

---

# 第 7 章

---

---

---

---

## 数据处理与分析

在数据处理与分析环节，可以利用统计学、机器学习和数据挖掘方法，并结合数据处理与分析技术，对数据进行处理与分析，得到有价值的结果，服务于生产和生活。统计学、机器学习和数据挖掘方法并非大数据时代的新生事物，但是，它们在大数据时代得到了新的发展——实现方式从单机程序发展到分布式程序，从而充分利用计算机集群的并行处理能力。MapReduce 和 Spark 等大数据处理技术，为高性能的大数据处理与分析提供了强有力的支撑。此外，大数据时代新生的数据仓库 Hive、流计算框架 Storm 和 Flink、大数据编程框架 Beam、查询分析系统 Dremel 等，有效地满足了企业不同应用场景的大数据处理与分析需求。

本章首先介绍数据处理与分析的概念，然后介绍机器学习和数据挖掘算法，接下来介绍大数据处理与分析技术，最后介绍大数据处理与分析代表性产品。

### 7.1 数据处理与分析的概念

数据分析可以分为广义的数据分析和狭义的数据分析，广义的数据分析包括狭义的数据分析和数据挖掘。广义的数据分析是指用适当的分析方法（来自统计学、机器学习和数据挖掘等领域），对收集来的数据进行分析，提取有用信息和形成结论的过程。可以看出，在广义的数据分析中，可以使用复杂的机器学习和数据挖掘算法，也可以根本不使用这些算法，而只使用一些简单的统计分析方法，比如汇总求和、求平均值、求均方差等。狭义的数据分析是指根据分析目的，用适当的统计分析方法和工具，对收集来的数据进行处理与分析，提取有价值的信息，发挥数据的作用。

本节介绍数据分析与数据挖掘、数据分析与数据处理，以及大数据处理与分析。

#### 7.1.1 数据分析与数据挖掘

狭义的数据分析和数据挖掘是有着明显的区分的，具体如下。

（1）在定义层面。狭义的数据分析在上面已经定义过。而数据挖掘是指从大量的数据中，通过统计学、人工智能、机器学习等方法，挖掘出未知的、且可能有价值的信息和知识的过程。

(2) 在作用层面。数据分析主要实现三大作用：现状分析、原因分析、预测分析（定量）。数据分析的目标明确，先做假设，然后通过数据分析来验证假设是否正确，从而得到相应的结论。数据挖掘主要侧重解决四类问题：分类、聚类、关联和预测（定量、定性）。数据挖掘的重点在于寻找未知的模式与规律；如著名的数据挖掘案例——啤酒与尿布，就是事先未知的，但又是非常有价值的信息。

(3) 在方法层面。数据分析主要采用对比分析、分组分析、交叉分析、回归分析等常用分析方法；数据挖掘主要采用决策树、神经网络、关联规则、聚类分析等统计学、人工智能、机器学习等方法进行挖掘。

(4) 在结果层面。数据分析一般都是得到一个指标统计量结果，如总和、平均值等，这些数据都需要与业务结合进行解读，才能发挥出数据的价值与作用。数据挖掘则是输出模型或规则，并且可相应得到模型得分或标签。模型得分如流失概率值、总和得分、相似度、预测值等，标签如高中低价值用户、流失与非流失、信用优良中差等。

### 7.1.2 数据分析与数据处理

数据分析过程通常会伴随着数据处理的发生（或者说伴随着大量数据计算），因此，分析和数据处理是一对关系紧密的概念，很多时候，二者是融合在一起的，很难割裂开来。也就是说，当用户在进行数据分析的时候，底层的计算机系统会根据数据分析任务的要求，使用程序进行大量的数据处理（或者说发生大量的数据计算）。例如，当用户进行决策树分析时，需要事先根据决策树算法编写分析程序，当分析开始以后，决策树分析程序就会从磁盘读取数据进行大量计算，最终给出计算结果（也就是决策树分析结果）。

### 7.1.3 大数据处理与分析

数据分析包含两个要素，即理论和技术。在理论层面，需要统计学、机器学习和数据挖掘等知识；在技术层面，包括单机分析工具（如 SPSS、SAS 等）或单机编程语言（如 Python、R），以及大数据处理与分析技术（如 MapReduce、Spark、Hive 等）。

数据分析可以是针对小规模数据的分析，也可以是针对大规模数据的分析（这时被称为“大数据分析”）。在大数据时代到来之前，数据分析主要以小规模的数据为主，一般使用统计学、机器学习和数据挖掘的相关方法，以单机分析工具或者单机编程的方式来实现分析程序。但是，到了大数据时代，数据量爆炸式地增长，很多时候需要对规模巨大的全量数据而不是小规模的数据进行分析。这时，单机工具和单机程序显得“无能为力”，就需要采用分布式实现技术，比如使用 MapReduce、Spark 或 Flink 编写分布式分析程序，借助于集群的多台机器进行并行数据处理分析，这个过程就被称为“大数据处理与分析”。

本章后续内容中，在数据分析理论层面，只介绍关于数据挖掘的理论知识（即机器学习和数据挖掘算法），对于使用统计学方法的狭义的数据分析理论知识（如对比分析、分组分析、交叉分析、预测分析、漏斗分析、A/B 测试分析、结构分析、因素分析、矩阵分析、相关分析、回归分析、聚类分析、判断分析、成分分析等）不做介绍，感兴趣的读者可以参考相关的统计学书籍。在数据分析技术

层面,介绍面向大规模数据的大数据处理与分析技术(如 MapReduce、Spark、Flink、Hive 等),对于单机工具和单机编程不做介绍,感兴趣的读者可以参考与 SPSS、SAS、Python 和 R 等相关的书籍。

## 7.2 机器学习和数据挖掘算法

机器学习和数据挖掘是计算机学科中最活跃的研究分支之一,数据处理与分析环节需要用到大量的机器学习和数据挖掘算法。

### 7.2.1 概述

机器学习是一门多领域交叉学科,涉及概率论、统计学、逼近论、凸分析、算法复杂度理论等多门学科,专门研究计算机怎样模拟或实现人类的学习行为,以获取新的知识或技能,重新组织已有的知识结构使之不断改善自身的性能。它是人工智能的核心,是使计算机具有智能的根本途径,其应用遍及人工智能的各个领域。

数据挖掘是指从大量的数据中通过算法搜索隐藏于其中的信息的过程。数据挖掘可以视为机器学习与数据库的交叉,它主要利用机器学习界提供的算法来分析海量数据,利用数据库界提供的存储技术来管理海量数据。从知识的来源角度来说,数据挖掘领域的很多知识“间接”来自统计学界,之所以说“间接”,是因为统计学界一般偏重于理论研究而不注重实用性,统计学界中的很多技术需要在机器学习界进行验证和实践并变成有效的机器学习算法以后,才可能进入数据挖掘领域,对数据挖掘产生影响。

虽然数据挖掘的很多技术都来自机器学习领域,但是,我们并不能因此就认为数据挖掘只是机器学习的简单应用。毕竟,机器学习通常只研究小规模的数据对象,往往无法应用到海量数据,数据挖掘领域必须借助于海量数据管理技术对数据进行存储和处理,同时对一些传统的机器学习算法进行改进,使其能够支持海量数据的情形。

典型的机器学习和数据挖掘算法包括分类、聚类、回归分析和关联规则等。

(1) 分类。分类是指找出数据库中的一组数据对象的共同特点,并按照分类模式将其划分为不同的类,其目的是通过分类模型,将数据库中的数据项映射到某个给定的类别中。分类可以应用到应用分类、趋势预测中,如淘宝商铺将用户在一段时间内的购买情况划分成不同的类,根据情况向用户推荐关联类的商品,从而增加商铺的销售量。

(2) 聚类。聚类类似于分类,但与分类的目的不同,是针对数据的相似性和差异性将一组数据分为几个类别。属于同一类别的数据之间的相似性很大,但不同类别之间数据的相似性很小,跨类的数据关联性很低。

(3) 回归分析。回归分析反映了数据库中数据的属性值的特性,通过函数表达数据映射的关系来发现属性值之间的依赖关系。它可以应用到对数据序列的预测和相关关系的研究中去。在市场营销中,回归分析可以被应用到各个方面。如通过对本季度销售的回归分析,对下一季度的销

售趋势做出预测并做出针对性的营销改变。

(4) 关联规则。关联规则是隐藏在数据项之间的关联或相互关系，即可以根据一个数据项的出现推导出其他数据项的出现。关联规则挖掘技术已经被广泛应用于金融行业的企业中来预测客户的需求，各银行在自己的 ATM 上通过捆绑客户可能感兴趣的信息供用户了解，并获取相应信息来改善自身的营销策略。

(5) 协同过滤。简单来说协同过滤就是利用兴趣相投、拥有共同经验的群体的喜好，来推荐用户感兴趣的信息，个人通过合作的机制给予信息相当程度的回应（如评分）并记录下来，以达到过滤的目的，进而帮助别人筛选信息。

## 7.2.2 分类

分类是一种重要的机器学习和数据挖掘技术。分类的目的是根据数据集的特点构造一个分类函数或分类模型（也常称作分类器），该模型能把未知类别的样本映射到给定类别中。

分类的具体规则可描述如下：给定一组训练数据的集合  $T$ ， $T$  的每一条记录包含由若干个属性组成的一个特征向量，记录用矢量  $\mathbf{X} = (x_1, x_2, \dots, x_n)$  表示。 $x_i$  可以有不同的值域，当一属性的值域为连续域时，该属性为连续属性（Numerical Attribute），否则为离散属性（Discrete Attribute）。用  $C = c_1, c_2, \dots, c_k$  表示类别属性，即数据集有  $k$  个不同的类别。那么， $T$  就隐含了一个从矢量  $\mathbf{X}$  到类别属性  $C$  的映射函数： $f(\mathbf{X}) \mapsto C$ 。分类的目的就是分析输入数据，通过在训练集中的数据表现出来的特性，为每一个类找到一种准确的描述或者模型，采用该方法（模型）将隐含函数表示出来。

构造分类模型的过程一般分为训练和测试两个阶段。在构造模型之前，将数据集随机地分为训练数据集和测试数据集。先使用训练数据集来构造分类模型，然后使用测试数据集来评估模型的分类准确率。如果认为模型的准确率可以接受，就可以用该模型对其他数据元组进行分类。一般来说，测试阶段的代价远低于训练阶段。

典型的分类方法包括决策树、朴素贝叶斯、支持向量机和人工神经网络等。

这里给出一个分类的应用实例。假设有一名植物学爱好者对她发现的鸢尾花的品种很感兴趣。她收集了每朵鸢尾花的一些测量数据：花瓣的长度和宽度以及花萼的长度和宽度。她还有一些鸢尾花分类的数据，也就是说，这些花之前已经被植物学专家鉴定为属于 *setosa*、*versicolor* 或 *virginica* 3 个品种之一。基于这些分类数据，她可以确定每朵鸢尾花所属的品种。于是，她可以构建一个分类算法，让算法从这些已知品种的鸢尾花测量数据中进行学习，得到一个分类模型，再使用分类模型预测新发现的鸢尾花的品种。

## 7.2.3 聚类

聚类又称群分析，是一种重要的机器学习和数据挖掘技术。聚类的目的是将数据集中的数据对象划分到若干个簇中，并且保证每个簇之间样本尽量接近，不同簇的样本间距离尽量远。通过聚类生成的簇是一组数据对象的集合，簇满足以下两个条件。

(1) 每个簇至少包含一个数据对象。

(2) 每个数据对象仅属于一个簇。

聚类的算法可形式化描述如下：给定一组数据的集合  $D$ ， $D$  的每一条记录包含由若干个属性组成的一个特征向量，用矢量  $\mathbf{x} = (x_1, x_2, \dots, x_n)$  表示。 $x_i$  可以有不同的值域，当一属性的值域为连续域时，该属性为连续属性，否则为离散属性。聚类算法将数据集  $D$  划分为  $k$  个不相交的簇  $\{C = c_1, c_2, \dots, c_k\}$ ，其中  $c_i \cap c_j = \emptyset, i \neq j$ ，且  $D = \bigcup_{i=1}^k c_i$ 。

聚类一般属于无监督分类的范畴，按照一定的要求和规律，在没有关于分类的先验知识情况下，对数据进行分类。聚类既能作为一个单独的过程，找寻数据内部的分布结构，也能作为分类等其他学习任务的前驱过程。聚类算法可分为划分法 (Partitioning Method)、层次法 (Hierarchical Method)、基于密度的方法 (Density-based Method)、基于网格的方法 (Grid-based Method)、基于模型的方法 (Model-Based Method) 等。这些方法没有统一的评价指标，因为不同聚类算法的目标函数相差很大。有些聚类是基于距离的(如 k-means)，有些是假设先验分布的(如 GMM、LDA)，有些是带有图聚类和谱分析性质的(如谱聚类)，还有些是基于密度的(如 DBSCAN)。聚类算法应该嵌入问题中进行评价。

聚类的常见应用场景如下。

(1) 目标用户的群体分类。通过对特定运营目的和商业目的所挑选出的指标变量进行聚类分析，把目标群体划分成几个具有明显特征区别的细分群体，从而可以在运营活动中为这些细分群体采取精细化、个性化的运营和服务，最终提升运营的效率和商业效果。

(2) 不同产品的价值组合。企业可以按照不同的商业目的，并依照特定的指标来为众多的产品种类进行聚类分析，把企业的产品体系进一步细分成具有不同价值、不同目的的多维度的产品组合，并且在此基础上分别制订相应的开发计划、运营计划和服务规划。

(3) 探测发现离群点和异常值。这里的离群点是指相对于整体数据对象而言的少数数据对象，这些对象的行为特征与整体的数据行为特征很不一致。例如，某 B2C 电商平台上，比较昂贵、频繁的交易，就有可能隐含欺诈的风险，需要风险控制部门提前关注。

## 7.2.4 回归分析

回归分析 (Regression Analysis) 指的是确定两种或两种以上变量间相互依赖的定量关系的一种统计分析方法。回归分析按照涉及的变量的多少，分为一元回归和多元回归分析；按照因变量的多少，可分为简单回归分析和多重回归分析；按照自变量和因变量之间的关系类型，可分为线性回归分析和非线性回归分析。

在大数据分析中，回归分析是一种预测性的建模技术，它研究的是因变量 (目标) 和自变量 (预测器) 之间的关系。这种技术通常用于预测分析、时间序列模型以及发现变量之间的因果关系。例如，司机的鲁莽驾驶与道路交通事故数量之间的关系，最好的研究方法就是回归。

回归分析的主要内容如下。

(1) 从一组数据出发，确定某些变量之间的定量关系式，即建立数学模型并估计其中的未知参数。估计参数的常用方法是最小二乘法。

(2) 对这些关系式的可信程度进行检验。

(3) 在许多自变量共同影响着一个因变量的关系中,判断哪个(或哪些)自变量的影响是显著的,哪个(或哪些)自变量的影响是不显著的,将影响显著的自变量加入模型中,剔除影响不显著的变量,通常用逐步回归、向前回归和向后回归等方法。

(4) 利用所求的关系式对某一生产过程进行预测或控制。

## 7.2.5 关联规则

关联规则最初是针对购物篮分析(Market Basket Analysis)问题提出的。假设零售商想了解更多地了解顾客的购物习惯,比如想知道顾客可能会在一次购物时同时购买哪些商品。为了回答该问题,可以对商店的顾客购物数据进行购物篮分析。该过程通过发现顾客放入“购物篮”中的不同商品之间的关联,分析顾客的购物习惯。这种关联的发现可以帮助零售商了解哪些商品频繁地被顾客购买,从而帮助他们制订更好的营销策略。

关联规则定义为:假设  $I=\{I_1, I_2, I_3, \dots, I_m\}$  是项的集合。给定一个交易数据库  $D$ , 其中每个事务  $t$  是  $I$  的非空子集,即每一个交易都与一个唯一的标识符 TID 对应。关联规则在  $D$  中的支持度是  $D$  中事务同时包含  $X$ 、 $Y$  的百分比,即概率;置信度是在  $D$  中事务已经包含  $X$  的情况下,包含  $Y$  的百分比,即条件概率。如果满足最小支持度阈值和最小置信度阈值,则认为关联规则是可信的。这些阈值是根据挖掘需要人为设定的。

这里举一个简单的例子进行说明。表 7-1 是数据库  $D$  中的顾客购买记录,包含 6 个事务。项集  $I=\{\text{乒乓球拍, 乒乓球, 运动鞋, 羽毛球}\}$ 。考虑关联规则(频繁二项集):乒乓球拍与乒乓球,事务 1、2、3、4、6 包含乒乓球拍,事务 1、2、6 同时包含乒乓球拍和乒乓球,这里用  $X$  表示购买了乒乓球,用  $Y$  表示购买了乒乓球拍,则  $X \wedge Y=3$ ,  $D=6$ , 支持度  $(X \wedge Y)/D=0.5$ ;  $X=5$ , 置信度  $(X \wedge Y)/X=0.6$ 。若给定最小支持度  $\alpha=0.5$ , 最小置信度  $\beta=0.6$ , 则认为购买乒乓球拍和购买乒乓球之间存在关联。

表 7-1 顾客购买记录

TID	乒乓球拍	乒乓球	运动鞋	羽毛球
1	1	1	1	0
2	1	1	0	0
3	1	0	0	0
4	1	0	1	0
5	0	1	1	1
6	1	1	0	0

常见的关联规则挖掘算法包括 Apriori 算法和 FP-Growth 算法等。

## 7.2.6 协同过滤

推荐技术从被提出到现在已有十余年,在多年的发展历程中诞生了很多新的推荐算法。协同过滤作为最早、最知名的推荐算法,不仅在学术界得到了深入研究,而且至今在业界仍有广泛的应用,已经被大量应用到电子商务的推荐系统中。协同过滤主要包括基于用户的协同过滤、基于

物品的协同过滤和基于模型的协同过滤。

基于用户的协同过滤算法（简称 UserCF 算法）是推荐系统中最古老的算法。可以说，UserCF 算法的诞生标志着推荐系统的诞生。该算法在 1992 年被提出，直到现在该算法都是推荐系统领域最著名的算法之一。UserCF 算法符合人们对于“趣味相投”的认知，即兴趣相似的用户往往有相同的物品喜好。当目标用户需要个性化推荐时，可以先找到和目标用户有相似兴趣的用户群体，然后将这个用户群体喜欢的、而目标用户没有听说过的物品推荐给目标用户，这种方法就称为“基于用户的协同过滤算法”。

基于物品的协同过滤算法（简称 ItemCF 算法）是目前业界应用最多的算法。无论是亚马逊还是 Netflix，其推荐系统的基础都是 ItemCF 算法。ItemCF 算法是给目标用户推荐那些和他们之前喜欢的物品相似的物品。ItemCF 算法并不利用物品的内容属性计算物品之间的相似度，而主要通过分析用户的行为记录来计算物品之间的相似度，该算法基于的假设是：物品 A 和物品 B 具有很大的相似度是因为喜欢物品 A 的用户大多也喜欢物品 B。例如，该算法会因为你购买过《数据挖掘导论》而给你推荐《机器学习实战》，因为，买过《数据挖掘导论》的用户多数也购买了《机器学习实战》。

基于模型的协同过滤算法（简称 ModelCF 算法）是通过已经观察到的用户给物品的打分，来推断每个用户的喜好并向用户推荐适合的物品。实际上，ModelCF 算法同时考虑了用户和物品两个方面，因此，它也可以看作 UserCF 算法和 ItemCF 算法的混合形式。

## 7.3 大数据处理与分析技术

大数据处理与分析面向的是大规模的海量数据，因此，需要一些特殊的处理与分析技术。本节首先介绍大数据处理与分析技术的四大类型，即批处理计算、流计算、图计算和查询分析计算，然后额外介绍一下流计算和图计算。

### 7.3.1 技术分类

大数据处理与分析技术主要包括四种类型，即批处理计算、流计算、图计算和查询分析计算（见表 7-2）。

表 7-2 大数据处理与分析技术类型、解决问题及其代表性产品

大数据处理与分析技术类型	解决问题	代表性产品
批处理计算	针对大规模数据的批量处理	MapReduce、Spark 等
流计算	针对流数据的实时计算	Flink、Storm、S4、Spark Streaming、Flume、Streams、Puma、DStream、Super Mario、银河流数据处理平台等
图计算	针对大规模图结构数据的处理	Pregel、Spark GraphX、Giraph、PowerGraph、Hama、GoldenOrb 等
查询分析计算	大规模数据的存储管理和查询分析	Dremel、Hive、Cassandra、Impala 等

### 1. 批处理计算

批处理计算主要解决针对大规模数据的批量处理，也是我们日常数据分析工作中非常常见的一类数据处理需求。MapReduce 是具有代表性的大数据批处理技术，它是一种分布式并行编程模型，用于大规模数据集的并行运算，它极大地方便了开发者的编程工作，允许开发者在不会分布式并行编程的情况下，将自己的程序运行在分布式系统上。

Spark 是一个针对超大数据集合的低延迟的集群分布式计算系统，比 MapReduce 快许多。Spark 启用了内存分布数据集，除了能够提供交互式查询外，还可以优化迭代工作负载。在 MapReduce 中，数据流从一个稳定的来源，进行一系列加工处理后，流出到一个稳定的文件系统（如 HDFS）。而 Spark 使用内存替代 HDFS 或本地磁盘来存储中间结果，因此，Spark 要比 MapReduce 的速度快许多。

### 2. 流计算

流数据（或数据流）也是大数据分析中的重要数据类型。流数据是指在时间分布和数量上无限的一系列动态数据集集合体，数据的价值随着时间的流逝而降低，因此，必须采用实时计算的方式给出秒级响应。流计算可以实时处理来自不同数据源的、连续到达的流数据，经过实时分析处理，给出有价值的分析结果。目前业内已涌现出许多的流计算框架与平台，第一类是商业级的流计算平台，包括 IBM InfoSphere Streams 和 IBM StreamBase 等；第二类是开源流计算框架，包括 Twitter Storm、Yahoo! S4（Simple Scalable Streaming System）、Spark Streaming、Flink 等；第三类是公司支持自身业务开发的流计算框架，如 Facebook 使用 Puma 和 HBase 相结合来处理实时数据，百度开发了通用实时流数据计算系统 DStream，淘宝开发了通用流数据实时计算系统——银河流数据处理平台。

### 3. 图计算

在大数据时代，许多大数据都以大规模图或网络的形式呈现，如社交网络、传染病传播途径、交通事故对路网的影响等。此外，许多非图结构的大数据，也常会被转换为图模型后进行处理分析。MapReduce 作为单输入、两阶段、粗粒度数据并行的分布式计算框架，在表达多迭代、稀疏结构和细粒度数据时，往往显得力不从心，不适合用来解决大规模图计算问题。因此，针对大型图的计算，需要采用图计算，目前已经出现了不少相关图计算产品。Pregel 是一种基于整体同步并行计算模型（Bulk Synchronous Parallel Computing Model, BSP 模型）实现的并行图处理系统。为了解决大型图的分布式计算问题，Pregel 搭建了一套可扩展的、有容错机制的平台，该平台提供了一套非常灵活的 API，可以描述各种各样的图计算。Pregel 主要用于图遍历、最短路径、PageRank 计算等。其他代表性的图计算产品还包括 Facebook 针对 Pregel 的开源实现 Giraph、Spark 下的 GraphX、图数据处理系统 PowerGraph 等。

### 4. 查询分析计算

针对超大规模数据的存储管理和查询分析，需要提供实时或准实时的响应，才能很好地满足企业经营管理需求。谷歌开发的 Dremel，是一种可扩展的、交互式的实时查询系统，用于只读嵌套数据的分析。通过结合多级树状执行过程和列式数据结构，它能做到几秒内完成对万亿张表的

聚合查询。系统可以扩展到成千上万的 CPU 上，满足谷歌上万用户操作 PB 级的数据，并且可以在 2~3s 完成 PB 级别数据的查询。此外，Cloudera 公司参考 Dremel 系统开发了实时查询引擎 Impala，它提供 SQL 语义，能快速查询存储在 Hadoop 的 HDFS 和 HBase 中的 PB 级大数据。

## 7.3.2 流计算

### 1. 流计算概念

流计算的示意图如图 7-1 所示，流计算平台实时获取来自不同数据源的海量数据，经过实时分析处理，获得有价值的信息。

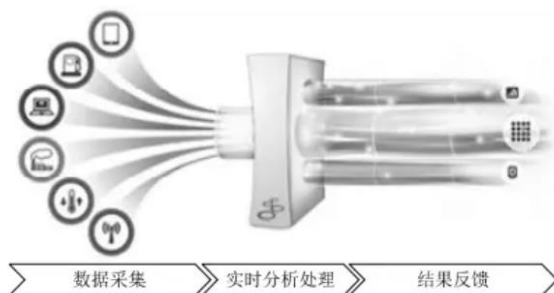


图 7-1 流计算的示意图

总的来说，流计算秉承一个基本理念，即数据的价值随着时间的流逝而降低。因此，当事件出现时就应该立即进行处理，而不是缓存起来进行批量处理。为了及时处理流数据，需要一个低延迟、可扩展、高可靠的处理引擎。对于一个流计算系统来说，它应达到如下需求。

- (1) 高性能。处理大数据的基本要求，如每秒处理几十万条数据。
- (2) 海量式。支持 TB 级甚至是 PB 级的数据规模。
- (3) 实时性。必须保证一个较低的延迟时间，达到秒级别，甚至是毫秒级别。
- (4) 分布式。支持大数据的基本架构，必须能够平滑扩展。
- (5) 易用性。能快速进行开发和部署。
- (6) 可靠性。能可靠地处理流数据。

针对不同的应用场景，相应的流计算系统会有不同的需求，但是针对海量数据的流计算，无论是数据采集还是数据处理都应达到秒级别响应的要求。

### 2. 流计算的处理流程

流计算处理流程包括数据实时采集、数据实时计算和实时查询服务。下面首先介绍传统的数据处理流程，然后详细介绍流计算处理流程的各个环节。

传统的数据处理流程如图 7-2 所示，需要先采集数据并存储在关系数据库等数据管理系统中，之后用户便可以通过查询操作和数据管理系统进行交互，最终得到查询结果。但是，这样一个流程隐含了两个前提。

- ① 存储的数据是旧的。当查询数据的时候，存储的静态数据已经是过去某一时刻的快照，这

些数据在查询时可能已不具备时效性了。

② 需要用户主动发出查询。也就是说，用户是主动发出查询来获取结果的。

流计算的数据处理流程如图 7-3 所示，一般包含 3 个阶段：数据实时采集、数据实时计算、实时查询服务。

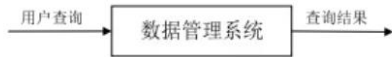


图 7-2 传统的数据处理流程

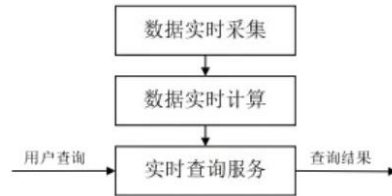


图 7-3 流计算的数据处理流程

### (1) 数据实时采集

数据实时采集阶段通常采集多个数据源的海量数据，需要保证实时性、低延迟与稳定可靠。以日志数据为例，由于分布式集群的广泛应用，数据分散存储在不同的机器上，因此需要实时汇总来自不同机器的日志数据。

目前有许多互联网公司发布的开源分布式日志采集系统均可满足每秒数百 MB 的数据采集和传输需求，如 Facebook 的 Scribe、LinkedIn 的 Kafka、淘宝的 TimeTunnel，以及基于 Hadoop 的 Chukwa 和 Flume 等。

数据采集系统的基本架构一般有 3 个部分（见图 7-4）。

- ① Agent：主动采集数据，并把数据推送到 Collector。
- ② Collector：接收多个 Agent 的数据，并实现有序、可靠、高性能的转发。
- ③ Store：存储 Collector 转发过来的数据。

但对于流计算，一般在 Store 部分不进行数据的存储，而是将采集的数据直接发送给流处理系统进行实时计算。

### (2) 数据实时计算

数据实时计算阶段对采集的数据进行实时的分析和计算。数据实时计算的流程如图 7-5 所示，流处理系统接收数据采集系统不断发来的实时数据，实时地进行分析计算，并反馈实时结果。经流处理系统处理后的数据，可视情况进行存储，以便之后进行分析计算。在时效性要求较高的场景中，处理之后的数据也可以直接丢弃。

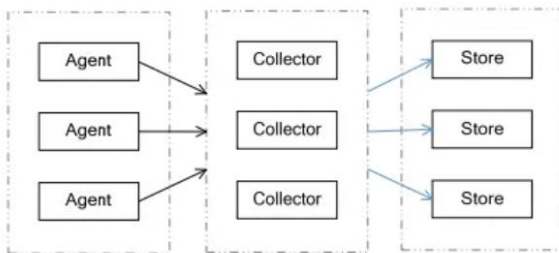


图 7-4 数据采集系统的基本架构



图 7-5 数据实时计算的流程

### (3) 实时查询服务

流计算的第3个阶段是实时查询服务，经由流计算框架得出的结果可供用户进行实时查询、展示或存储。在传统的数据处理流程中，用户需要主动发出查询才能获得想要的结果。而在流处理流程中，实时查询服务可以不断更新结果，并将用户所需的结果实时推送给用户。虽然通过对传统的数据处理系统进行定时查询也可以实现不断更新结果和结果推送，但通过这样的方式获取的结果仍然是根据过去某一时刻的数据得到的结果，与实时结果有着本质的区别。

由此可见，流处理系统与传统的系统有如下不同之处。

① 流处理系统处理的是实时的数据，而传统的系统处理的是预先存储好的静态数据。

② 用户通过流处理系统获取的是实时结果，而通过传统的系统获取的是过去某一时刻的结果。并且，流处理系统无须用户主动发出查询，实时查询服务可以主动将实时结果推送给用户。

## 7.3.3 图计算

在实际应用中，存在许多图计算问题，如最短路径、集群、网页排名、最小切割、连通分支等。图计算算法的性能直接关系到应用问题解决的高效性，尤其对于大型图（如社交网络和网络图）而言，更是如此。下面首先指出传统图计算解决方案的不足之处，然后介绍两大类通用图计算软件。

### 1. 传统图计算解决方案的不足之处

在很长一段时间内，一直都缺少一个可扩展的通用系统来解决大型图的计算问题。很多传统的图计算算法都存在以下几个典型问题：常表现出比较差的内存访问局部性；针对单个顶点的处理工作过少；计算过程中伴随着并行度的改变。

针对大型图的计算问题，可能的解决方案及其不足之处具体如下。

(1) 为特定的图应用定制相应的分布式实现。不足之处是通用性不好，在面对新的图算法或图表示方式时，就需要做大量的重复开发。

(2) 基于现有的分布式计算平台进行图计算。比如，MapReduce 作为一个优秀的大规模数据处理框架，有时也能够用来对大规模图对象进行挖掘，不过在性能和易用性方面往往无法达到最优。

(3) 使用单机的图算法库，比如 BGL、LEAD、NetworkX、JDSL、Stanford GraphBase 和 FGL 等。但是，这种单机方式在可以解决的问题的规模方面具有很大的局限性。

(4) 使用已有的并行图计算系统。Parallel BGL 和 CGM Graph 等库实现了很多并行图算法，但是对大规模分布式系统非常重要的一些特性（如容错），无法提供较好的支持。

### 2. 通用图计算软件

正是因为传统的图计算解决方案无法解决大型图的计算问题，所以需要设计能够用来解决这些问题的通用图计算软件。针对大型图的计算，目前通用的图计算软件主要包括两种：第一种主

要是基于遍历算法的、实时的图数据库，如 Neo4j、OrientDB、DEX 和 InfiniteGraph；第二种则是以图顶点为中心的、基于消息传递批处理的并行引擎，如 Hama、Golden Orb、Giraph 和 Pregel。

第二种图计算软件（如 Pregel）主要是基于 BSP 模型实现的并行图处理系统。BSP 模型是由哈佛大学 Viliant 和牛津大学 Bill Mc Coll 提出的并行计算模型，全称为“整体同步并行计算模型”（Bulk Synchronous Parallel Computing Model），又名“大同步模型”。创始人希望 BSP 模型像冯·诺依曼体系结构那样，架起计算机程序语言和体系结构间的桥梁，故又称为“桥模型”。一个 BSP 模型由大量通过网络相互连接的处理器组成，每个处理器都有快速的本地内存和不同的计算线程，一次 BSP 计算过程包括一系列全局超步（超步是指计算中的一次迭代），每个超步主要包括以下 3 个组件。

（1）局部计算。每个参与的处理器都有自身的计算任务，它们只读取存储在本地内存中的值，不同处理器的计算任务都是异步并且独立的。

（2）通信。处理器群相互交换数据，交换的形式是，由一方发起推送（Put）和获取（Get）操作。

（3）栅栏同步（Barrier Synchronization）。当一个处理器遇到“路障”（或栅栏），会等其他所有处理器完成它们的计算步骤；每一次同步也是一个超步的完成和下一个超步的开始。一个超步的垂直结构如图 7-6 所示。

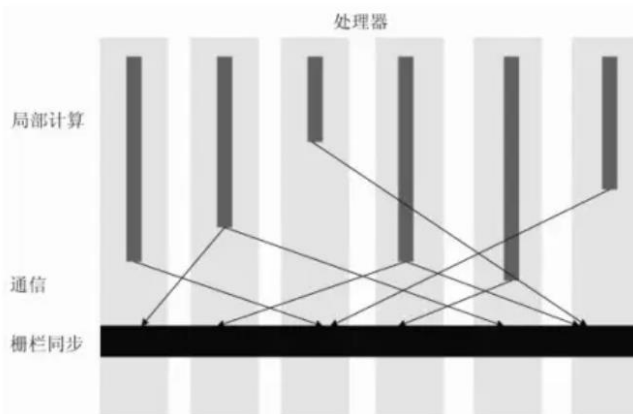


图 7-6 一个超步的垂直结构

## 7.4 大数据处理与分析代表性产品

本节介绍大数据处理与分析领域具有代表性的产品，包括分布式计算框架 MapReduce、数据仓库 Hive 和 Impala、基于内存的分布式计算框架 Spark、机器学习框架 TensorFlowOnSpark、流计算框架 Storm 和 Flink、大数据编程框架 Beam 和查询分析系统 Dremel 等。关于这些产品的详细介绍，可以参考《大数据技术原理与应用》和《Spark 编程基础》等书籍。

## 7.4.1 分布式计算框架 MapReduce

### 1. MapReduce 简介

谷歌在 2003—2006 年连续发表了 3 篇很有影响力的文章，分别阐述了 GFS、MapReduce 和 Bigtable 的核心思想。其中，MapReduce 是谷歌的核心计算模型。MapReduce 将复杂的、运行于大规模集群上的并行计算过程高度地抽象为两个函数：Map 和 Reduce，这两个函数及其核心思想都源自函数式编程语言。

在 MapReduce 中，一个存储在分布式文件系统的大规模数据集会被切分成许多独立的小数据块，这些小数据块可以被多个 Map 任务并行处理。MapReduce 框架会为每个 Map 任务输入一个数据子集，Map 任务生成的结果会继续作为 Reduce 任务的输入，最终由 Reduce 任务输出最后结果，并写入分布式文件系统。特别需要注意的是，适合用 MapReduce 来处理的数据集需要满足一个前提条件：待处理的数据集可以分解成许多小的数据集，而且每一个小数据集都可以完全并行地进行处理。

MapReduce 设计的一个理念就是“计算向数据靠拢”，而不是“数据向计算靠拢”。因为移动数据需要大量的网络传输开销，尤其是在大规模数据环境下，这种开销尤为惊人，所以，移动计算要比移动数据更加经济。本着这个理念，在一个集群中，只要有可能，MapReduce 框架就会将 Map 程序就近地在 HDFS 数据所在的节点运行，即将计算节点和存储节点放在一起运行，从而减少节点间的数据移动开销。

### 2. MapReduce 工作流程

大规模数据集的处理包括分布式存储和分布式计算两个核心环节。谷歌用分布式文件系统 GFS 实现分布式数据存储，用 MapReduce 实现分布式计算；而 Hadoop 则使用分布式文件系统 HDFS 实现分布式数据存储，用 Hadoop MapReduce 实现分布式计算。MapReduce 的输入和输出都需要借助于分布式文件系统进行存储，这些文件被分布存储到集群中的多个节点上。

MapReduce 的核心思想可以用“分而治之”来描述，如图 7-7 所示，也就是把一个大的数据集拆分成多个小数据块在多台机器上并行处理。也就是说，一个大的 MapReduce 作业，首先会被拆分成许多个 Map 任务在多台机器上并行执行，每个 Map 任务通常运行在数据存储的节点上。这样计算和数据就可以放在一起运行，不需要额外的数据传输开销。当 Map 任务结束后，会生成以<key,value>形式表示的许多中间结果。然后，这些中间结果会被分发到多个 Reduce 任务在多台机器上并行执行，具有相同 key 的<key,value>会被发送到同一个 Reduce 任务，Reduce 任务会对中间结果进行汇总计算得到最后结果，并输出到分布式文件系统。

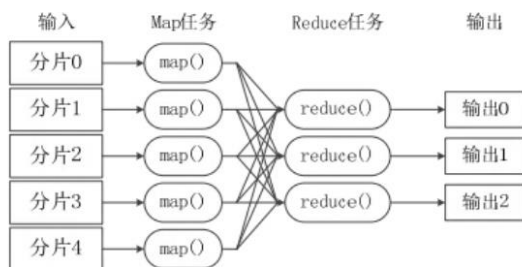


图 7-7 MapReduce 的工作流程

### 3. MapReduce 的不足之处

总体而言，Hadoop 的 MapReduce 存在以下缺点。

(1) 表达能力有限。计算都必须转化成 Map 和 Reduce 两种操作，但这并不适合所有的情况，难以描述复杂的数据处理过程。

(2) 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磁盘中，IO 开销较大。

(3) 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 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，因此难以胜任复杂、多阶段的计算任务。

由于 MapReduce 是基于磁盘的分布式计算框架，因此，性能方面要逊色于基于内存的分布式计算框架（如 Spark 和 Flink 等）。所以，随着 Spark 和 Flink 的发展，MapReduce 的市场空间逐渐被挤压，地位也逐渐被边缘化，但是，不可否认的是，MapReduce 的一些优秀的设计思想在其他框架中得到了很好的继承。

## 7.4.2 数据仓库 Hive

Hive 是一个基于 Hadoop 的数据仓库工具，可以对存储在 Hadoop 文件中的数据集进行数据整理、特殊查询和分析处理。Hive 的学习门槛比较低，因为它提供了类似于关系数据库 SQL 的查询语言——HiveQL。当采用 MapReduce 作为执行引擎时，Hive 可以通过 HiveQL 语句快速实现简单的 MapReduce 任务，Hive 自身可以将 HiveQL 语句快速转换成 MapReduce 任务进行运行，而不必开发专门的 MapReduce 应用程序，因而十分适合数据仓库的统计分析。

### 1. Hive 简介

Hive 是一个构建在 Hadoop 之上的数据仓库工具，由 Facebook 开发，并在 2008 年 8 月开源。Hive 在某种程度上可以看作用户编程接口，其本身并不存储和处理数据，而是依赖 HDFS 来存储数据，依赖 MapReduce（或者 Tez、Spark）来处理数据。Hive 定义了简单的类似 SQL 的查询语言——HiveQL，它与大部分 SQL 语法兼容。

当采用 MapReduce 作为执行引擎时，HiveQL 语句可以快速实现简单的 MapReduce 任务，这样用户通过编写的 HiveQL 语句就可以运行 MapReduce 任务，不必编写复杂的 MapReduce 应用程序。对于 Java 开发工程师，就不必花费大量精力在记忆常见的数据运算与底层的 MapReduce Java API 的对应关系上；对于数据库管理员，可以很容易地把原来构建在关系数据库上的数据仓库应用程序移植到 Hadoop 平台上。所以说，Hive 是一个可以有效、合理、直观地组织和使用数据的分析工具。

现在，Hive 作为 Hadoop 平台上的数据仓库工具，应用已经十分广泛，主要是因为它具有的特点非常适合数据仓库应用程序。首先，当采用 MapReduce 作为执行引擎时，Hive 把 HiveQL 语句转换成 MapReduce 任务后，采用批处理的方式对海量数据进行处理。数据仓库存储的是静态数据，构建于数据仓库上的应用程序只进行相关的静态数据分析，不需要快速响应给出结果，而且

数据本身也不会频繁变化，因而很适合采用 MapReduce 进行批处理。其次，Hive 本身提供了一系列对数据进行抽取、转换、加载的工具，可以存储、查询和分析存储在 Hadoop 中的大规模数据。这些工具能够很好地满足数据仓库各种应用场景，包括维护海量数据、对数据进行挖掘、形成意见和报告等。

## 2. Hive 与 Hadoop 生态系统中其他组件的关系

图 7-8 描述了当采用 MapReduce 作为执行引擎时，Hive 与 Hadoop 生态系统中其他组件的关系。HDFS 作为高可靠的底层数据存储系统，可以存储海量数据。MapReduce 对这些海量数据进行批处理，实现高性能计算。Hive 架构在 MapReduce、HDFS 之上，其自身并不存储和处理数据，而是分别借助于 HDFS 和 MapReduce 实现数据的存储和处理，用 HiveQL 语句编写的处理逻辑，最终都要转换成 MapReduce 任务来运行。Pig 可以作为 Hive 的替代工具，是一种数据流语言和运行环境，适用于在 Hadoop 平台上查询半结构化数据集，常用于

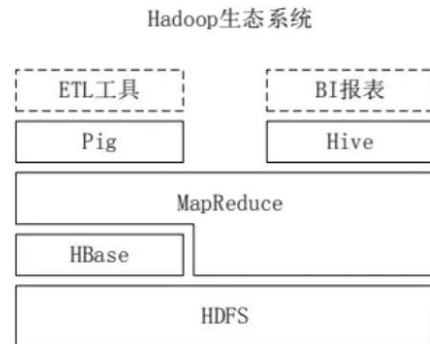


图 7-8 Hive 与 Hadoop 生态系统中其他组件的关系

ETL 过程的一部分，即将外部数据装载到 Hadoop 集群中，然后转换为用户需要的数据格式。HBase 是一个面向列的、分布式的、可伸缩的数据库，它可以提供数据的实时访问功能，而 Hive 只能处理静态数据，主要是 BI 报表数据。就设计初衷而言，在 Hadoop 上设计 Hive，是为了减少复杂 MapReduce 应用程序的编写工作，在 Hadoop 上设计 HBase 则是为了实现对数据的实时访问，所以，HBase 与 Hive 的功能是互补的，它实现了 Hive 不能提供的功能。

## 3. Hive 系统架构

Hive 系统架构主要由以下 3 个模块组成（见图 7-9）：用户接口模块、驱动模块以及元数据存储模块。用户接口模块包括 CLI、HWI（Hive Web Interface）、JDBC、ODBC、Thrift Server 等，用来实现外部应用对 Hive 的访问。CLI 是 Hive 自带的一个命令行客户端工具，但是，这里需要注意的是，Hive 还提供了另外一个命令行客户端工具 Beeline，在 Hive 3.0 以上版本中，Beeline 取代了 CLI。HWI 是 Hive 的一个简单网页，JDBC、ODBC 和 Thrift Server 可以向用户提供进行编程访问的接口。其中，Thrift Server 基于 Thrift 软件框架开发，它提供 Hive 的 RPC 通信接口。驱动模块（Driver）包括编译器、优化器、执行器等，所采用的执行引擎可以是 MapReduce、Tez 或 Spark 等。当采用 MapReduce 作为执行引擎时，驱动模块负责把 HiveQL 语句转换成一系列 MapReduce 作业，所有命令和查询都会进入驱动模块，通过该模块对输入进行解析编译，对计算过程进行优化，然后按照指定的步骤执行。元数据存储模块（Metastore）是一个独立的关系数据库，通常是与 MySQL 数据库连接后创建的一个 MySQL 实例，也可以是 Hive 自带的 derby 数据库实例。元数据存储模块中主要保存表模式和其他系统元数据，如表的名称、表的列及其属性、表的分区及其属性、表的属性、表中数据所在位置信息等。

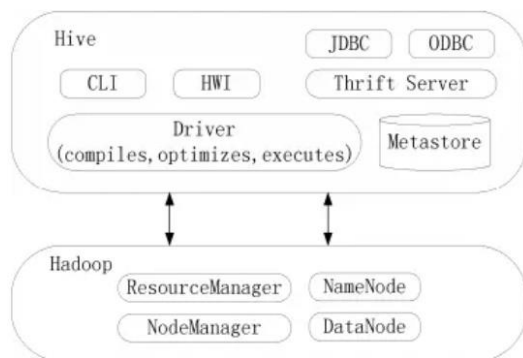


图 7-9 Hive 系统架构

### 7.4.3 数据仓库 Impala

Hive 作为比较流行的数据仓库分析工具之一，得到了广泛的应用。但是，由于 Hive 在采用 MapReduce 执行引擎时使用 MapReduce 来完成批量数据处理，而 MapReduce 是一个面向批处理的非实时计算框架，因此，实时性不好，查询延迟较高，不能满足查询的实时交互性。Impala 作为开源大数据分析引擎，支持实时计算，它提供了与 Hive 类似的功能，并在性能上比 Hive 高出 3~30 倍。

Impala 是由 Cloudera 开发的查询系统，它提供了 SQL 语义，能查询存储在 Hadoop 的 HDFS 和 HBase 上的 PB 级别海量数据。Impala 最初是参照 Dremel 系统进行设计的，Dremel 系统是由 Google 开发的交互式数据分析系统，可以在 2~3s 分析 PB 级别的海量数据。所以，Impala 也可以实现大数据的快速查询。

需要指出的是，虽然 Impala 的实时查询性能要比 Hive 好很多，但是，Impala 的目的并不在于替换现有的包括 Hive 在内的 MapReduce 工具，而是提供一个统一的平台用于实时查询。事实上，Impala 的运行依然需要依赖 Hive 的元数据。总体而言，Impala 与其他组件的关系如图 7-10 所示。

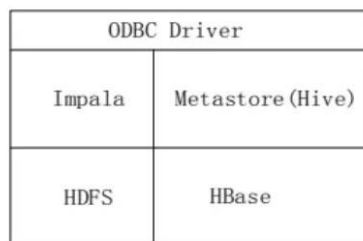


图 7-10 Impala 与其他组件的关系

与 Hive 类似，Impala 也可以直接与 HDFS 和 HBase 进行交互。当采用 MapReduce 作为执行引擎时，Hive 底层执行使用的是 MapReduce，所以主要用于处理长时间运行的批处理任务，例如批量抽取、转换、加载任务。而 Impala 采用了与商用 MPP 并行关系数据库类似的分布式查询引擎，可以直接从 HDFS 或者 HBase 中用 SQL 语句查询数据，而不需要把 SQL 语句转换成 MapReduce 任务来执行，从而大大降低了延迟，可以很好地满足实时查询的要求。另外，Impala 和 Hive 采用相同的 SQL 语法、ODBC 驱动程序和用户接口。

### 7.4.4 基于内存的分布式计算框架 Spark

#### 1. Spark 简介

Spark 最初由美国加州大学伯克利分校 (University of California, Berkeley) 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, 开始迅速地发展, 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 23min 的时间里完成了 100 TB 数据的排序, 而 Hadoop 是使用 2000 个节点在 72min 的时间里才完成同样数据的排序。也就是说, Spark 仅使用了约十分之一的计算资源, 获得了约比 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、高效的大数据计算框架。

Spark 具有如下 4 个主要特点。

(1) 运行速度快。Spark 使用先进的有向无环图 (Directed Acyclic Graph, DAG) 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以上。

(2) 容易使用。Spark 支持使用 Scala、Java、Python 和 R 进行编程, 简洁的 API 设计有助于用户轻松构建并程序, 并且可以通过 Spark Shell 进行交互式编程。

(3) 通用性强。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。

(4) 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 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。

## 2. Spark 相对于 MapReduce 的优点

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 主要具有以下优点。

(1) Spark 的计算模式也属于 MapReduce, 但不局限于 Map 和 Reduce, 还提供了多种数据集操作类型, 编程模型比 MapReduce 更灵活。

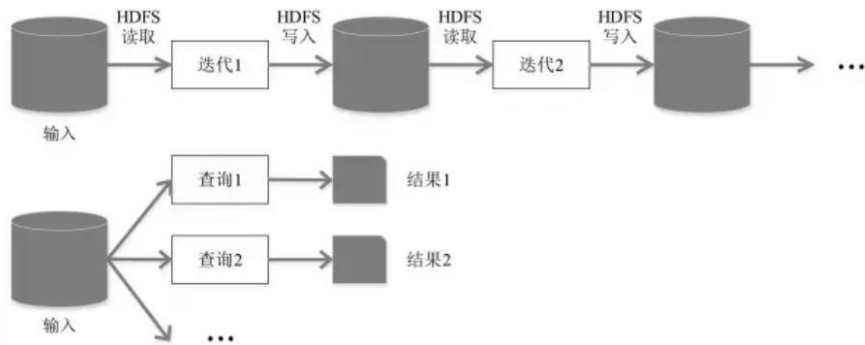
(2) Spark 提供了内存计算, 中间结果直接放到内存中, 带来了更高的迭代运算效率。

(3) Spark 基于 DAG 的任务调度执行机制, 要优于 MapReduce 的迭代执行机制。

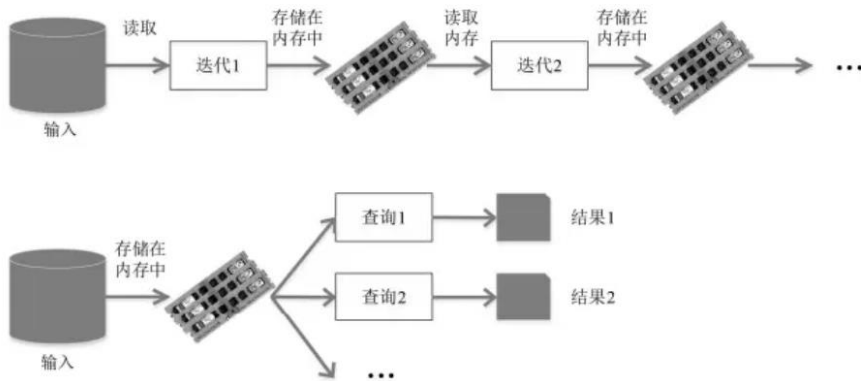
Hadoop 与 Spark 的执行流程对比如图 7-11 所示。由图 7-11 可以看到, Spark 最大的特点就是计算数据、中间结果都存储在内存中, 大大减少了 IO 开销, 因而 Spark 更适合于迭代运算比较多的数据挖掘与机器学习运算。

使用 MapReduce 进行迭代计算非常耗资源, 因为每次迭代计算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 7-12 所示, Hadoop 与 Spark 在执行逻辑回归

时所需的时间相差巨大。



(a) Hadoop MapReduce的执行流程



(b) Spark的执行流程

图 7-11 Hadoop 与 Spark 的执行流程对比

在实际进行开发时,使用 MapReduce 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程序,MapReduce 的代码量要比 Spark 多 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。

近几年来,大数据机器学习和数据挖掘的并行化算法研究成为大数据领域一个较为重要的研究热点。在 Spark 崛起之前,学界和业界普遍关注的是 Hadoop 平台上的并行化算法设计。但是,MapReduce 的网络和磁盘读写开销大,难以高效地实现需要大量迭代计算的机器学习并行化算法。因此,近年来国内外的研究重点开始转到如何在 Spark 平台上实现各种机器学习和数据挖掘的并行化算法设计。为了方便一般应用领域的数据分析人员使用所熟悉的 R 语言在 Spark 平台上完成数据分析,Spark 提供了一个称为 Spark R 的编程接口,使得一般应用领域的数据分析人员,可以在 R 语言的环境里方便地使用 Spark 的并行化编程接口和强大的计算能力。

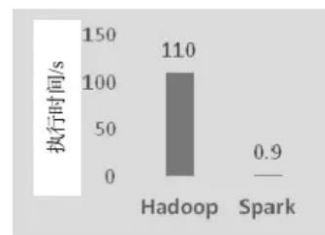


图 7-12 Hadoop 与 Spark 执行逻辑回归所需时间的对比

### 3. Spark 与 Hadoop 的关系

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框架。目前，越来越多的企业放弃 MapReduce，转而使用 Spark 开发企业应用。但是，需要指出的是，Spark 作为计算框架，只能解决数据计算问题，无法解决数据存储问题，Spark 只是取代了 Hadoop 生态系统中的计算框架 MapReduce，而 Hadoop 中的其他组件依然在企业大数据系统中发挥着重要的作用。比如，企业在采用 Spark 解决数据计算问题的同时，依然需要依赖 Hadoop 分布式文件系统 HDFS 和分布式数据库 HBase，来实现不同类型数据的存储和管理，并借助于 YARN 实现集群资源的管理和调度。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 MapReduce、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 上统一部署各个计算框架（见图 7-13）。这些不同的计算框架统一运行在 YARN 中，可以带来如下好处。

- (1) 计算资源按需伸缩。
- (2) 不同负载应用混搭，集群利用率高。
- (3) 共享底层存储，避免数据跨集群迁移。

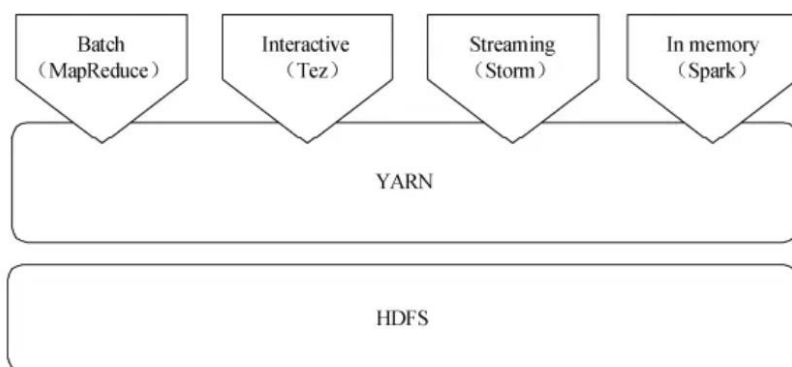


图 7-13 在 YARN 上统一部署各个计算框架

### 4. Spark 生态系统

在实际应用中，大数据处理主要包括以下 3 个情景。

- (1) 复杂的批量数据处理：时间跨度通常在数十分钟到数小时。
- (2) 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。
- (3) 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。

目前已有很多相对成熟的开源软件用于处理以上 3 种情景，比如，可以利用 Hadoop MapReduce 进行批量数据处理，可以用 Impala 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 3 种场景，就需要同时部署 3 种不同的软件，这样做难免会带来一些问题。

- (1) 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。
- (2) 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。
- (3) 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 生态系统足以应对上述 3 种场景，即同时支持批处理、交互式查询和流数据处理。

Spark 生态系统主要包含了 Spark Core、Spark SQL、Spark Streaming、Structured Streaming、MLlib 和 GraphX 等组件（见图 7-14），各个组件的具体功能如下。

(1) Spark Core: Spark Core 包含 Spark 最基础和最核心的功能，如内存计算、任务调度、部署模式、故障恢复、存储管理等，主要面向批数据处理。Spark Core 建立在统一的抽象弹性分布式数据集（Resilient Distributed Dataset, 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；需要注意的是，Spark Core 通常被简称为 Spark。

(2) Spark SQL: Spark SQL 是 Spark 中用于结构化数据处理的组件，提供了一种通用的访问多种数据源的方式，可访问的数据源包括 Hive、Avro、Parquet、ORC、JSON 和 JDBC 等。Spark SQL 采用了 DataFrame 数据模型，支持用户在 Spark SQL 中执行 SQL 语句，实现对结构化数据的处理。

(3) Spark Streaming: Spark Streaming 是一种流计算框架，可以支持高吞吐量、可容错处理的实时流数据处理，其核心思路是将流数据分解成一系列短小的批处理作业，每个短小的批处理作业都可以使用 Spark Core 进行快速处理。Spark Streaming 支持多种数据输入源，如 Kafka、Flume 和 TCP 套接字等。

(4) Structured Streaming: Structured Streaming 是一种基于 Spark SQL 引擎构建的、可扩展且容错的流处理引擎。通过一致的 API，Structured Streaming 使得使用者可以像编写批处理程序一样编写流处理程序，降低了使用者的使用难度。

(5) 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，开发者只要具备一定的理论知识就能进行机器学习的工作。

(6) 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。

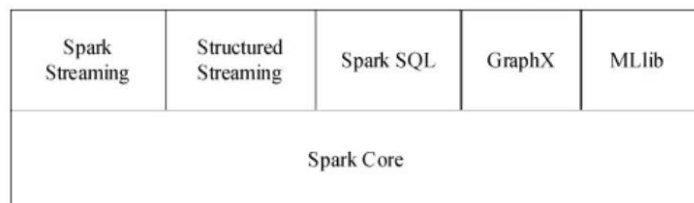


图 7-14 Spark 生态系统

需要说明的是,无论是 Spark SQL、Spark Streaming、Structured Streaming、MLlib,还是 GraphX,都可以使用 Spark Core 的 API 处理问题,它们的使用方法几乎是通用的,处理的数据也可以共享,不同应用之间的数据可以无缝集成。

表 7-3 给出了在不同的应用场景下,可以选用的其他框架和 Spark 生态系统中的组件。

表 7-3 Spark 的应用场景、可以选用的其他框架和 Spark 生态系统中的组件

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 Core
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	毫秒级、秒级	Storm、S4、Flink	Spark Streaming Structured 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

### 5. Spark 运行架构

如图 7-15 所示,Spark 运行架构包括集群管理器 (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 Program, 或简称为 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。可以看出,就系统架构而言,Spark 采用“主从架构”,包含一个 Master (即 Driver) 和若干个 Worker。

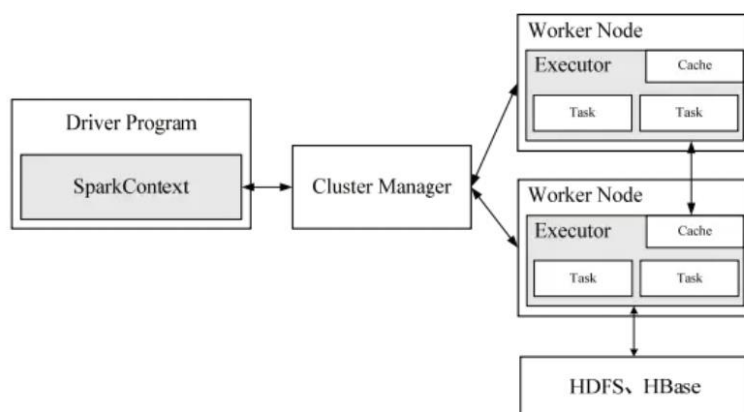


图 7-15 Spark 运行架构

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,会将内存和磁盘共同作为存储设备 (默认使用内存,当内存不够时,会写到磁盘),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,从而有效减少 IO 开销,或者在交互式查询场景下,Executor 预先将表缓存到该存储系统上,从而提

高读写 IO 的性能。

总体而言，在 Spark 中，当执行一个应用时，任务控制节点 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 上执行任务，运行结束后，执行结果会返回给任务控制节点 Driver，或者写到 HDFS 或者其他数据库中。

## 6. Spark 的数据抽象 RDD

Spark 建立在统一的抽象 RDD 之上，这使得 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。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 可以分成多个分区，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 提供了一组丰富的操作以支持常见的数据运算，分为“行动”(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(如 map、filter、groupBy、join 等)接受 RDD 并返回 RDD，而行动操作(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 比较适合对数据集中元素执行相同操作的批处理式应用，而不适用于需要异步、细粒度状态的应用，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，可以具备很多现有计算框架(如 MapReduce、SQL、Pregel 等)的表达能力，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。

Spark 用 Scala 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(或者内存中的集合)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，供给下一个转换操作使用。
- (3) 最后一个 RDD 经行动操作进行处理，并输出到外部数据源(或者变成 Scala 集合或标量)。

## 7. Spark 的部署方式

目前，Spark 支持 5 种不同类型的部署方式，包括 Local、Standalone、Spark on Mesos、Spark on YARN 和 Spark on Kubernetes。其中，Local 模式属于单机部署模式，其他属于分布式部署模式，具体如下。

(1) Standalone 模式。与 MapReduce1.0 框架类似，Spark 框架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，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 节点构成，并且以槽(Slot)作为资源分配单位。不同的是，Spark 中的槽不再像 MapReduce1.0 那样分为 Map 槽和 Reduce 槽，而是只设计了统一的一种槽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 的支持。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采用这种模式。

(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 7-16 所示，资源管理和调度依赖于 YARN，分布式存储则依赖于 HDFS。

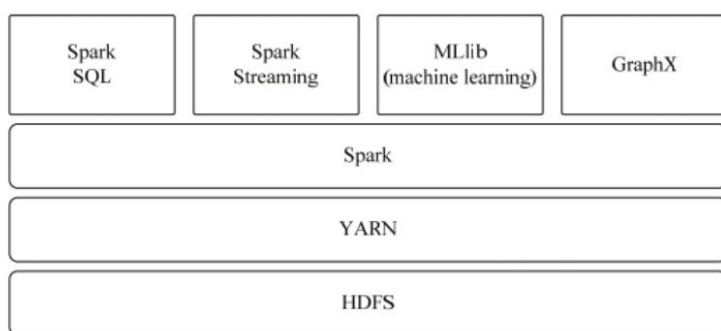


图 7-16 Spark on YARN 架构

(4) Spark on Kubernetes 模式。Kubernetes 作为一个广受欢迎的开源容器协调系统，是 Google 于 2014 年酝酿的项目。Kubernetes 自 2014 年以来热度一路飙升，短短几年时间就已超越了大数据分析领域的明星产品 Hadoop。Spark 从 2.3.0 版本引入了对 Kubernetes 的原生支持，可以将编写好的数据处理程序直接通过 spark-submit 提交到 Kubernetes 集群。

## 8. Spark SQL

本节介绍 Spark SQL 的前身——Shark、Spark SQL 的架构以及诞生原因。

### (1) 从 Shark 说起

Hive 是一个基于 Hadoop 的数据仓库工具，提供了类似于关系数据库 SQL 的查询语言——HiveQL，用户可以通过 HiveQL 语句快速实现简单的 MapReduce 作业，Hive 自身可以自动将 HiveQL 语句快速转换成 MapReduce 任务运行。当用户向 Hive 输入一段命令或查询（即 HiveQL 语句）时，Hive 需要与 Hadoop 交互来完成该操作。该命令或查询首先进入驱动模块，由驱动模块中的编译器进行解析编译，并由优化器对该操作进行优化计算，然后交给执行器去执行，执行器通常的任务是启动一个或多个 MapReduce 任务。图 7-17 描述了用户提交一段 SQL 查询后，Hive 把 SQL 语句转换成 MapReduce 作业进行执行的详细过程。

Shark 提供了类似 Hive 的功能，与 Hive 不同的是，Shark 把 SQL 语句转换成 Spark 作业，而不是 MapReduce 作业。为了实现与 Hive 兼容，Shark 重用了 Hive 中的 HiveQL 解析、逻辑查询计划翻译、执行计划优化等逻辑（见图 7-18）。可以近似认为，Shark 仅将物理执行计划从 MapReduce 作业替换成了 Spark 作业，也就是通过 Hive 的 HiveQL 解析功能，把 HiveQL 翻译成 Spark 上的 RDD。Shark 的出现，使得 SQL-on-Hadoop 的性能比 Hive 有了 10~100 倍的提高。

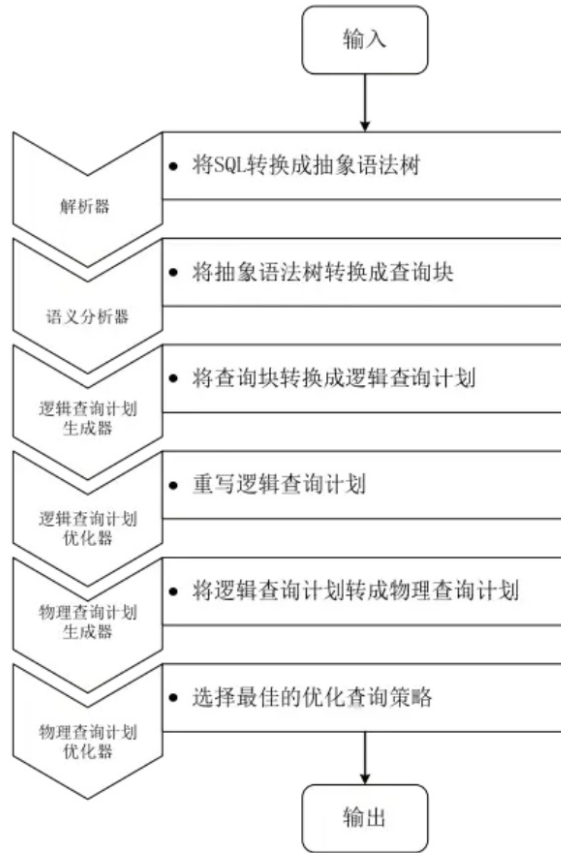


图 7-17 Hive 中 SQL 查询的 MapReduce 作业转化过程

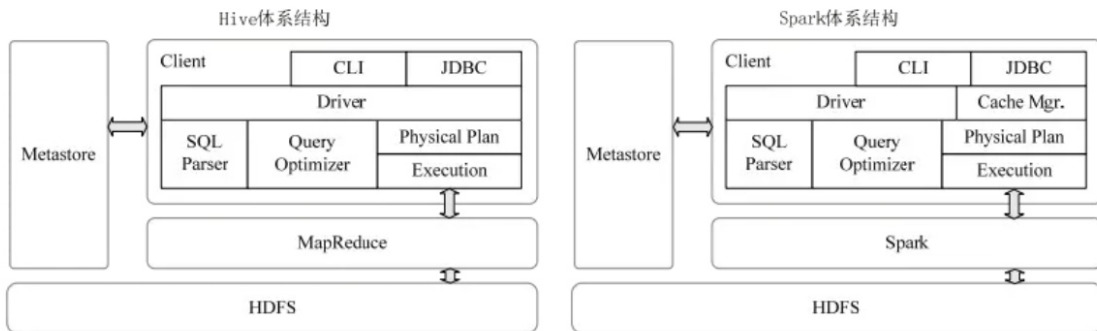


图 7-18 Shark 直接继承了 Hive 的各个组件

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 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，从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，是整个项目的瓶颈。因此，在 2014 年的时候，Shark 项目中止，并转向 Spark SQL 的开发。

### (2) Spark SQL 架构

Spark SQL 架构如图 7-19 所示, 在 Shark 原有的架构基础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 HiveQL 被解析成抽象语法树起, 剩余的工作都全部由 Spark SQL 接管。Spark SQL 逻辑执行计划的生成和优化都由 Catalyst (函数式关系查询优化框架) 负责。

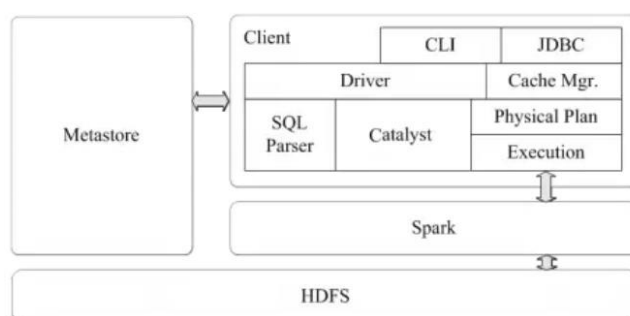


图 7-19 Spark SQL 架构

Spark SQL 增加了 DataFrame (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 HDFS、Hive、Cassandra 等外部数据源, 数据格式既可以是 HDFS、Hive、Cassandra, 也可以是 JSON 格式。Spark SQL 目前支持 Scala、Python、Java 等编程语言, 支持 SQL-92 规范 (见图 7-20)。

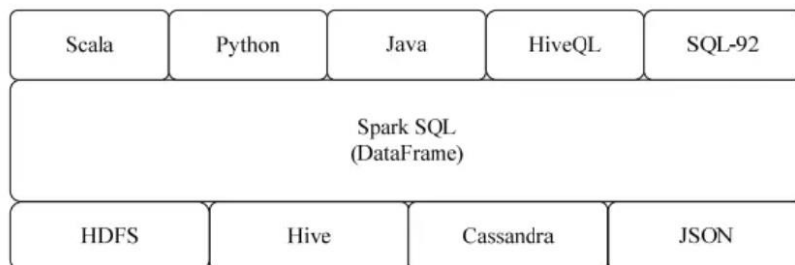


图 7-20 Spark SQL 支持的数据格式和编程语言

### (3) 为什么推出 Spark SQL

关系数据库已经流行多年, 最早是由图灵奖得主、有“关系数据库之父”之称的埃德加·弗兰克·科德于 1970 年提出的。由于具有规范的行和列结构, 因此, 存储在关系数据库中的数据通常也被称为“结构化数据”, 用来查询和操作关系数据库的语言被称为“结构化查询语言”。由于关系数据库具有完备的数学理论基础、完善的事务管理机制和高效的查询处理引擎, 因此得到了广泛的应用, 并从 20 世纪 70 年代到 21 世纪前 10 年一直占据商业数据库应用的主流位置。目前主流的关系数据库有 Oracle、DB2、SQL Server、Sybase、MySQL 等。

尽管数据库的事务和查询机制较好地满足了银行、电信等各类商业公司的业务数据管理需求, 但是, 关系数据库在大数据时代已经不能满足各种新增的用户需求。首先, 用户需要从不同数据

源执行各种操作，包括结构化和非结构化数据；其次，用户需要执行高级分析，比如机器学习和图像处理。在实际大数据应用中，经常需要融合关系查询和复杂分析算法（如机器学习或图像处理），但是，一直以来都缺少这样的系统。

Spark SQL 的出现，填补了这个“鸿沟”。首先，Spark SQL 可以提供 DataFrame API，可以对内部和外部各种数据源执行各种关系操作；其次，可以支持大量的数据源和数据分析算法，组合使用 Spark SQL 和 Spark MLlib，可以融合传统关系数据库的结构化数据管理能力和机器学习算法的数据处理能力，有效满足各种复杂的应用需求。

### 9. Spark 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结合分析的应用场景。

Spark Streaming 是 Spark 的核心组件之一，为 Spark 提供了可扩展、高吞吐量、容错的流计算能力。Spark Streaming 可整合多种输入数据源，如 Kafka、Flume、HDFS 等，甚至是普通的 TCP 套接字，如图 7-21 所示。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。

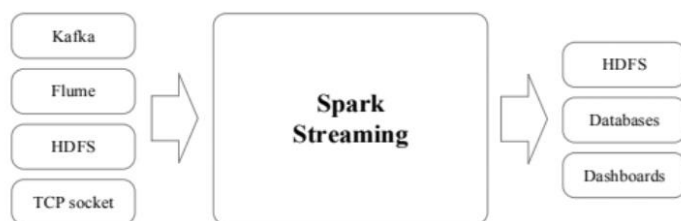


图 7-21 Spark Streaming 支持的输入、输出数据源

Spark Streaming 实际上以一系列微小批处理来模拟流计算。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 7-22 所示。

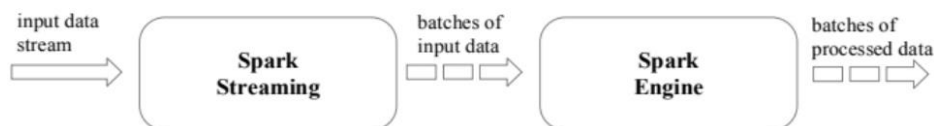


图 7-22 Spark Streaming 的执行流程

图 7-23 展示了 Spark Streaming 的工作机制。在 Spark Streaming 中，会有一个组件 Receiver，作为一个长期运行的任务（Task）运行在一个 Executor 上，每个 Receiver 组件都会负责一个 DStream 输入流（如从文件中读取数据的文件流、套接字流，或者从 Kafka 中读取的一个输入流等）。Receiver 组件接收到数据源发来的数据后，会提交给 Spark Streaming 程序进行处理。处理后的结果，可以交给可视化组件进行可视化展示，也可以写入 HDFS、HBase 中。

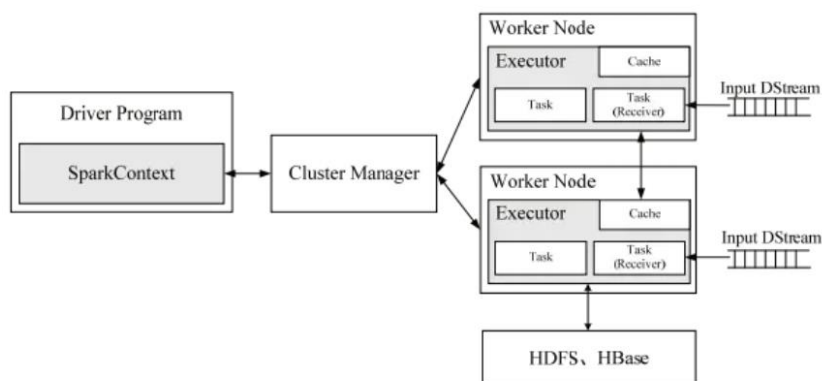


图 7-23 Spark Streaming 的工作机制

## 10. Structured Streaming

由于 Spark Streaming 组件的延迟较高，最快响应时间要在秒级，无法满足一些需要更快响应时间的企业应用的需求，所以，Spark 社区推出了 Structured Streaming。

### (1) Structured Streaming 简介

Structured Streaming 是一种基于 Spark SQL 引擎构建的、可扩展且容错的流处理引擎。通过一致的 API，Structured Streaming 使得使用者可以像编写批处理程序一样编写流处理程序，降低了使用者的使用难度。提供端到端的完全一致性是 Structured Streaming 设计背后的关键目标之一。为了实现这一点，Spark 设计了输入源、执行引擎和接收器，以便对处理的进度进行更可靠的跟踪，使之可以通过重启或重新处理，来处理任何类型的故障。如果所使用的源具有偏移量来跟踪流的读取位置，那么，引擎可以使用检查点和预写日志，来记录每个触发时期正在处理的数据的偏移范围。此外，如果使用的接收器是“幂等”的，那么通过使用重放、对“幂等”接收数据进行覆盖等操作，Structured Streaming 可以确保在任何故障下达到端到端的完全一致性。

Spark 一直在不停更新中，从 Spark 2.3.0 开始引入了持续流式处理模型，可以将原先流处理的延迟降低到毫秒级别。

### (2) Structured Streaming 的关键思想

Structured Streaming 的关键思想是将实时数据流视为一张正在不断添加数据的表，这种新的流处理模型与批处理模型非常类似。可以把流计算等同于在一个静态表上的批处理查询，Spark 会在不断添加数据的无界输入表上运行计算，并进行增量查询。如图 7-24 所示，输入流输入的每一项数据项被原样添加到无界表，最终形成了一个新的无界表。

在无界表上对输入的查询将生成结果表，系统每隔一定的周期会触发对无界表的计算并更新结果表。如图 7-25 所示，在时间线上，每秒为一个触发周期，在  $t=1$  时刻，数据量较少，查询出结果后，输入接收器。在  $t=2$  时刻，数据量增加，查询出结果后，输入接收器。在  $t=3$  时刻，数据量再次增加，如同前面 2 秒一样查询并输出。

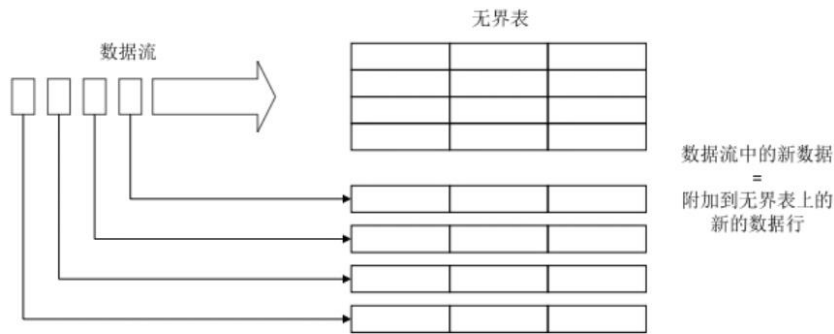


图 7-24 无界表

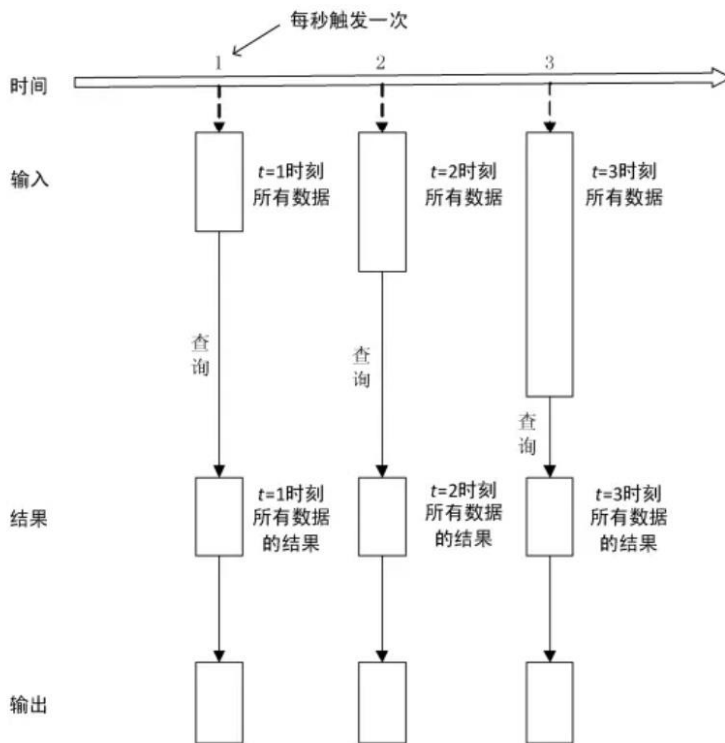


图 7-25 Structured Streaming 编程模型

### (3) Structured Streaming 的两种处理模型

Structured Streaming 包括微批处理和持续处理两种处理模型，默认使用微批处理模型。

#### ① 微批处理。

Structured Streaming 默认使用微批处理执行模型，这意味着 Spark 流计算引擎会定期检查流数据源，并对上一批次结束后到达的新数据执行批量查询。如图 7-26 所示，在微批处理模型的体系结构中，Driver 驱动程序通过将当前待处理数据的偏移量保存到预写日志中，来对数据处理进度设置检查点，以便今后可以使用它来重新启动或恢复查询。为了获得确定性的重新执行 (Deterministic Re-executions) 和端到端语义，在下一个微批处理之前，就要将该微批处理所要处理的数据的偏移范围保存到日志中。所以，当前到达的数据需要等待先前的微批作业处理完成，

且它的偏移量范围被记入日志后，才能在下一个微批作业中得到处理，这会导致数据到达和得到处理并输出结果之间的延时超过 100ms。

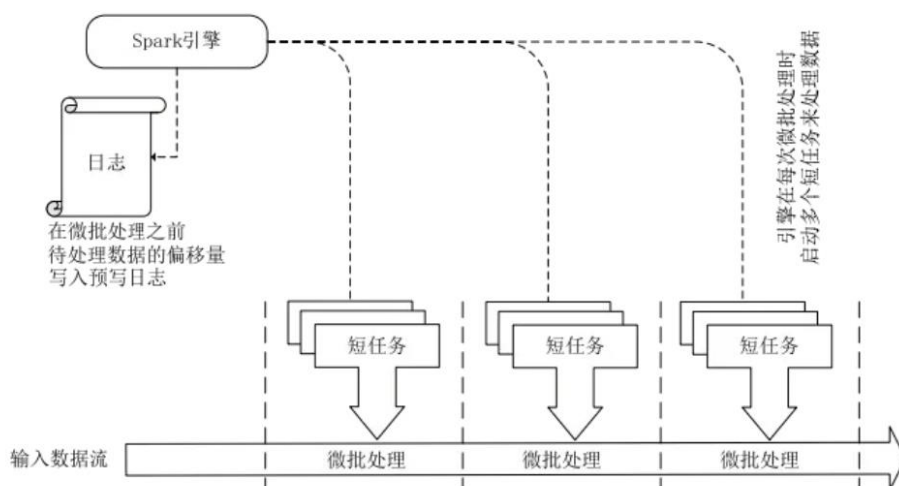


图 7-26 Structured Streaming 的微批处理模型

## ② 持续处理。

微批处理的数据延迟对于大多数实际的流式工作负载（如 ETL 和监控）已经足够了。然而，一些场景确实需要更低的延迟，比如在金融行业的信用卡欺诈交易识别中，需要在犯罪嫌疑人盗刷信用卡后立刻识别并阻止，但是又不能让合法交易的用户感觉到延迟，从而影响到用户的使用体验，这就需要在 10~20ms 的时间内对每笔交易进行欺诈识别，这时就不能使用微批处理模型，而需要使用持续处理模型。

Spark 从 2.3.0 版本开始引入了持续处理的试验性功能，可以实现流计算的毫秒级延迟。在持续处理模式下，Spark 不再根据触发器来周期性启动任务，而是根据一系列的连续读取、处理和写入结果的长时间运行来启动任务。如图 7-27 所示，为了缩短延迟，引入新的算法对查询设置检查点，在每个任务的输入数据流中，一个特殊标记的记录被注入，当任务遇到标记时，任务把处理后的最后偏移量异步地报告给引擎，引擎接收到所有写入接收器的任务的偏移量后，写入预写日志。由于检查点的写入是完全异步的，任务可以持续处理，因此，延迟可以缩短到毫秒级。也正是由于写入是异步的，会导致数据流在故障后可能被处理超过一次以上，持续处理只能做到“至少一次”的一致性。因此，需要注意到，虽然持续处理模型能比微批处理模型获得更好的实时响应性能，但是，这是以牺牲一致性为代价的。微批处理可以保证“端到端”的完全一致性，而持续处理只能做到“至少一次”的一致性。

## (4) Structured Streaming 和 Spark SQL、Spark Streaming 关系

Structured Streaming 与 Spark Streaming 一样，处理的也是源源不断的数据流，区别在于，Spark Streaming 采用的数据抽象是 DStream（本质上就是一系列 RDD），而 Structured Streaming 采用的数据抽象是 DataFrame。Structured Streaming 可以使用 Spark SQL 的 DataFrame/Dataset 来处理数据流。虽然 Spark SQL 也采用 DataFrame 作为数据抽象，但是，Spark SQL 只能处理静态的数据，

而 Structured Streaming 可以处理结构化的数据流。这样，Structured Streaming 就将 Spark SQL 和 Spark Streaming 二者的特性结合了起来。Structured Streaming 可以对 DataFrame/Dataset 应用各种操作，包括 select、where、groupBy、map、filter、flatMap 等。此外，Spark Streaming 只能实现秒级的实时响应，而 Structured Streaming 由于采用了全新的设计方式，采用微批处理模型时可以实现 100ms 级别的实时响应，采用持续处理模型时可以实现毫秒级的实时响应。

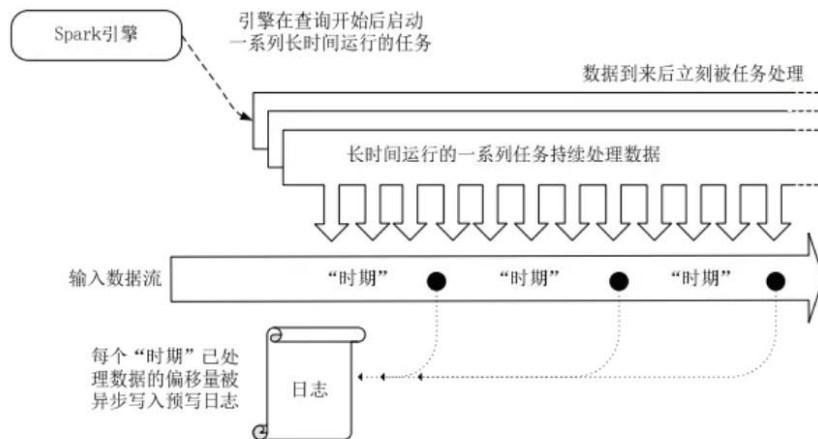


图 7-27 Structured Streaming 的持续处理模型

## 11. Spark MLlib

传统的机器学习算法，由于技术和单机存储的限制，只能基于少量数据使用，因此，传统的统计、机器学习算法依赖于数据抽样。但是在实际应用中，样本往往很难做到随机，导致学习的模型不是很准确，在测试数据上的效果也不太好。随着 HDFS 等分布式文件系统的出现，我们可以对海量数据进行存储和管理，并利用 MapReduce 框架在全量数据上进行机器学习，这在一定程度上解决了统计随机性的问题，提高了机器学习的精度。但是，MapReduce 自身存在缺陷，延迟高，磁盘开销大，无法高效支持迭代计算，这使得 MapReduce 无法高效地实现分布式机器学习算法。因为通常情况下，机器学习算法参数学习的过程都是迭代计算，本次计算的结果要作为下一次迭代计算的输入。在这个过程中，MapReduce 只能把中间结果存储到磁盘中，然后在下一次计算的时候重新从磁盘读取数据；对于迭代计算频发的算法，这是制约其性能的瓶颈。相比而言，Spark 立足于内存计算，天然地适用于迭代式计算，能很好地与机器学习算法相匹配。这也是近年来 Spark 平台流行的重要原因，业界的很多业务纷纷从 Hadoop 平台转向 Spark 平台。

基于大数据的机器学习，需要处理全量数据并进行大量的迭代计算，这就要求机器学习平台具备强大的处理能力和分布式计算能力。然而，对于普通开发者来说，实现一个分布式机器学习算法，仍然是一件极具挑战的事情。为此，Spark 提供了一个基于海量数据的机器学习库，它提供了常用机器学习算法的分布式实现。对于开发者而言，只需要有 Spark 编程基础，并且了解机器学习算法的基本原理和方法中相关参数的含义，就可以轻松地通过调用相应的 API，来实现基

于海量数据的机器学习过程。同时，spark-shell 也提供即席（Ad Hoc）查询的功能，算法工程师可以边写代码、边运行、边看结果。Spark 提供的各种高效的工具，使得机器学习过程更加直观便捷。比如，可以通过 sample()函数非常方便地进行抽样。Spark 发展到目前，已经拥有了批处理、SQL、流计算、图计算等模块，成为了一个全平台的系统，把机器学习作为关键模块加入 Spark 中也是大势所趋。

MLlib（Machine Learning Library）是 Spark 的机器学习库，旨在简化机器学习的工程实践，并能够方便地扩展到更大规模数据。MLlib 提供了主要的机器学习算法，包括用于特征预处理的数理统计方法，特征提取、转换和选择，以及分类、回归、聚类、关联规则、推荐、优化、算法的评估等。MLlib 由一些通用的学习算法和工具组成，包括分类、回归、聚类、协同过滤、降维等，同时还包括底层的优化原语和高层的管道 API。具体来说，MLlib 主要包括以下几方面的内容。

- 算法工具：常用的学习算法，如分类、回归、聚类和协同过滤。
- 特征化工具：特征抽取、转化、降维和选择工具。
- 流水线（Pipeline）：用于构建、评估和调整机器学习工作流的工具。
- 持久性：保存和加载算法、模型和管道。
- 实用工具：线性代数、统计、数据处理等工具。

Spark 在机器学习方面的发展非常快，已经支持了主流的统计和机器学习算法。纵观所有基于分布式架构的开源机器学习库，MLlib 以计算效率高著称。表 7-4 列出了目前 MLlib 支持的主要的机器学习算法。

表 7-4 MLlib 支持的主要机器学习算法

类型	算法
基本统计 ( Basic Statistics )	Summary Statistics、Correlations、Stratified Sampling、Hypothesis Testing、Random Data Generation
分类和回归 ( Classification and Regression )	Support Vector Machines ( SVM )、Logistic Regression、Linear Regression、Naive Bayes、Decision Trees、Random Forest、Gradient-Boosted Trees
协同过滤 ( Collaborative Filtering )	Alternating Least Squares ( ALS )
聚类 ( Clustering )	k-means、Gaussian Mixture Model、Latent Dirichlet allocation (LDA)、Bisecting k-means
频繁项挖掘 ( Frequent Pattern Mining )	FP-Growth
降维 ( Dimensionality Reduction )	Singular Value Decomposition( SVD )、Principal Component Analysis ( PCA )
特征抽取和转换 ( Feature Extraction And Transformation )	Term Frequency-Inverse Document Frequency( TF-IDF )、Word2Vec、StandardScaler、Normalizer

## 7.4.5 机器学习框架 TensorFlowOnSpark

TensorFlow 是一个开源的、基于 Python 的机器学习框架，它由谷歌开发，并在图形分类、音频处理、推荐系统和自然语言处理等场景下有着丰富的应用，是目前最热门的机器学习框架。TensorFlow 是一个采用数据流图 (Data Flow Graph)、用于数值计算的开源软件库。数据流图中的节点 (Nodes) 表示数学操作，图中的线则表示节点间的相互联系的多维数据组，即张量 (Tensor)。在计算过程中，张量从图的一端流动到另一端，这也是这个工具取名为“TensorFlow”的原因。一旦输入端的所有张量准备好，节点将被分配到各种计算设备完成异步并行地执行运算。利用 TensorFlow 我们可以在多种平台上展开数据分析与计算，如 CPU (或 GPU)、台式机、服务器、甚至移动设备等。

尽管 TensorFlow 也开放了自己的分布式运行框架，但在目前公司的技术架构和使用环境上不是那么友好，如何将 TensorFlow 加入现有的环境中 (Spark/YARN)，并为用户提供更加方便易用的环境，成为了目前所要解决的问题。

TensorFlowOnSpark 项目是 Yahoo! 开源的一个软件包，能将 TensorFlow 与 Spark 结合在一起使用，为 Apache Hadoop 和 Apache Spark 集群带来可扩展的深度学习功能，使 Spark 能够利用 TensorFlow 拥有深度学习和 GPU 加速计算的能力。传统情况下处理数据需要跨集群 (深度学习集群和 Hadoop/Spark 集群)，Yahoo! 为了解决跨集群传递数据的问题开发了 TensorFlowOnSpark 项目。TensorFlowOnSpark 目前被用于雅虎私有云的 Hadoop 集群中，主要进行大规模分布式深度学习。

TensorFlowOnSpark 在设计时充分考虑了 Spark 本身的特性和 TensorFlow 的运行机制，大大保证了两者的兼容性，使得可以通过较少的修改来运行已经存在的 TensorFlow 程序。在独立的 TensorFlowOnSpark 程序中，TensorFlow 能够与 SparkSQL、MLlib 和其他 Spark 库一起工作处理数据 (见图 7-28)。

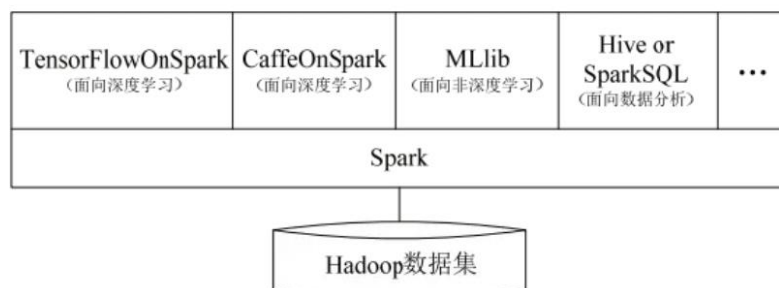


图 7-28 TensorFlowOnSpark 与 Spark 的集成

TensorFlowOnSpark 的体系架构较为简单 (见图 7-29)，Spark Driver 程序并不会参与 TensorFlow 内部相关的计算和处理。其设计思路像是将一个 TensorFlow 集群运行在 Spark 上，它会在每个 Spark Executor 中启动 TensorFlow 程序，然后通过 gRPC 或 RDMA 方式进行数据传递与交互。

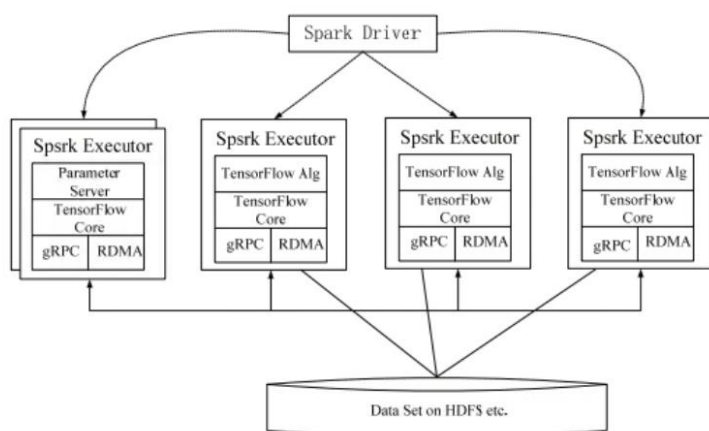


图 7-29 TensorFlowOnSpark 的体系架构

TensorFlowOnSpark 的 Spark 应用程序包括 4 个基本过程。

- (1) 预留：组建 TensorFlow 集群，并在每个 Executor 进程上预留监听端口，启动“数据/控制”消息的监听程序。
- (2) 启动：在每个 Executor 进程上启动 TensorFlow 程序。
- (3) 训练/推理：在 TensorFlow 集群上完成模型的训练/推理。
- (4) 关闭：关闭 Executor 进程上的 TensorFlow 程序，释放相应的系统资源（消息队列）。

## 7.4.6 流计算框架 Storm

### 1. Storm 简介

Twitter Storm 是一个免费、开源的分布式实时计算系统，也是大数据领域中第一个开源的流处理框架。Storm 对于实时计算的意义类似于 Hadoop 对于批处理的意义。Storm 可以简单、高效、可靠地处理流数据，并支持多种编程语言。Storm 框架可以方便地与数据库系统进行整合，从而开发出强大的实时计算系统。目前，Storm 框架已成为 Apache 软件基金会旗下的项目，读者可以在其官方网站了解更多信息。

Twitter 是全球访问量最大的社交网站之一，Twitter 之所以开发 Storm 流处理框架也是为了应对其不断增长的流数据实时处理需求。为了处理实时数据，Twitter 采用了由实时处理系统和批处理系统组成的分层数据处理架构（见图 7-30），一方面，由 Hadoop 和 ElephantDB（专门用于从 Hadoop 中导出 key/value 数据的数据库）组成批处理系统，另一方面，由 Storm 和 Cassandra（非关系数据库）组成实时系统。在计算查询时，该系统会同时查询批处理视图和实时视图，并把它们合并起来以得到最终的结果。实时系统处理的结果最终会由批处理系统来修正，这种设计方式使得 Twitter 的数据处理系统显得与众不同。

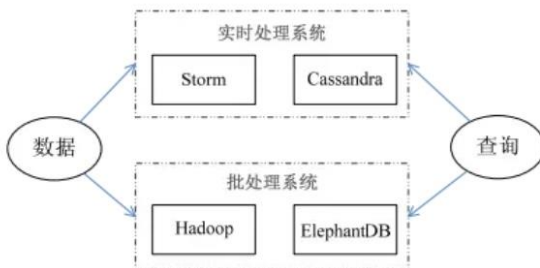


图 7-30 Twitter 的分层数据处理架构

## 2. Storm 的特点

Storm 具有以下主要特点。

- (1) 整合性。Storm 可方便地与队列系统和数据库系统进行整合。
- (2) 简易的 API。Storm 的 API 在使用上既简单又方便。
- (3) 可扩展性。Storm 的并行特性使其可以运行在分布式集群中。
- (4) 容错性。Storm 可以自动进行故障节点重启, 以及节点故障时任务重新分配。
- (5) 可靠的消息处理。Storm 保证每个消息都能完整处理。
- (6) 支持各种编程语言。Storm 支持使用各种编程语言来定义任务。
- (7) 快速部署。Storm 仅需要少量的安装和配置工作就可以快速进行部署和使用。
- (8) 免费、开源。Storm 是一款开源框架, 可以免费使用。

Storm 可用于许多领域, 如实时分析, 在线机器学习, 持续计算, 远程 RPC, 数据抽取、加载和转换等。由于 Storm 具有可扩展、高容错性、能可靠地处理消息等特点, 目前已经广泛应用于流计算。此外, Storm 是开源免费的, 用户可以轻易地进行搭建、使用, 这大大降低了学习和使用成本。

## 3. Storm 的框架设计

Storm 运行在分布式集群中, 其运行任务的方式与 Hadoop 类似: 在 Hadoop 上运行的是 MapReduce 作业, 而在 Storm 上运行的是“Topology”。但两者的任务大不相同, 其中主要的不同是, 一个 MapReduce 作业最终会完成计算并结束运行, 而一个 Topology 将持续处理消息 (直到人为终止)。

Storm 集群采用“Master-Worker”的节点方式, 其中 Master 节点运行名为“Nimbus”的后台程序 (类似 Hadoop 中的“JobTracker”), 负责在集群范围内分发代码、为 Worker 分配任务和监测故障。而每个 Worker 节点运行名为“Supervisor”的后台程序, 负责监听分配给它所在机器的工作, 即根据 Nimbus 分配的任务来决定启动或停止 Worker 进程。

Storm 集群框架示意如图 7-31 所示, Storm 采用了 ZooKeeper 来作为分布式协调组件, 负责 Nimbus 和多个 Supervisor 之间的所有协调工作 (一个完整的拓扑可能被分为多个子拓扑, 并由多个 Supervisor 完成)。

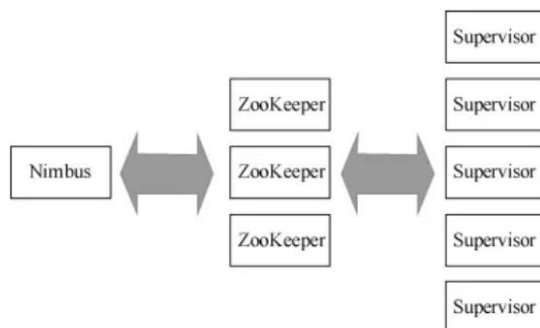


图 7-31 Storm 集群框架示意

此外, Nimbus 后台进程和 Supervisor 后台进程都是快速失败 (Fail-fast) 和无状态 (Stateless) 的, Master 节点并没有直接和 Worker 节点通信, 而是借助 ZooKeeper 将状态信息存放在 ZooKeeper

中或本地磁盘中，以便节点故障时进行快速恢复。这意味着，若 Nimbus 进程或 Supervisor 进程终止，一旦进程重启，它们将恢复到之前的状态并继续工作。这种设计使 Storm 极其稳定。

基于这样的框架设计，Storm 的工作流程示意如图 7-32 所示，包含 4 个过程。

- (1) 客户端提交 Topology 到 Storm 集群中。
- (2) Nimbus 将分配给 Supervisor 的任务写入 ZooKeeper。
- (3) Supervisor 从 ZooKeeper 中获取所分配的任务，并启动 Worker 进程。
- (4) Worker 进程执行具体的任务。

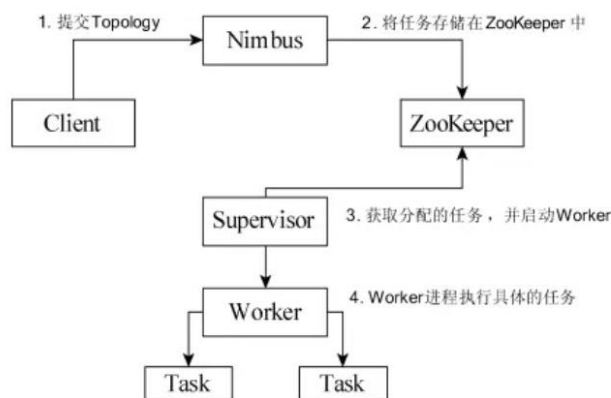


图 7-32 Storm 的工作流程示意

#### 4. Spark 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批处理窗口大小（通常在 0.5~2s）分解为一系列批处理作业，在这个过程中会产生多个 Spark 作业，且每一段数据的处理都会经过 Spark DAG 分解、任务调度过程，因此无法实现毫秒级响应。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 左右）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。

### 7.4.7 流计算框架 Flink

#### 1. Flink 简介

Flink 是 Apache 软件基金会的顶级项目之一，是一个针对流数据和批数据的分布式计算框架，设计思想主要源于 Hadoop、MPP 数据库、流计算系统等。Flink 主要是由 Java 代码实现的，目前

主要还是依靠开源社区的贡献而发展。Flink 所要处理的主要场景是流数据，批数据只是流数据的一个特例而已。也就是说，Flink 会把所有任务当成流来处理。Flink 可以支持本地的快速迭代计算和一些环形的迭代计算任务。

Flink 以层级式系统形式组建其软件栈（见图 7-33），不同层的栈建立在其下层的基础上。具体而言，Flink 的典型特性如下。

（1）提供了面向流处理的 DataStream API 和面向批处理的 DataSet API。DataSet API 支持 Java、Scala 和 Python，DataStream API 支持 Java 和 Scala。

（2）提供了多种候选部署方案，比如本地模式（Local）、集群模式（Cluster）和云模式（Cloud）。对于集群模式而言，可以采用独立模式（Standalone）或 YARN。

（3）提供了一些类库，包括 Table（处理逻辑表查询）、FlinkML（机器学习）、Gelly（图像处理）和 CEP（复杂事件处理）。

（4）提供了较好的 Hadoop 兼容性，不仅可以支持 YARN，还可以支持 HDFS、HBase 等数据源。

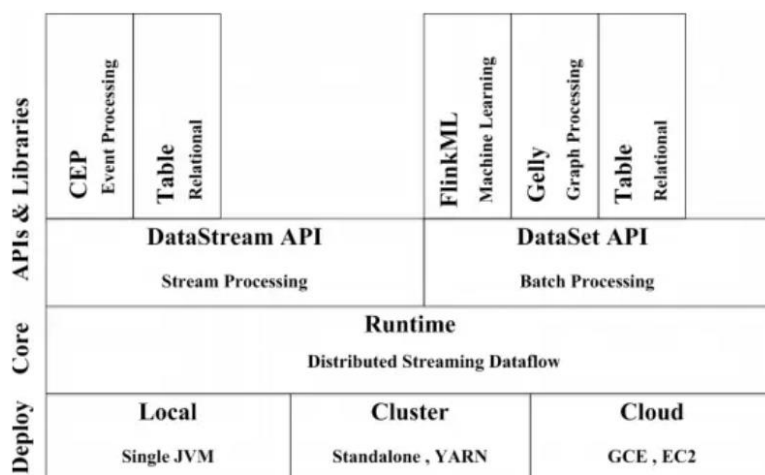


图 7-33 Flink 框架

Flink 和 Spark 一样，都是基于内存的计算框架，因此，都可以获得较好的实时计算性能。当两者都运行在 Hadoop YARN 之上时，Flink 的性能甚至略好于 Spark。因为，Flink 支持增量迭代计算，具有对迭代计算进行自动优化的功能。Flink 和 Spark Streaming 都支持流计算，二者的区别在于，Flink 是一行一行地处理数据，而 Spark Streaming 是基于 RDD 的小批量处理数据，所以，Spark Streaming 在流式处理方面，不可避免地会增加一些延迟，实时性没有 Flink 好。Flink 的计算性能和 Storm 差不多，可以支持毫秒级的响应，而 Spark Streaming 只能支持秒级响应。总体而言，Flink 和 Spark 都是非常优秀的基于内存的分布式计算框架，但是，Spark 的市场影响力和社区活跃度明显超过 Flink，这在一定程度上限制了 Flink 的发展空间。

## 2. Flink 是理想的流计算框架

流处理架构需要具备低延迟、高吞吐和高性能的特性，而目前从市场上已有的产品来看，只有

Flink 可以满足要求。Storm 虽然可以做到低延迟，但是无法实现高吞吐，也不能在故障发生时准确地处理计算状态。Spark Streaming 通过采用微批处理模型实现了高吞吐和容错性，但是牺牲了低延迟和实时处理能力。Structured Streaming 采用持续处理模型可以支持毫秒级的实时响应，但是，这是以牺牲一致性为代价的，持续处理模型只能做到“至少一次”的一致性，而无法保证端到端的完全一致性。

Flink 实现了 Google Dataflow 流计算模型，是一种兼具高吞吐、低延迟和高性能的实时流计算框架，并且同时支持批处理和流处理。此外，Flink 支持高度容错的状态管理，防止状态在计算过程中因为系统异常而出现丢失。因此，Flink 成为了能够满足流处理架构要求的理想的流计算框架。

### 3. Flink 体系架构

如图 7-34 所示，Flink 体系架构主要由两个组件组成，分别为 JobManager 和 TaskManager，Flink 体系架构也遵循 Master/Slave 架构设计原则，JobManager 为 Master 节点，TaskManager 为 Slave 节点。

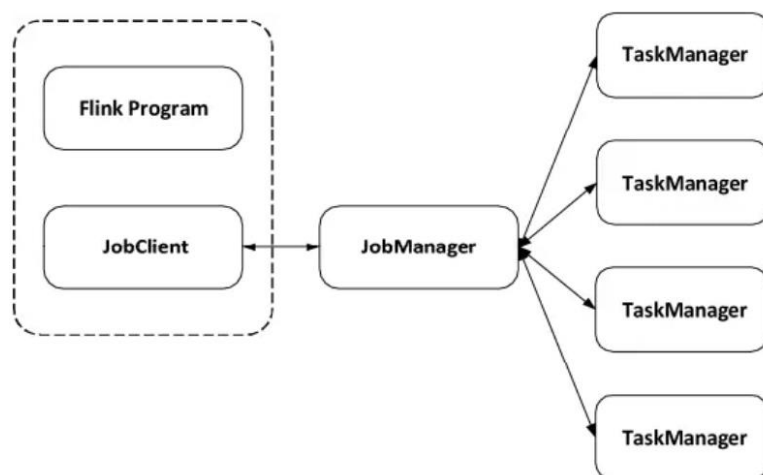


图 7-34 Flink 体系架构

在执行 Flink 程序时，Flink 程序需要首先提交给 JobClient，然后，JobClient 将任务提交给 JobManager。JobManager 负责协调资源分配和任务执行，它首先要做的是分配所需的资源。资源分配完成后，任务将提交给相应的 TaskManager。在接收任务时，TaskManager 启动一个线程以开始执行。执行到位时，TaskManager 会继续向 JobManager 报告状态更改。任务可以有各种状态，例如开始执行、正在进行或已完成。任务执行完成后，其结果将发送回客户端（JobClient）。

## 7.4.8 大数据编程框架 Beam

在大数据处理领域，开发者经常要用到很多不同的技术、框架、API、开发语言和 SDK。根据不同的企业业务系统开发需求，开发者很可能会用 MapReduce 进行批处理，用 Spark SQL 进行交互式查询，用 Flink 实现实时流处理，还有可能用到基于云端的机器学习框架。大量的开源大数据产品（如 MapReduce、Spark、Flink、Storm、Apex 等），为大数据开发者提供了丰富的工具的同时，增加了开发者选择合适工具的难度，尤其对于新入行的开发者来说更是如此。新的分布式处理框架可能带来更高的性能、更强大的功能和更低的延迟，但是，用户切换到新的分布式处

理框架的代价也非常大——需要学习一个新的大数据处理框架，并重塑所有的业务逻辑。解决这个问题的思路包括两个部分：首先，需要一个编程范式，能够统一、规范分布式数据处理的需求，例如，统一批处理和流处理的需求；其次，生成的分布式数据处理任务，应该能够在各个分布式执行引擎（如 Spark、Flink 等）上执行，用户可以自由切换分布式数据处理任务的执行引擎与执行环境。Apache Beam 的出现，就是为了解决这个问题。

Beam 是由谷歌贡献的 Apache 软件基金会的顶级项目，它的目标是开发者提供一个易于使用、又很强大的数据并行处理模型，能够支持流处理和批处理，并兼容多个运行平台。Beam 是一个开源的统一的编程模型，开发者可以使用 Beam SDK 来创建数据处理管道，然后，这些程序可以在任何支持的执行引擎上运行，比如运行在 Apex、Spark、Flink、Cloud Dataflow 上。Beam SDK 定义了开发分布式数据处理任务业务逻辑的 API，即提供一个统一的编程接口给上层应用的开发者。开发者不需要了解底层的具体的大数据平台的开发接口是什么，不管输入是用于批处理的有限数据集，还是用于流处理的无限数据集，直接通过 Beam SDK 的接口，就可以开发数据处理的加工流程。对于有限或无限的输入数据，Beam SDK 都使用相同的类来表现，并且使用相同的转换操作进行处理。

如图 7-35 所示，终端用户用 Beam 来实现自己所需的流计算功能，使用的终端语言可能是 Python、Java 等，Beam 为每种语言提供了一个对应的 SDK，用户可以使用相应的 SDK 创建数据处理管道，用户写出的程序可以被运行在各个 Runner 上，每个 Runner 都实现了从 Beam 管道到平台功能的映射。目前主流的大数据处理框架 Flink、Spark、Apex 以及谷歌的 Cloud DataFlow 等，都有了支持 Beam 的 Runner。通过这种方式，Beam 使用一套高层抽象的 API 屏蔽了多种计算引擎的区别，开发者只需要编写一套代码就可以运行在不同的计算引擎之上（如 Apex、Spark、Flink、Cloud Dataflow 等）。

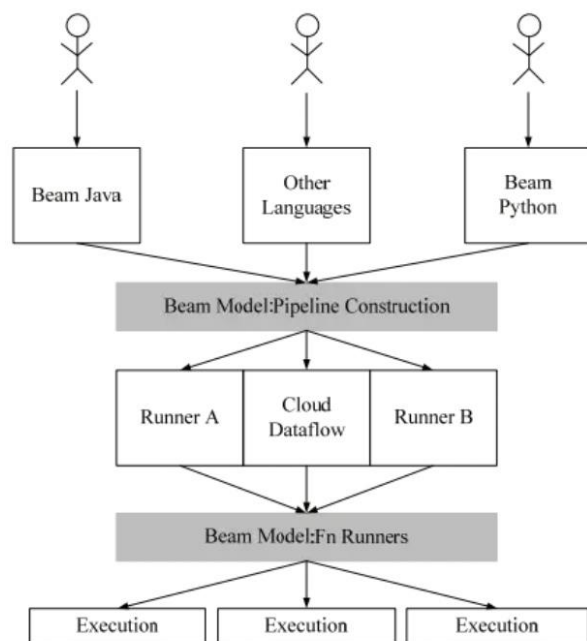


图 7-35 Beam 使用一套高层抽象的 API 屏蔽多种计算引擎的区别

## 7.4.9 查询分析系统 Dremel

随着 Hadoop 的流行,大规模的数据分析系统已经越来越普及。但是, Hadoop 比较适用于大规模数据的批量处理,而对于实时的交互式处理就显得力不从心,比如, Hadoop 通常无法做到让用户在 2~3s 迅速完成 PB 级别数据的查询。因此,大数据分析师需要一个能够快速分析数据的交互式系统,这样就可以非常方便快捷地浏览数据,建立分析模型。谷歌设计的 Dremel 就是一个能够满足这种实时交互式处理的系统。

Dremel 具有以下几个主要的特点。

(1) Dremel 是一个大规模、稳定的系统。在一个 PB 级别的数据集上面,将任务缩短到秒级,无疑需要大量的并发计算,这就需要大量机器的参与,但是,机器越多,出问题的概率越大。如此大的集群规模,需要有足够的容错考虑,保证整个分析的速度不被集群中的个别慢(坏)节点所影响。

(2) Dremel 是 MapReduce 交互式查询能力不足的补充。和 MapReduce 一样, Dremel 也需要和数据运行在一起,将计算移动到数据上面,所以,它也需要 GFS 这样的文件系统作为存储层。在设计之初, Dremel 并非是 MapReduce 的替代品,它只是可以执行非常快的分析,在使用的时候,常用它来处理 MapReduce 的结果集或者用来建立分析原型。

(3) Dremel 的数据模型是嵌套的。互联网数据常是非关系的,这就要求 Dremel 必须有一个灵活的数据模型,这个数据模型对于获得高性能的交互式查询而言至关重要。因此, Dremel 采用了嵌套数据模型,有点类似于 JSON。嵌套数据模型相对于关系模型而言具有明显的优势。传统的关系模型中不可避免地存在大量连接操作,因此,在处理如此大规模的数据的时候,往往是有心无力的。而嵌套数据模型可以在 PB 级别数据上“一展身手”。

(4) Dremel 中的数据是用列式存储的。当对采用列式存储的数据进行分析的时候就可以只扫描需要的那部分数据,从而大大减少磁盘的访问量。同时,列式存储是压缩友好的,可以实现更高的压缩率,使 CPU 和磁盘发挥最大的效能。

(5) Dremel 结合了 Web 搜索和并行 DBMS 的技术。首先,它借鉴了 Web 搜索中的“查询树”的概念,将一个相对巨大、复杂的查询分割成较小、较简单的查询。“大事化小,小事化了”,能并发地在大量节点上查询。其次,和并行 DBMS 类似, Dremel 可以提供一个类似 SQL 的接口,就像 Hive 和 Pig 那样。

Dremel 从 2006 年就已经投入开发,并且在谷歌已经有了几千用户。多种多样的 Dremel 实例被部署在谷歌里,每个实例拥有数十至数千个节点。使用 Dremel 系统的例子如下。

- 分析网络文档。
- 追踪 Android 市场应用程序的安装数据。
- 谷歌产品的崩溃报告分析。
- Google Books 的 OCR 结果。
- 垃圾邮件分析。

- Google Maps 里地图部件调试。
- 管理中的 Bigtable 实例的 Tablet 迁移。
- 谷歌分布式构建系统中的测试结果分析。
- 数万个硬盘的磁盘 IO 统计信息。
- 谷歌数据中心上运行的任务的资源监控。
- 谷歌代码库的符号和依赖关系分析。

## 7.5 本章小结

大数据处理与分析是传统数据处理与分析的进一步发展。虽然它依然采用具有多年历史的统计学、机器学习和数据挖掘方法，但是，在实现方式层面有了质的变革，已经从传统的单机程序演进到了分布式程序。分布式程序开发本是一个比较复杂的工作，具有很高的门槛，但是，MapReduce、Spark 和 Flink 等分布式计算框架的出现，大大降低了分布式程序开发者的工作负担，只需要简单地编程，就可以实现复杂的分布式计算程序，从而可以充分利用计算机集群构建起强大的数据分析能力。

本章对数据处理与分析的概念做了介绍，并详细阐述了大数据处理分析技术及其代表性产品。

## 7.6 习题

1. 试述数据分析的概念及其与数据处理的关系。
2. 试述机器学习的概念及其与数据挖掘的关系。
3. 试述常见的机器学习和数据挖掘算法有哪些。
4. 试述协同过滤算法有哪些种类。
5. 试述典型的大数据处理与分析技术有哪几种类型，并给出代表性产品。
6. 试述流计算的概念及其处理流程。
7. 试述通用的图计算软件有哪几种。
8. 试述 MapReduce 的工作流程。
9. 试述 MapReduce 有哪些不足之处。
10. 请将数据仓库 Hive 和传统数据库进行对比分析。
11. 试述数据仓库 Hive 的体系架构。
12. 试述 Spark 相对于 MapReduce 的优点。
13. 试述 Spark 与 Hadoop 的关系。
14. 试述 Spark 的体系架构包含哪些组件。

15. 试述 Spark 的部署方式有哪几种。
16. 试述为什么推出 Spark SQL。
17. 试述 Spark Streaming 的基本原理。
18. 试述 Structured Streaming 有哪几种处理模型。
19. 请将 Structured Streaming 和 Spark SQL、Spark Streaming 进行对比分析。
20. 试述 Spark MLlib 的功能以及它提供了哪些工具。
21. 试述 TensorFlowOnSpark 的 Spark 应用程序包括哪几个基本过程。
22. 请画出 Storm 的集群架构并加以简要说明。
23. 试述 Storm 的工作流程。
24. 请对 Spark Streaming 和 Storm 进行简要对比。
25. 试述为什么流计算场景比较适合采用 Flink。
26. 试述 Flink 的体系架构包含哪些组件。
27. 试述 Beam 的设计目标。
28. 试述查询分析系统 Dremel 具有哪些特点。