

S00036ro! 6H01010007510179050CD10051710960.0forer015101CIQ(1tof016

0t0i⁰10 6!30100!((010

0i⁰0 16Cf0.030 e

L6010gl00100C01e6.0uk 0

**第13章** **Spark 及其生态系统**



**13.1** **简** **介**

Apache Spark 是一个开源的大数据处理框架，它与Hadoop 并驾齐驱，是当前主流的 大数据处理框架之一。Spark 于2009年由加利福尼亚大学伯克利分校 (UC Berkeley) 的 AMP 实验室开发，并且在2010年开源，成为一个Apache 项目。Spark 是一个速度快、 易用、通用的集群计算系统，能够与Hadoop 生态系统和数据源良好地兼容。目前，很多 机构和组织已经在大规模集群上(上千节点)运行 Spark, 对它们的数据进行分析。根据 Spark FAQ(Frequently Asked Questions), 到目前为止，实际部署的最大的集群达到 8000个节点。

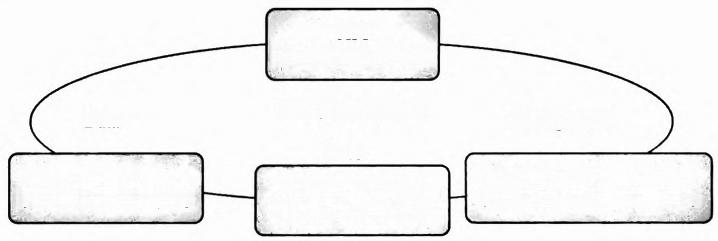
用户可以使用Shell 程序，交互式地对数据进行查询。除了简单的Map 和 Reduce 操 作 ，Spark 本身提供了超过80个数据处理的操作原语 (Operator Primitive), 方便用户编 写数据处理程序。 Spark 提供了 Java,Scala 以 及 Python 语言的应用程序编程接口 (API) 。 用户可以使用这些操作，完成SQL 查询、流数据处理、机器学习以及图数据处 理，还可以在一个数据处理的工作流 (Data Processing Workflow) 中把这些功能整合 起来。

**13.1.1** **Spark软件架构**

Spark 软件架构包含4个主要部分，分别是Spark 的核心组件和相关组件、数据存储 (Data Storage)、应用程序编程接口 (API) 和资源管理框架 (Management Framework) (见图13- 1)。其中，Spark 的核心组件和相关组件包括Spark Core,Spark Streaming, Spark SQL,GraphX,MLLib 等组件，将在下文详细介绍。

在数据存储方面，Spark 一般使用HDFS 分布式文件系统存储数据，它支持Hbase, Cassandra 等数据源。Spark 的编程接口包括Scala API,Java API 以 及Python API 等，





API

(Scala,Java,Python)

Compute Interface

Base

Spark Core &

Related Components

Management

Distributed Computing

(Standalone,YARN,Mesos)

**Data**

**Storage**

**(HDFS,other formats)**

**图13-1** **Spark 软件架构**

使得开发人员可以使用Java,Scala 以 及Python 等语言进行编程。在资源管理框架方面， Spark 可以独立部署 (Standalone Mode Mode), 或者部署到YARN 或 者Mesos 等资源管 理框架上。

**13.1.2 Spark 的主要优势**

相对于 Hadoop 系统，Spark 的主要优势包括：

(1)数据类型与计算的表达能力。首先， Spark 可以管理各种类型 (Variety) 的 数 据集，包括文本数据、图数据等。它是一个通用的平台，支持以DAG ( 有 向 无 环 图 ) 形式表达的复杂的计算，支持批处理、流数据处理、图数据处理、机器学习等众多应用 场景。

(2)数据处理速度。当数据完全驻留于内存时， Spark 的数据处理速度达到 Hadoop 系统的几十至上百倍。当数据保存在磁盘上时，需要从磁盘装载数据以后才能进行处理， 它的处理速度能够达到Hadoop 系统的10倍左右。

2014年， Spark 击败Hadoop, 在1/10的节点数量上，以快三倍速度完成100TB 数 据的排序 (Daytona Gray Sort Contest)。同时，它也是到目前为止在PB 级数据排序方面 最快的开源引擎，具体情况如下。

2014年， Spark 赢 得 了 Daytona Gray Sort 比 赛 (<http://sortbenchmark.org/)。>

Databricks 公 司 ( 对 Spark 进行商业化的公司)的研究人员在206台 EC2 虚拟机上用23 分钟完成了100TB 数据的排序。之前， Hadoop/MapReduce 创造的100TB 数据的排序纪 录，是使用2100台机器耗时72分钟完成的。排序是在磁盘数据上进行的 (HDFS), 没 有使用内存Cache 。 这个结果显示， Spark 能够使用更少的硬件资源(只用10%),对数据 进行更快(快3倍)的排序。

赢得这个比赛是Spark 发展的重要里程碑。为此， Databricks公司对Spark 平台进行 了多方面的改进，采用了一系列新技术，包括Sort-based Shuffle,Netty-based Transport Module,External Shuffle Service等 。

此外，虽然没有PB 级别的比赛，但是Databricks 公司接着进行了PB 级(10 Trillion 条记录)数据的排序实验，在190台机器上耗时4个小时完成排序。这个时间比之前Ha- doop/MapReduce 的结果(在3800台机器上耗时16个小时完成排序)快了4倍。表13- 1给出了排序实验的参数设置以及性能指标，并且和之前Hadoop/MapReduce 的结果进行 了比较。



**表13-1** **Hadoop 和Spark 100TB 数据排序结果的比较**

|  |  |  |  |
| --- | --- | --- | --- |
|  | 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 nodes | 6592 virtualized nodes | 6080 virtualized nodes |
| Cluster disk throughput | 3150 GB/s(est.) | 618GB/s | 570GB/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 |

资料来源：Databricks.

Spark 开源社区和 Databricks 公司的研发人员从各个方面持续改进 Spark, 包括其扩 展性、可靠性以及性能等。现在，当整个数据集不能完全放到内存里时， Spark 的操作符 可以完成外部(磁盘)操作。 PB 排序结果显示.Spark 已经能够超越整个集群所有内存的 限制，对更大的数据集进行处理。

需要注意的是，并不是所有的数据处理任务都能够达到这样的加速比。随着 Hadoop 新版本的推出 (YARN), 以及新的查询处理引擎(比如 Tez) 的设计和实现， Hadoop 和 Spark 的差距在缩小。

**13.2 Hadoop的局限和Spark的诞生**

Hadoop 作为大数据处理的有效工具已经存在了十年，从1.0版本发展到了2.0版本。 传统的 Hadoop 平台使用MapReduce 计算模型对数据进行处理。对于只需一遍扫描处理 的计算任务((ne Pass Computation) 来讲，MapReduce 计算模型非常有效，但是某些计 算任务需要在数据上进行多遍扫描和处理 (Multi Pass Computation),MapReduce 计算模 型的效率就大大降低了。

在 Hadoop/MapReduce 平台上数据处理流程分为Map 阶段和Reduce 阶段，人们需要 把任何计算任务转换成 MapReduce 计算模式才能使用这套系统，也就是当我们需要执行 某些复杂的数据处理任务时，需要把它翻译成一系列的 MapReduce 作业，然后依次执行 这些作业。在计算过程的各个步骤之间，各个作业.(Job) 输出的中间结果需要存储到分 布式文件系统，然后才能被下一个步骤使用。因为多副本复制和磁盘存取的关系， Ha- doop 执行翻译成多个MapReduce 作业的计算任务时效率不高。

从计算模型来看， Spark 把计算任务表达成 DAG (有向无环图， Directed Acyclic Graph) 作业，从而把整个数据处理流程的多个步骤完整地表达出来。因此， Spark 能够 支持对数据进行复杂的处理操作，无须像 Hadoop 那样，把复杂任务分解成一系列的Ma- pReduce 作业。Spark 支持把数据驻留在内存中.作业(表达为 DAG 形式)的多个阶段

间实现数据不落地(写入磁盘)的流水线 (Pipeline) 处理。Spark 不仅能够用DAG 更加 自然地表达各种复杂计算，而且其性能也比 Hadoop 高。

这并不是说Hadoop 大数据处理系统毫无意义。从某种意义上讲，正是 Hadoop 的成 功和不足催生了Spark 这样的系统。Hadoop 系统本身也在不断发展，包括新的计算模型 的支持，比如流数据处理、图数据处理以及 DAG 表达的计算任务的处理。我们应该把 Spark 看作Hadoop 的一个备选项，而不是完全地替代Hadoop 本身。

目 前 ，Spark 可以在 Hadoop 分布式文件系统HDFS 上运行，把 HDFS 作为数据存储 使用。Spark 应用程序可以部署到 Hadoop 2.0YARN 集群，或者Apache Mesos 集群上， YARN 和 Mesos 是可以互相替换的资源管理和调度软件。

**13.3 Spark特性总结**

(1)Spark 的数据处理速度比 Hadoop 要快很多，这主要得益于其基于内存的数据处 理技术。Spark 本身支持内存数据集的处理，以及存储在磁盘上的数据集的处理。 Spark 尽量把更多的数据驻留在内存中，必要时才把数据写入磁盘。它支持把部分数据保存在内 存中，剩下的部分保存在磁盘上。此外，对于中间结果，Spark 也把它保存在内存中而不 是写人磁盘，对于需要对数据进行多次处理操作的计算任务来讲，避免I/O 操作，可以极 大提高处理效率。Spark 的基于内存的数据处理技术，使它获得比其他大数据处理框架更 高的性能。

(2)Spark 和 Hadoop 生态系统兼容，能够集成和操作 HDFS,Amazon S3,Hive. HBase,Cassandra 等数据源。Spark 可以运行在 Hadoop YARN 或者Apache Mesos 上， 也可以独立运行 (Standalone Mode)。

(3)除了Map 函数和Reduce 函数，Spark 提供其他大量的数据操作原语，使用这些 操作可以表达复杂的计算任务。

(4)Spark 可以对任意DAG 表达的算法进行优化。

(5)对于数据查询操作， Spark 采用延迟计算的方式 (Lazy Evaluation) 来执行，从 而帮助优化器对整个数据处理的工作流进行优化。

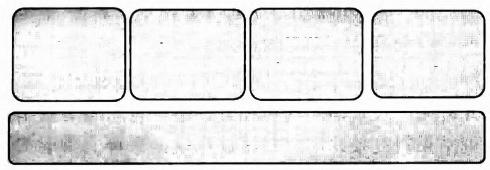
(6)Spark 系统使用Scala语言编写，经过编译以后在JVM(Java Virtual Machine) 上运行。Spark 系统提供了简洁、一致的编程接口，支持各种编程语言。目前Spark 支持 的开发语言包括Scala,Java,Python.Clojure,R 等。

(7)Spark 提供了一个交互式的Shell, 用户可以使用Scala 或者Python 语言，交互式 地对数据进行操作，并且马上看到结果。

**13.4 Spark生态系统**

整个 Spark 软件系统包含核心模块 (Spark Core) 以及若干数据处理分析模块，这些 模块一起构成整个 Spark 大数据处理的生态环境 (Ecosystem) (见图13-2)。

\



MLlib

(Machine learning)

Apache Spark Core

Spark

Streaming

GraphX

(Graph)

**Spark**

SQL

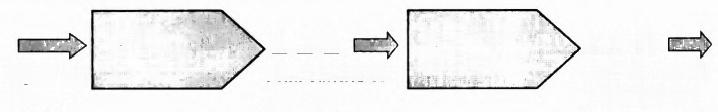
**图13-2** **Spark 生态系统(主要模块)**

**1.Spark 核心模块** **(Spark Core)**

Spark 核心模块是整个系统进行大规模并行和分布式数据处理的基础。它的主要功能 是内存管理和容错保证、集群环境下的作业调度和监控，以及和存储系统的接口和交互 等。核心模块上的其他模块，介绍如下。

**2. 流数据处理模块** **(Spark Streaming)**

用于处理实时流数据，比如 Web 服务器的日志文件 (Log File)、社交媒体如 Twitter 以及各种消息队列如 Kafka 消息队列等。它采用小批量 (Micro Batch) 数据处理的方式， 即把接收的数据流分解成一系列小的 RDD, 交 给 Spark 引擎进行处理，实现流数据处理， 其结果也是以批量 (In Batch) 的方式生成的(见图13 - 3)。



■■□

Mini batches of input data

□□ □

Mini batches of processed data

Spark Core Engine

**Spark**

**Streaming**

Input data Stream

**图13-3** **Spark Streaming**

**3.SQL 查询与结构化数据处理** **(Spark SQL)**

该模块当初的目的是试图把Apache Hive移植到Spark 平台上运行，现在它已经是 Spark 生态系统的主要模块之一。该模块把 Spark 数据集通过JDBC API暴露出来，让客 户程序可以在上面运行SQL 查询，也可以把传统的BI(Business Intelligence) 和可视化 工具连接到该数据集，利用JDBC 进行数据查询、汇总和展示等。Spark SQL 模块支持不 同外部数据源(比如JSON、Parquet 列存储以及关系数据库等)的导入、转换和装载， 并且支持即席 (Ad-Hoc) 查询。

**4. 机器学习** **(Spark MLlib)**

MLlib 是Spark 生态系统里可扩展的机器学习模块，它实现了众多的算法，包括分类 (Classification) 、 回 归 (Regression) 、 聚 类 (Clustering) 、 协同过滤 (Collaborative Filte- ring) 、降维 (Dimensionality Reduction) 等。

由于Spark 核心模块的高性能和通用性，目前Apache Mahout(Hadoop 平台上的一 个机器学习库)已经放弃MapReduce 计算模型，转而选择Spark 作为其执行引擎。

**5. 图数据处理** **(Spark GraphX)**

GraphX 支持图数据的并行处理。用户可以利用该模块对图数据进行探索式分析 (Ex- ploratory Analysis) 以及迭代式计算 (Iterative Graph Computation)。

GraphX 对 RDD 进行了扩展，称为 RDPG(Resilient Distributed Property Graph)。 RDPG 是一个把属性赋予各个节点和各条边的有向图 (Multi Graph)。为了支持图数据的 处 理 ，GraphX 提供了 一 系列的操作供用户使用，包括子图 (Sub Graph)、顶 点 连 接 (Join Vertices)、消息聚集 (Aggregate Message) 等，还提供了Pregel(Google 公司的图

数据处理软件) API 的变种。在此基础上，GraphX 包含了经典的图处理算法，比如Pag- eRank 等，方便在此之上开发更加复杂的图数据分析软件。

此外，BlinkDB 是一个近似查询处理引擎，可以在大规模数据集上交互式地执行SQL 查询。这个查询引擎允许用户在查询的精度和查询响应时间之间做出折中。用户可以指定 查询响应时间或者查询结果精度要求，BlinkDB 在数据采样 (Data Sample) 上执行查询， 获得一个近似结果。查询结果打上一个错误范围 (Error Bar) 标签，帮助用户做出决断。

Tachyon 是一款基于内存的分布式文件系统，支持不同集群计算框架之间的(比如 Spark 和 MapReduce) 可靠的文件共享。它把工作集 (Workset) 的文件缓存在内存中，避免 了对磁盘的存取。如果数据集不断被扫描和处理，那么数据处理的速度将获得极大的提升。 Tachyon 支持不同的上层查询处理框架，以极高的速度对集群内存里的文件进行访问。

整个Spark 生态环境还集成了一系列的适配软件模块，包括Spark Cassandra Connector, SparkR 等，以支持Spark 系统和Cassandra,R 等软件系统的协同工作。比如通过Cassandra Connector, 可以使用Spark 去存取存在Cassandra数据库里的数据，进行必要的数据分析。

**13.5** **RD** **D及其处理**

弹性分布数据集 (Resilient Distributed Dataset,RDD) 是 Spark 软件系统的核心概 念，它是一个容错的、不可更新的 (Immutable) 分布式数据集.支持并行处理。

简单来讲，RDD可以看作数据库里的一张表，但是它可以存放任何类型的数据。Spark 把隶属于一个RDD的数据划分成不同的分区 (Partition), 分区是RDD的下级概念。对RDD 进行分区，分布到集群环境，这样的数据组织方式有利于对数据进行并行处理。

RDD 采用基于血统 (Lineage) 的容错机制，也就是它记住每个 RDD 是如何从其他 RDD 转换 (Transformation) 而来的。当某个 RDD 损坏时，Spark 系统从上游RDD 重新 计算和创建本RDD 的数据。

RDD 数据集是不可更新的 (Immutable) 。 我们可以对一个 RDD 进行转换 (Transfor- mation), 但是转换之后会返回一个新的RDD, 原来的RDD 保持不变。

RDD 数据集支持两种操作，分别是转换 (Transformation) 和动作 (Action) 。 对 一 个 RDD 施加转换操作，将返回一个新的RDD 。典型的转换操作包括map,filter,flat-

Map,groupByKey,reduceByKey,aggregateByKey,pipe 以 及 coalesce 等操作。动作操 作施加于RDD 数据集，经过对RDD 数据集的计算，返回一个新的值 (New Value)。典 型的动作包括reduce,collect,count,first,take,countByKey 以及 foreach 等操作。

转换操作是延迟 (Lazy) 执行的，也就是这个操作不会马上执行。当某个动作 (Action) 操作被一个驱动程序 (Driver Program, 不是设备驱动程序而是一个客户端软件)调用DAG 的动作操作 (Action) 时，动作操作的一系列前导 (Proceeding)转换操作才会被启动。

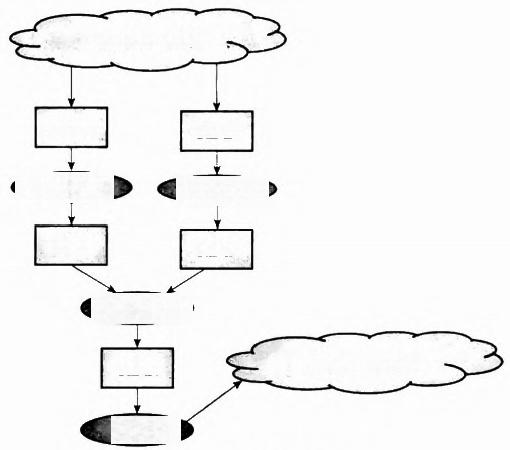
**13.5.1 DAG、宽依赖与窄依赖**

RDD、 转换操作、动作操作等一起构成一个 DAG, 用以表达复杂的计算。 一般来讲， 对 DAG中的每个RDD, 当需要在上面执行某个转换/动作时，将重新从上游 RDD 进行计

算。我们也可以对 RDD 进行缓存或者持久化。在这种情况下， Spark 会保留这个RDD, 以便后面可以再次存取它，从而获得更高的查询速度。缓存指的是把数据缓存在内存中。 Spark 提供了三种持久化的RDD 的存储选项，分别是内存中的反序列化的Java 对象 (In- memStorage as Deserialized Java Objects)、内存中的序列化的数据 (In-mem Storage as Serialized Data) 或者磁盘存储 (On-Disk Storage)。

一个实际的DAG (表达一个数据处理工作流)如图13-4所示。首先，这个工作流 把两个 HDFS 文件里的数据分别装载到两个RDD 中，然后对RDD (包括中间生成的各个 RDD) 施加一系列的转换 (map,flatMap,filter,groupByKey,join 等)操作。对一个 RDD施加转换操作后，生成一个新的RDD 。然后后续的转换操作继续对新生成的RDD 进 行操作。最后，一个动作操作 (count,collect,save,take 等)施加于最后的 RDD, 生

成最终结果，并且写入存储设备。

Extern world

RDD RDD

Transform Transform

RDD RDD

*tansfom*

RDD

Extern world

Action

**图13-4** **一个典型的DAG**

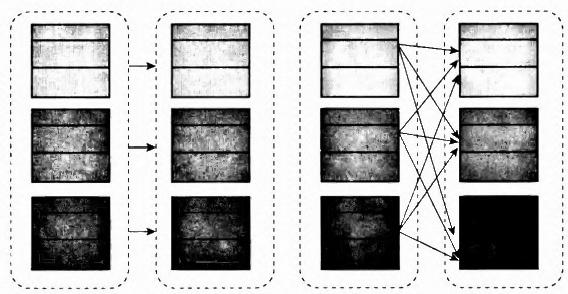
在 DAG 里，父子RDD 的各个分区 (Partition) 之间有两种依赖关系，分别是宽依赖 和窄依赖。窄依赖指的是每个父RDD 的分区，最多被一个子RDD 的分区使用到(父 RDD的某个分区的数据经过转换操作，产生子RDD 的分区)。宽依赖指的是多个子RDD

的分区，依赖于同一个父 RDD 的分区的情形， groupByKey,reduceByKey,sortByKey 等操作需要用到宽依赖，以便获得正确的结果。

窄依赖的处理(从父RDD 分区生成一个子RDD 分区)可以在一台机器上完成，无须 在网络上进行数据的传输 (Data Shuffling), 但是宽依赖一般都涉及数据的网络传输 (Shuffle) (见图13-5)。

**13.5.2 DAG的** **调** **度** **执** **行**

DAGScheduler 是面向阶段 (Stage-(riented) 的 DAG 执行调度器。 DAGScheduler 使用作业 (Job) 和阶段 (Stage) 等基本概念进行作业调度。 一个作业是一个提交到 DAGScheduler 的顶层的工作项目 (Work Item)。一个作业表达成一个DAG, 并且以一个



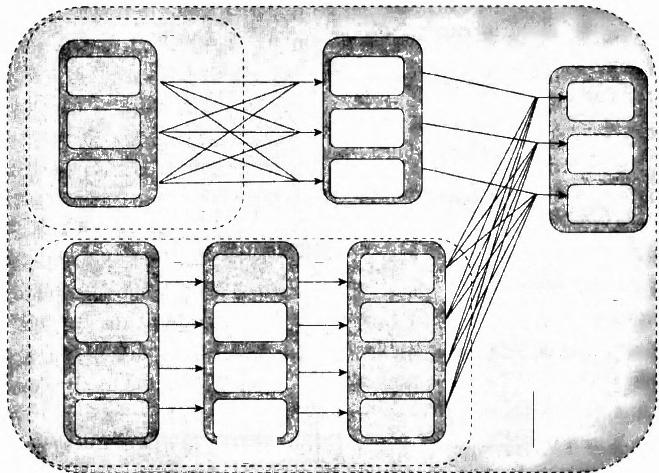
Narrow Transformation Wide Transformation

**图13-5** **窄依赖与宽依赖**

RDD 结束。阶段是一组并行任务，每个任务对应RDD 的一个分区。每个阶段是Spark 作 业的一部分，负责计算部分结果，它是数据处理的基本单元。

DAGScheduler 检查依赖的类型，它把一系列窄依赖RDD 组织成一个阶段。对于宽依 赖则需要跨越连续的阶段。

图13-6展示了一个作业及其对应的各个阶段。这个作业所表达的计算是把两个表连 接起来，然后进行聚集操作，这个作业包含三个阶段。



A

F

Stage 1 Group by

D E

C

Join **Stage 3**

Cached partition

RDD

Filter Stage 2

□ 日

Map

B

**图13-6** **一个Spark作业(Job)** **和对应的各个阶段** **(Stage)**

在 Stage 1里，对RDD A(是一张表，整个是一个RDD, 划分成一系列分区，分布在各 个节点上)进行Group by处理，生成RDD B 。RDD A和 RDDB 之间的依赖关系是宽依赖。

在 Stage 2里，对RDDC (是另外一张表，整个是一个RDD, 划分成一系列分区，分布 在各个节点上)进行Map,Filter 等操作，依次生成RDD D,RDDE等中间结果， RDDC 与 RDD D之间以及RDDD 与 RDDE 之间的依赖关系是窄依赖。

在 Stage 3里，对RDD B和 RDDE 做 Join 操作。RDD B和 RDD F之间的依赖关系



是窄依赖，RDDE 和 RDDF 之间的依赖关系是宽依赖。

通过这三个阶段的处理，实现表 A 和表 B 之间的连接操作。

DAGScheduler 在上述分析基础上，为作业产生一系列的阶段以及它们的依赖关系， 并且确定需要对哪些 RIDD 和哪些阶段的输出进行持久化，找到一个运行这个作业的最小 代价的调度方案，然后把这些阶段提交给Task Scheduler来执行。DAGScheduler 同时根 据当前的缓存信息 (Cache Status). 确定运行每个任务的优选位置，把这些信息一并交给 Task Scheduler。

DAGScheduler 负责对Shuffle 输出文件 (Shuffle Output File) 丢失的情况进行处理， 有时候需要把一些已经执行过的阶段 (Stage) 重新进行提交，以便重建丢失的数据。阶 段内的失败情况(非Shuffle输出文件丢失情况)则由 Task Scheduler本身进行处理，它 将尝试执行该任务一定的次数.如果还是失败，则取消整个阶段 (Stage)。

**13.5.3 共享变量** **(Shared Variable)**

共享变量可以是一个值，也可以是一个数据集。Spark 提供了两种类型的共享变量， 以方便在集群上运行的Spark 程序在各个节点间交换数据。这两种共享变量分别是广播变 量 (Broadcast Variable) 和收集变量 (Accumulator)。

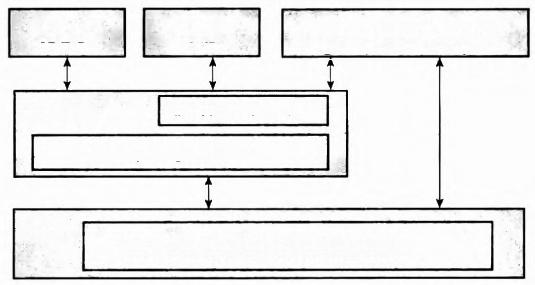
广播变量可以缓存在各个节点上，而不是随着计算任务的调度发送这些变量的拷贝。 在大规模集群上，广播变量的使用可以避免大量的数据移动。

收集变量可以用来实现计数器 (Counter) 或者计算一个总和 (SUM) 。 集群上运行的 各个任务可以向这个变量进行增加操作，但是不能读取它的值。只有驱动者程序 (Driver Program) 可以读取收集变量的值。由于累加操作符合结合律，所以集群上各个任务对收 集变量的累加是没有顺序要求的.可以乱序并行执行。

**13.6 SparkSQL**

SparkSQI 是 Spark 大数据处理框架的重要组成部分，它使得用户可以对结构化数据 采用SQL 语言进行查询分析。用户可以通过ETL(Extraction Transformation and Load- ing) 工具，从不同格式的数据源装载数据，然后运行一些即席查询 (Ad-Hoc Query)。 SparkSQL 和其他系统组件的关系如图13-7所示，其中Catalyst 为SparkSQL 的查询优化器。

SparkSQL 的主要特性包括：(1) DataFrame。 新版本SparkSQL 在数据存储层面通 过数据模式 (Schema) 完成对数据的结构化描述，称为DataFrame 。 在查询处理层面，通 过 SQLContext 支持SQL 查询功能。(2) Data Source 。Spark 增加了Data Source 应用程 序编程接口 (Application Programming Interface,API)。通过这个API.SparkSQL 可以 存取以不同格式保存的结构化数据.包括 Parquet 格式、JSON 格式以及Apache Avro 数 据序列化格式 (Data Serialization) 等。JDBC Data Source 是 SparkSQL 支持的一个数据 源类型。用户可以通过JDBC 编程接口 (API), 从关系数据库读取数据.并且把读取的数 据和其他的数据集进行连接 (Join) 操作。老版本的JDBCRDD 对象则没有这样的连接操 作的支持。(3) JDBC 服务器 (Server) 。SparkSQL 内置一个JDBC 服务器，客户端程序



User

**JDBC** CLI

Program(Java,Scala,Python)

SparkSQL DataFrame API

Catalyst Optimizer

**Spark**

RDD-Resilient Distributed Datasets

**图13-7** **SparkSQL 和其他系统组件的关系**

可以通过JDBC 驱动程序，连接到该服务器，存取SparkSQL 数据库表。

下面我们介绍 SparkSQL 的两个重要对象，DataFrame 和 SQLContext。

DataFrame 是SparkSQL 的一个重要的软件抽象层。DataFrame 是给不同属性列命名 (Named Column) 的分布式数据集，它和关系数据库的数据库表结构非常类似。

DataFrame 从SparkSQL 前一个版本的SchemaRDD 发展而来，并且进行了大量的代 码改写。SchemaRDD 是在RDD 的基础上增加了一个模式 (Schema) 层，以便对数据集 的各个数据列进行命名和数据类型描述。开发人员可以通过在Scala或者Java 程序中调用 DataFrame API, 把过程性处理和关系处理 (Relational Processing, 即对表格的选择、投 影、连接等操作)集成起来。 DataFrame API的各个操作是以延迟 (Lazy) 的方式执行 的，这就使得Spark 系统可以对关系操作以及整个数据处理工作流进行深入优化。

结构化的DataFrame 可以重新转换成无结构的RDD 数据集。通过调用DataFrame 的 rdd() 方法，把DataFrame 的内容(各行数据)以RDD 格式返回。

用户可以从不同的数据源创建DataFrame, 包括已经存在的RDD 数据集、结构化数 据文件 (Structured Data File) 、JSON 数据集、Hive 表格、外部数据库表等。

SparkSQL 通过SQLContext 对象，把所有的关系操作包装起来。用户首先需要创建 SQLContext, 然后调用其方法，实现SQL 查询处理。实际上， SQLContext 是在Spark- Context 基础上进行包装，生成的一个新对象。

下面这行代码展示了如何创建一个SQLContext 对象，其中，一个 SparkContext 对象 sc作为参数传递给SQLContext 构造函数。

val sqlContext=new org.apache.spark.sql.SQLContext(sc)

SparkSQL 还包含了一个HiveContext 对象，它提供了SQLContext 功能的一个超集 (Su- perset), 可以用HQL(Hive Query Language) 来写查询，从Hive 数据库表中查询数据。

**SparkSQL 应用程序**

我们通过一个实例①,展示如何进行SparkSQL 应用程序的编写。该实例从一个文本 文件，装载客户信息，创建一个 DataFrame 对象，然后从 DataFrame 里选择部分数据。

数据文件的名字为customer.txt, 它的内容为：



① 该 实 例 使 用scala 语言，详情请参考<https://www.infoq.com/articles/apache-spark-sql。>

|  |
| --- |
| 100,John Smith,Austin,TX,78727  200,Joe Johnson,Dallas,TX,75201  300,Bob Jones,Houston,TX,77028  400,Andy Davis,San Antonio,TX,78227  500,James Williams,Austin,TX,78727 |

下面的代码片段展示了如何通过 Scala语言编写的程序，调用SparkSQL 的功能，实 现数据处理。这些代码可以在 Spark Shell 控制台交互式地运行。具体的代码如下，代码 给出了注释，说明了每行代码的功能。

|  |
| --- |
| //创建 SQLContext 对象，参数是一个 SparkContext 对象(SparkContext 是 Spark Shell 的内置对象)  val sqlContext =new org.apache.spark.sql.SQLContext(sc)  //导入必要的模块(implicitly convert an RDD to a DataFrame) import sqlContext.implicits.\_  //创建一个 custom 类，用来表示客户(Customer)  case class Customer(customer\_id:Int,name:String,city:String,state: String,zip\_code:String)  //从文本文件(数据集),创建一个DataFrame,里面包含一系列的 Customer 对象 val dfCustomers =sc.textFile("data/customers.txt").map(\_.split(",")). map(p =>Customer(p(0).trim.toInt,p(1),p(2),p(3),p(4))).toDF()  //把DataFrame 注册为一个临时表  dfCustomers.registerTempTable("customers")  //显示 dfCustomers(DataFrame) 的内容 dfCustomers.show()  //打印dfCustomers 的模式(schema)  dfCustomers.printSchema()  **//选择客户的姓名属性列，并且显示**  **dfCustomers.select("name").show()**  //选择客户的姓名、所在城市属性列，并且显示 dfCustomers.select("name","city").show()  //利用基于id 的查询条件，选择某个客户，并且显示  dfCustomers.filter(dfCustomers("customer\_id").equalTo(500)).show()  //根据 zip code 对客户进行分组，显示每组的客户数量 dfCustomers.groupBy("zip\_code").count().show() |

在上述例子中，数据模式通过反射机制推断出来。这种方法的好处是客户数据的属性 列(数据模式)无须事先定义，反射机制可以从数据中推断出数据模式。当然，数据模式 也可以通过编程方式指定。

下面展示另外一个例子，这个例子使用数据类型类 (Data Type Classe), 包括Struct- Type,StringType,StructField 等来指定数据的模式。具体的代码如下，代码给出了注 释，说明了每行代码的功能。

|  |
| --- |
| //该实例以编程的方式，指定 schema  //创建 SQLContext 对象 . 参数是 一 个 SparkContext 对 象 (SparkContext 是 Spark Shell的内置对象)  val sqlContext =new org.apache.spark.sql.SQLContext(sc)  //装载文本文件，创建一个RDD  val rddCustomers=sc.textFile("data/customers.txt")  //用一个字符串表示数据的模式  val schemaString="customer\_id name city state zip\_code"  //导入必要的模块 (Spark SQL data types and Row) import org.apache.spark.sql.\_  import org.apache.spark.sql.types.\_;  //利用上述字符串，创建一个 schema  val schema =StructType(schemaString.split("").map(fieldName => StructField(fieldName,StringType,true)))  //把 RDD 数据集 (rddCustomers) 的每一行，转换成Row 对象  val rowRDD=rddCustomers.map(\_.split(",")).map(p =>Row(p(0). trim,p(1),p(2),p(3),p(4)))  //把 schema 应 用 (apply) 到 RDD 上，赋予其一定的数据结构  val dfCustomers=sqlContext.createDataFrame(rowRDD,schema)  //把 DataFrame 注册为一个临时表  dfCustomers.registerTempTable("customers")  //sqlContext 提供了sql 方法，用于执行SQL查询  val custNames =sqlContext.sql("SELECT name FROM customers")  //SQL查询的结果是一个DataFrame, 支持所有的RDD 操作  //结果集的每一行的每一列，可以通过序号来访问，序号从0开始 custNames.map(t=>"Name:"+t(0)).collect().foreach(println) |

|  |
| --- |
| //sqlContext 提供了sql 方法，用于执行 SQL查询  val customersByCity=sqIContext.sql("SELECT name,zip\_code FROM custom- ers ORDER BY zip\_code")  //SQL查询的结果是一个DataFrame,支持所有的RDD 操作  //结果集的每一行的每一列，可以通过序号来访问，序号从0开始  customersByCity.map(t =>t(0)+","+t(1)).collect().foreach  (println) |

正如上文所述，用户还可以从JSON 数据文件、Hive 数据库表、关系数据库表(通 过JDBC Data Source) 装载数据，创建DataFrame 。之后就可以在DataFrame 上执行各种 查询，也可以进行这些DataFramc 之间的连接操作。

至此，我们看到 SparkSQL 提供了一个友好的SQL 接口，使得用户可以方便地使用 SQL 语言，对从各种数据源装载的数据进行查询分析。这个功能对于数据分析者和 DBA 都是非常有用的，因为他们早已熟悉SQL 语言，或者可以很快上手学习SQL 语言 (SQL 语言简单易学)。让他们先学习和掌握Java,Scala 语言，再编写程序去处理数据则困难 得 多 。

**1** **3** **.** **7** **S** **p** **a** **r** **k** **应** **用** **案** **例**

到目前为止，我们已经对Spark的底层技术、整个生态系统和各个构件进行了详细描 述。接下来我们将通过一个实例来展示如何在实际应用中使用Spark。

我们将展示的实例，是WwW2010 的一篇论文提出的“通过分析Twitter 数据流以探 测地震的发生”数据分析案例。①这个实例比日本的气象机构更快地探测到地震将要或者 正在发生。论文使用了其他技术实现该功能，在这里，我们将展示如何使用Spark 生态系 统的工具实现同样的功能。

在整个数据处理的流程中，第一步是把与“地震” (Earthquake) 和“震动” (Sha- king) 相关的Tweet 过滤出来。我们可以使用Spark Streaming 组件完成这项工作，下面 展示了相应的代码片段。

|  |
| --- |
| TwitterUtils.createStream(…)  .filter(\_.getText.contains("earthquake")I\_.getText.contains("sha- king")) |

接下来，需要对Tweet 进行语义分析 (Semantic Analysis),以便确定这些 Tweet 是 否真正地描述正在发生的地震。类似于 “Earthquake!” 或者 “Now it is shaking” 这样的 Tweet, 可以认为是和正在发生地震相关的。类似于 “Attending an Earthquake Confer- ence” 或者 “The earthquake yesterday was scary” 则不能和当前正在发生的地震相关联。

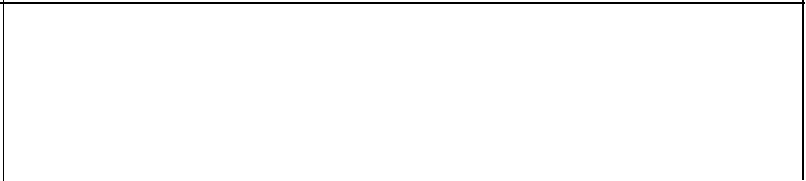
① 详 情 请 参 考 [https://www.toptal.com/spark/introduction-to-apache spark](https://www.toptal.com/spark/introduction-to-apachespark) 。Twitter 是一个新闻和社交类网 站，用户通过发布短文本消息进行交互，消息称为tweet, 消息的大小限制为140个字符。

这篇论文的作者使用支持向量机 (Support Vector Machine,SVM) 来进行 Tweet 的分 类。使用MLLib 对流数据进行处理，代码片段如下：

|  |
| --- |
| //事先准备关于地震的(earthquake)tweet, 然后以LIBSVM 格式装载  val data =MLUtils.loadLibSVMFile(sc,"sample\_earthquate\_tweets.txt")  //把数据集划分成训练集(60%)和测试集(40%)  val splits=data.randomSplit(Array(0.6,0.4),seed=11L)  val training=splits(0).cache()  val test =splits(1)  //训练模型，迭代次数为100  val numIterations=100  val model=SVMWithSGD.train(training,numIterations)  //清除缺省的阈值(threshold)  model.clearThreshold()  //计算测试集上的得分(raw scores)  val scoreAndLabels=test.map{point => val score =model.predict(point.features)  (score,point.label)  }  //计算评价指标(evaluation metrics)  val metrics =new BinaryClassificationMetrics(scoreAndLabels)  val auROC=metrics.areaUnderROC()  println("Area under ROC="+auROC) |

如果训练后的模型预测准确率达到我们的要求，就可以用其进行Tweet 数据流的处 理，探测是否正在发生地震并且采取相应措施。为了确认正在发生地震，需要在预定的时 间窗口内出现一定数量和地震相关 (Positive Match) 的 Tweet。

如果用户允许共享他们的 Tweet 的地理位置信息 (Twitter Location Services Ena- bled), 那么我们就能够了解到某个用户是否处在地震带。可以通过 SparkSQL 查询一张 Hive 表(这张表保存了希望接收地震预警信息的用户),提取用户的 Email 地址，把警报 信息通过 Email 发送给他们。



//sc 是一个 SparkContext 对象

val sqlContext =new org.apache.spark.sql.hive.HiveContext(sc)

//sendEmail是一个用户自定义函数

sqlContext.sql(“FROM earthquake\_warning\_users SELECT firstName,lastName, city,email”).collect().foreach(sendEmail)

**Spark** **的其他应用案例**

Spark 生态系统的工具集可以有效地应对大数据的数据量 (Volume)、 速度 (Veloci- ty) 、 数据多样性 (Diversity) 等多重挑战。下面将列举部分可以使用Spark 技术的实际 应用 。

在游戏行业，如果能够对游戏中的大量实时事件 (Real Time Event) 进行分析，并且 发现一些模式，就可以进行快速响应，比如进行精准广告投放 (Targeted Advertising)、 玩家的挽留 (Retention) 以及自动调整游戏的复杂度等，这些分析将是非常有利可图的。

在电商领域，实时的交易信息可以作为一个数据流，在数据流上可以运行一些聚类算 法，比如 K-Means 算法或者协同过滤算法 (Collaborative Filtering), 所获得的分析结果 可以结合其他的非结构化数据源，比如客户对商品的评价信息等，用于不断优化和调整展 示给用户的推荐信息。

在金融以及网络安全领域， Spark 生态系统工具可以用于欺诈检测 (Fraud Detec- tion) 、入侵检测 (Intrusion Detection) 等。通过对大量的日志进行分析，并且和外部数 据源进行合并，比如关于数据泄露、受损账户、连接和请求发出的 IP 地址所在位置的地 理信息以及时间等信息，我们就可以获得更加准确的预测效果。

**13.8** **小结**

简而言之，Spark 是一个高性能的通用的大数据处理平台，它可以处理大规模的实时 或者批量数据，它可以处理结构化和非结构化数据，它集成了复杂的机器学习和图数据处 理算法。不同类型数据的保存以及各种复杂的处理任务，都可以在一个平台上完成，帮助 我们从数据中发现隐藏的规律和模式。

**13.9** **思考题**

(1)Spark 的软件架构。

(2)Spark 的优势。

(3)Spark 生态系统。

(4)RDD 分布式数据集，及其 Transformation 操作、Action 操作。

(5)宽依赖、窄依赖。

(6)DAG 的作业调度执行，作业实例。

(7)DataFrame 与 SparkSQL。

(8)Spark 应用案例。