# 《大数据与（统计）机器学习》

（2021年秋季学期）

翟祥
北京林业大学

E-mail: zhaixbh@126.com

# 第4章

# 数据处理与分析

# 提纲

**4.1 数据处理与分析的概念**
**4.2机器学习和数据挖掘算法**
**4.3 大数据处理与分析技术**
**4.4大数据处理与分析代表性产品**

# 4.1 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，广义的数据分析就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及工具，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。

## 4.1.1 数据分析与数据挖掘

## 4.1.2数据分析与数据处理

## 4.1.3大数据处理与分析

# 4.1.1 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，这些指标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。

# 4.1.1 数据分析与数据挖掘

综上所述，数据分析（狭义）与数据挖掘的本质都是一样的，都是从数据里面发现关于业务的知识（有价值的信息），从而帮助业务运营、改进产品以及帮助企业做更好的决策。所以，数据分析（狭义）与数据挖掘构成广义的数据分析。

机器学习：自动化

# 4.1.2数据分析与数据处理

数据分析过程通常会伴随着发生数据处理（或者说伴随着大量数据计算），因此，数据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）。

# 4.1.3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（比如SPSS、SAS等）或单机编程语言（比如Python、R），以及大数据处理与分析技术（比如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抽样数据为主，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（比如SPSS和SAS）或者单机编程（比如Python、R）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抽样数据进行分析，这时，单机工具和单机程序已经显得“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”。

# 4.1.3大数据处理与分析

本章后续内容中，在数据分析理论层面，只介绍属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。

# 4.2机器学习和数据挖掘算法

4.2.1概述
4.2.2 分类
4.2.3聚类
4.2.4 回归分析
4.2.5关联规则
4.2.6协同过滤

# 4.2.1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。从知识的来源角度而言，数据挖掘领域的很多知识也“间接”来自于统计学界，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。

# 4.2.1概述

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的情形，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。

# 4.2.2 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。

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它数据元组进行分类。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。

# 4.2.2 分类

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三个品种之一。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鸢尾花的品种。

# 4.2.2 分类

聚类分析的常见应用场景包括：
（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，并依照特定的指标标量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，并且在此基础分别制定和相应的开发计划、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，比如，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，需要风控部门提前关注。

# 4.2.3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。聚类分析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。

聚类分析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，对数据进行区分和分类。聚类既能作为一个单独过程，用于找寻数据内部的分布结构，也可以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）。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）。聚类算法应该嵌入到问题中进行评价。

# 4.2.4 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。

# 4.2.5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。假设零售商想更多地了解顾客的购物习惯。特别是，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，可以对商店的顾客事物零售数量进行购物篮分析。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同时购买，从而帮助他们开发更好的营销策略。

关联规则定义为：假设是$I=\{I_1,I_2,I_3,\ldots,I_m\}$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$D$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，则认为关联规则是有趣的。这些阈值是根据挖掘需要人为设定的。

# 4.2.5关联规则

这里举一个简单的例子进行说明。表4-1是顾客购买记录的数据库*D*，包含6个事务。项集*I*={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*^*Y*=3, *D*=6，支持度(*X*^*Y*)/*D*=0.5；*X*=5, 置信度(*X*^*Y*)/*X*=0.6。若给定最小支持度α= 0.5，最小置信度β= 0.6，认为购买乒乓球拍和购买乒乓球之间存在关联。

表　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。

# 4.2.6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”。

# 4.2.6协同过滤

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（ModelCF）是通过已经观察到的所有用户给产品的打分，来推断每个用户的喜好并向用户推荐适合的产品。实际上，ModelCF同时考虑了用户和物品两个方面，因此，它也可以看作是UserCF和ItemCF的混合形式。

# 4.3 大数据处理与分析技术

4.3.1技术分类
4.3.2 流计算
4.3.3 图计算

# 4.3.1技术分类

表 大数据计算模式及其代表产品

| 大数据计算模式 | 解决问题 | 代表产品 |
| --- | --- | --- |
| 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、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 |

# 4.3.2 流计算

**1.流计算概念**

流计算：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。

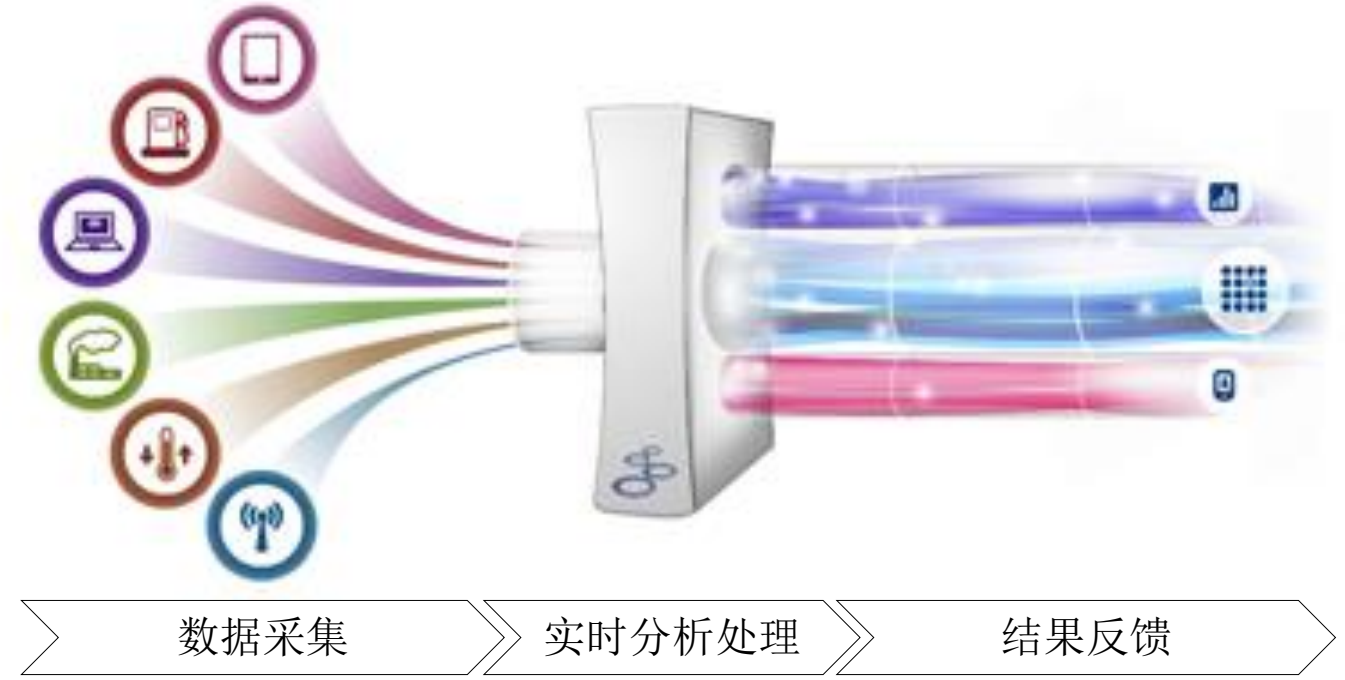



图4-1 流计算示意图

# 4.3.2 流计算

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，**如用户点击流。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，就需要一个低延迟、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：

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

# 4.3.2 流计算

**2.流计算的处理流程**

传统的数据处理流程，需要先采集数据并存储在关系数据库等数据管理系统中，之后由用户通过查询操作和数据管理系统进行交互

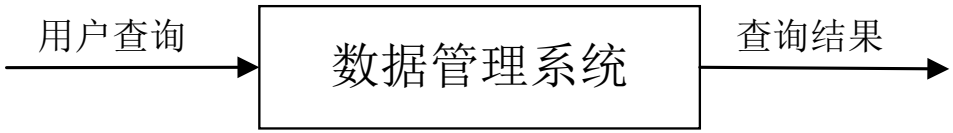



图4-2 传统的数据处理流程示意图

传统的数据处理流程隐含了两个前提：

•**存储的数据是旧的**。存储的静态数据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

•**需要用户主动发出查询来获取结果**

# 4.3.2 流计算

流计算的处理流程一般包含三个阶段：数据实时采集、数据实时计算、实时查询服务

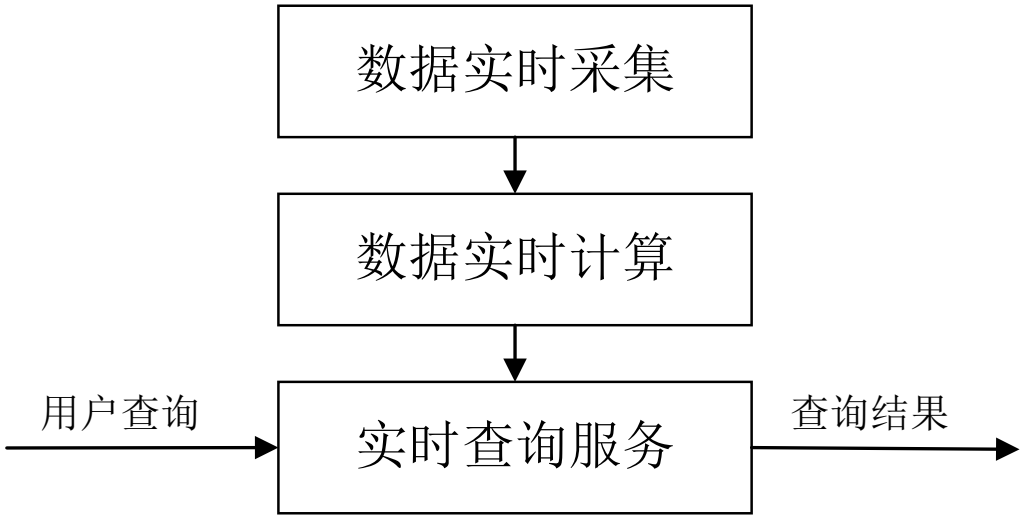
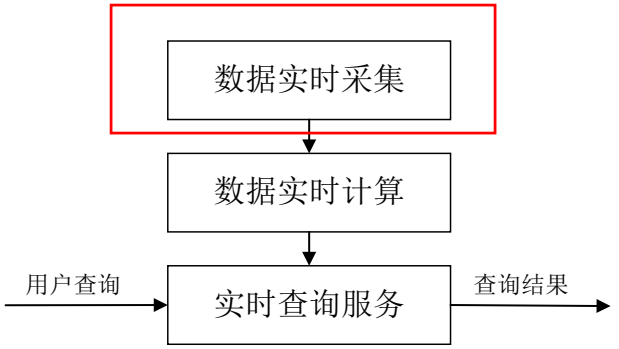



图4-3 流计算处理流程示意图

# 4.3.2 流计算

- 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上的日志数据
- 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Time Tunnel
  - 基于Hadoop的Chukwa和Flume



# 4.3.2 流计算

- 数据采集系统的基本架构一般有以下三个部分：
  - Agent：主动采集数据，并把数据推送到Collector部分
  - Collector：接收多个Agent的数据，并实现有序、可靠、高性能的转发
  - Store：存储Collector转发过来的数据（对于流计算不存储数据）

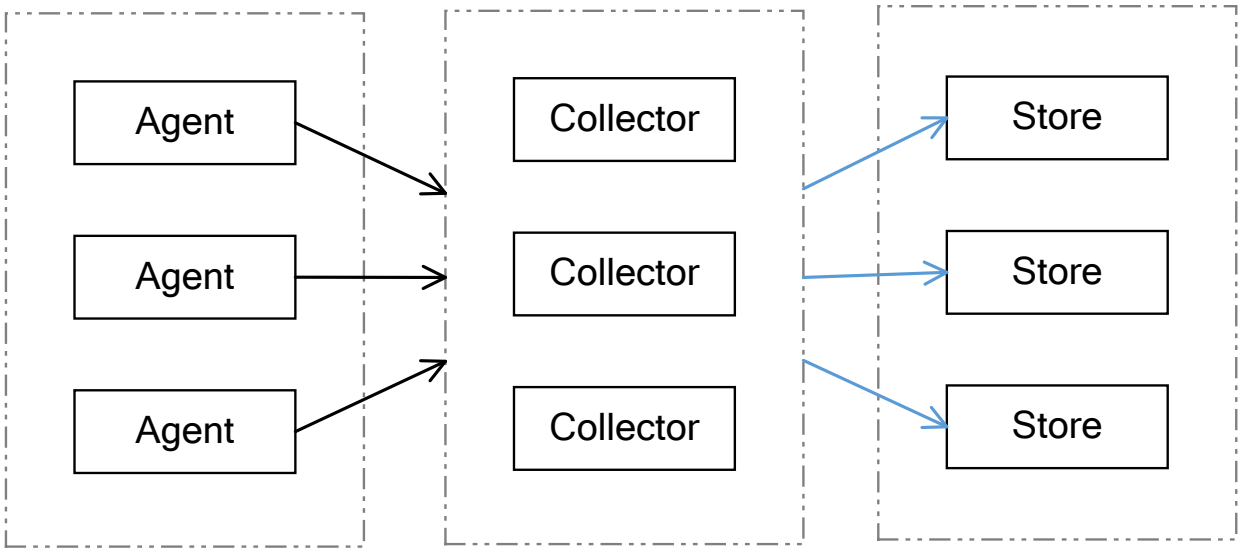



图4-4 数据采集系统基本架构

# 4.3.2 流计算

- 数据实时计算阶段对采集的数据进行实时的分析和计算，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，以便之后再进行分析计算。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

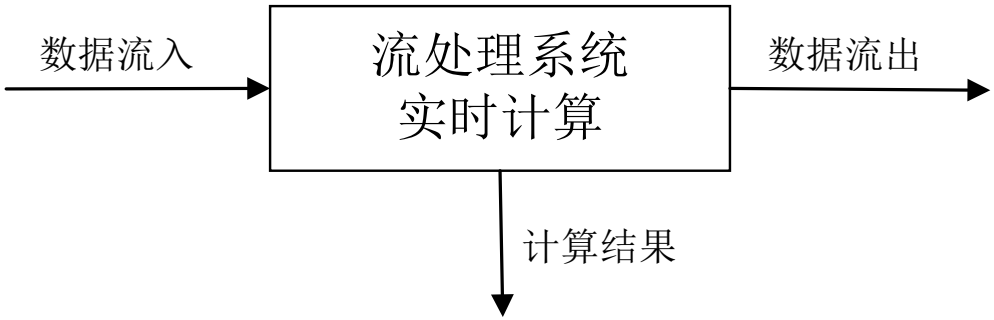
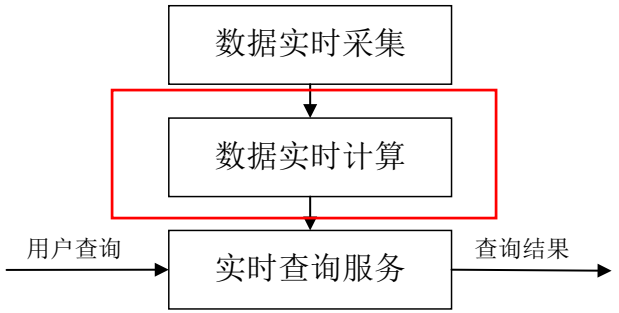
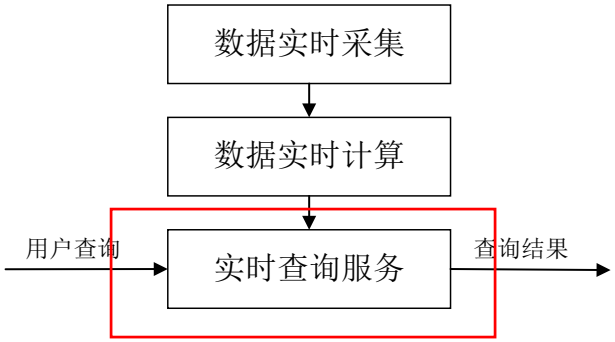



图4-5 数据实时计算流程



# 4.3.2 流计算

- 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、展示或储存
- 传统的数据处理流程，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，也可以实现不断地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



# 4.3.2 流计算

- 可见，流处理系统与传统的数据处理系统有如下不同：
  - 流处理系统处理的是实时的数据，而传统的数据处理系统处理的是预先存储好的静态数据
  - 用户通过流处理系统获取的是实时结果，而通过传统的数据处理系统，获取的是过去某一时刻的结果
  - 流处理系统无需用户主动发出查询，实时查询服务可以主动将实时结果推送给用户

# 4.3.3 图计算

## 1.传统图计算解决方案的不足之处

很多传统的图计算算法都存在以下几个典型问题：

（1）常常表现出比较差的内存访问局部性

（2）针对单个顶点的处理工作过少

（3）计算过程中伴随着并行度的改变

# 4.3.3 图计算

针对大型图（比如社交网络和网络图）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：**通用性不好
- **（2）基于现有的分布式计算平台进行图计算：**在性能和易用性方面往往无法达到最优
  - 现有的并行计算框架像MapReduce还无法满足复杂的关联性计算
  - 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，力不从心
  - 比如，有公司利用MapReduce进行社交用户推荐，对于5000万注册用户，50亿关系对，利用10台机器的集群，需要超过10个小时的计算
- **（3）使用单机的图算法库：**比如BGL、LEAD、NetworkX、JDSL、Standford GraphBase和FGL等，但是，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：**比如，Parallel BGL和CGM Graph，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方面（比如容错），无法提供较好的支持

# 4.3.3 图计算

## 2.图计算通用软件

- 传统的图计算解决方案无法解决大型图的计算问题，因此，就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 Infinite Graph
  - 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GoldenOrb、Giraph、Pregel和Hama，这些图处理软件主要是基于BSP模型实现的并行图处理系统

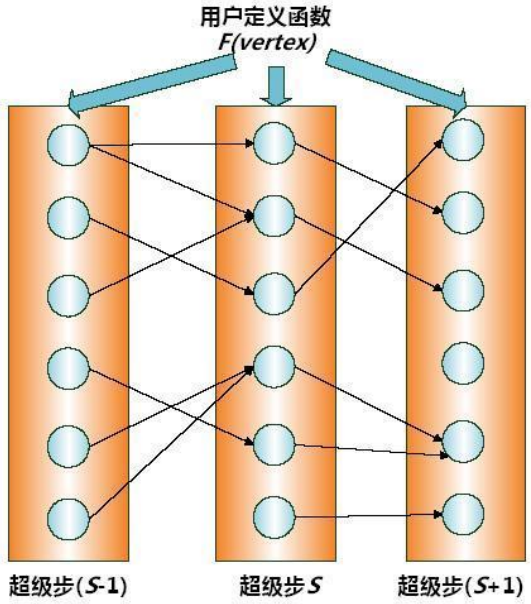
# 4.3.3 图计算

一次BSP(Bulk Synchronous Parallel Computing Model，又称“大同步”模型)计算过程包括一系列全局超步（所谓的超步就是计算中的一次迭代），每个超步主要包括三个组件：

•**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

•**通讯**：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

•**栅栏同步**(Barrier Synchronization)：当一个处理器遇到“路障”（或栅栏），会等到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



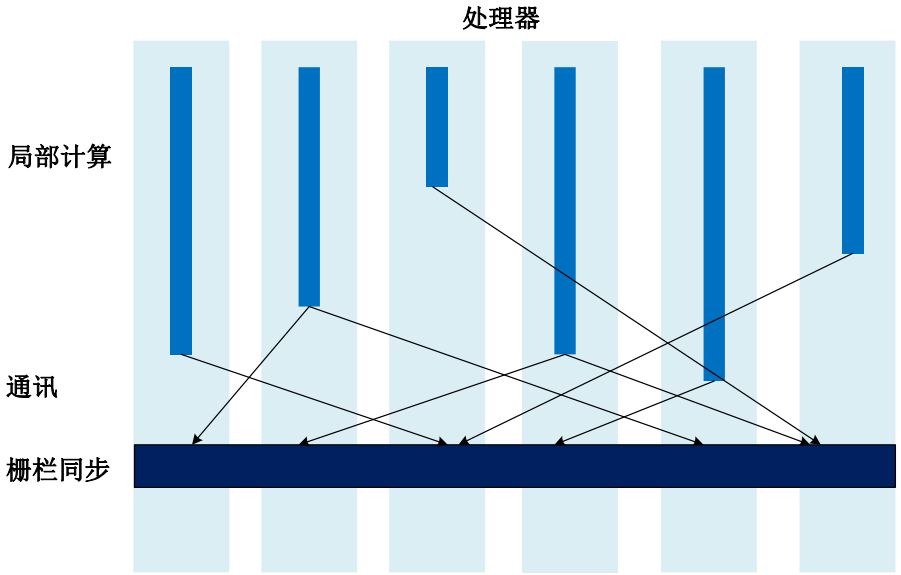




图4-6 一个超步的垂直结构图

# 4.4大数据处理与分析代表性产品

4.4.1 分布式计算框架MapReduce
4.4.2 数据仓库Hive
4.4.3数据仓库Impala
4.4.4 基于内存的分布式计算框架Spark
4.4.5 TensorFlowOnSpark
4.4.6 流计算框架Storm
4.4.7 流计算框架Flink
4.4.8大数据编程框架Beam
4.4.9查询分析系统Dremel

# 4.4.1 分布式计算框架MapReduce

## 1.MapReduce简介

谷歌在2003年～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是谷歌公司的核心计算模型。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到两个函数：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中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。

# 4.4.1 分布式计算框架MapReduce

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，从而减少了节点间的数据移动开销。

# 4.4.1 分布式计算框架MapReduce

## 2.MapReduce工作流程

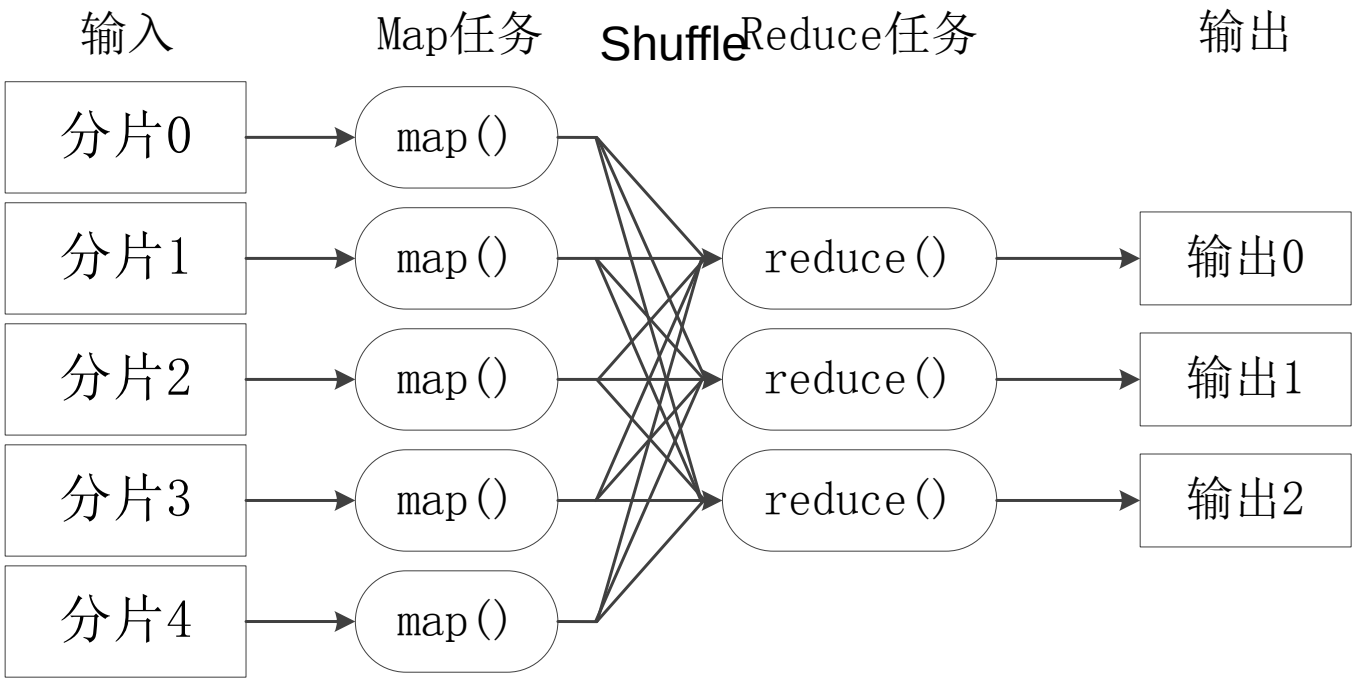



图4-7 MapReduce工作流程

- 不同的Map任务之间不会进行通信
- 不同的Reduce任务之间也不会发生任何信息交换
- 用户不能显式地从一台机器向另一台机器发送消息
- 所有的数据交换都是通过MapReduce框架自身去实现的

# 4.4.1 分布式计算框架MapReduce

**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。
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到磁盘中，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，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（比如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还是在其他框架中得到了很好的继承。

# 4.4.2 数据仓库Hive

## 1. Hive简介

- Hive是一个构建于Hadoop顶层的数据仓库工具
- 某种程度上可以看作是用户编程接口，本身不存储和处理数据
- 依赖分布式文件系统HDFS存储数据
- 依赖分布式并行计算模型MapReduce处理数据
- 定义了简单的类SQL 查询语言——HiveQL
- 用户可以通过编写的HiveQL语句运行MapReduce任务
- 是一个可以提供有效、合理、直观组织和使用数据的模型

# 4.4.2 数据仓库Hive

**2.Hive与Hadoop生态系统中其他组件的关系**

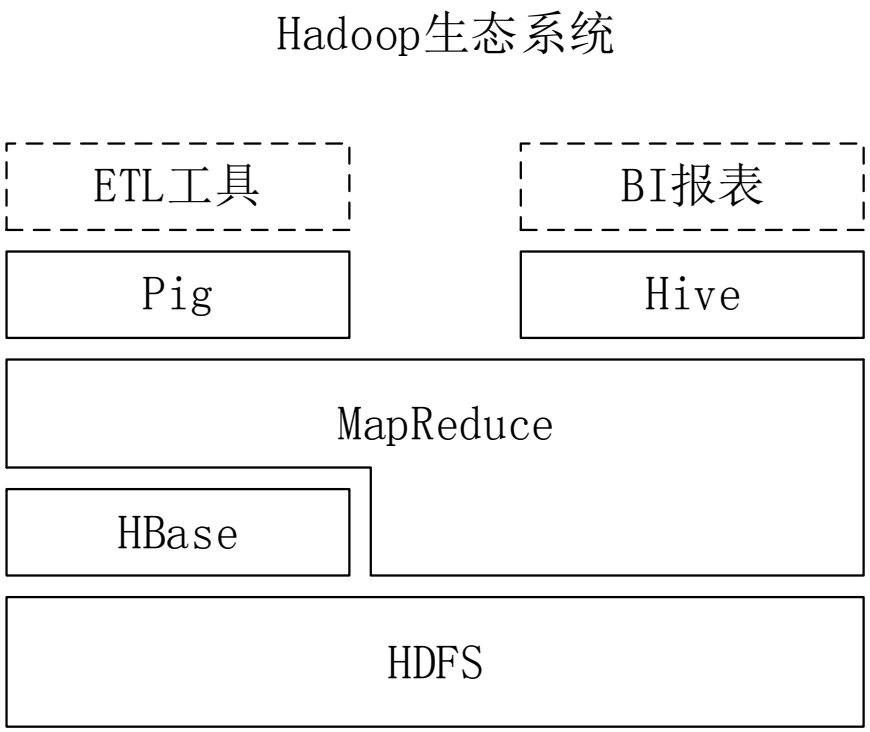



图4-8 Hadoop生态系统中Hive与其他部分的关系

# 4.4.2 数据仓库Hive

- **Hive依赖于HDFS 存储数据**

HDFS作为高可靠性的底层存储，用来存储海量数据

- **Hive依赖于MapReduce 处理数据**

MapReduce对这些海量数据进行处理，实现高性能计算，用HiveQL语句编写的处理逻辑最终均要转化为MapReduce任务来运行

- **Pig可以作为Hive的替代工具**

pig是一种数据流语言和运行环境，适合用于Hadoop和MapReduce平台上查询半结构化数据集。常用于ETL过程的一部分，即将外部数据装载到Hadoop集群中，然后转换为用户期待的数据格式

- **HBase 提供数据的实时访问**

HBase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，所以HBase与Hive的功能是互补的，它实现了Hive不能提供功能。

# 4.4.2 数据仓库Hive

## 3. Hive系统架构

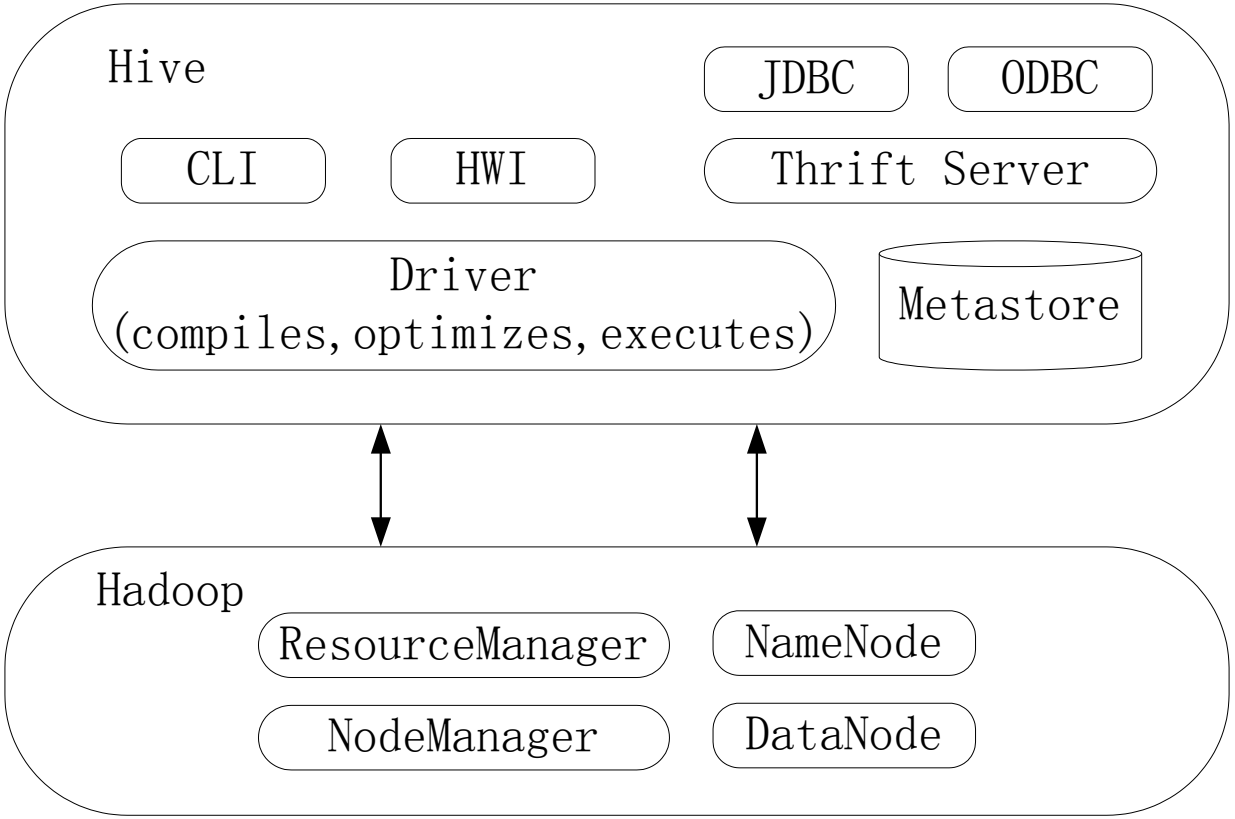



图4-9 Hive系统架构

# 4.4.2 数据仓库Hive

- **用户接口模块：**包括CLI、HWI、JDBC、ODBC、Thrift Server等

  CLI是Hive自带的一个命令行界面；

  HWI是Hive的一个简单网页界面；

  JDBC、ODBC以及Thrift Server可以向用户提供进行编程访问的接口。
- **驱动模块：**包括编译器、优化器、执行器等

所有命令和查询都会进入到驱动模块，通过该模块对输入进行解析编译，对需求的计算进行优化，然后按照指定的步骤进行执行。

- **元数据存储模块（Metastore）：**是一个独立的关系型数据库

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。

# 4.4.3数据仓库Impala

- Impala是由Cloudera公司开发的新型查询系统，它提供SQL语义，能查询存储在Hadoop的HDFS和HBase上的PB级大数据。Impala最开始是参照 Dremel系统进行设计的，Impala的目的不在于替换现有的MapReduce工具，而是提供一个统一的平台用于实时查询。

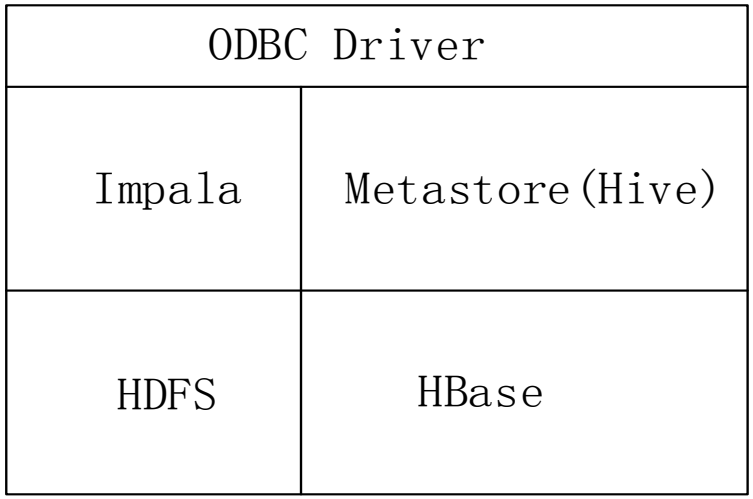



图4-10 Impala与其他组件关系

# 4.4.3数据仓库Impala

- 与Hive类似，Impala也可以直接与HDFS和HBase进行交互。
- 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，例如批量提取、转化、加载类型的任务。
- Impala通过与商用并行关系数据库中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，从而大大降低了延迟，主要用于实时查询。
- Impala和Hive采用相同的SQL语法、ODBC 驱动程序和用户接口。

# 4.4.4 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（UC Berkeley）的AMP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
- 2013年Spark加入Apache孵化器项目后发展迅猛，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（Hadoop、Spark、Storm）
- Spark在2014年打破了Hadoop保持的基准排序纪录
  - Spark/206个节点/23分钟/100TB数据
  - Hadoop/2000个节点/72分钟/100TB数据
  - Spark用十分之一的计算资源，获得了比Hadoop快3倍的速度

# 4.4.4 基于内存的分布式计算框架Spark

Spark具有如下几个主要特点：

•运行速度快：使用DAG执行引擎以支持循环数据流与内存计算
•容易使用：支持使用Scala、Java、Python和R语言进行编程，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
•运行模式多样：可运行于独立的集群模式中，可运行于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

# 4.4.4 基于内存的分布式计算框架Spark

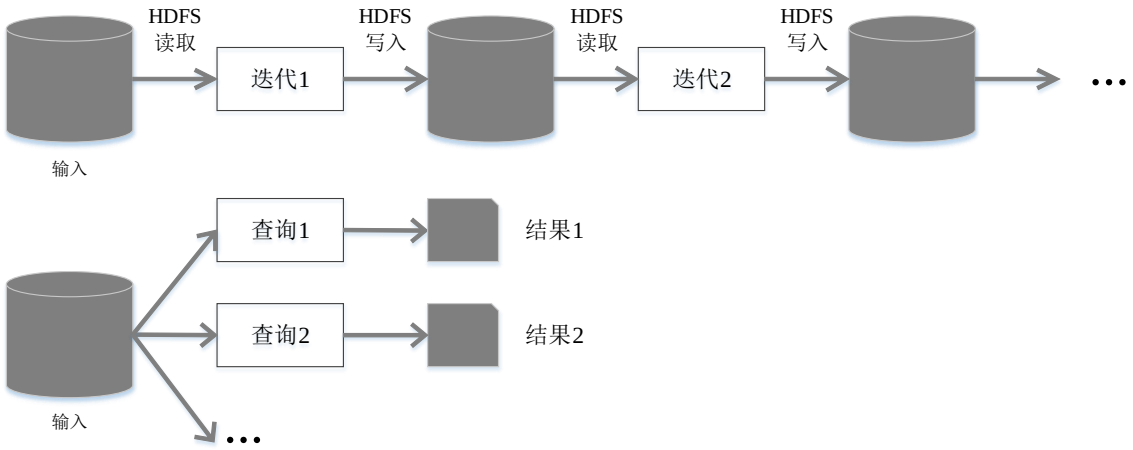
**2.Spark相对于MapReduce的优点**

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Hadoop 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Hadoop MapReduce更灵活
•Spark提供了内存计算，可将中间结果放到内存中，对于迭代运算效率更高
Spark基于DAG的任务调度执行机制，要优于Hadoop MapReduce的迭代执行机制

# 4.4.4 基于内存的分布式计算框架Spark




(a) Hadoop MapReduce执行流程

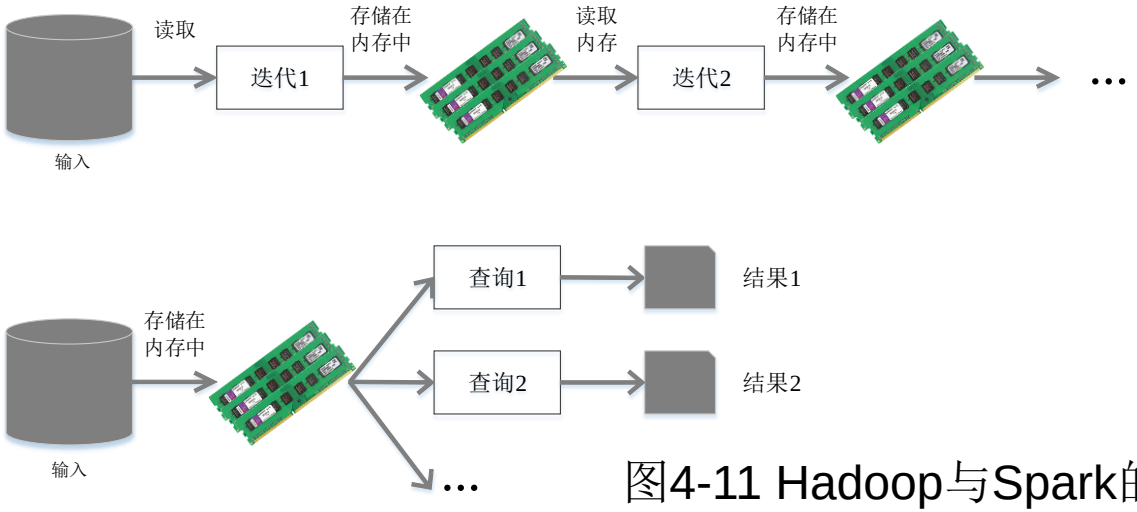



图4-11 Hadoop与Spark的执行流程对比

(b) Spark执行流程

# 4.4.4 基于内存的分布式计算框架Spark

•使用Hadoop进行迭代计算非常耗资源
•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

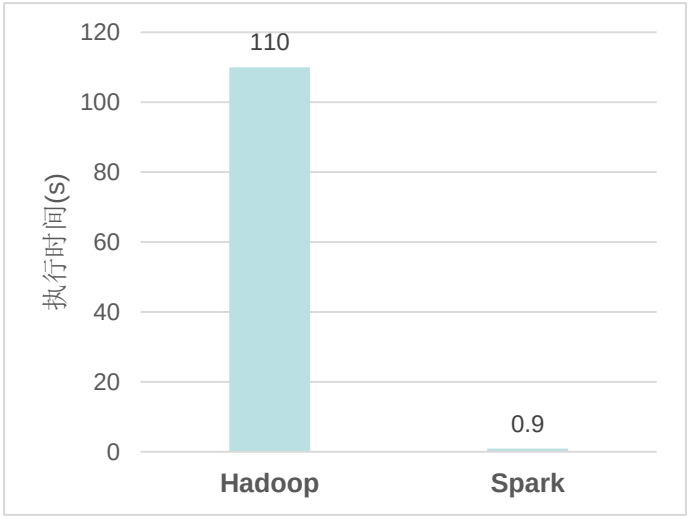



图4-12 Hadoop与Spark执行逻辑回归的时间对比

# 4.4.4 基于内存的分布式计算框架Spark

## 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平台。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。

# 4.4.4 基于内存的分布式计算框架Spark

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之上统一部署各个计算框架（如图所示）。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。

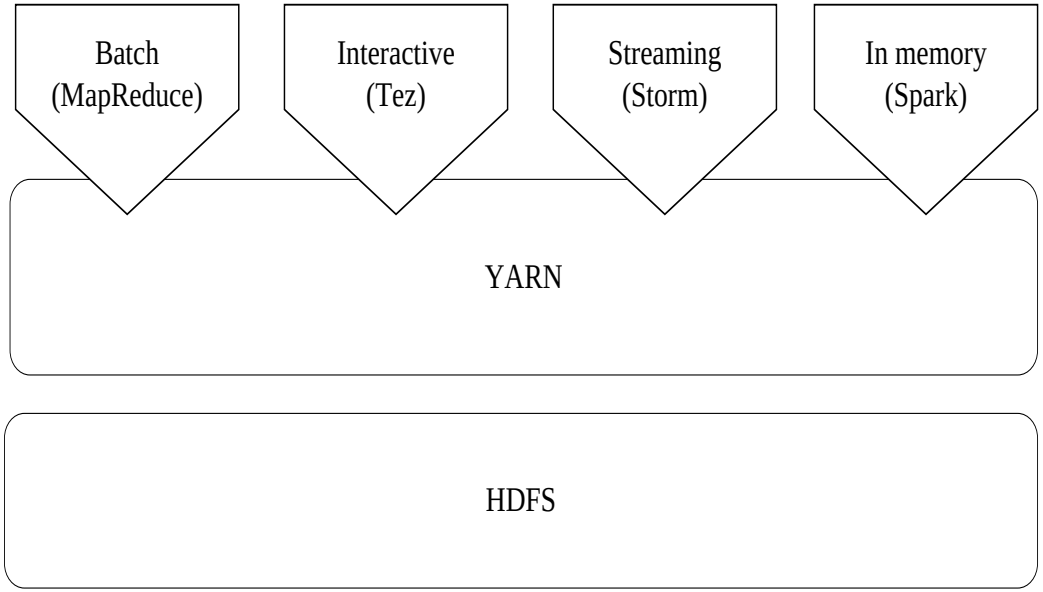



图4-13 Hadoop和Spark的统一部署

# 4.4.4 基于内存的分布式计算框架Spark

**4. Spark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：通常时间跨度在数十分钟到数小时之间
- 基于历史数据的交互式查询：通常时间跨度在数十秒到数分钟之间
- 基于实时数据流的数据处理：通常时间跨度在数百毫秒到数秒之间

当同时存在以上三种场景时，就需要同时部署三种不同的软件

- 比如: MapReduce / Impala / Storm

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

# 4.4.4 基于内存的分布式计算框架Spark

- 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

# 4.4.4 基于内存的分布式计算框架Spark

Spark的生态系统主要包含了Spark Core、Spark SQL、Spark Streaming、Structured Streaming、MLlib和GraphX 等组件

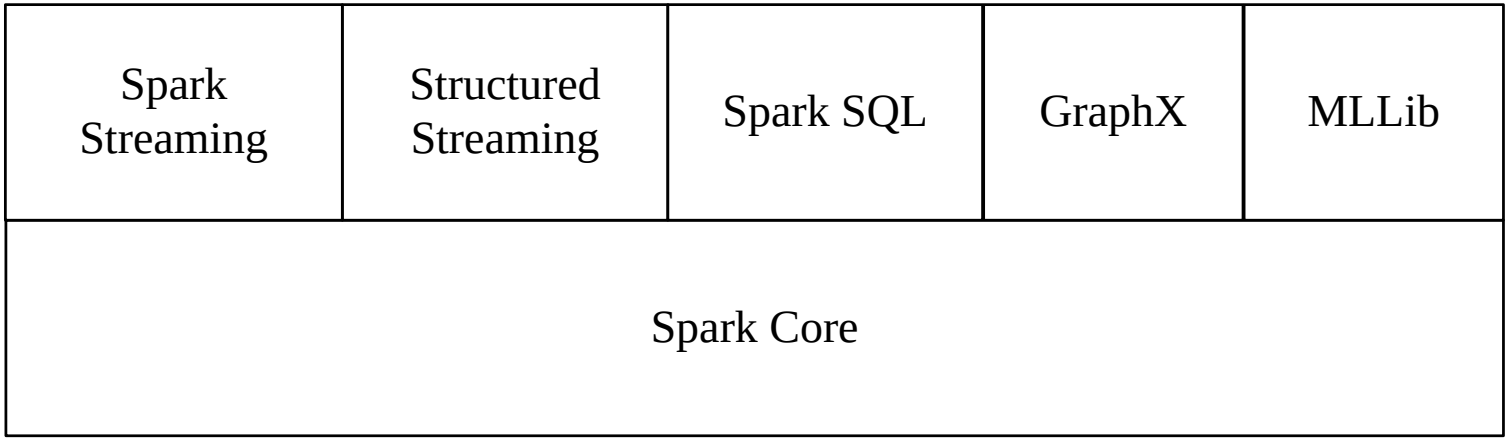



图4-14 Spark生态系统

# 4.4.4 基于内存的分布式计算框架Spark

表 Spark生态系统组件的应用场景

| 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 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 |

# 4.4.4 基于内存的分布式计算框架Spark

**5.Spark体系架构**

•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）

•资源管理器可以自带或Mesos或YARN

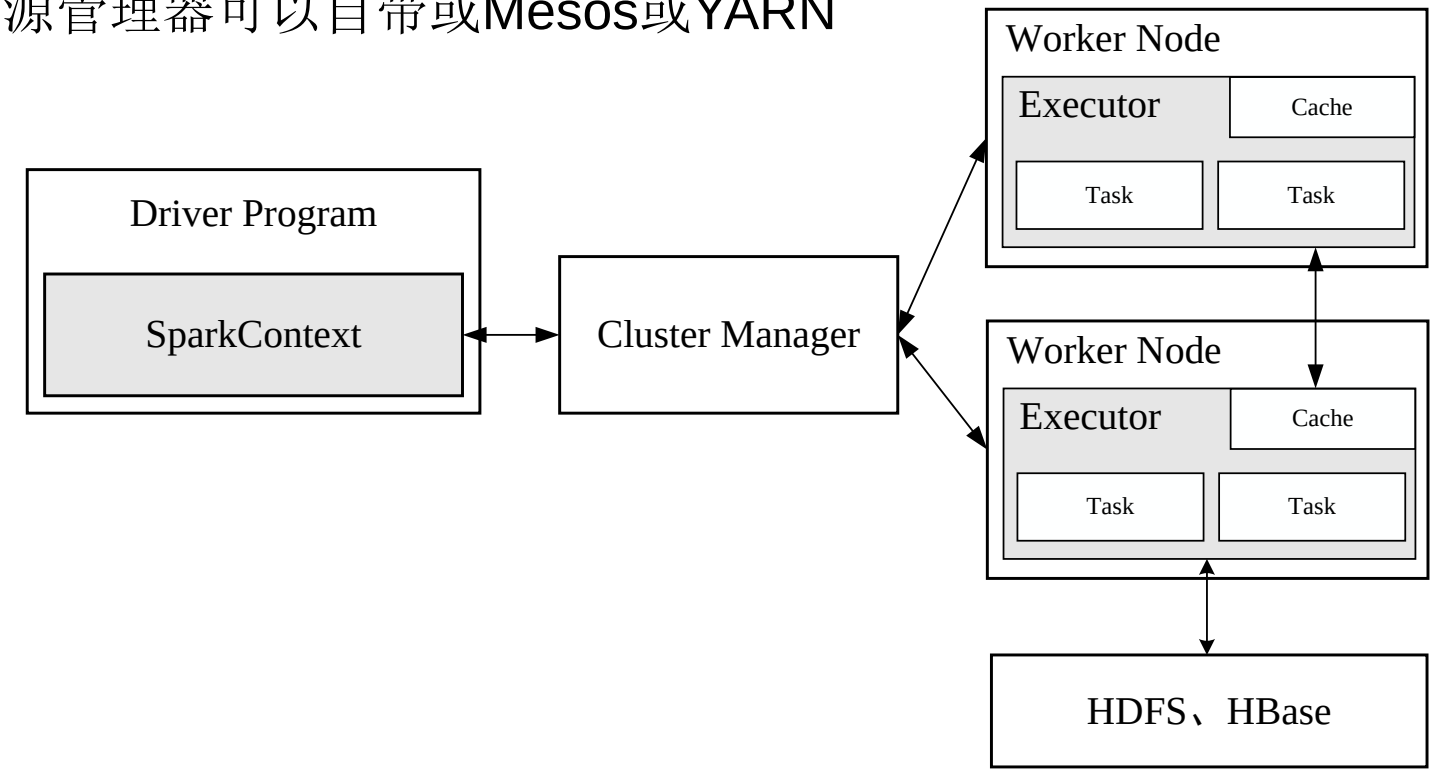



图4-15 Spark运行架构

# 4.4.4 基于内存的分布式计算框架Spark

**6.Spark的数据抽象RDD**

Spark Core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。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。

# 4.4.4 基于内存的分布式计算框架Spark

- 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
- 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（不适合网页爬虫）
- 表面上RDD的功能很受限、不够强大，实际上RDD已经被实践证明可以高效地表达许多框架的编程模型（比如MapReduce、SQL、Pregel）
- Spark提供了RDD的API，程序员可以通过调用API实现对RDD的各种操作

# 4.4.4 基于内存的分布式计算框架Spark

RDD典型的执行过程如下：

•RDD读入外部数据源进行创建

•RDD经过一系列的转换（Transformation）操作，每一次都会产生不同的RDD，供给下一个转换操作使用

•最后一个RDD经过“动作”操作进行转换，并输出到外部数据源

# 4.4.4 基于内存的分布式计算框架Spark

**4.Spark的部署方式**

Spark支持五种不同类型的部署方式，包括：

- Local
- Standalone（类似于MapReduce1.0，slot为资源分配单位）
- Spark on Mesos（和Spark有血缘关系，更好支持Mesos）
- Spark on YARN
- Spark on Kubernetes

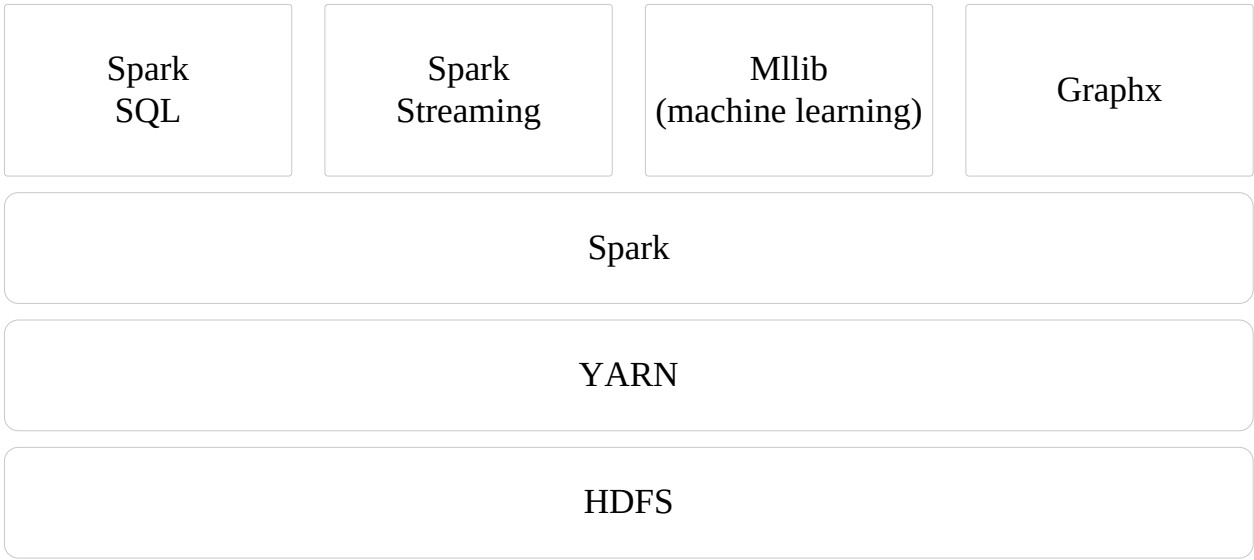


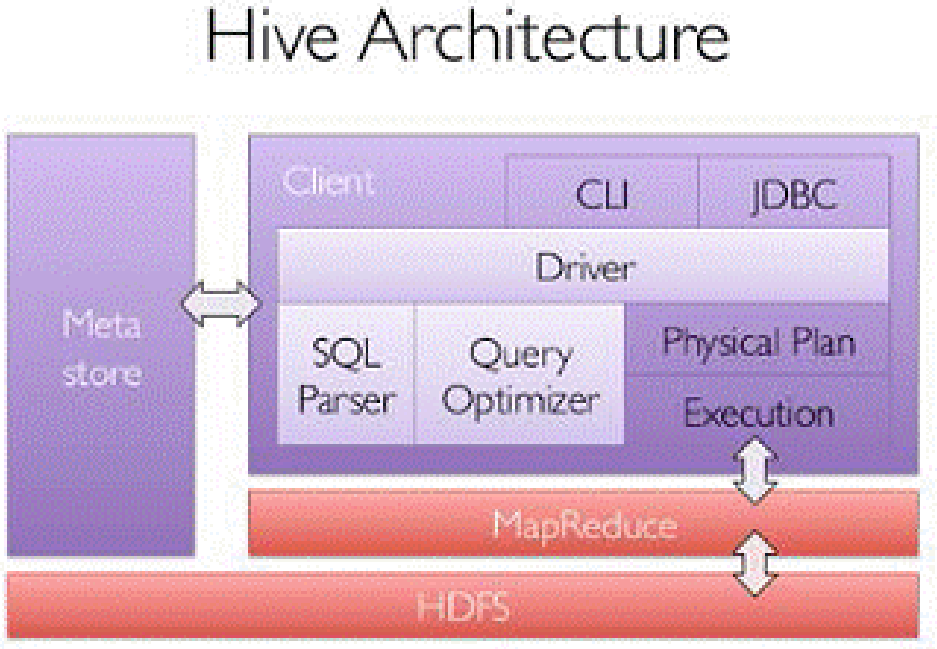

图4-16 Spark on YARN架构

# 4.4.4 基于内存的分布式计算框架Spark

**8.Spark SQL**

（1）从Shark说起

Hive: SQL-on-Hadoop



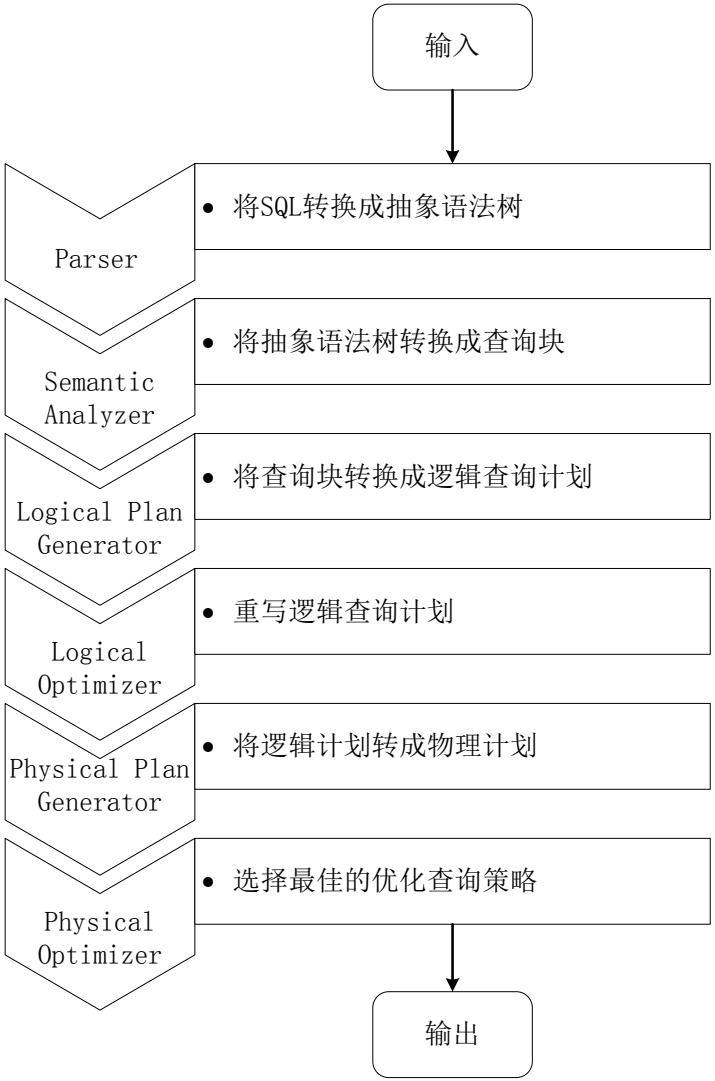




图4-17 Hive中SQL查询的MapReduce作业转化过程

# 4.4.4 基于内存的分布式计算框架Spark

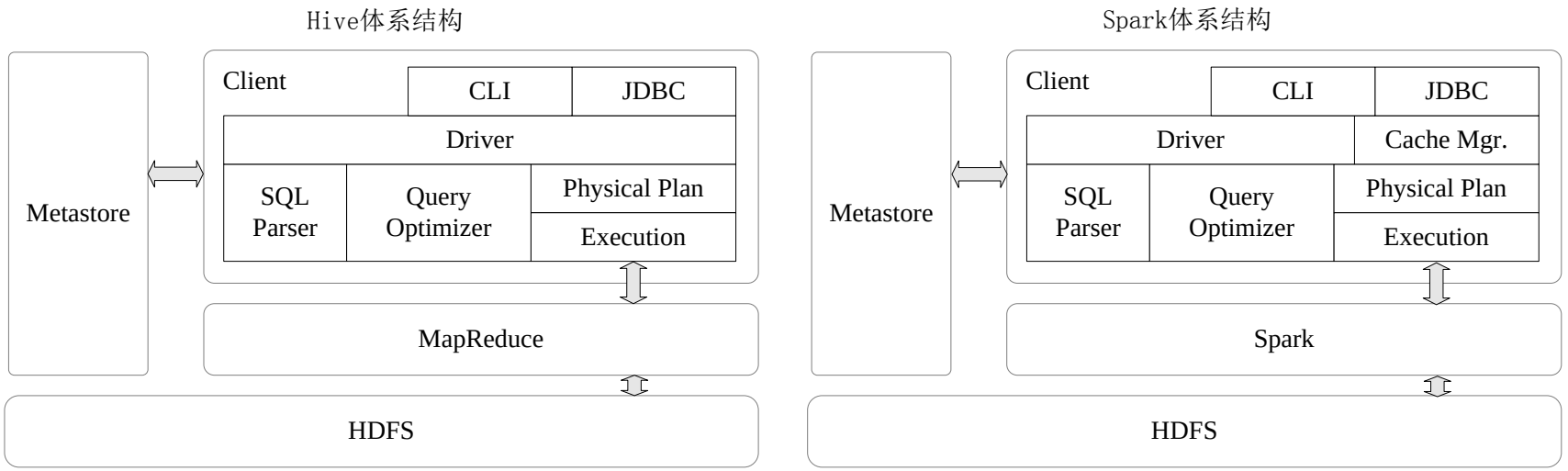
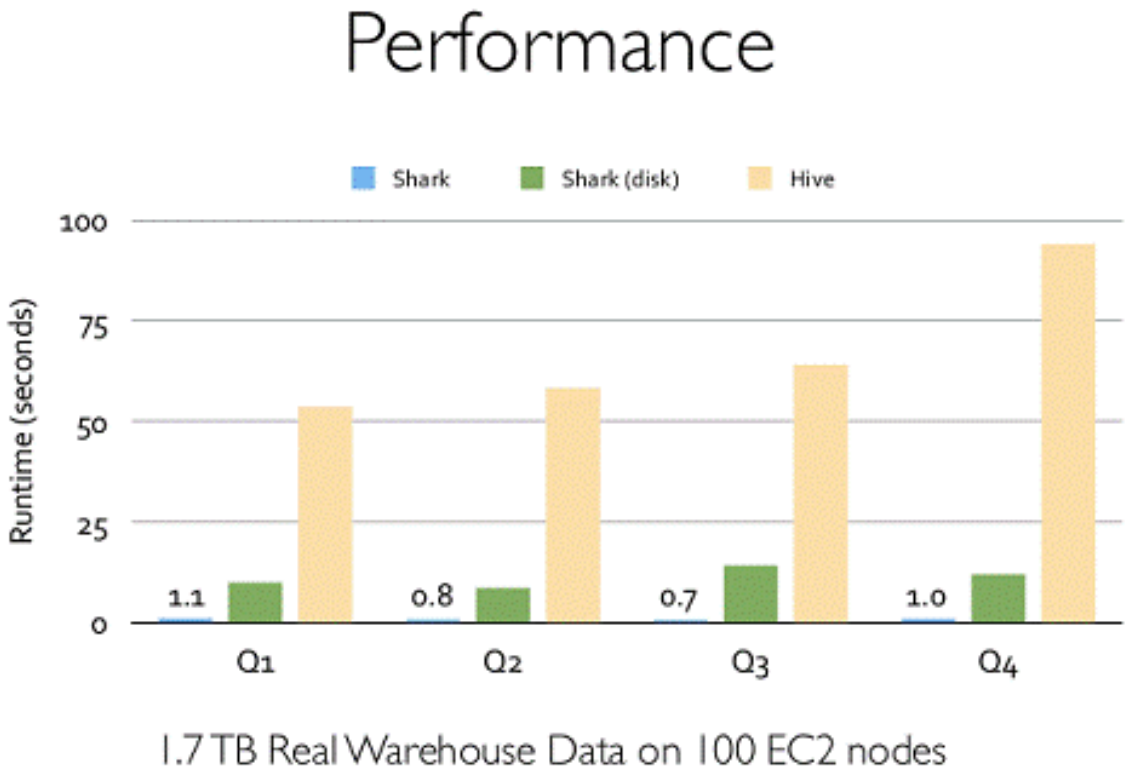



图4-18 Shark直接继承了Hive的各个组件

•Shark即Hive on Spark，为了实现与Hive兼容，Shark在HiveQL方面重用了Hive中HiveQL的解析、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

# 4.4.4 基于内存的分布式计算框架Spark

Shark的出现，使得SQL-on-Hadoop的性能比Hive有了10-100倍的提高



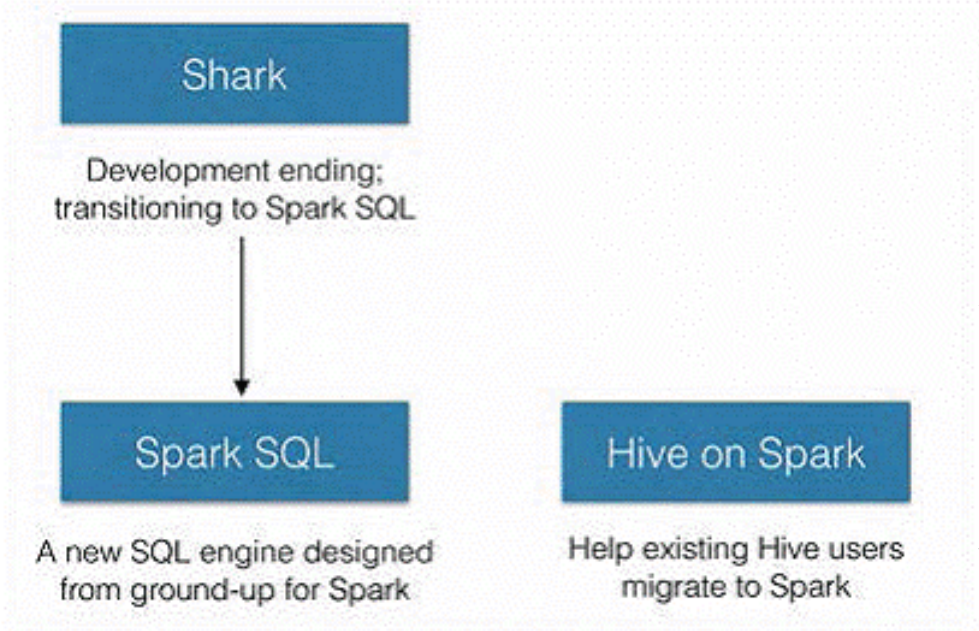
# 4.4.4 基于内存的分布式计算框架Spark

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

# 4.4.4 基于内存的分布式计算框架Spark

2014年6月1日Shark项目和Spark SQL项目的主持人Reynold Xin宣布：停止对Shark的开发，团队将所有资源放在Spark SQL项目上，至此，Shark的发展画上了句号，但也因此发展出两个分支：Spark SQL和Hive on Spark




- Spark SQL作为Spark生态的一员继续发展，而不再受限于Hive，只是兼容Hive
- Hive on Spark是一个Hive的发展计划，该计划将Spark作为Hive的底层引擎之一，也就是说，Hive将不再受限于一个引擎，可以采用Map-Reduce、Tez、Spark等引擎

# 4.4.4 基于内存的分布式计算框架Spark

## （2） Spark SQL架构

Spark SQL在Hive兼容层面仅依赖HiveQL解析、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

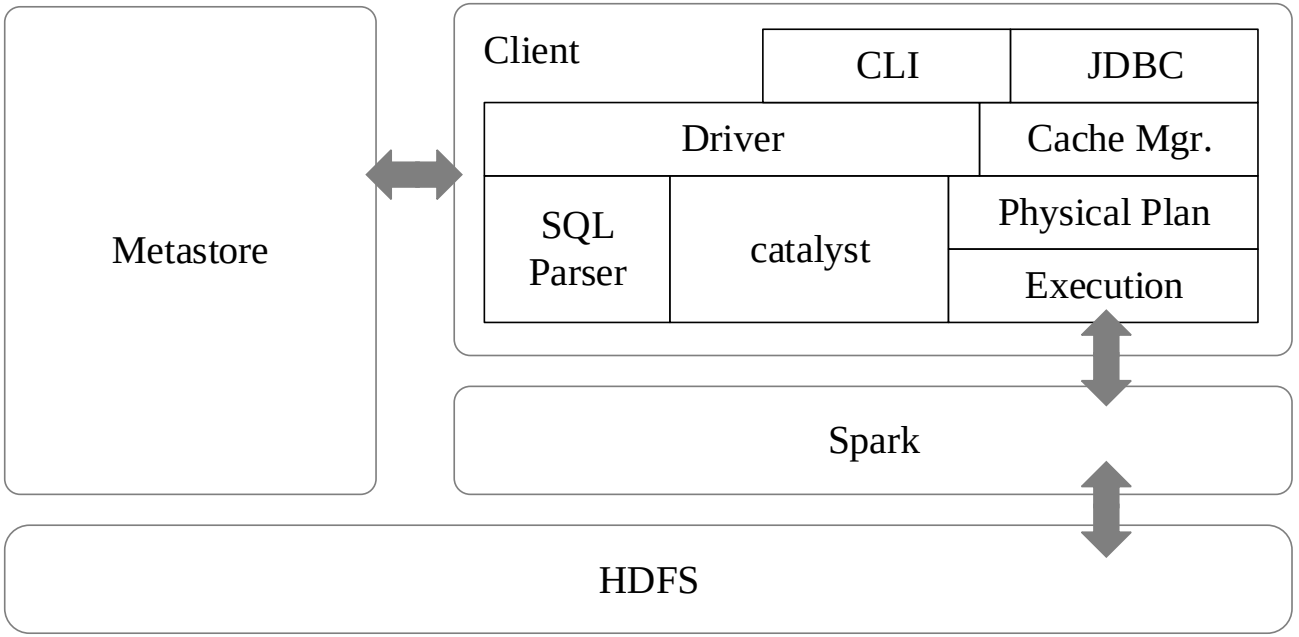



图4-19 Spark SQL架构

# 4.4.4 基于内存的分布式计算框架Spark

•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，也可以是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

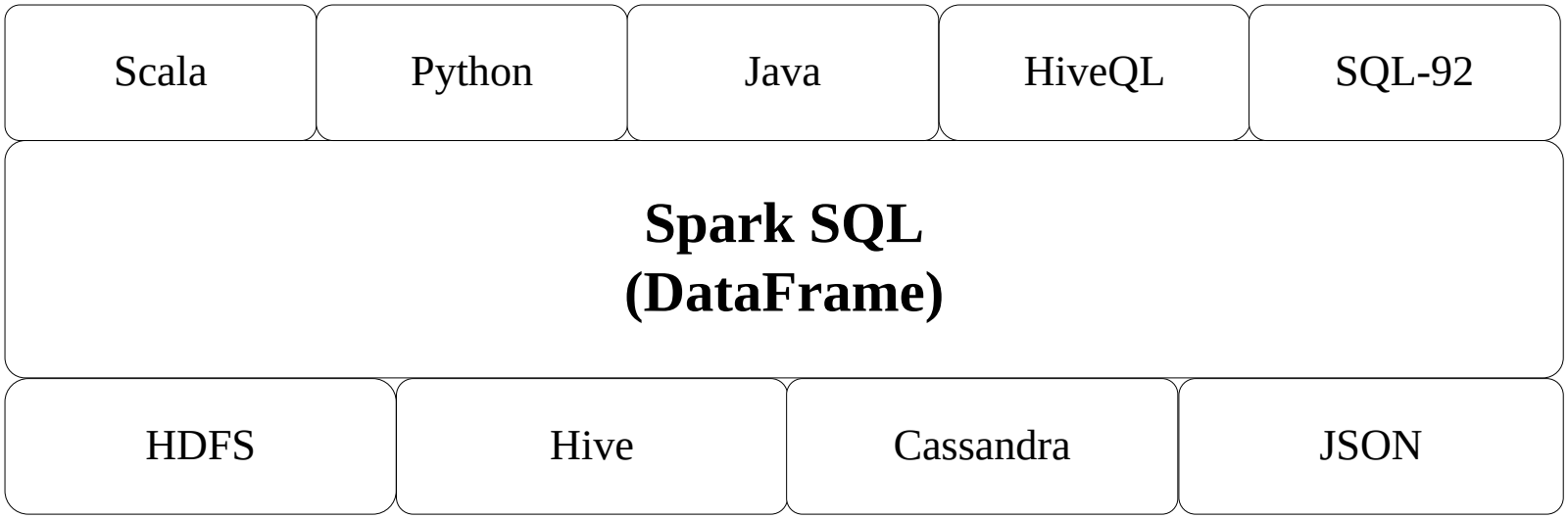



图4-20　Spark SQL支持的数据格式和编程语言

# 4.4.4 基于内存的分布式计算框架Spark

**（3）为什么推出Spark SQL**

**Spark SQL: Relational Data Processing in Spark**

Michael Armbrust[†], Reynold S. Xin[†], Cheng Lian[†], Yin Huai[†], Davies Liu[†], Joseph K. Bradley[†], Xiangrui Meng[†], Tomer Kaftan[‡], Michael J. Franklin[†‡], Ali Ghodsi[†], Matei Zaharia[†*]

[†]Databricks Inc. [*]MIT CSAIL [‡]AMPLab, UC Berkeley

While the popularity of relational systems shows that users often prefer writing declarative queries, the relational approach is insufficient for many big data applications. First, users want to perform ETL to and from various data sources that might be semi- or unstructured, requiring custom code. Second, users want to perform advanced analytics, such as machine learning and graph processing, that are challenging to express in relational systems. In practice, we have observed that most data pipelines would ideally be expressed with a combination of both relational queries and complex procedural algorithms. Unfortunately, these two classes of systems—relational and procedural—have until now remained largely disjoint,

# 4.4.4 基于内存的分布式计算框架Spark

**Spark SQL: Relational Data Processing in Spark**

Michael Armbrust[†], Reynold S. Xin[†], Cheng Lian[†], Yin Huai[†], Davies Liu[†], Joseph K. Bradley[†], Xiangrui Meng[†], Tomer Kaftan[‡], Michael J. Franklin[†‡], Ali Ghodsi[†], Matei Zaharia[†*]

[†]Databricks Inc. [*]MIT CSAIL [‡]AMPLab, UC Berkeley

Spark SQL bridges the gap between the two models through two contributions. First, Spark SQL provides a *DataFrame API* that can perform relational operations on both external data sources and Spark's built-in distributed collections. This API is similar to the widely used data frame concept in R [32], but evaluates operations lazily so that it can perform relational optimizations. Second, to support the wide range of data sources and algorithms in big data, Spark SQL introduces a novel extensible optimizer called *Catalyst*. Catalyst makes it easy to add data sources, optimization rules, and

# 4.4.4 基于内存的分布式计算框架Spark

- 关系数据库已经很流行
- 关系数据库在大数据时代已经不能满足要求
  - 首先，用户需要从不同数据源执行各种操作，包括结构化、半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（比如机器学习或图像处理），但是，缺少这样的系统

Spark SQL填补了这个鸿沟：

- 首先，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型操作
- 其次，可以支持大数据中的大量数据源和数据分析算法

Spark SQL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

# 4.4.4 基于内存的分布式计算框架Spark

## 9. Spark Streaming

•Spark Streaming是构建在Spark Core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。
•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。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，如图所示。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。

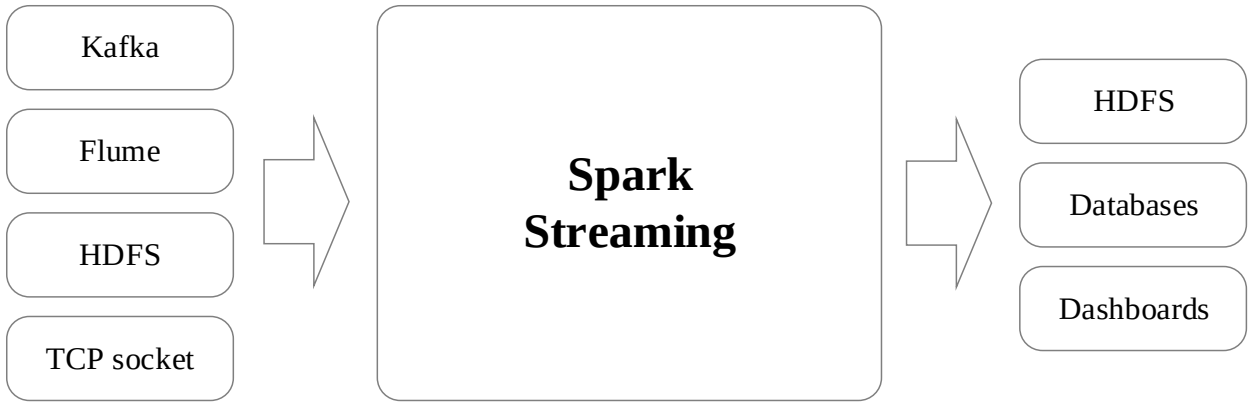



图4-21 Spark Streaming支持的输入、输出数据源

# 4.4.4 基于内存的分布式计算框架Spark

Spark Streaming实际上是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，执行流程如图所示。

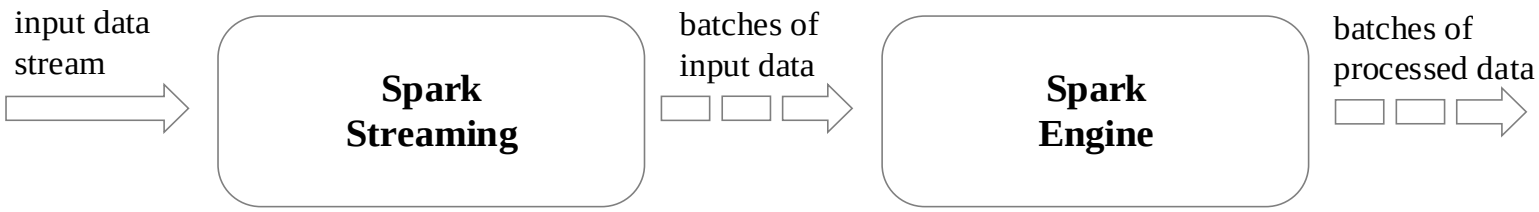



图4-22　Spark Streaming执行流程

# 4.4.4 基于内存的分布式计算框架Spark

下图展示了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）跑在一个Executor上，每个Receiver都会负责一个DStream输入流（比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，或者也可以写入到HDFS、HBase中。

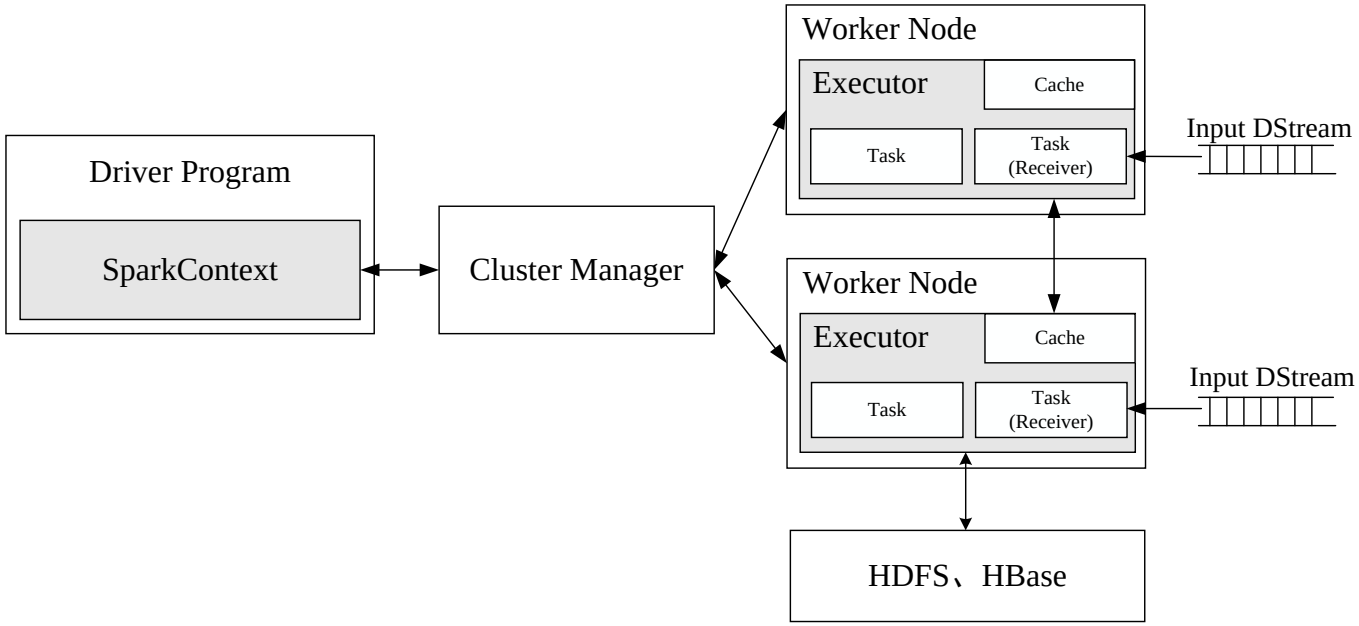



图4-23 Spark Streaming工作机制

# 4.4.4 基于内存的分布式计算框架Spark

## 10.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使得使用者可以像写批处理程序一样编写流处理程序，简化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地跟踪，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版本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。

# 4.4.4 基于内存的分布式计算框架Spark

**（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

•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

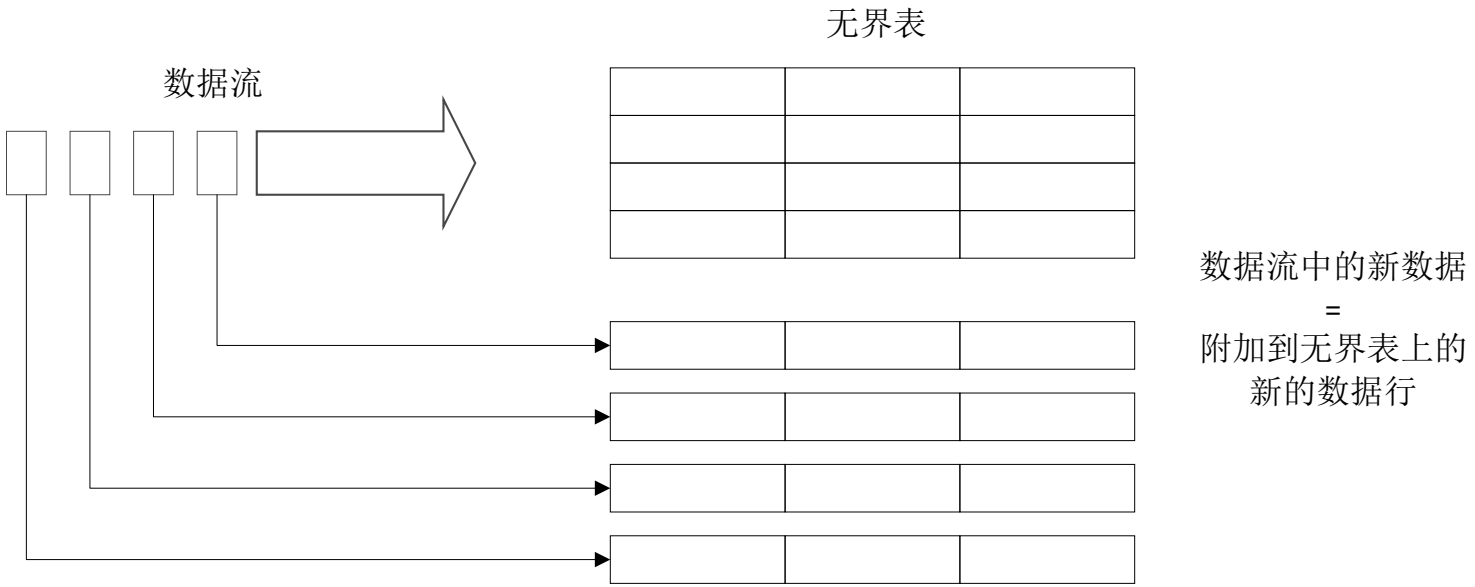



图4-24 无界表

# 4.4.4 基于内存的分布式计算框架Spark

•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

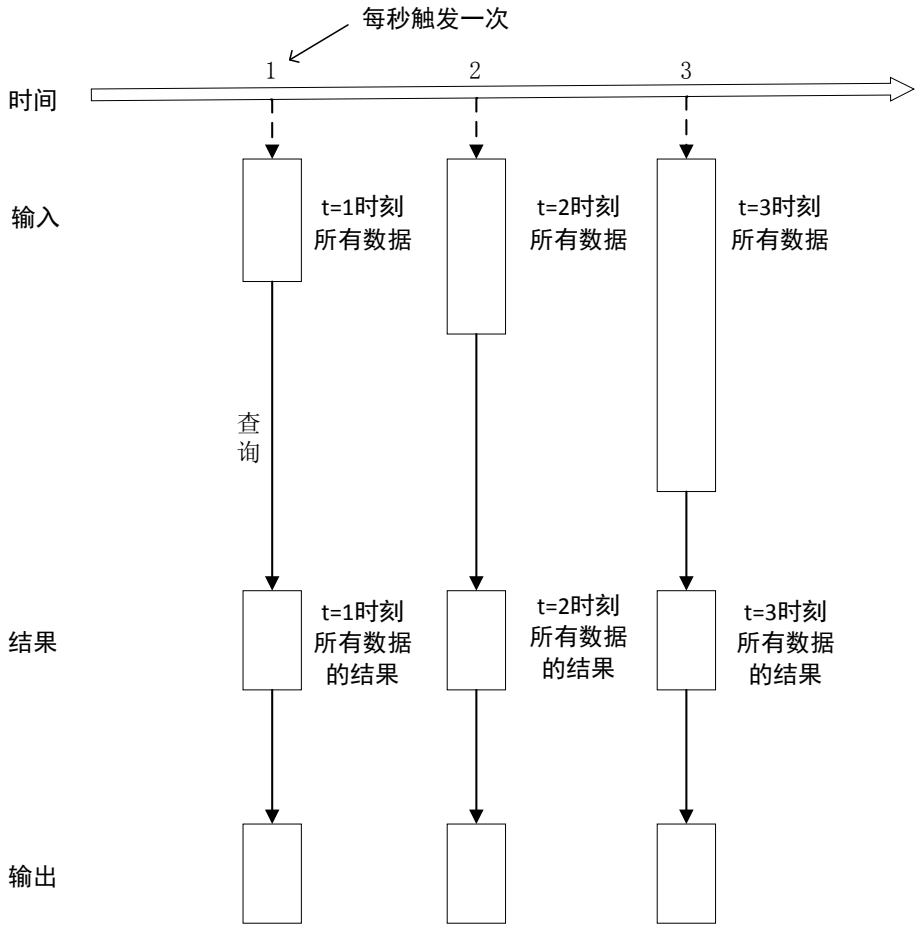



图4-25 Structured Streaming编程模型

# 4.4.4 基于内存的分布式计算框架Spark

**（3）Structured Streaming的两种处理模型**

**（a）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，并对自上一批次结束后到达的新数据执行批量查询
- 数据到达和得到处理并输出结果之间的延时超过100毫秒

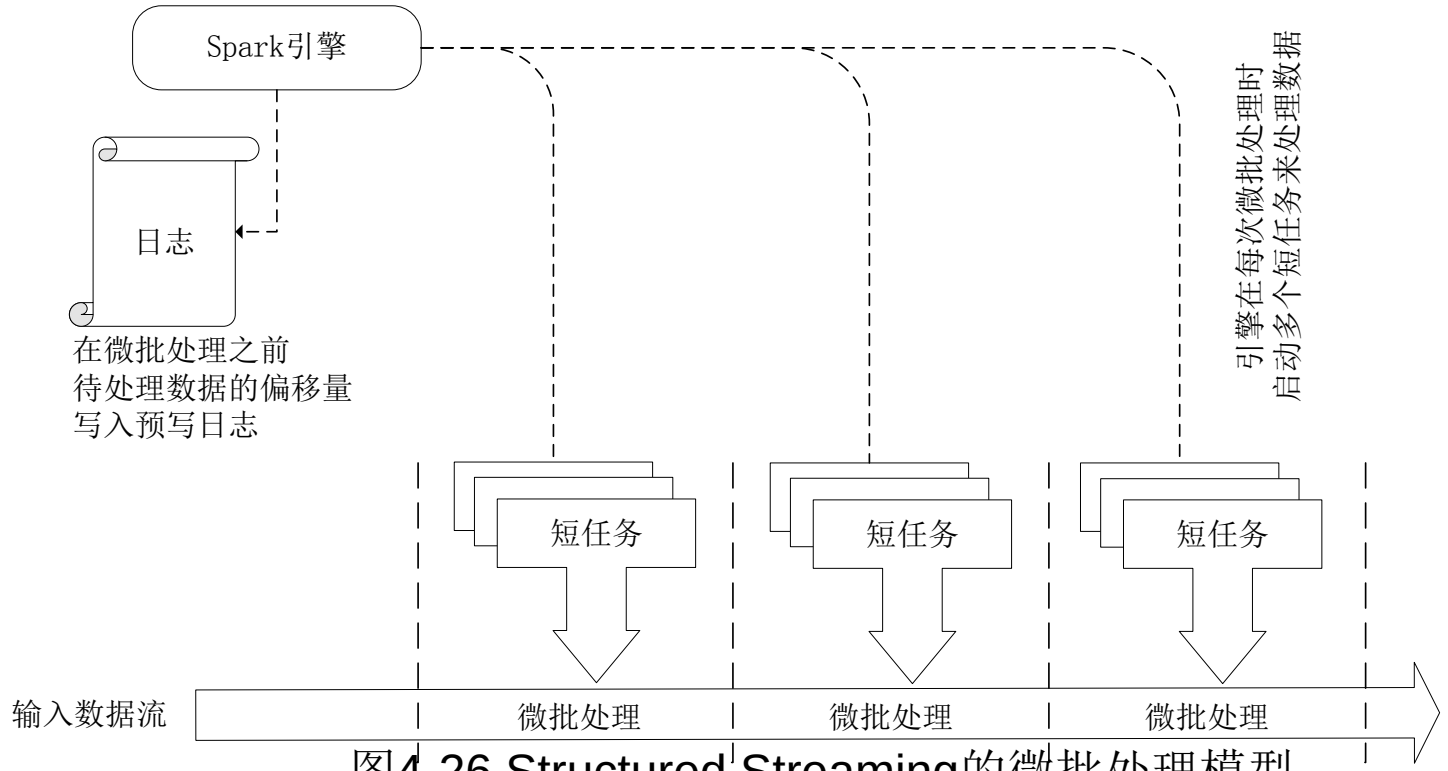



图4-26 Structured Streaming的微批处理模型

# 4.4.4 基于内存的分布式计算框架Spark

**（2）持续处理**

•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，而是启动一系列的连续读取、处理和写入结果的长时间运行的任务

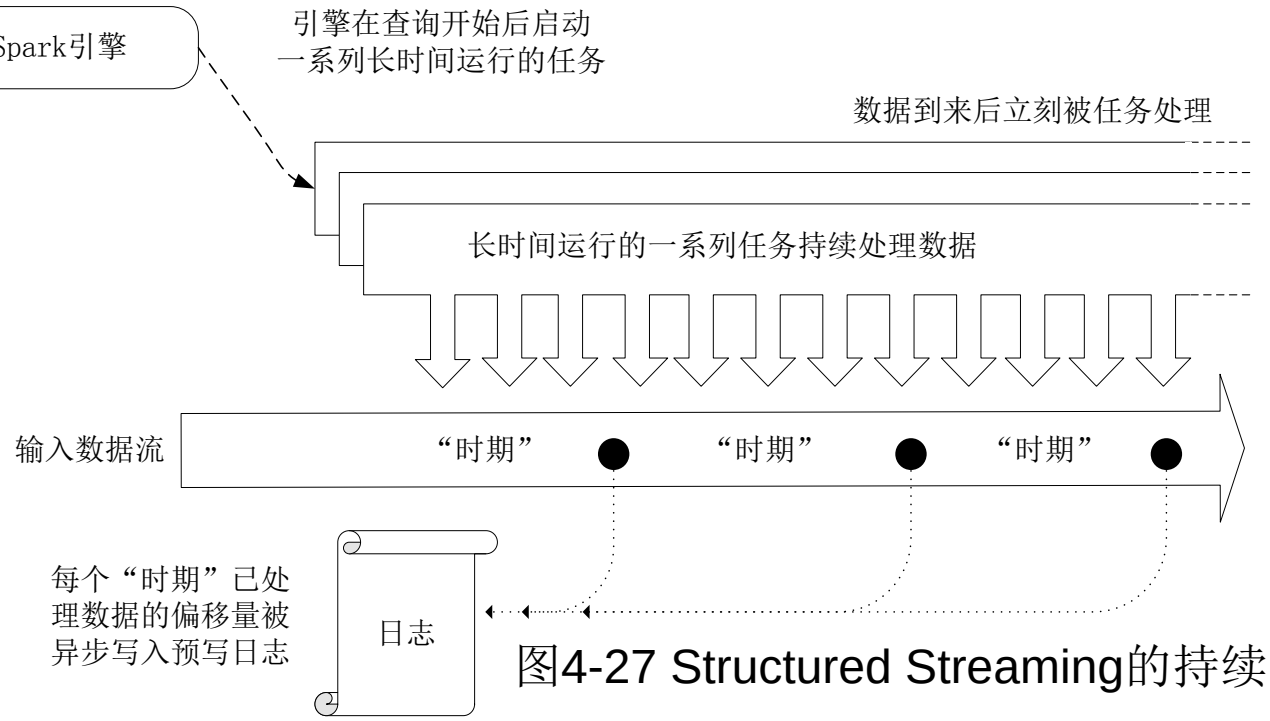



图4-27 Structured Streaming的持续处理模型

# 4.4.4 基于内存的分布式计算框架Spark

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

•Structured Streaming处理的数据跟Spark Streaming一样，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也是采用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。

# 4.4.4 基于内存的分布式计算框架Spark

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

•Structured Streaming可以对DataFrame/Dataset应用前面章节提到的各种操作，包括select、where、groupBy、map、filter、flatMap等。

•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100毫秒级别的实时响应，采用持续处理模型时可以支持毫秒级的实时响应。

# 4.4.4 基于内存的分布式计算框架Spark

**11.Spark MLlib**

- 传统的机器学习算法，由于技术和单机存储的限制，只能在少量数据上使用，依赖于数据抽样
- 大数据技术的出现，可以支持在全量数据上进行机器学习
- 机器学习算法涉及大量迭代计算
- 基于磁盘的MapReduce不适合进行大量迭代计算
- 基于内存的Spark比较适合进行大量迭代计算

# 4.4.4 基于内存的分布式计算框架Spark

**11.Spark MLlib**

•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

•开发者只需要有 Spark 基础并且了解机器学习算法的原理，以及方法相关参数的含义，就可以轻松的通过调用相应的 API 来实现基于海量数据的机器学习过程

•pyspark的**即席查询**也是一个关键。算法工程师可以边写代码边运行，边看结果

# 4.4.4 基于内存的分布式计算框架Spark

**11.Spark MLlib**

•MLlib是Spark的机器学习（Machine Learning）库，旨在简化机器学习的工程实践工作

•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流水线（Pipeline）API，具体如下：

•**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**：特征提取、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。

# 4.4.4 基于内存的分布式计算框架Spark

**11.Spark MLlib**

MLlib目前支持4种常见的机器学习问题: 分类、回归、聚类和协同过滤

| | 离散数据 | 连续数据 |
| --- | --- | --- |
| 监督学习 | Classification、LogisticRegression(with Elastic-Net)、SVM、DecisionTree、RandomForest、GBT、NaiveBayes、MultilayerPerceptron、OneVsRest | Regression、LinearRegression(with Elastic-Net)、DecisionTree、RandomFores、GBT、AFTSurvivalRegression、IsotonicRegression |
| 无监督学习 | Clustering、KMeans、GaussianMixture、LDA、PowerIterationClustering、BisectingKMeans | Dimensionality Reduction, matrix factorization、PCA、SVD、ALS、WLS |

# 4.4.5 TensorFlowOnSpark

- TensorFlow是一个开源的、基于Python的机器学习框架，它是由谷歌公司开发的，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、甚至移动设备等等。

# 4.4.5 TensorFlowOnSpark

•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加入到现有的环境中（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项目是由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中的Hadoop集群，主要进行大规模分布式深度学习。

# 4.4.5 TensorFlowOnSpark

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TFOnSpark程序中能够与SparkSQL、MLlib和其他 Spark 库一起工作处理数据（如图所示）。

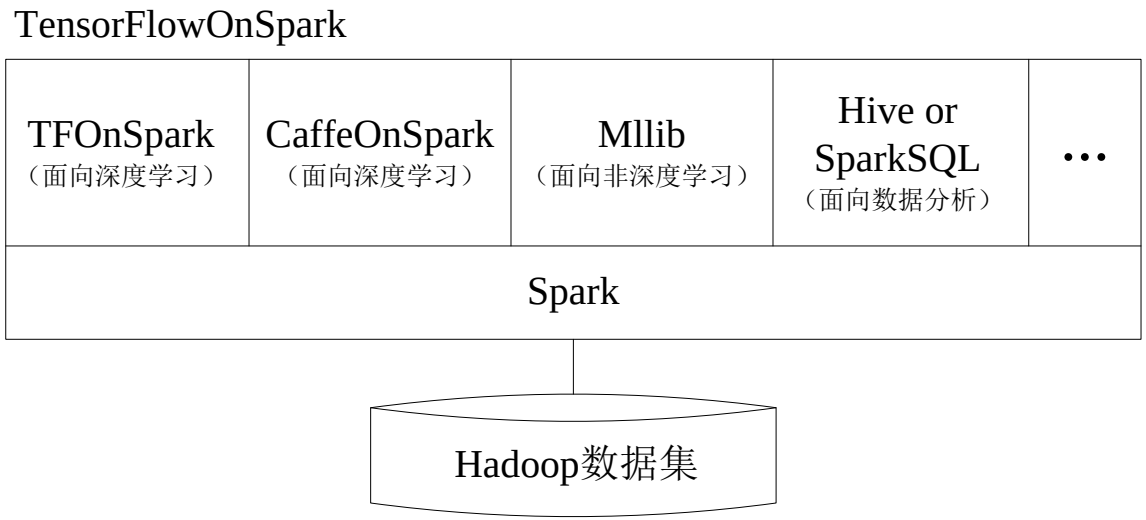



图4-28 TensorFlowOnSpark与Spark的集成

# 4.4.5 TensorFlowOnSpark

TensorFlowOnSpark的体系架构较为简单（如图所示），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集群运行在了Spark上，它会在每个Spark Executor中启动TensorFlow应用程序，然后通过gRPC或RDMA方式进行数据传递与交互。

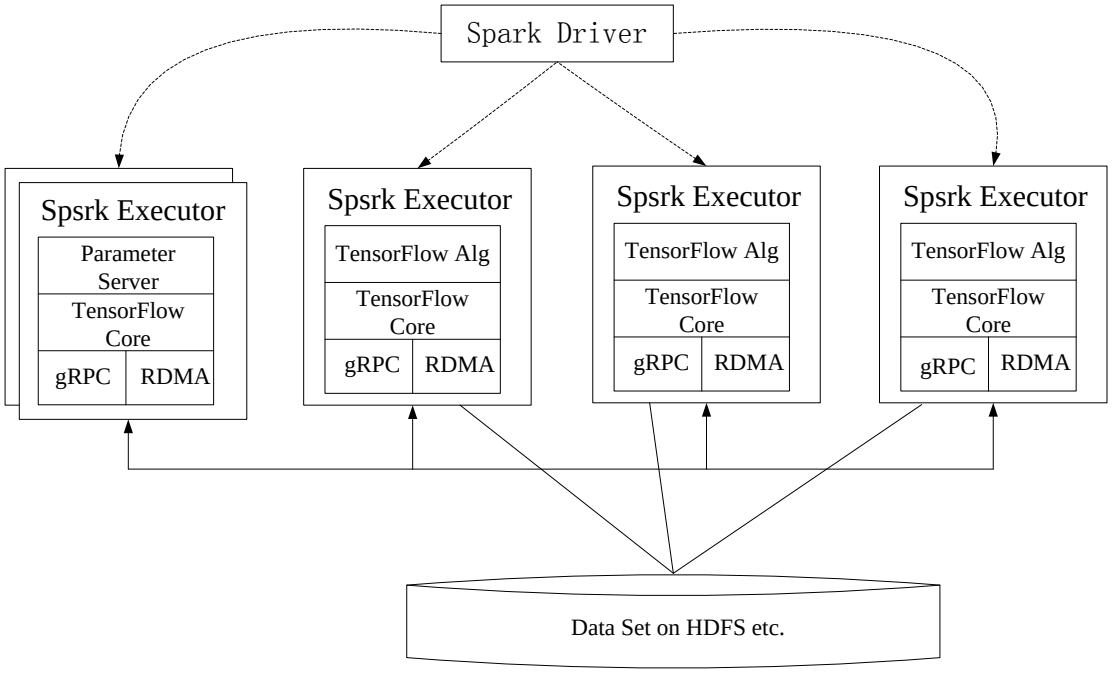



图4-29 TensorFlowOnSpark体系架构

# 4.4.5 TensorFlowOnSpark

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应用程序；
（3）训练/推理：在TensorFlow集群上完成模型的训练或推理；
（4）关闭：关闭Executor进程上的TensorFlow应用程序，释放相应的系统资源(消息队列)。

# 4.4.6 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，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

# 4.4.6 流计算框架Storm

- Twitter是全球访问量最大的社交网站之一，Twitter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

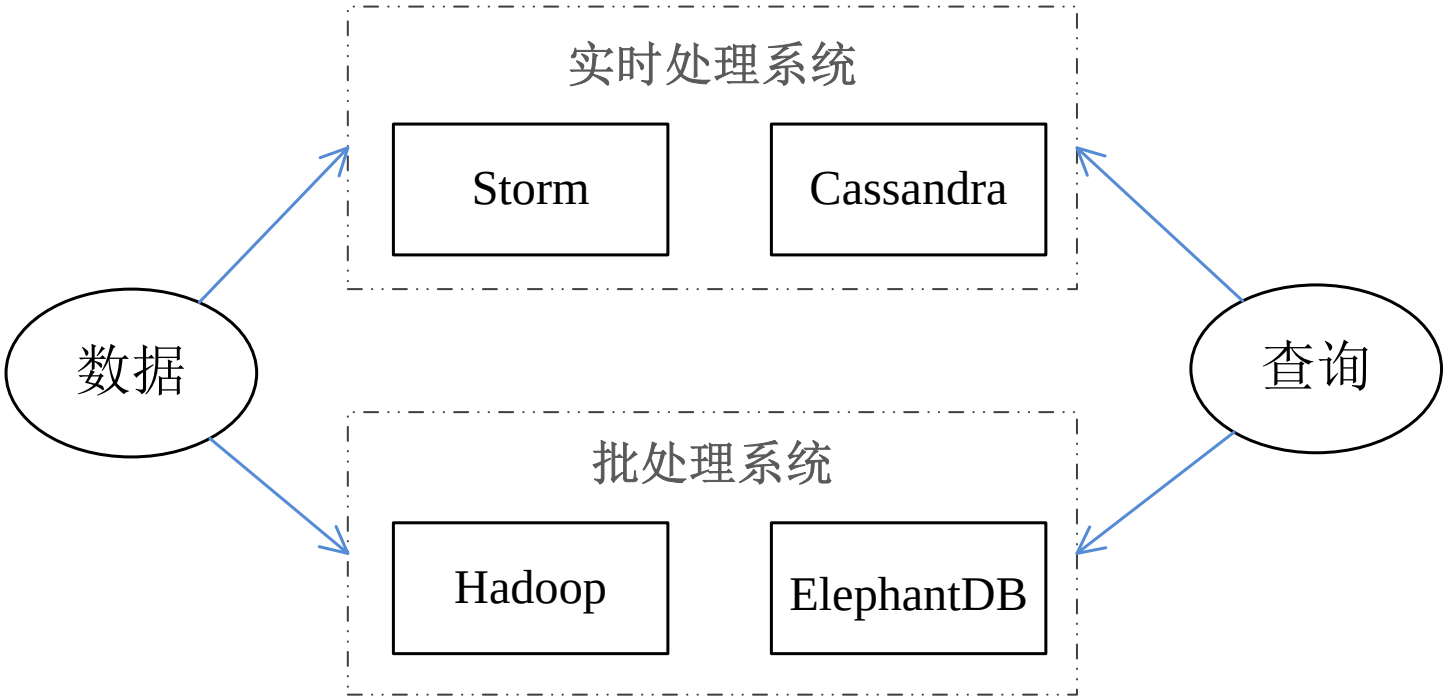



图4-30 Twitter的分层数据处理架构

# 4.4.6 流计算框架Storm

**2. Storm的特点**

- Storm可用于许多领域中，如实时分析、在线机器学习、持续计算、远程RPC、数据提取加载转换等
- Storm具有以下主要特点：
  - 整合性：Storm可方便地与队列系统和数据库系统进行整合
  - 简易的API：Storm的API在使用上即简单又方便
  - 可扩展性：Storm的并行特性使其可以运行在分布式集群中
  - 容错性：Storm可自动进行故障节点的重启、任务的重新分配
  - 可靠的消息处理：Storm保证每个消息都能完整处理
  - 支持各种编程语言：Storm支持使用各种编程语言来定义任务
  - 快速部署：Storm可以快速进行部署和使用
  - 免费、开源：Storm是一款开源框架，可以免费使用

# 4.4.6 流计算框架Storm

**3.Storm的框架设计**

•Storm运行任务的方式与Hadoop类似：Hadoop运行的是MapReduce作业，而Storm运行的是“Topology”

•但两者的任务大不相同，主要的不同是：MapReduce作业最终会完成计算并结束运行，而Topology将持续处理消息（直到人为终止）

Storm和Hadoop架构组件功能对应关系

| | Hadoop | Storm |
|---|---|---|
| 应用名称 | Job | Topology |
| 系统角色 | JobTracker | Nimbus |
| | TaskTracker | Supervisor |
| 组件接口 | Map/Reduce | Spout/Bolt |

# 4.4.6 流计算框架Storm

- Storm集群采用“Master—Worker”的节点方式：
  - 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
  - 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，一个Worker节点上同时运行若干个Worker进程

# 4.4.6 流计算框架Storm

- Storm使用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。借助于Zookeeper，若Nimbus进程或Supervisor进程意外终止，重启时也能读取、恢复之前的状态并继续工作，使得Storm极其稳定

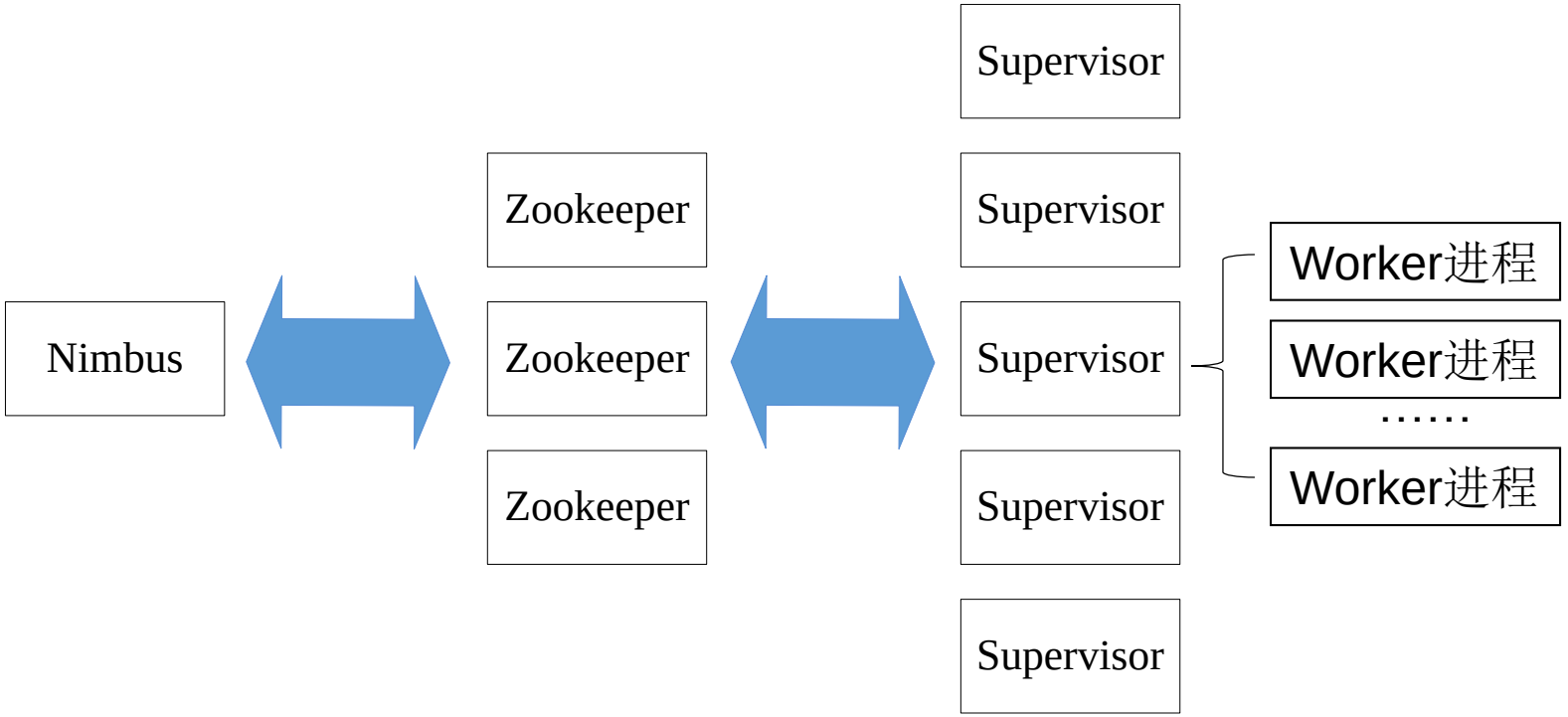



图4-31 Storm集群架构示意图

# 4.4.6 流计算框架Storm

- 基于这样的架构设计，Storm的工作流程如下图所示：

•所有Topology任务的提交必须在Storm客户端节点上进行，提交后，由Nimbus节点分配给其他Supervisor节点进行处理

•Nimbus节点首先将提交的Topology进行分片，分成一个个Task，分配给相应的Supervisor，并将Task和Supervisor相关的信息提交到Zookeeper集群上

•Supervisor会去Zookeeper集群上认领自己的Task，通知自己的Worker进程进行Task的处理

•说明：在提交了一个Topology之后，Storm就会创建Spout/Bolt实例并进行序列化。之后，将序列化的组件发送给所有的任务所在的机器(即Supervisor节点)，在每一个任务上反序列化组件

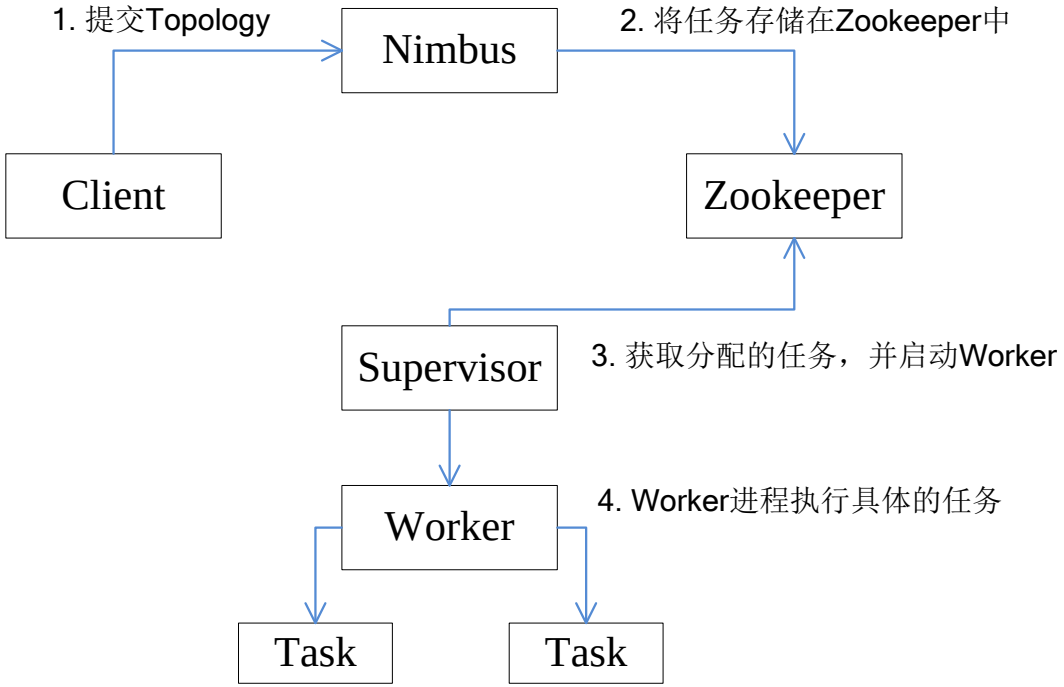



图4-32 Storm工作流程示意图

# 4.4.6 流计算框架Storm

**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

•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

# 4.4.6 流计算框架Storm

- 从编程的灵活性来讲，Storm是比较理想的选择，它使用Apache Thrift，可以用任何编程语言来编写拓扑结构（Topology）
- 当需要在一个集群中把流计算和图计算、机器学习、SQL查询分析等进行结合时，可以选择Spark Streaming，因为，在Spark上可以统一部署Spark SQL，Spark Streaming、MLlib，GraphX等组件，提供便捷的一体化编程模型
- 当应用场景需要毫秒级响应时，可以选择Storm，因为Spark Streaming无法实现毫秒级的流计算

# 4.4.7 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，设计思想主要来源于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以及一些环形的迭代任务。

# 4.4.7 流计算框架Flink

Flink以层级式系统形式组建其软件栈（如图所示），不同层的栈建立在其下层基础上。具体而言，Flink的典型特性如下：

•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；

•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）或者YARN；

•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）；

•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。

# 4.4.7 流计算框架Flink

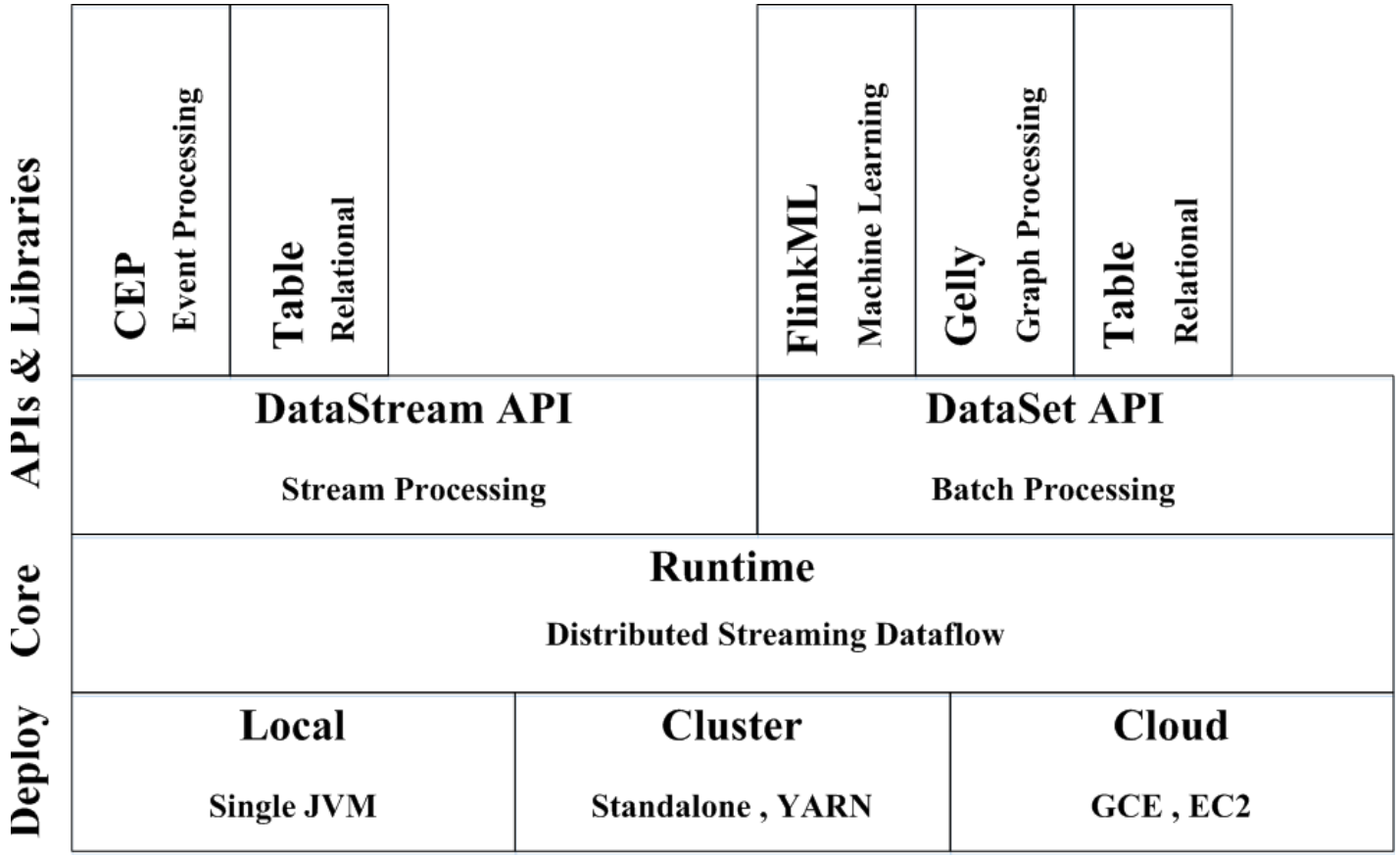



图4-33 Flink架构图

# 4.4.7 流计算框架Flink

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。当全部运行在Hadoop YARN之上时，Flink的性能甚至还要略好于Spark，因为，Flink支持增量迭代，具有对迭代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，所以，Spark Streaming在流式处理方面，不可避免地会增加一些延时，实时性没有Flink好。Flink的流计算性能和Storm差不多，可以支持毫秒级的响应，而Spark Streaming则只能支持秒级响应。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。

# 4.4.7 流计算框架Flink

**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通过采用微批处理方法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采用持续处理模型时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就成为了能够满足流处理架构要求的理想的流计算框架。

# 4.4.7 流计算框架Flink

### 3. Flink体系架构

如图所示，Flink系统主要由两个组件组成，分别为JobManager和TaskManager，Flink 架构也遵循Master-Slave架构设计原则，JobManager为Master节点，TaskManager为Slave节点。

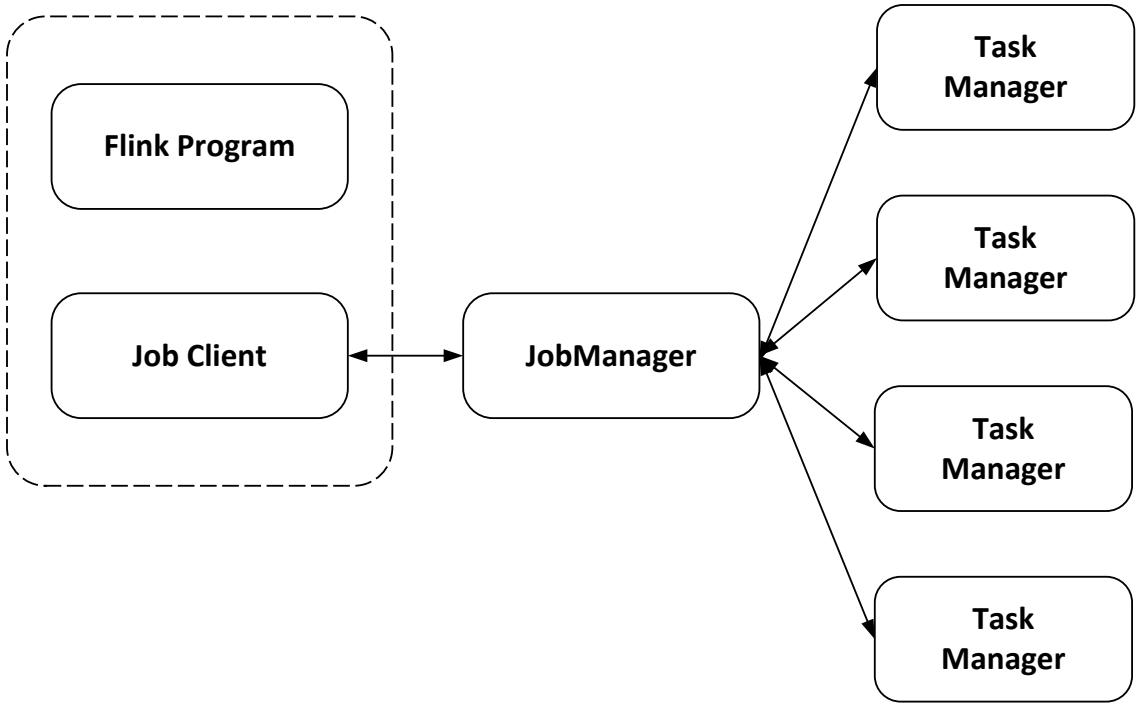



图4-34 Flink体系架构

# 4.4.8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（比如MapReduce、Spark、Flink、Storm、Apex等），为大数据开发者提供了丰富的工具的同时，也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，并重写所有的业务逻辑。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。

# 4.4.8大数据编程框架Beam

Beam是由谷歌贡献的Apache顶级项目，它的目标是为开发者提供一个易于使用、却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接口，即提供一个统一的编程接口给到上层应用的开发者，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，直接通过Beam SDK的接口，就可以开发数据处理的加工流程，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。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。

# 4.4.8大数据编程框架Beam

如图所示，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（比如Apex、Spark、Flink、Cloud Dataflow等）。

# 4.4.8大数据编程框架Beam

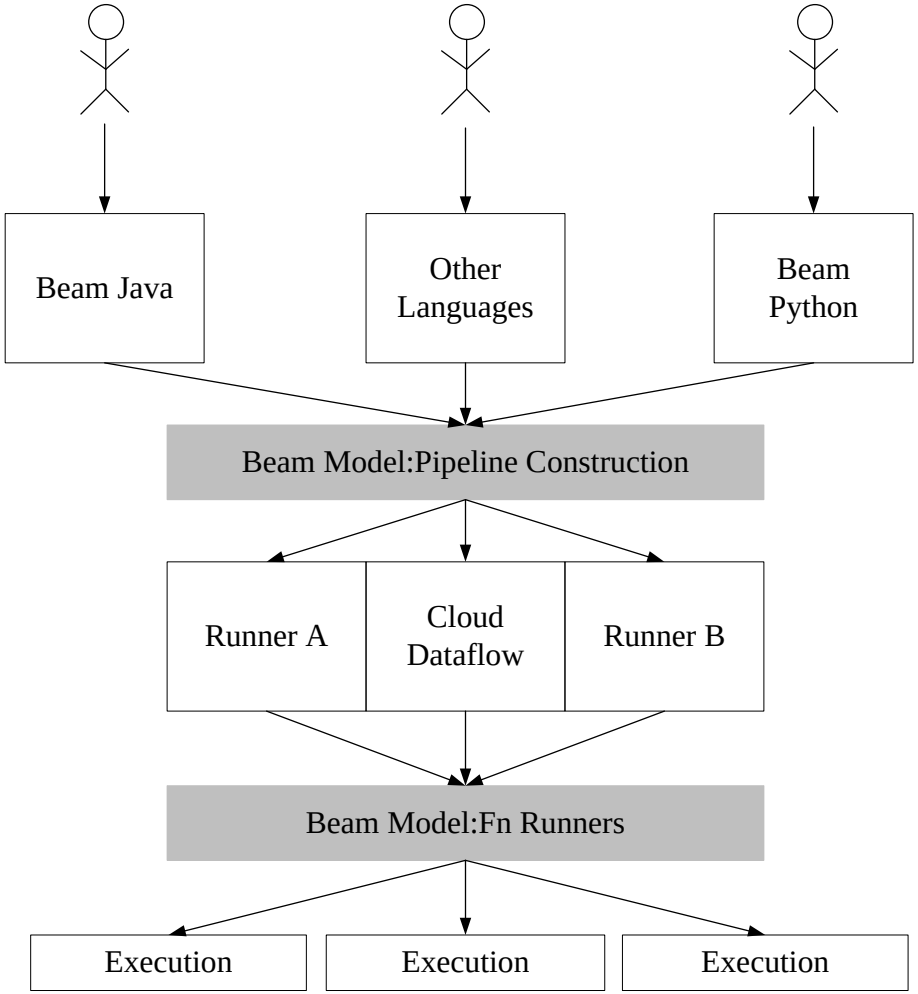



图4-35 Beam使用一套高层抽象的API屏蔽多种计算引擎的区别

# 4.4.9查询分析系统Dremel

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，满足谷歌公司上万用户操作PB级的数据，可以在2到3秒内完成PB级别数据的查询。

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。
（2）Dremel是MapReduce交互式查询能力不足的补充。
（3）Dremel的数据模型是嵌套的。
（4）Dremel中的数据是用列式存储的。
（5） Dremel结合了Web搜索和并行DBMS（Database Management System）的技术。

# 4.5 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，此时，虽然它依然采用已经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人员的工作负担，只需要简单的编程，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。

本章内容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。