark 顶尖技术人员峰会 Spark Summit，吸引了使用 Spark 的一线技术公司及专家汇聚一堂，共同探讨目前 Spark 在企业的落地情况及未来 Spark 的发展方向和挑战。Spark Summit 的参会人数从 2014 年的不到 500 人暴涨到 2015 年的 2000 多人，足以反映 Spark 社区的旺盛人气。

Spark 如今已吸引了国内外各大公司的注意，如腾讯、淘宝、百度、亚马逊等公司均不同程度地使用了 Spark 来构建大数据分析应用，并应用到实际的生产环境中。相信在将来，Spark 会在更多的应用场景中发挥重要作用。

16.1.2Scala 简介

Scala 是一门现代的多范式编程语言，平滑地集成了面向对象和函数式语言的特性，旨在以简练、优雅的方式来表达常用编程模式。Scala 语言的名称来自于“可伸展的语言”，从写个小脚本到建立个大系统的编程任务均可胜任。Scala 运行于 Java 平台（JVM，Java 虚拟机）上，并兼容现有的 Java 程序。

Spark 的设计目的之一就是使程序编写更快更容易，这也是 Spark 选择 Scala 的原因所在。总体而言，Scala 具有以下突出的优点：

- Scala 具备强大的并发性，支持函数式编程，可以更好地支持分布式系统；
- Scala 语法简洁，能提供优雅的 API；
- Scala 兼容 Java，运行速度快，且能融合到 Hadoop 生态圈中。

实际上，AMP 实验室的大部分核心产品都是使用 Scala 开发的。Scala 近年来也吸引了不少开发者的眼球，例如，知名社交网站 Twitter 已将代码从 Ruby 转到了 Scala。

Scala 是 Spark 的主要编程语言，但 Spark 还支持 Java、Python、R 作为编程语言，因此，若仅仅是编写 Spark 程序，并非一定要用 Scala。Scala 的优势是提供了 REPL（Read-Eval-Print Loop，交互式解释器），因此，在 Spark Shell 中可进行交互式编程（即表达式计算完成就会输出结果，而不必等到整个程序运行完毕，因此可即时查看中间结果，并对程序进行修改），这样可以在很大程度上提升开发效率。

16.1.3Spark 与 Hadoop 的对比

Hadoop 虽然已成为大数据技术的事实标准，但其本身还存在诸多缺陷，最主要的缺陷是其 MapReduce 计算模型延迟过高，无法胜任实时、快速计算的需求，因而只适用于离线批处理的应用场景。

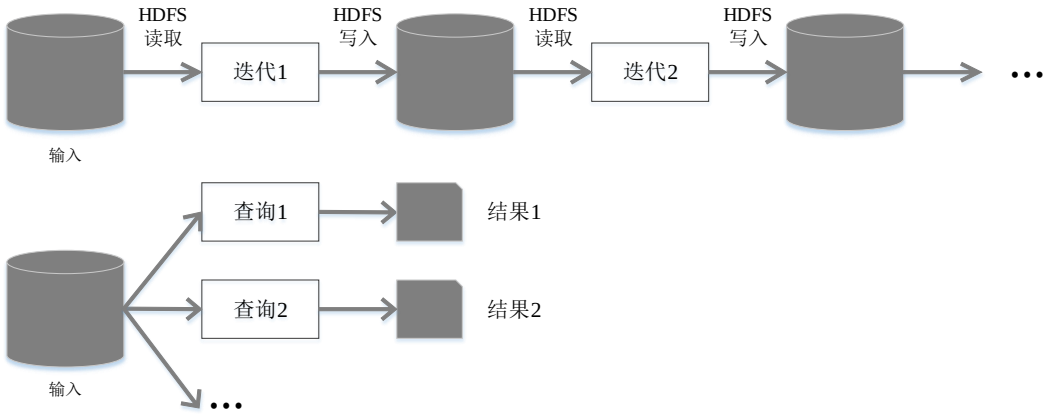
回顾 Hadoop 的工作流程，可以发现 Hadoop 存在如下一些缺点：

- 表达能力有限。计算都必须要转化成 Map 和 Reduce 两个操作，但这并不适合所有的情况，难以描述复杂的数据处理过程；
- 磁盘 IO 开销大。每次执行时都需要从磁盘读取数据，并且在计算完成后需要将中间结果写入到磁盘中，IO 开销较大；
- 延迟高。一次计算可能需要分解成一系列按顺序执行的 MapReduce 任务，任务之间的衔接由于涉及到 IO 开销，会产生较高延迟。而且，在前一个任务执行完成之前，其他任务无法开始，难以胜任复杂、多阶段的计算任务。

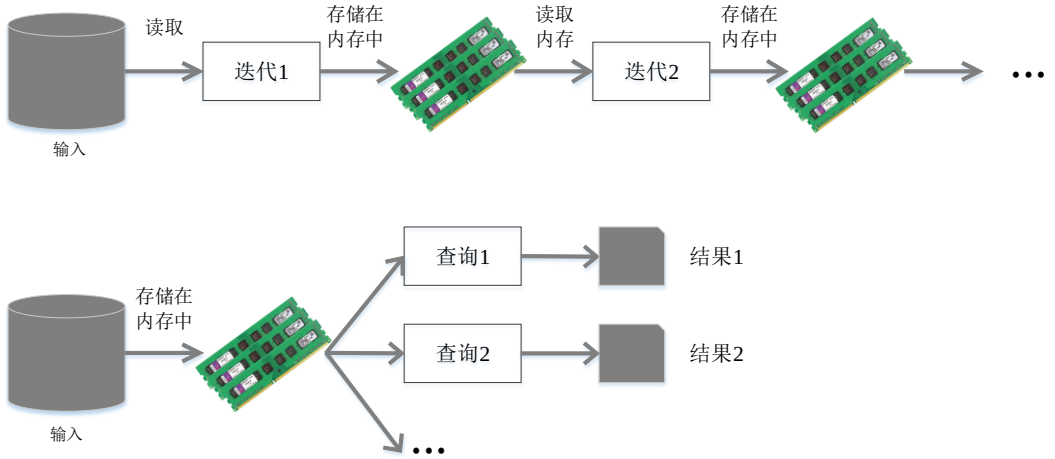
Spark 在借鉴 Hadoop MapReduce 优点的同时，很好地解决了 MapReduce 所面临的问题。相比于 MapReduce，Spark 主要具有如下优点：

- Spark 的计算模式也属于 MapReduce，但不局限于 Map 和 Reduce 操作，还提供了多种数据集操作类型，编程模型比 MapReduce 更灵活；
- Spark 提供了内存计算，中间结果直接放到内存中，带来了更高的迭代运算效率；
- Spark 基于 DAG 的任务调度执行机制，要优于 MapReduce 的迭代执行机制。

如图 16-2 所示，对比 Hadoop 与 Spark 的执行流程可以看到，Spark 最大的特点就是将计算数据、中间结果都存储在内存中，大大减少了 IO 开销，因而，Spark 更适合于迭代运算比较多的数据挖掘与机器学习运算。



(a) Hadoop MapReduce执行流程



(b) Spark执行流程

图 16-2 Hadoop 与 Spark 的执行流程对比

使用 Hadoop 进行迭代计算非常耗资源，因为每次迭代都需要从磁盘中写入、读取中间数据，IO 开销大。而 Spark 将数据载入内存后，之后的迭代计算都可以直接使用内存中的中间结果作运算，避免了从磁盘中频繁读取数据。如图 16-3 所示，Hadoop 与 Spark 在执行逻辑回归时所需的时间相差巨大。

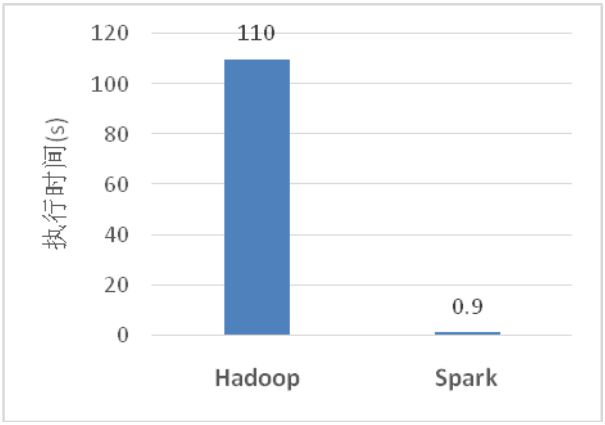


图 16-3 Hadoop 与 Spark 执行逻辑回归的时间对比

在实际进行开发时，使用 Hadoop 需要编写不少相对底层的代码，不够高效。相对而言，Spark 提供了多种高层次、简洁的 API，通常情况下，对于实现相同功能的应用程序，Spark 的代码量要比 Hadoop 少 2-5 倍。更重要的是，Spark 提供了实时交互式编程反馈，可以方便地验证、调整算法。

尽管 Spark 相对于 Hadoop 而言具有较大优势，但 Spark 并不能完全替代 Hadoop，主要用于替代 Hadoop 中的 MapReduce 计算模型。实际上，Spark 已经很好地融入了 Hadoop 生态圈，并成为其中的重要一员，它可以借助于 YARN 实现资源调度管理，借助于 HDFS 实现分布式存储。此外，Hadoop 可以使用廉价的、异构的机器来做分布式存储与计算，但是，Spark 对硬件的要求稍高一些，对内存与 CPU 有一定的要求。

16.2Spark 生态系统

在实际应用中，大数据处理主要包括以下三个类型：

- 复杂的批量数据处理：时间跨度通常在数十分钟到数小时之间；
- 基于历史数据的交互式查询：时间跨度通常在数十秒到数分钟之间；
- 基于实时数据流的数据处理：时间跨度通常在数百毫秒到数秒之间。

目前已有很多相对成熟的开源软件用于处理以上三种情景，比如，可以利用 Hadoop MapReduce 来进行批量数据处理，可以用 Impala 来进行交互式查询（Impala 与 Hive 相似，但底层引擎不同，提供了实时交互式 SQL 查询），对于流式数据处理可以采用开源流计算框架 Storm。一些企业可能只会涉及其中部分应用场景，只需部署相应软件即可满足业务需求，但是，对于互联网公司而言，通常会同时存在以上三种场景，就需要同时部署三种不同的软件，这样做难免会带来一些问题：

- 不同场景之间输入输出数据无法做到无缝共享，通常需要进行数据格式的转换；
- 不同的软件需要不同的开发和维护团队，带来了较高的使用成本；
- 比较难以对同一个集群中的各个系统进行统一的资源协调和分配。

Spark 的设计遵循“一个软件栈满足不同应用场景”的理念，逐渐形成了一套完整的生态系统，既能够提供内存计算框架，也可以支持 SQL 即席查询、实时流式计算、机器学习和图计算等。Spark 可以部署在资源管理器 YARN 之上，提供一站式的大数据解决方案。因此，Spark 所提供的生态系统足以应对上述三种场景，即同时支持批处理、交互式查询和流数据处理。

现在，Spark 生态系统已经成为伯克利数据分析软件栈 BDAS（Berkeley Data Analytics Stack）的重要组成部分。BDAS 的架构如图 16-4 所示，从中可以看出，Spark 专注于数据的处理分析，而数据的存储还是要借助于 Hadoop 分布式文件系统 HDFS、Amazon S3 等来实现的。

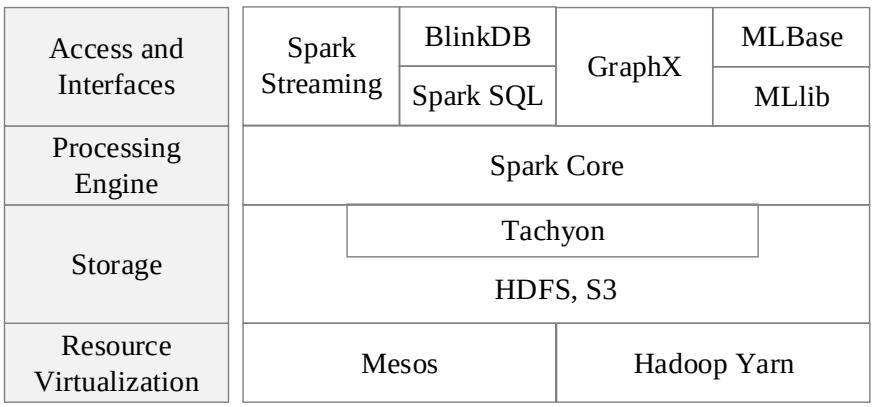


图 16-4 BDAS 架构

Spark 的生态系统主要包含了 Spark Core、Spark SQL、Spark Streaming、MLLib 和 GraphX 等组件，各个组件的具体功能如下：

- Spark Core：Spark Core 包含 Spark 的基本功能，如内存计算、任务调度、部署模式、故障恢复、存储管理等。Spark 建立在统一的抽象 RDD 之上，使其可以以基本一致的方式应对不同的大数据处理场景；通常所说的 Apache Spark，就是指 Spark Core；
- Spark SQL：Spark SQL 允许开发人员直接处理 RDD，同时也可查询 Hive、HBase 等外部数据源。Spark SQL 的一个重要特点是其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 命令进行查询，并进行更复杂的数据分析；
- Spark Streaming：Spark Streaming 支持高吞吐量、可容错处理的实时流数据处理，其核心思路是将流式计算分解成一系列短小的批处理作业。Spark Streaming 支持多种数据输入源，如 Kafka、Flume 和 TCP 套接字等；
- MLlib（机器学习）：MLlib 提供了常用机器学习算法的实现，包括聚类、分类、回归、协同过滤等，降低了机器学习的门槛，开发人员只要具备一定的理论知识就能进行机器学习的工作；
- GraphX（图计算）：GraphX 是 Spark 中用于图计算的 API，可认为是 Pregel 在 Spark 上的重写及优化，Graphx 性能良好，拥有丰富的功能和运算符，能在海量数据上自如地运行复杂的图算法。

表 16-1 给出了在不同的应用场景下，可以选用的 Spark 生态系统中的组件和其他框架。

表 16-1 Spark 的应用场景

应用场景	时间跨度	其他框架	Spark 生态系统中的组件
复杂的批量数据处理	小时级	MapReduce、Hive	Spark
基于历史数据的交互式查询	分钟级、秒级	Impala、Dremel、Drill	Spark SQL
基于实时数据流的数据处理	毫秒、秒级	Storm、S4	Spark Streaming
基于历史数据的数据挖掘	-	Mahout	MLlib
图结构数据的处理	-	Pregel、Hama	GraphX

16.3Spark 运行架构

本节首先介绍 Spark 的基本概念和架构设计方法，然后介绍 Spark 运行基本流程，最后介绍 RDD 的运行原理。

16.3.1 基本概念

在具体讲解 Spark 运行架构之前，需要先了解几个重要的概念：

- RDD：是 Resillient Distributed Dataset（弹性分布式数据集）的简称，是分布式内存的一个抽象概念，提供了一种高度受限的共享内存模型；
- DAG：是 Directed Acyclic Graph（有向无环图）的简称，反映 RDD 之间的依赖关系；
- Executor：是运行在工作节点（Worker Node）上的一个进程，负责运行 Task；

- Application：用户编写的 Spark 应用程序；
- Task：运行在 Executor 上的工作单元；
- Job：一个 Job 包含多个 RDD 及作用于相应 RDD 上的各种操作；
- Stage：是 Job 的基本调度单位，一个 Job 会分为多组 Task，每组 Task 被称为 Stage，或者也被称为 TaskSet，代表了一组关联的、相互之间没有 Shuffle 依赖关系的任务组成的任务集；

16.3.2 架构设计

如图 16-5 所示，Spark 运行架构包括集群资源管理器（Cluster Manager）、运行作业任务的工作节点（Worker Node）、每个应用的任务控制节点（Driver）和每个工作节点上负责具体任务的执行进程（Executor）。其中，集群资源管理器可以是 Spark 自带的资源管理器，也可以是 YARN 或 Mesos 等资源管理框架。

与 Hadoop MapReduce 计算框架相比，Spark 所采用的 Executor 有两个优点：一是利用多线程来执行具体的任务（Hadoop MapReduce 采用的是进程模型），减少任务的启动开销；二是 Executor 中有一个 BlockManager 存储模块（类似于 KV 系统），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块里，下次需要时，就可以直接读该存储模块里的数据，而不需要读写到 HDFS 等文件系统里，因而有效减少了 IO 开销；或者在交互式查询场景下，预先将表缓存到该存储系统上，从而可以提高读写 IO 性能。

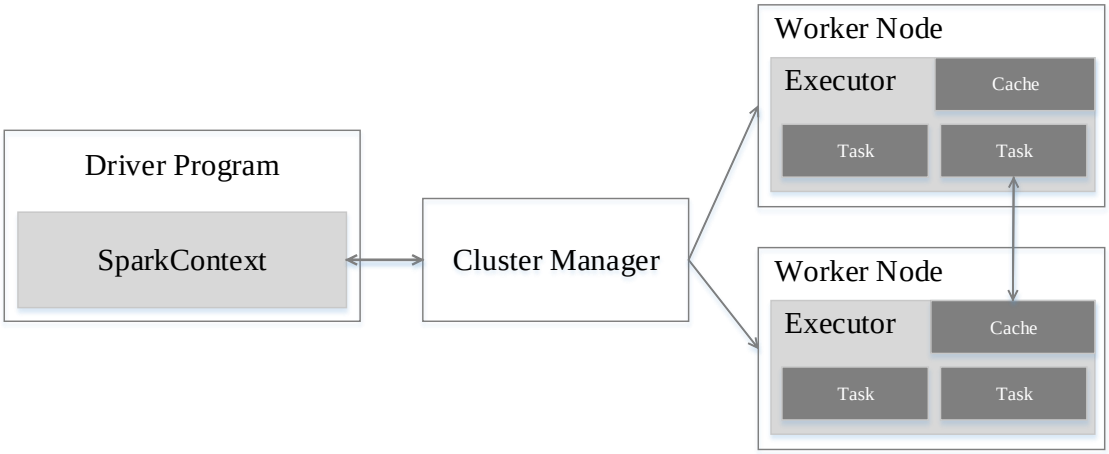


图 16-5 Spark 运行架构

总体而言，如图 16-6 所示，在 Spark 中，一个 Application 由一个 Driver 和若干个 Job 构成，一个 Job 由多个 Stage 构成，一个 Stage 由多个没有 Shuffle 关系的 Task 组成。当执行一个 Application 时，Driver 会向集群管理器申请资源，启动 Executor，并向 Executor 发送应用程序代码和文件，然后在 Executor 上执行 Task，运行结束后，执行结果会返回给 Driver，或者写到 HDFS 或者其他数据库中。

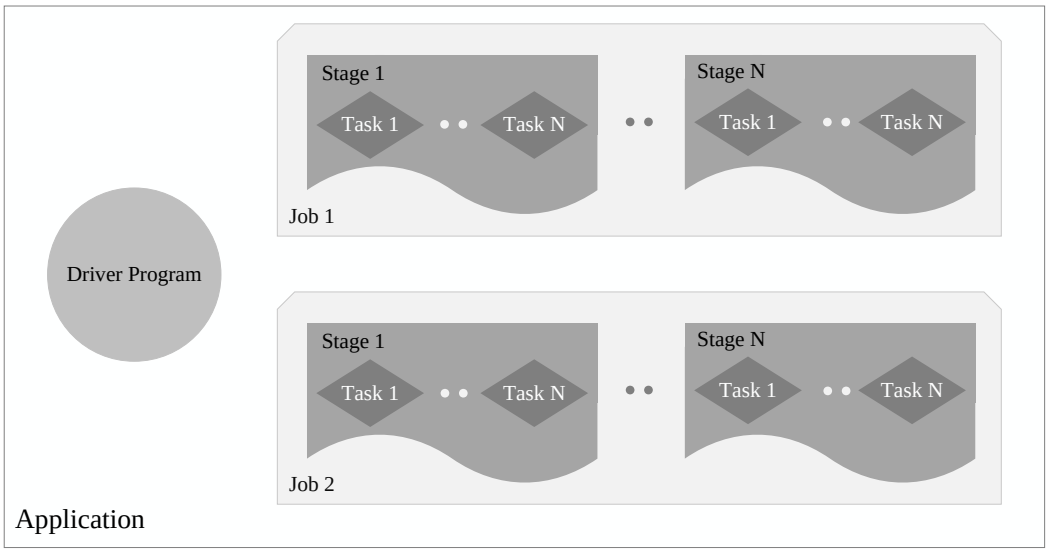


图 16-6 Spark 中各种概念之间的相互关系

16.3.3Spark 运行基本流程

如图 16-7 所示，Spark 的基本运行流程如下：

（1）当一个 Spark Application 被提交时，首先需要为这个应用构建起基本的运行环境，即由 Driver 创建一个 SparkContext，由 SparkContext 负责和资源管理器（Cluster Manager）的通信以及进行资源的申请、任务的分配和监控等。SparkContext 会向资源管理器注册并申请运行 Executor 的资源；

（2）资源管理器为 Executor 分配资源，并启动 Executor 进程，Executor 运行情况将随着“心跳”发送到资源管理器上；

（3）SparkContext 根据 RDD 的依赖关系构建 DAG 图，DAG 图提交给 DAGScheduler 进行解析，将 DAG 图分解成 Stage，并且计算出各个 Stage 之间的依赖关系，然后把一个个 TaskSet 提交给底层调度器 TaskScheduler 进行处理；Executor 向 SparkContext 申请 Task，Task Scheduler 将 Task 发放给 Executor 运行，同时，SparkContext 将应用程序代码发放给 Executor；

（4）Task 在 Executor 上运行，把执行结果反馈给 TaskScheduler，然后反馈给 DAGScheduler，运行完毕后写入数据并释放所有资源。

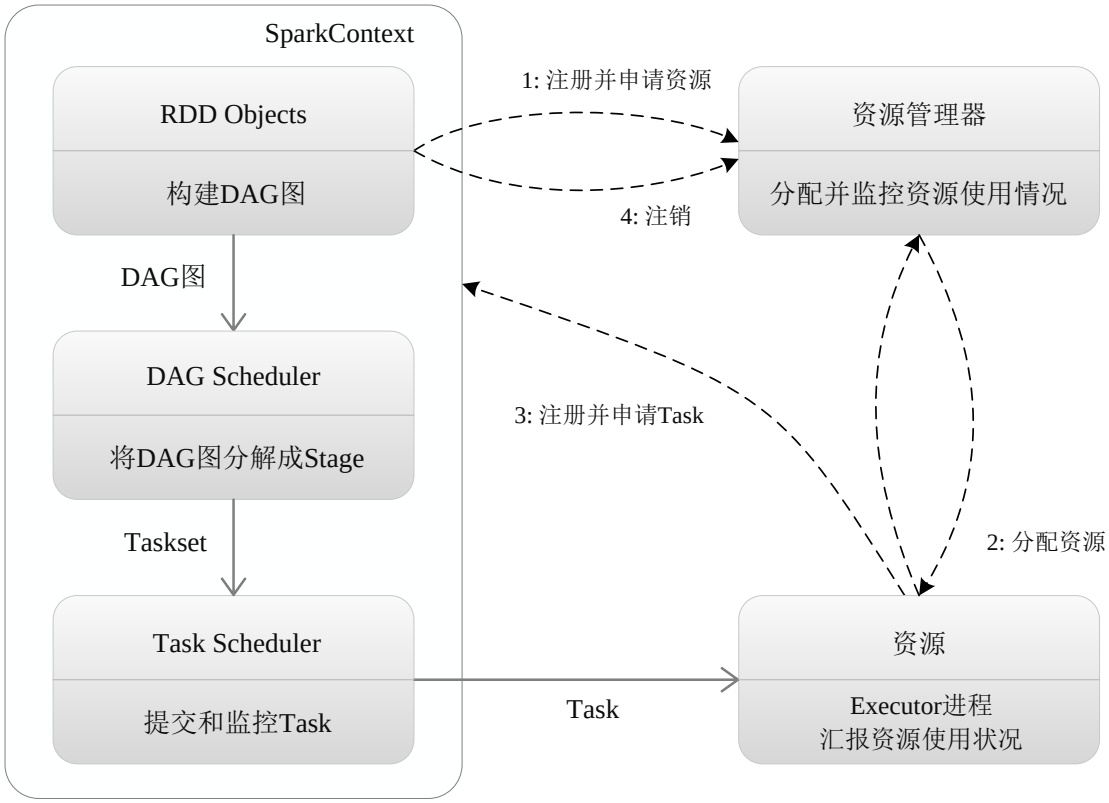


图 16-7 Spark 运行基本流程图

总体而言，Spark 运行架构具有以下特点：

（1）每个 Application 都有自己专属的 Executor 进程，并且该进程在 Application 运行期间一直驻留。Executor 进程以多线程的方式运行 Task；

（2）Spark 运行过程与资源管理器无关，只要能够获取 Executor 进程并保持通信即可；

（3）Task 采用了数据本地性和推测执行等优化机制。数据本地性是尽量将计算移到数据所在的节点上进行，即“计算向数据靠拢”，因为移动计算比移动数据所占的网络资源要少得多。而且，Spark 采用了延时调度机制，可以在更大的程度上实现执行过程优化。比如，拥有数据的节点当前正被其他的 Task 占用，那么，在这种情况下是否需要将数据移动到其他的空闲节点呢？答案是不一定。因为，如果经过预测发现当前节点结束当前任务的时间要比移动数据的时间还要少，那么，调度就会等待，直到当前节点可用。

16.3.4RDD 的设计与运行原理

Spark 的核心是建立在统一的抽象 RDD 之上，使得 Spark 的各个组件可以无缝进行集成，在同一个应用程序中完成大数据计算任务。RDD 的设计理念源自 AMP 实验室发表的论文《Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing》。

1.RDD 设计背景

在实际应用中，存在许多迭代式算法（比如机器学习、图算法等）和交互式数据挖掘工具，这些应用场景的共同之处是，不同计算阶段之间会重用中间结果，即一个阶段的输出结果会作为下一个阶段的输入。但是，目前的 MapReduce 框架都是把中间结果写入到 HDFS 中，带来了大量的数据复制、磁盘 IO 和序列化开销。虽然，类似 Pregel 等图计算框架也是

将结果保存在内存当中，但是，这些框架只能支持一些特定的计算模式，并没有提供一种通用的数据抽象。RDD 就是为了满足这种需求而出现的，它提供了一个抽象的数据架构，我们不必担心底层数据的分布式特性，只需将具体的应用逻辑表达为一系列转换处理，不同 RDD 之间的转换操作形成依赖关系，可以实现管道化，从而让避免了中间结果的存储，大大降低了数据复制、磁盘 IO 和序列化开销。

2.RDD 概念

一个 RDD 就是一个分布式对象集合，本质上是一个只读的分区记录集合，每个 RDD 可分成多个分区，每个分区就是一个数据集片段，并且一个 RDD 的不同分区可以被保存到集群中不同的节点上，从而可以在集群中的不同节点上进行并行计算。RDD 提供了一种高度受限的共享内存模型，即 RDD 是只读的记录分区的集合，不能直接修改，只能基于稳定的物理存储中的数据集创建 RDD，或者通过在其他 RDD 上执行确定的转换操作（如 map、join 和 group by）而创建得到新的 RDD。RDD 提供了一组丰富的操作以支持常见的数据运算，分为“动作”（Action）和“转换”（Transformation）两种类型，前者用于执行计算并指定输出的形式，后者指定 RDD 之间的相互依赖关系。两类操作的主要区别是，转换操作（比如 map、filter、groupBy、join 等）接受 RDD 并返回 RDD，而动作操作（比如 count、collect 等）接受 RDD 但是返回非 RDD（即输出一个值或结果）。RDD 提供的转换接口都非常简单，都是类似 map、filter、groupBy、join 等粗粒度的数据转换操作，而不是针对某个数据项的细粒度修改。因此，RDD 比较适合对于数据集中元素执行相同操作的批处理式应用，而不适合于需要异步、细粒度更细状态的应用，比如 Web 应用系统、增量式的网页爬虫等。正因为这样，这种粗粒度转换接口设计，会使人直觉上认为 RDD 的功能很受限、不够强大。但是，实际上 RDD 已经被实践证明可以很好地应用于许多并行计算应用中，可以高效地表达许多框架的编程模型，比如 MapReduce、SQL、Pregel 等，并且可以应用于这些框架处理不了的交互式数据挖掘应用，因为，这些应用一般都是在多个数据项上运用相同的操作。

Spark 用 Scala 语言实现了 RDD 的 API，程序员可以通过调用 API 实现对 RDD 的各种操作。RDD 典型的执行过程如下：

1. RDD 读入外部数据源进行创建；
2. RDD 经过一系列的转换操作，每一次都会产生不同的 RDD，供给下一个转换使用；
3. 最后一个 RDD 经动作操作进行处理，并输出到外部数据源。

需要说明的是，RDD 采用了惰性调用，即在 RDD 的执行过程中，真正的计算发生在 RDD 的动作操作，对于动作之前的所有转换操作，Spark 只是记录下 RDD 生成的轨迹，即相互之间的依赖关系，而不会触发真正的计算。例如，在图 16-8 中，从输入中逻辑上生成 A 和 C 两个 RDD，经过一系列转换操作，逻辑上生成了 F（也是一个 RDD），之所以说是逻辑上，是因为这时候计算并没有发生，Spark 只是记录了 RDD 的生成和依赖关系。当 F 要进行输出时，也就是当 F 进行动作操作的时候，Spark 才会根据 RDD 的依赖关系生成 DAG，并从起点开始真正的计算。

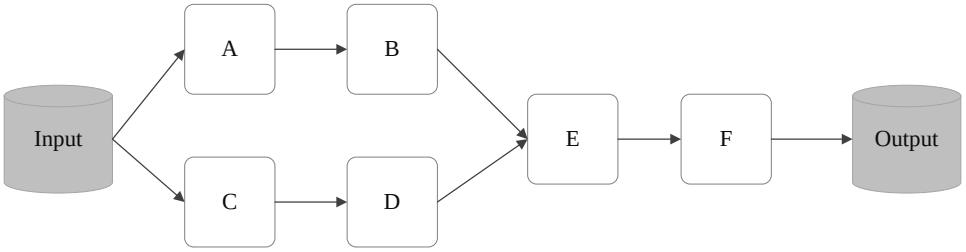


图 16-8 RDD 执行过程的一个实例

上述这一系列处理称为一个 Lineage（血缘关系），即 DAG 拓扑排序的结果。采用惰性

调用，通过血缘关系连接起来的一系列 RDD 操作就可以实现管道化（pipeline），避免了多次转换操作之间数据同步的等待，而且不用担心有过多的中间数据，因为这些具有血缘关系的操作都管道化了，一个操作得到的结果不需要保存为中间数据，而是直接管道式地流入到下一个操作进行处理。同时，这种通过血缘关系把一系列操作进行管道化连接的设计方式，也使得管道中每次操作的计算变得相对简单，保证了每个操作在处理逻辑上的单一性，而不用像 MapReduce 那样，为了尽可能地减少 MapReduce 过程，在单个 MapReduce 中写入过多复杂的逻辑。

3.RDD 特性

总体而言，Spark 采用 RDD 以后能够实现高效计算的主要原因如下：

（1）高效的容错性。现有的分布式共享内存、键值存储、内存数据库等，为了实现容错，必须在集群节点之间进行数据复制或者记录日志，也就是在节点之间会发生大量的数据传输，这对于数据密集型应用而言会带来很大的开销。在 RDD 的设计中，数据只读，不可修改，如果需要修改数据，必须从父 RDD 转换到子 RDD，由此在不同 RDD 之间建立了血缘关系。所以，RDD 是一种天生具有容错机制的特殊集合，不需要通过数据冗余的方式（比如检查点）实现容错，而只需通过 RDD 父子依赖（血缘）关系重新计算得到丢失的分区来实现容错，无需回滚整个系统，这样就避免了数据复制的高开销，而且重算过程可以在不同节点之间并行进行，实现了高效的容错。此外，RDD 提供的转换操作都是一些粗粒度的操作（比如 map、filter 和 join），RDD 依赖关系只需要记录这种粗粒度的转换操作，而不需要记录具体的数据和各种细粒度操作的日志（比如对哪个数据项进行了修改），这就大大降低了数据密集型应用中的容错开销；

（2）中间结果持久化到内存。数据在内存中的多个 RDD 操作之间进行传递，不需要“落地”到磁盘上，避免了不必要的读写磁盘开销；

（3）存放的数据可以是 Java 对象，避免了不必要的对象序列化和反序列化。

4. RDD 之间的依赖关系

RDD 中不同的操作会使得不同 RDD 中的分区会产生不同的依赖。RDD 中的依赖关系分为窄依赖（Narrow dependency）与宽依赖（Wide Dependency），图 16-9 展示了两种依赖之间的区别。

窄依赖表现为一个父 RDD 的分区对应于一个子 RDD 的分区，或多个父 RDD 的分区对应于一个子 RDD 的分区；比如图 16-9(a)中，RDD1 是 RDD2 的父 RDD，RDD2 是子 RDD，RDD1 的分区 1，对应于 RDD2 的一个分区（即分区 4）；再比如，RDD6 和 RDD7 都是 RDD8 的父 RDD，RDD6 中的分区（分区 15）和 RDD7 中的分区（分区 18），两者都对应于 RDD8 中的一个分区（分区 21）。

宽依赖则表现为存在一个父 RDD 的一个分区对应一个子 RDD 的多个分区。比如图 16-9(b)中，RDD9 是 RDD12 的父 RDD，RDD9 中的分区 24 对应了 RDD12 中的两个分区（即分区 27 和分区 28）。

总体而言，如果父 RDD 的一个分区只被一个子 RDD 的一个分区所使用就是窄依赖，否则就是宽依赖。窄依赖典型的操作包括 map、filter、union 等，宽依赖典型的操作包括 groupByKey、sortByKey 等。对于连接（join）操作，可以分为两种情况，如果连接操作使用的每个分区仅仅和已知的分区进行连接，就是窄依赖（如图 16-9(a)中的连接操作），其他情况下的连接操作都是宽依赖（如图 16-9(b)中的连接操作）。宽依赖的情形通常伴随着 Shuffle 操作。

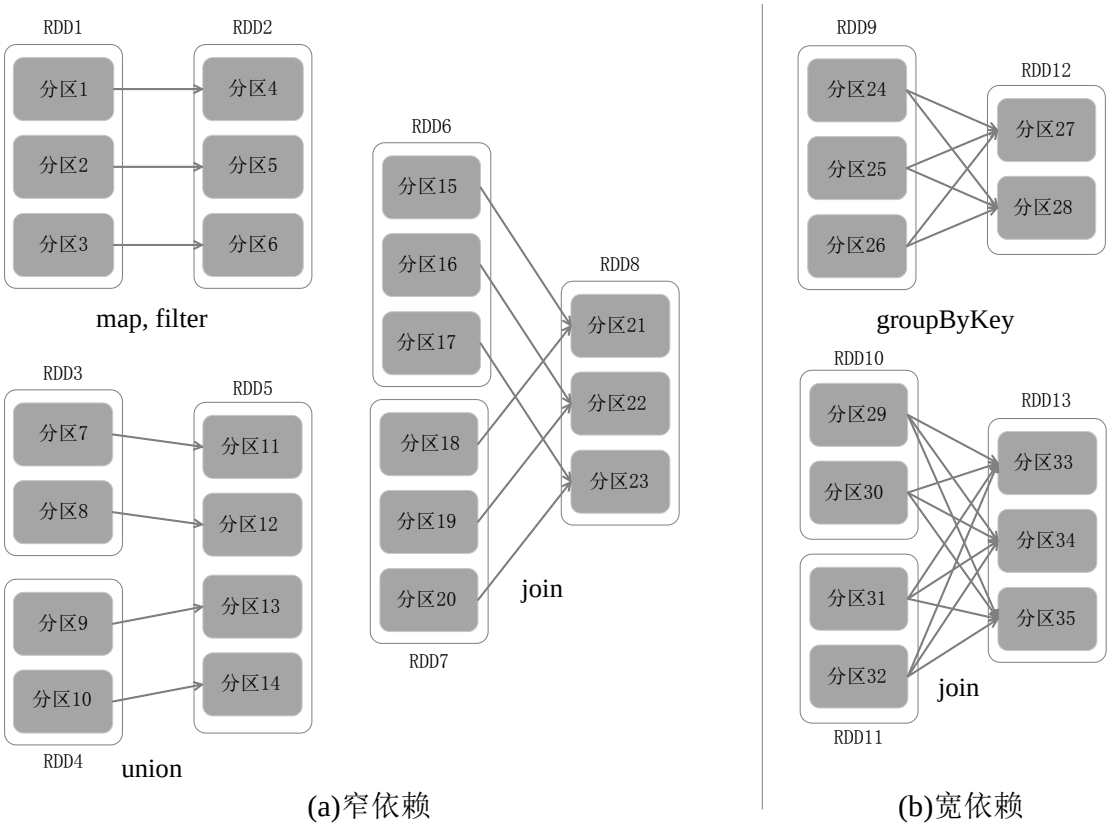


图 16-9 窄依赖与宽依赖的区别

Spark 的这种依赖关系设计，使其具有了天生的容错性，大大加快了 Spark 的执行速度。因为，RDD 数据集通过血缘关系（Lineage）记住了它是如何从其它 RDD 中演变过来的，血缘关系记录的是粗颗粒度的转换操作行为，当这个 RDD 的部分分区数据丢失时，它可以通过血缘关系获取足够的信息来重新运算和恢复丢失的数据分区，由此带来了性能的提升。相对而言，在两种依赖关系中，窄依赖的失败恢复更为高效，它只需要根据父 RDD 分区重新计算丢失的分区即可，而且可以并行地在不同节点进行重新计算。而对于宽依赖而言，重新计算过程则会涉及到多个父 RDD 分区。

5.Stage 的划分

Spark 通过分析各个 RDD 的依赖关系生成了 DAG，再通过分析各个 RDD 中的分区之间的依赖关系来决定如何划分 Stage，具体划分方法是：在 DAG 中进行反向解析，遇到宽依赖就断开，遇到窄依赖就把当前的 RDD 加入到 Stage 中；将窄依赖尽量划分在同一个 Stage 中，可以实现流水线计算（具体的 Stage 划分算法请参见 AMP 实验室发表的论文《Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing》）。例如，如图 16-10 所示，假设从 HDFS 中读入数据生成 3 个不同的 RDD（即 A、C 和 E），通过一系列转换操作后再将计算结果保存回 HDFS。对 DAG 进行解析时，在依赖图中从右往左进行反向解析，可以得到三个 Stage，可以看出，在 Stage2 中，从 map 到 union 都是窄依赖，这两步操作可以形成一个流水线操作，比如，分区 7 通过 map 操作生成的分区 9，可以不用等待分区 8 到分区 9 这个转换操作的计算结束，而是继续进行 union 操作，转换得到分区 13，这样流水线执行大大提高了计算的效率。

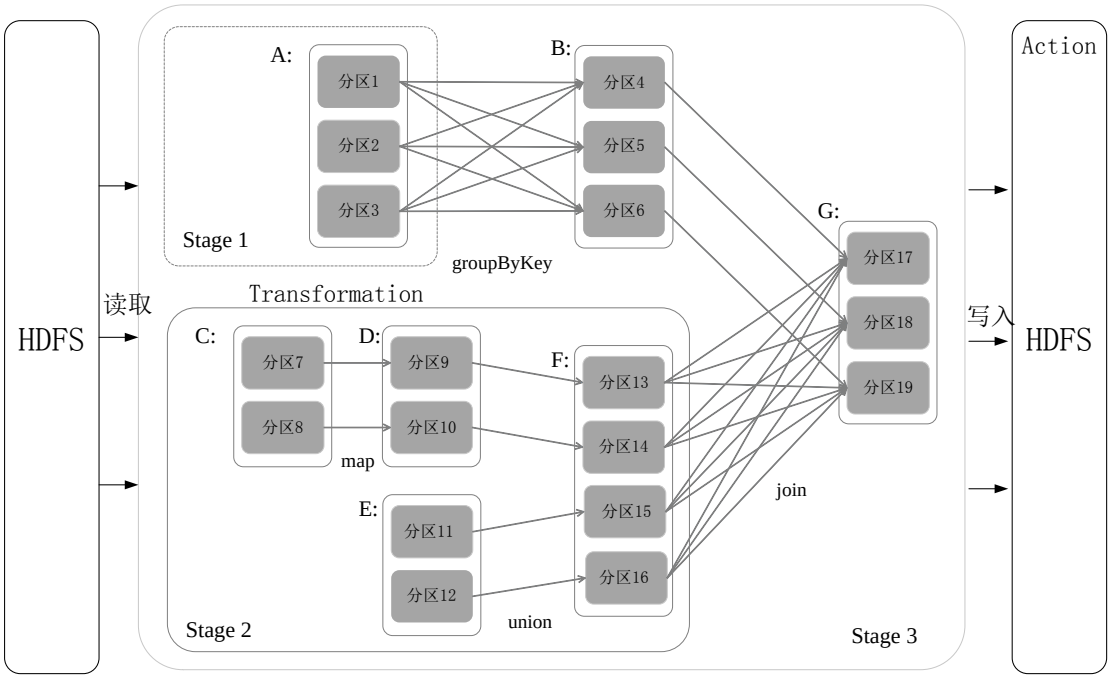


图 16-10 根据 RDD 分区的依赖关系划分 Stage

从上述划分可以看出，Stage 的类型包括两种：ShuffleMapStage 和 ResultStage，具体如下：

（1）ShuffleMapStage：不是最终的 Stage，在它之后还有其他 Stage，所以，它的输出一定需要经过 Shuffle 过程，并作为后续 Stage 的输入；这种 Stage 是以 Shuffle 为输出边界，其输入边界可以是从外部获取数据，也可以是另一个 ShuffleMapStage 的输出，其输出可以是另一个 Stage 的开始；在一个 Job 里可能有该类型的 Stage，也可能没有该类型 Stage；

（2）ResultStage：最终的 Stage，没有输出，而是直接产生结果或存储。这种 Stage 是直接输出结果，其输入边界可以是从外部获取数据，也可以是另一个 ShuffleMapStage 的输出。在一个 Job 里必定有该类型 Stage。

因此，一个 Job 含有一个或多个 Stage，其中至少含有一个 ResultStage。

5.RDD 运行过程

通过上述对 RDD 概念、依赖关系和 Stage 划分的介绍，结合之前介绍的 Spark 运行基本流程，这里再总结一下 RDD 在 Spark 架构中的运行过程：

（1）创建 RDD 对象；

（2）SparkContext 负责计算 RDD 之间的依赖关系，构建 DAG；

（3）DAGScheduler 负责把 DAG 图分解成多个 Stage，每个 Stage 中包含了多个 Task，每个 Task 会被 TaskScheduler 分发给各个 WorkerNode 上的 Executor 去执行。

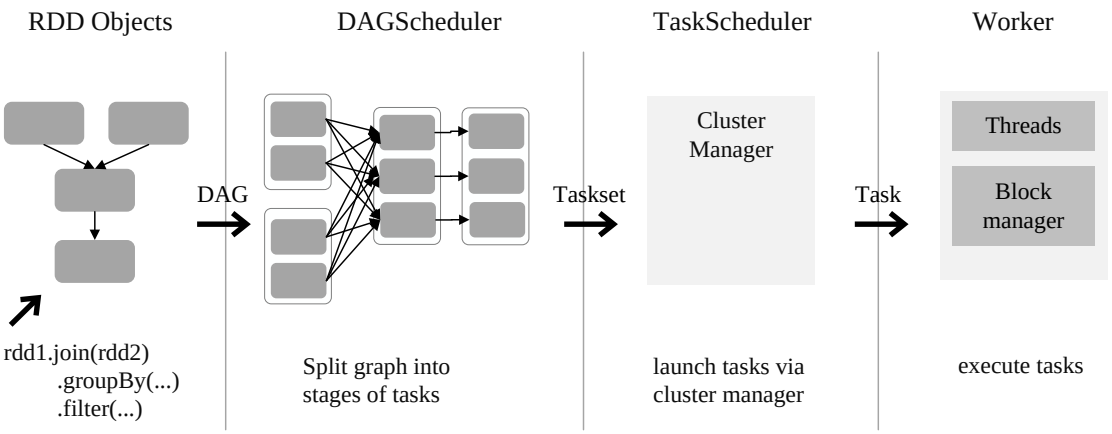


图 16-11 RDD 在 Spark 中的运行过程

16.4Spark SQL

Spark SQL 是 Spark 生态系统中非常重要的组件，其前身为 Shark。Shark 是 Spark 上的数据仓库，最初设计成与 Hive 兼容，但是该项目于 2014 年开始停止开发，转向 Spark SQL。Spark SQL 全面继承了 Shark，并进行了优化。

16.4.1 从 Shark 说起

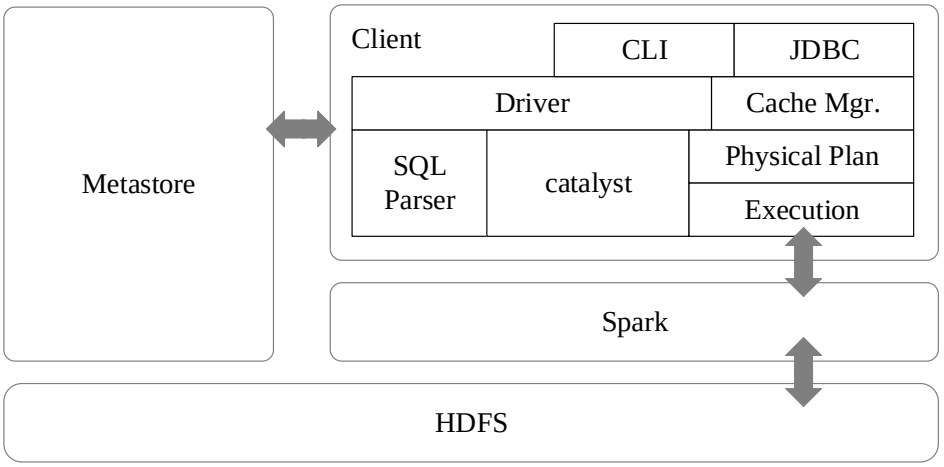
Shark 即 Hive on Spark，为了实现与 Hive 兼容，Shark 在 HiveQL 方面重用了 Hive 中的 HiveQL 解析、逻辑执行计划翻译、执行计划优化等逻辑，可以近似认为仅将物理执行计划从 MapReduce 作业替换成了 Spark 作业，通过 Hive 的 HiveQL 解析，把 HiveQL 翻译成 Spark 上的 RDD 操作。

Shark 的设计导致了两个问题：一是执行计划优化完全依赖于 Hive，不方便添加新的优化策略；二是因为 Spark 是线程级并行，而 MapReduce 是进程级并行，因此，Spark 在兼容 Hive 的实现上存在线程安全问题，导致 Shark 不得不使用另外一套独立维护的打了补丁的 Hive 源码分支。

Shark 的实现继承了大量的 Hive 代码，因而给优化和维护带来了大量的麻烦，特别是基于 MapReduce 设计的部分，成为整个项目的瓶颈。因此，在 2014 年的时候，Shark 项目中止，并转向 Spark SQL 的开发。

16.4.2Spark SQL 设计

Spark SQL 的架构如图



16-12

图所示，在 Shark 原有的架构上重写了逻辑执行计划的优化部分，解决了 Shark 存在的问题。Spark SQL 在 Hive 兼容层面仅依赖 HiveQL 解析和 Hive 元数据，也就是说，从 HQL 被解析成抽象语法树（AST）起，就全部由 Spark SQL 接管了。Spark SQL 执行计划生成和优化都由 Catalyst（函数式关系查询优化框架）负责。

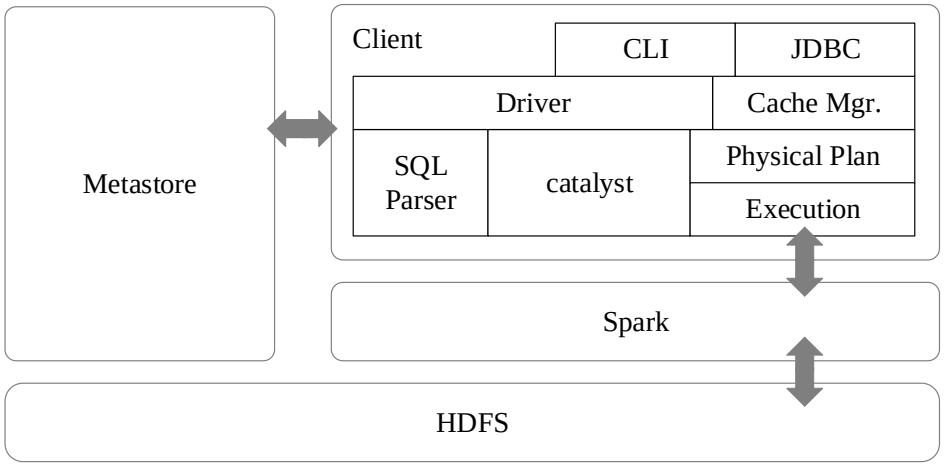


图 16-12 Spark SQL 架构

Spark SQL 增加了 SchemaRDD（即带有 Schema 信息的 RDD），使用户可以在 Spark SQL 中执行 SQL 语句，数据既可以来自 RDD，也可以来自 Hive、HDFS、Cassandra 等外部数据源，还可以是 JSON 格式的数据。Spark SQL 目前支持 Scala、Java、Python 三种语言，支持 SQL-92 规范（如图 16-13 所示）。

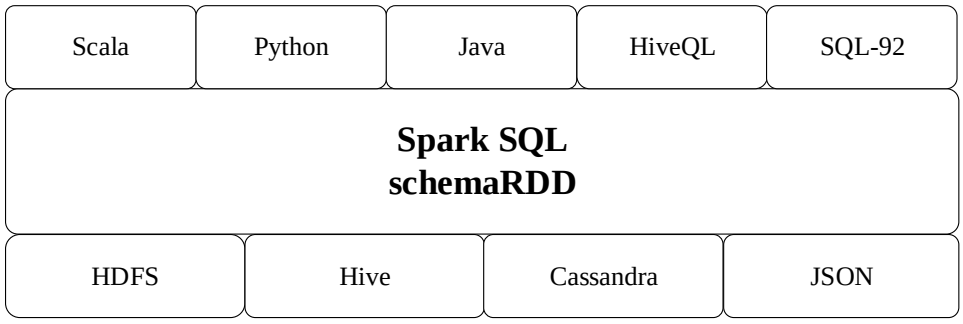


图 16-13 Spark SQL 支持的数据格式和编程语言

16.5Spark Streaming

Spark Streaming 是构建在 Spark 上的实时计算框架，它扩展了 Spark 处理大规模流式数据的能力。Spark Streaming 可结合批处理和交互查询，适合一些需要对历史数据和实时数据进行结合分析的应用场景。

16.5.1Spark Streaming 设计

Spark Streaming 是 Spark 的核心组件之一，为 Spark 提供了可拓展、高吞吐、容错的流计算能力。如图 16-14 所示，Spark Streaming 可整合多种输入数据源，如 Kafka、Flume、HDFS，甚至是普通的 TCP 套接字。经处理后的数据可存储至文件系统、数据库，或显示在仪表盘里。

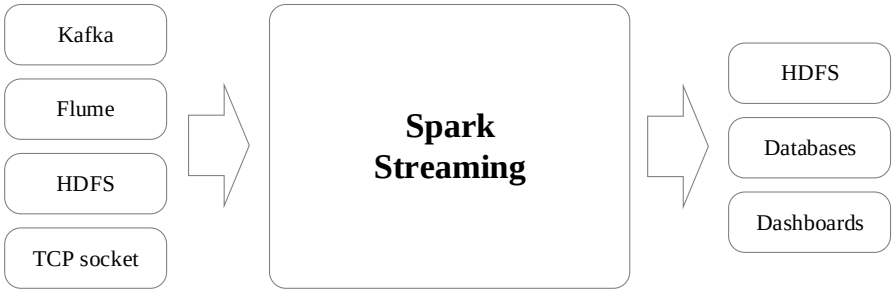


图 16-14 Spark Streaming 支持的输入、输出数据源

Spark Streaming 的基本原理是将实时输入数据流以时间片（秒级）为单位进行拆分，然后经 Spark 引擎以类似批处理的方式处理每个时间片数据，执行流程如图 16-15 所示。

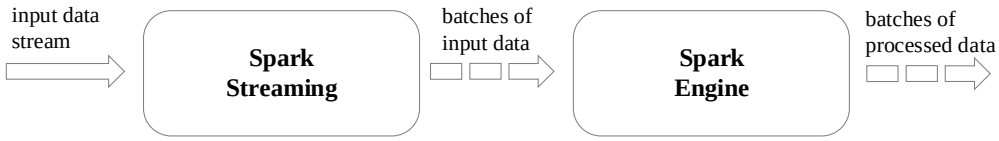


图 16-15 Spark Streaming 执行流程

Spark Streaming 最主要的抽象是 DStream（Discretized Stream，离散化数据流），表示连续不断的数据流。在内部实现上，Spark Streaming 的输入数据按照时间片（如 1 秒）分成一段一段的 DStream，每一段数据转换为 Spark 中的 RDD，并且对 DStream 的操作都最终转变为对相应的 RDD 的操作。例如，图 16-16 展示了进行单词统计时，每个时间片的数据（存储句子的 RDD）经 flatMap 操作，生成了存储单词的 RDD。整个流式计算可根据业务的需求对这些中间的结果进一步处理，或者存储到外部设备中。

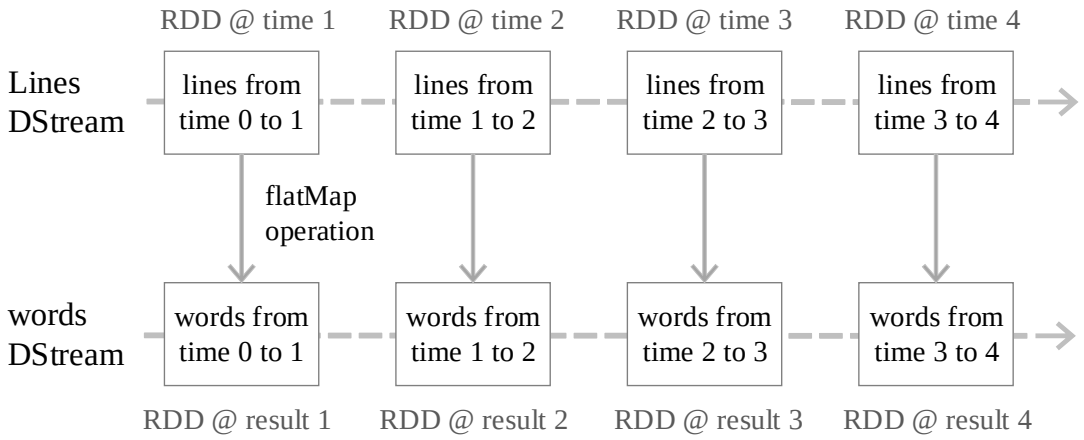


图 16-16 DStream 操作示意图

16.5.2Spark Streaming 与 Storm 的对比

Spark Streaming 和 Storm 最大的区别在于，Spark Streaming 无法实现毫秒级的流计算，而 Storm 可以实现毫秒级响应。

Spark Streaming 无法实现毫秒级的流计算，是因为其将流数据按 batch size（通常在 0.5~2 秒之间）分解为一系列批处理作业，在这个过程中，会产生多个 Spark Job，且每一段数据的处理都会经过 Spark DAG 图分解、任务调度过程，因此，无法实现毫秒级相应。Spark Streaming 难以满足对实时性要求非常高（如高频实时交易）的场景，但足以胜任其他流式准实时计算场景。相比之下，Storm 处理的单位为 Tuple，只需要极小的延迟。

Spark Streaming 构建在 Spark 上，一方面是因为 Spark 的低延迟执行引擎（100ms+）可以用于实时计算，另一方面，相比于 Storm，RDD 数据集更容易做高效的容错处理。此外，Spark Streaming 采用的小批量处理的方式使得它可以同时兼容批量和实时数据处理的逻辑和算法，因此，方便了一些需要历史数据和实时数据联合分析的特定应用场合。

16.6Spark 的部署和应用方式

本节首先介绍 Spark 支持的三种典型部署方式，即 standalone、Spark on Mesos 和 Spark on YARN；然后，介绍在企业中是如何具体部署和应用 Spark 框架的，在企业实际应用环境中，针对不同的应用场景，可以采用不同的部署应用方式，或者采用 Spark 完全替代原有的 Hadoop 架构，或者采用 Spark 和 Hadoop 一起部署的方式。

16.6.1Spark 三种部署方式

目前，Spark 支持三种不同类型的部署方式，包括 standalone、Spark on Mesos 和 Spark on YARN。

1.standalone 模式

与 MapReduce1.0 框架类似，Spark 框架本身也自带了完整的资源调度管理服务，可以独立部署到一个集群中，而不需要依赖其他系统来为其提供资源管理调度服务。在架构的设

计上，Spark 与 MapReduce1.0 完全一致，都是由一个 Master 和若干个 Slave 构成，并且以 slot 作为资源分配单位。不同的是，Spark 中的 slot 不再像 MapReduce1.0 那样分为 Map slot 和 Reduce slot，而是只设计了统一的一种 slot 提供给各种任务来使用。

2.Spark on Mesos 模式

Mesos 是一种资源调度管理框架，可以为运行在它上面的 Spark 提供服务。由于 Mesos 和 Spark 存在一定的血缘关系，因此，Spark 这个框架在进行设计开发的时候，就充分考虑到了对 Mesos 的充分支持，因此，相对而言，Spark 运行在 Mesos 上，要比运行在 YARN 上更加灵活、自然。目前，Spark 官方推荐采用这种模式，所以，许多公司在实际应用中也采用该模式。

3. Spark on YARN 模式

Spark 可运行于 Yarn 之上，与 Hadoop 进行统一部署，即“Spark on Yarn”，其架构如图 16-17 所示，资源管理和调度依赖 YARN，分布式存储则依赖 HDFS。

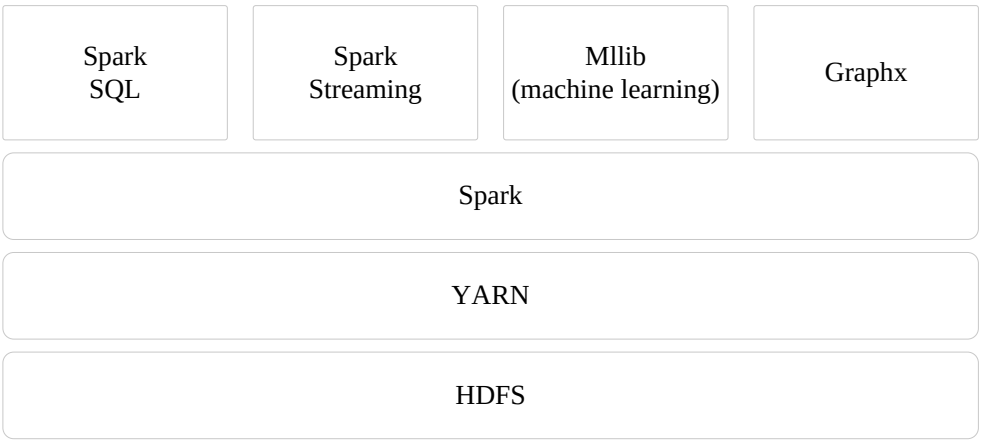


图 16-17 Spark on Yarn 架构

16.6.2 从 Hadoop+Storm 架构转向 Spark 架构

为了能同时进行批处理与流处理，企业应用中通常会采用 Hadoop+Storm 的架构（也称为 Lambda 架构）。图 16-18 给出了采用 Hadoop+Storm 部署方式的一个案例，在这种部署架构中，Hadoop 和 Storm 框架部署在资源管理框架 YARN（或 Mesos）之上，接受统一的资源管理和调度，并共享底层的数据存储（HDFS、HBase、Cassandra 等）。Hadoop 负责对批量历史数据的实时查询和离线分析，而 Storm 则负责对流数据的实时处理。

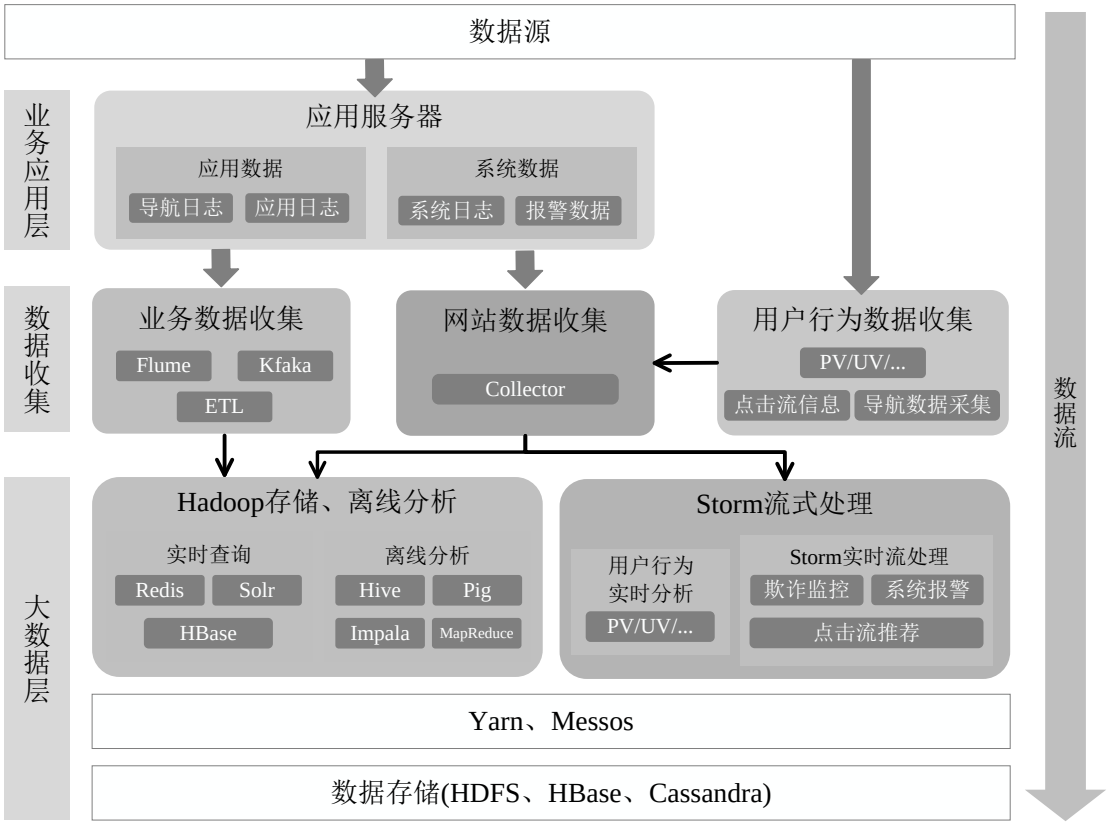


图 16-18 采用 Hadoop+Storm 部署方式的一个案例

但是，上面这种架构部署较为繁琐。由于 Spark 同时支持批处理与流处理，因此，对于一些类型的企业应用而言，从 Hadoop+Storm 架构转向 Spark 架构（如图 16-19 所示）就成为一种很自然的选择。采用 Spark 架构具有如下优点：

- 实现一键式安装和配置、线程级别的任务监控和告警；
- 降低硬件集群、软件维护、任务监控和应用开发的难度；
- 便于做成统一的硬件、计算平台资源池。

需要说明的是，正如前面介绍的那样，Spark Streaming 无法实现毫秒级的流计算，因此，对于需要毫秒级实时响应的企业应用而言，仍然需要采用流计算框架（如 Storm）。

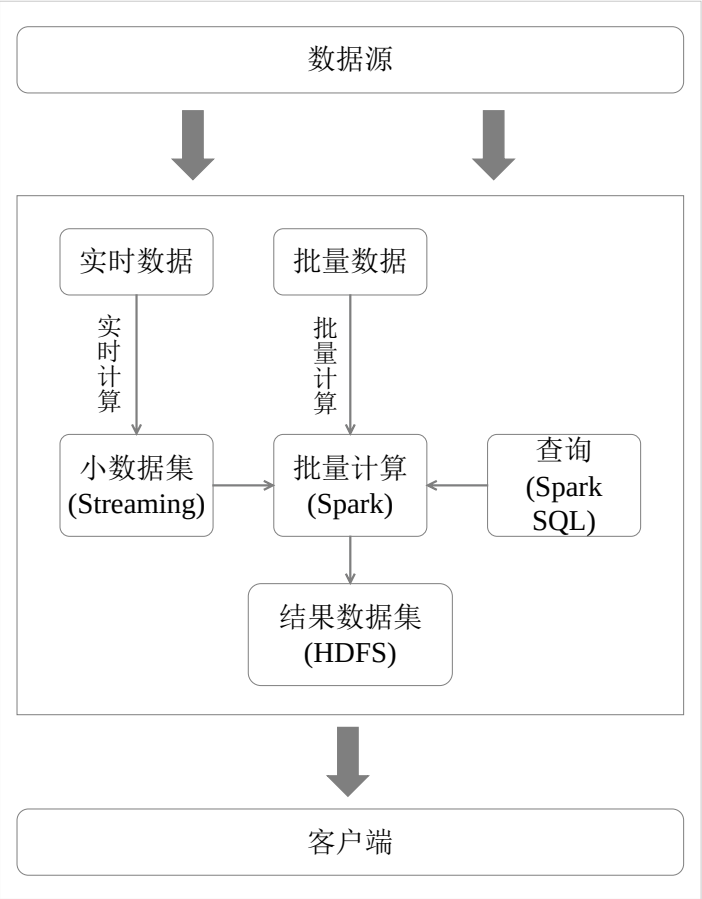


图 16-19 用 Spark 架构满足批处理和流处理需求

16.6.3Hadoop 和 Spark 的统一部署

一方面，由于 Hadoop 生态系统中的一些组件所实现的功能，目前还是无法由 Spark 取代的，比如，Storm 可以实现毫秒级响应的流计算，但是，Spark 则无法做到毫秒级响应。另一方面，企业中已经有许多现有的应用，都是基于现有的 Hadoop 组件开发的，完全转移到 Spark 上需要一定的成本。因此，在许多企业实际应用中，Hadoop 和 Spark 的统一部署是一种比较现实合理的选择。

由于 Hadoop MapReduce、HBase、Storm 和 Spark 等，都可以运行在资源管理框架 YARN 之上，因此，可以在 YARN 之上进行统一部署（如图 16-20 所示）。这些不同的计算框架统一运行在 YARN 中，可以带来如下好处：

- 计算资源按需伸缩；
- 不用负载应用混搭，集群利用率高；
- 共享底层存储，避免数据跨集群迁移。

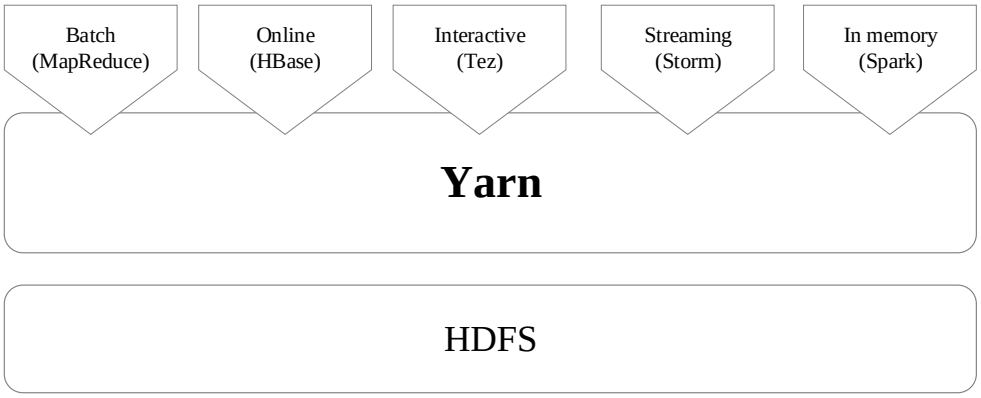


图 16-20 Hadoop 和 Spark 的统一部署

16.7Spark 编程实践

本节介绍 Spark 的安装和启动、Spark RDD 的基本操作、Spark SQL 与 DataFrame 的基本操作，最后介绍如何编译、打包、运行 Spark 应用程序。

16.7.1Spark 安装

Spark 的运行需要 Java 环境和 Hadoop 环境，因此，在安装 Spark 之前需要首先安装 Hadoop。之前的章节已介绍过 Linux 系统中 Hadoop 的安装，在此不再赘述，这里假定读者已在 Linux 系统中安装好了 Java 环境和 Hadoop 环境。

Spark 官网（http://spark.apache.org）提供了软件安装包的下载，进入官网后，可以点击主页右侧的“Download Spark”按钮进入下载页面，下载页面中提供了几个下载选项，主要是 Spark release 及 Package type 的选择，如图 16-21 所示。第 1 项 Spark release 一般默认选择最新的发行版本，如截止至 2016 年 3 月份的最新版本为 1.6.0。第 2 项 package type 则选择“Pre-build with user-provided Hadoop [can use with most Hadoop distributions]”，可适用于多数 Hadoop 版本。选择好之后，再点击第 4 项给出的链接就可以下载 Spark 了。

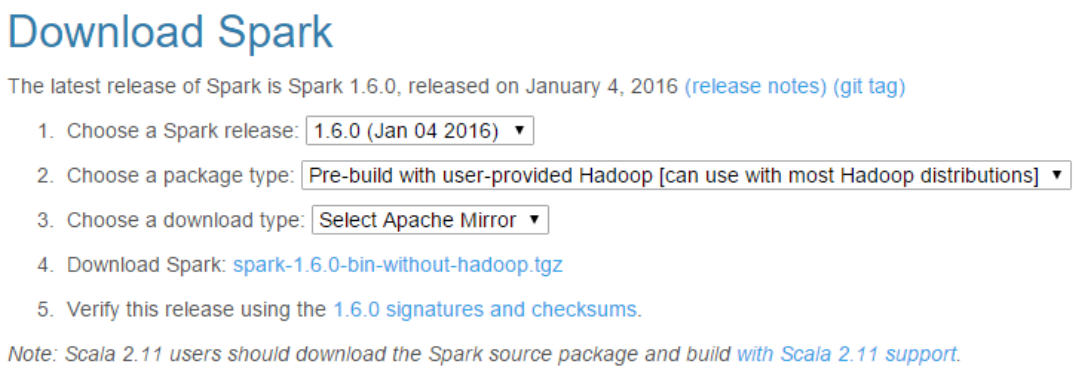


图 16-21 Spark 下载选项

下载后执行如下命令进行安装（假设下载路径为”~/下载”，安装在”/usr/local/spark”中）：

```
$ sudo tar -zxf ~/下载/spark-1.6.0-bin-without-hadoop.tgz -C /usr/local/    # 解压到 /usr/local 中
$ cd /usr/local
$ sudo mv ./spark-1.6.0-bin-without-hadoop/ ./spark     # 更改文件夹名
$ sudo chown -R hadoop ./spark                          # 此处的 hadoop 为系统用户名
```

解压安装后，需设置 Spark 的 Classpath。首先执行如下命令，拷贝配置文件：

```
$ cd /usr/local/spark
$ cp ./conf/spark-env.sh.template ./conf/spark-env.sh
```

接着编辑该配置文件，在文件最后面加上如下一行内容：

```
export SPARK_DIST_CLASSPATH=$(/usr/local/hadoop/bin/hadoop classpath)
```

保存配置文件后，Spark 就可以启动运行了。Spark 包含多种运行模式，可使用单机模式，也可以使用伪分布式、完全分布式模式，为简单起见，这里使用单机模式运行 Spark。此外，如果需要使用 HDFS 中的文件，则在使用 Spark 前需要启动 Hadoop。

16.7.2 启动 Spark Shell

Spark shell 提供了简单的方式来学习 Spark API，且能以实时、交互的方式来分析数据。Spark Shell 支持 Scala 和 Python，这里选择使用 Scala 进行编程实践，了解 Scala 有助于更好地掌握 Spark。

执行如下命令启动 Spark Shell：

```
$ ./bin/spark-shell
```

启动 Spark shell 成功后在输出信息的末尾可以看到“Scala >”的命令提示符，如**错误！未找到引用源。**-22 所示。

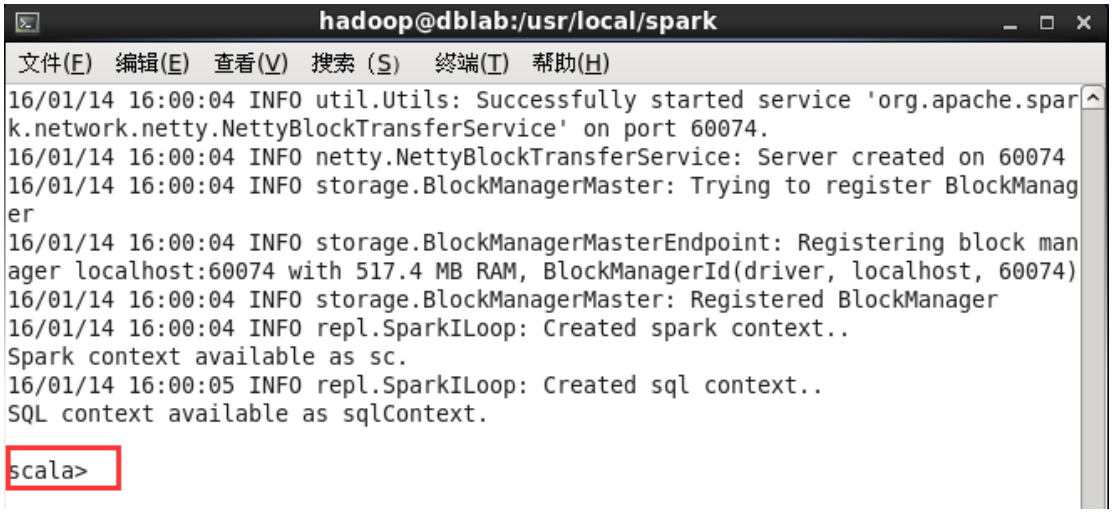


图 16-22 启动 Spark Shell

16.7.3Spark RDD 基本操作

Spark 的主要操作对象是 RDD，RDD 可以通过多种方式灵活创建，可通过导入外部数据源建立（如位于本地或 HDFS 中的数据文件），或者从其他的 RDD 转化而来。

在 Spark 程序中必须创建一个 SparkContext 对象，该对象是 Spark 程序的入口，负责创建 RDD、启动任务等。在启动 Spark Shell 后，该对象会自动创建，可以通过变量 sc 进行访问。作为示例，这里选择以 Spark 安装目录中的“README.md”文件作为数据源新建一个 RDD，代码如下（后续出现的 Spark 代码中，“Scala >”表示一行代码的开始，与代码位于同一行的注释内容表示该代码的说明，代码下面的注释内容表示交互式输出结果）：

```
Scala > val textFile = sc.textFile("file:///usr/local/spark/README.md") // 通过 file:前缀指定读取本地文件
```

Spark RDD 支持两种类型的操作：

- 动作（action）：在数据集上进行运算，返回计算值；
- 转换（transformation）： 基于现有的数据集创建一个新的数据集。

Spark 提供了非常丰富的 API，表 16-2 和表 16-3 仅列出了几个常用的动作、转换 API，更详细的 API 及说明可查阅官方文档。

表 16-2 常用的几个 Action API 介绍

Action API	说明
count()	返回数据集中的元素个数
collect()	以数组的形式返回数据集中的所有元素
first()	返回数据集中的第一个元素
take(n)	以数组的形式返回数据集中的前 n 个元素
reduce(func)	通过函数 func（输入两个参数并返回一个值）聚合数据集中的元素
foreach(func)	将数据集中的每个元素传递到函数 func 中运行

表 16-3 常用的几个 Transformation API 介绍

Transformation API	说明
filter(func)	筛选出满足函数 func 的元素，并返回一个新的数据集
map(func)	将每个元素传递到函数 func 中，并将结果返回为一个新的数据集
flatMap(func)	与 map()相似，但每个输入元素都可以映射到 0 或多个输出结果
groupByKey()	应用于(K,V)键值对的数据集时，返回一个新的(K, Iterable<V>)形式的数据集
reduceByKey(func)	应用于(K,V)键值对的数据集时，返回一个新的(K, V)形式的数据集，其中的每个值是将每个 key 传递到函数 func 中进行聚合

例如，在下面的实例中，使用 count()这个 Action API 就可以统计出一个文本文件的行数，命令如下（输出结果“Long=95”表示该文件共有 95 行内容）：

```
Scala > textFile.count()
// Long = 95
```

再比如，在下面实例中，使用 filter()这个 Transformation API 就可以筛选出只包含“Spark”的行，命令如下（第一条命令会返回一个新的 RDD，因此不影响之前 RDD 的内容；输出结果“Long=17”表示该文件中共有 17 行内容包含“Spark”）：

```
Scala > val linesWithSpark = textFile.filter(line => line.contains("Spark"))
Scala > linesWithSpark.count()
// Long = 17
```

在上面计算过程中，中间输出结果采用 linesWithSpark 变量进行保存，然后再使用 count()计算出行数。假设这里只需要得到包含“Spark”的行数，而不需要了解每行的具体内容，那么，使用 linesWithSpark 变量存储筛选后的文本数据就是多余的，因为这部分数据在计算得到行数后就不再使用到了。实际上，借助于强大的链式操作（即在同一条代码中同时使用多个 API），Spark 可连续进行运算，一个操作的输出直接作为另一个操作的输入，不需要采用临时变量存储中间结果，这样不仅可以使 Spark 代码更加简洁，也优化了计算过程。如上述两条代码可合并为如下一行代码：

```
Scala > val linesCountWithSpark = textFile.filter(line => line.contains("Spark")).count()
// Long = 17
```

从上面代码可以看出， Spark 基于整个操作链，仅储存、计算所需的数据，提升了运行效率。

Spark 属于 MapReduce 计算模型，因此也可以实现 MapReduce 的计算流程，如实现单词统计，可以首先使用 flatMap()将每一行的文本内容通过空格进行划分为单词；然后，使用 map()将单词映射为(K,V)的键值对，其中 K 为单词，V 为 1；最后，使用 reduceByKey()将相同单词的计数进行相加，最终得到该单词总的出现的次数。具体实现命令如下：

```
Scala > val wordCounts = textFile.flatMap(line => line.split(" ")).map(word => (word,
```

```
1)).reduceByKey((a, b) => a + b)
Scala > wordCounts.collect() // 输出单词统计结果
// Array[(String, Int)] = Array((package,1), (For,2), (Programs,1), (processing.,1), (Because,1),
(The,1)...)
```

16.7.4Spark SQL、DataFrame 基本操作

Spark SQL 是 Spark 内嵌的模块，主要用于处理结构化数据，其前身是 Shark，而 Shark 的前身则是 Hadoop 中的 Hive。Spark 从 1.3 版本开始，在其 API 中加入了 DataFrame，其目的是为列表数据处理提供更好的支持。DataFrame 是一个以命名列方式组织的分布式数据集，等同于关系型数据库中的一个表。DataFrame 提供了通用的方式来连接多种数据源，现已支持 Hive、MySQL、PostgreSQL 等外部数据源，支持使用 JDBC 连接数据库，也支持 JSON 格式的数据源，并且可以在多种数据源之间执行连接（join）操作。

在启动 Spark Shell 时，会初始化一个对象 sqlContext，通过该对象来使用 Spark SQL 的功能。使用 sqlContext 可以从现有的 RDD 或其他数据源创建 DataFrames，作为示例，这里以 Spark 提供的一个 JSON 格式的数据源文件（./examples/src/main/resources/people.json）来进行演示，该数据源内容如下：

```
{"name":"Michael"}
{"name":"Andy", "age":30}
{"name":"Justin", "age":19}
```

执行如下命令导入数据源，创建一个 DataFrame，并输出内容：

```
Scala > val df =
sqlContext.read.json("file:///usr/local/spark/examples/src/main/resources/people.json")
Scala > df.show()        // 输出数据源内容
// +----+-------+
// | age |    name|
// +----+-------+
// |null | Michael|
// | 30 |    Andy |
// | 19 |    Justin |
// +----+-------+
```

DataFrame 提供了丰富、简明的 API 来操作数据，如使用如下的命令可实现通常 SQL 中的 where 条件语句：

```
Scala > df.filter(df("age") > 21).show()        // 相当于 select * from table where age > 21
// +----+-------+
// | age | name |
// +----+------+
```

```
// | 30 | Andy |
// +----+------+
```

而将 DataFrame 注册为临时表后，可以使用 SQL 语句来进行操作，命令如下：

```
Scala > df.registerTempTable("people")        // 将 DataFrame 注册为临时表 people
Scala > val result = sqlContext.sql("SELECT name, age FROM people WHERE age >= 13 AND
age <= 19")
Scala > result.show()
// +------+---+
// | name| age|
// +------+---+
// |Justin | 19 |
// +------+---+
```

此外，DataFrame 也包含了丰富的函数用于字符串处理、日期计算、数学计算等，大大方便了对结构化数据的操作，在此不再赘述，详细的 API 可查阅 Spark 官网文档。

16.7.5Spark 应用程序

Spark 应用程序支持采用 Scala、Python、Java、R 等语言进行开发。在 Spark Shell 中进行交互式编程时，可以采用 Scala 和 Python 语言，主要是方便对代码进行调试，但需要以逐行代码的方式运行。一般等到代码都调试好之后，可选择将代码打包成独立的 Spark 应用程序，然后提交到 Spark 中运行。如果不是在 Spark Shell 中进行交互式编程，比如使用 Java 语言进行 Spark 应用程序开发，也需要编译打包后再提交给 Spark 运行。采用 Scala 编写的程序，需要使用 sbt 进行编译打包；采用 Java 编写的程序，建议使用 Maven 进行编译打包；采用 Python 编写的程序，可以直接通过 spark-submit 提交给 Spark 运行。

sbt（Simple Build Tool）是对 Scala 或 Java 语言进行编译的一个工具，类似于 Maven 或 Ant，需要 JDK1.6 或更高版本的支持，并且可以在 Windows 和 Linux 两种环境下安装使用。sbt 需要下载安装，可以访问 http://pan.baidu.com/s/1eRyFddw 下载 sbt-launch.jar，保存到下载目录。假设下载目录为“~/下载”，安装目录为“/usr/local/sbt”，则执行如下命令将下载后的文件拷贝至安装目录中：

```
sudo mkdir /usr/local/sbt                  # 创建安装目录
cp ~/下载/sbt-launch.jar /usr/local/sbt   //把下载目录下的安装文件复制到安装目录下
sudo chown -R hadoop /usr/local/sbt     # 此处的 hadoop 为系统当前用户名
```

接着在安装目录中创建一个 Shell 脚本文件（vim /usr/local/sbt/sbt）用于启动 sbt，该脚本文件中的代码如下：

```
#!/bin/bash
SBT_OPTS="-Xms512M -Xmx1536M -Xss1M -XX:+CMSClassUnloadingEnabled
-XX:MaxPermSize=256M"
```

```
java $SBT_OPTS -jar `dirname $0`/sbt-launch.jar "$@"
```

保存后，还需要为该 Shell 脚本文件增加可执行权限：

```
chmod u+x /usr/local/sbt/sbt
```

现在，就可以使用“/usr/local/sbt/sbt package”的命令来打包 Scala 编写的 Spark 程序了。

这里以一个简单的程序为例，介绍如何打包并运行 Spark 程序，该程序的功能是统计文本文件中包含字母 a 和字 b 的各有多少行。首先执行如下命令创建程序根目录，并创建程序所需的文件夹结构：

```
mkdir ~/sparkapp                          # 创建程序根目录
mkdir -p ~/sparkapp/src/main/scala     # 创建程序所需的文件夹结构
```

接着创建一个 SimpleApp.scala 文件（vim ~/sparkapp/src/main/scala/SimpleApp.scala），该文件是程序的代码内容，具体代码如下：

```
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf

object SimpleApp {
  def main(args: Array[String]) {
    val logFile = "file:///usr/local/spark/README.md" // 用于统计的文本文件
    val conf = new SparkConf().setAppName("Simple Application")
    val sc = new SparkContext(conf)
    val logData = sc.textFile(logFile, 2).cache()
    val numAs = logData.filter(line => line.contains("a")).count()
    val numBs = logData.filter(line => line.contains("b")).count()
    println("Lines with a: %s, Lines with b: %s".format(numAs, numBs))
  }
}
```

然后创建一个 simple.sbt 文件（vim ~/sparkapp/simple.sbt），该文件用于声明该应用程序的信息以及与 Spark 的依赖关系，具体内容如下：

```
name := "Simple Project"
version := "1.0"
scalaVersion := "2.10.5"
libraryDependencies += "org.apache.spark" %% "spark-core" % "1.6.0"
```

最后，执行如下命令使用 sbt 进行打包：

```
cd ~/sparkapp
```

```
/usr/local/sbt/sbt package
```

打包成功后，如图 16-23 所示会输出程序 jar 包的位置以及“Done Packaging”的提示。

图 16-23 使用 sbt 打包 Spark 程序

有了最终生成的 jar 包后，再通过 spark-submit 就可以提交到 Spark 中运行了，命令如下：

```
/usr/local/spark/bin/spark-submit --class "SimpleApp" ~/sparkapp/target/scala-2.10/simple-project_2.10-1.0.jar
```

该应用程序的执行结果如下：

```
Lines with a: 58, Lines with b: 26
```

本章小结

本章首先介绍了 Spark 的起源与发展，分析了 Hadoop 存在的缺点与 Spark 的优势。接着介绍了 Spark 的相关概念、生态系统与核心设计。Spark 的核心是统一的抽象 RDD，在此之上形成了结构一体化、功能多元化的完整的大数据生态系统，支持内存计算，SQL 既席查询、实时流式计算、机器学习和图计算。

本章最后介绍了 Spark 基本的编程实践，包括 Spark 的安装与 Spark Shell 的使用，并演示了 Spark RDD、Spark SQL 的基本操作。Spark 提供了丰富的 API，让开发人员可以用简洁的方式来处理复杂的数据计算与分析。

习题

1. Spark 是基于内存计算的大数据计算平台，试述 Spark 的主要特点。
2. Spark 的出现是为了解决 Hadoop MapReduce 的不足，试列举 Hadoop MapReduce 的几个缺陷，并说明 Spark 具备哪些优点。
3. 伯克利大学提出的数据分析的软件栈 BDAS 认为目前的大数据处理可以分为哪三个类型？

4. Spark 已打造出结构一体化、功能多样化的大数据生态系统，试述 Spark 的生态系统。
5. 从 Hadoop+Storm 架构转向 Spark 架构可带来哪些好处？
6. 试述“Spark on Yarn”的概念？
7. 试述如下 Spark 的几个主要概念：RDD、DAG、Stage、Partition、Narrow dependency、Wide dependency。
8. 试述 Spark Streaming 的基本原理？
9. Spark Streaming 和 Storm 都可用于流计算，两者最大的区别是？
10. Spark 对 RDD 的操作主要分为动作（action）和转换（transformation）两种类型，两种操作的区别是什么？
11. 试列举 Spark SQL 支持的几种外部数据源。

附录 1:任课教师介绍

林子雨(1978－),男,博士,厦门大学计算机科学系助理教授,主要研究领域为数据库,实时主动数据仓库,数据挖掘.
主讲课程：《大数据技术基础》
办公地点：厦门大学海韵园科研 2 号楼
E-mail: ziyulin@xmu.edu.cn
个人主页：http://www.cs.xmu.edu.cn/linziyu
数据库实验室网站：http://dblab.xmu.edu.cn

附录 2：课程教材介绍

《大数据技术原理与应用——概念、存储、处理、分析与应用》，由厦门大学计算机科学系教师林子雨博士编著，是中国高校第一本系统介绍大数据知识的专业教材。本书定位为大数据技术入门教材，为读者搭建起通向“大数据知识空间”的桥梁和纽带，以“构建知识体系、阐明基本原理、引导初级实践、了解相关应用”为原则，为读者在大数据领域“深耕细作”奠定基础、指明方向。

全书共有 13 章，系统地论述了大数据的基本概念、大数据处理架构 Hadoop、分布式文件系统 HDFS、分布式数据 库 HBase、NoSQL 数据库、云数据库、分布式并行编程模型 MapReduce、流计算、图计算、数据可视化以及大数据在互联网、生物医学和物流等各个领域的应用。在 Hadoop、HDFS、HBase 和 MapReduce 等重要章节，安排了入门级的实践操作，让读者更好地学习和掌握大数据关键技术。

本书可以作为高等院校计算机专业、信息管理等相关专业的大数据课程教材，也可供相关技术人员参考、学习、培训之用。

欢迎访问《大数据技术原理与应用——概念、存储、处理、分析与应用》教材官方网站：

http://dblab.xmu.edu.cn/post/bigdata

扫一扫访问教材官网

附录 3：中国高校大数据课程公共服务平台介绍

中国高校大数据课程
公 共 服 务 平 台

中国高校大数据课程公共服务平台，由中国高校首个“数字教师”的提出者和建设者——林子雨老师发起，由厦门大学数据库实验室全力打造，由厦门大学云计算与大数据研究中心、海峡云计算与大数据应用研究中心携手共建。这是国内第一个服务于高校大数据课程建设的公共服务平台，旨在促进国内高校大数据课程体系建设，提高大数据课程教学水平，降低大数据课程学习门槛，提升学生课程学习效果。平台服务对象涵盖高校、教师和学生。平台为高校开设大数据课程提供全流程辅助，为教师开展教学工作提供一站式服务，为学生学习大数据课程提供全方位辅导。平台重点打造“9 个 1 工程”，即 1 本教材（含官网）、1 个教师服务站、1 个学生服务站、1 个公益项目、1 堂巡讲公开课、1 个示范班级、1 门在线课程、1 个交流群（QQ 群、微信群）和 1 个保障团队。

平台主页：http://dblab.xmu.edu.cn/post/bigdata-teaching-platform/

扫一扫访问平台主页