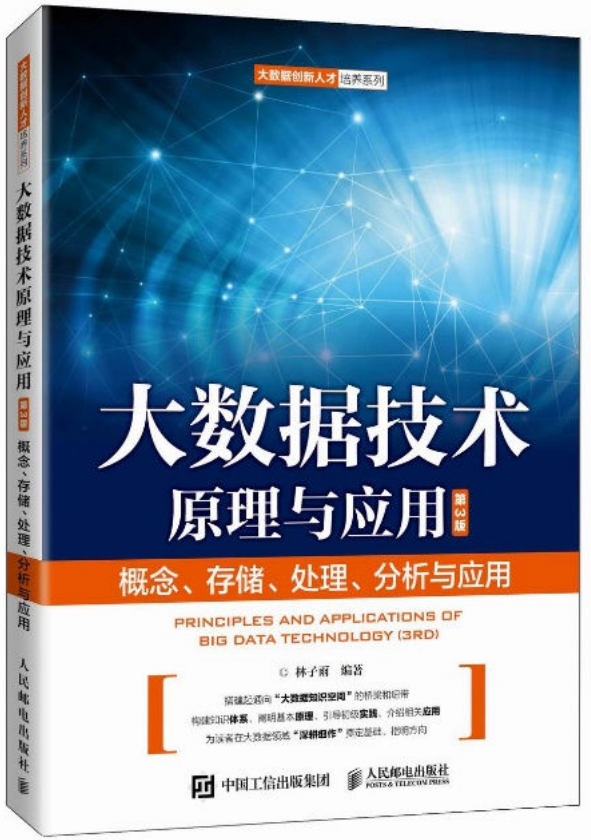
大数据技术原理与应用

9. 数据仓库 Hive

陈建文
电子信息与通信学院
chenjw@hust.edu.cn

9. 数据仓库Hive



9.1 概述

- 9.1.1 数据仓库概念
- 9.1.2 Hive简介
- 9.1.3 Hive与Hadoop生态系统中其他组件的关系
- 9.1.4 Hive与传统数据库的对比分析
- 9.1.5 Hive在企业中的部署和应用

9.1.1 数据仓库概念

数据仓库（**Data Warehouse**）是一个面向主题的（**Subject Oriented**）、集成的（**Integrated**）、相对稳定的（**Non-Volatile**）、反映历史变化（**Time Variant**）的数据集合，用于支持管理决策。

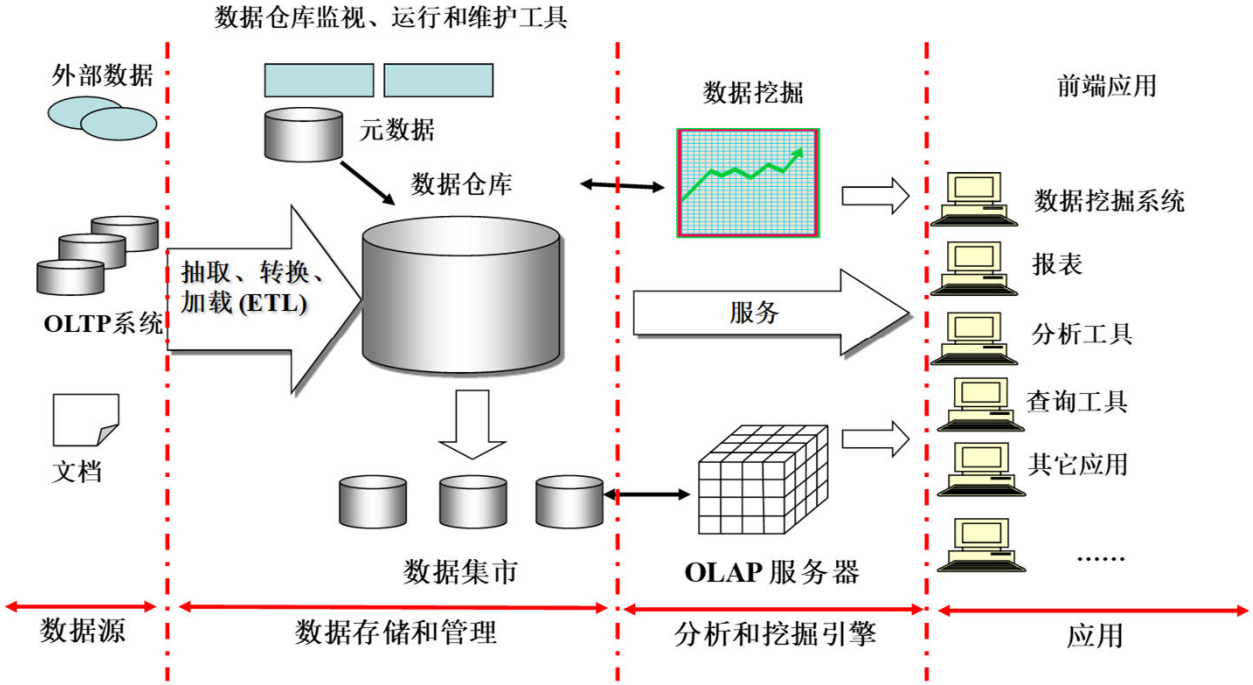


图9-1 数据仓库的体系结构

9.1.2 Hive简介

- Hive是一个构建于Hadoop顶层的数据仓库工具
- 支持大规模数据存储、分析，具有良好的可扩展性
- 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
- 定义了简单的类似SQL 的查询语言-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分析工具

Hive具有的特点非常适用于数据仓库:

- 采用批处理方式处理海量数据
  - Hive需要把HiveQL语句转换成MapReduce任务进行运行
  - 数据仓库存储的是静态数据，对静态数据的分析适合采用批处理方式，不需要快速响应给出结果，而且数据本身也不会频繁变化
- 提供适合数据仓库操作的工具
  - Hive本身提供了一系列对数据进行提取、转换、加载（ETL）的工具，可以存储、查询和分析存储在Hadoop中的大规模数据
  - 非常适合数据仓库应用程序维护海量数据、对数据进行挖掘、形成意见和报告等

9.1.3 Hive与Hadoop生态系统中其他组件的关系

Hadoop生态系统中Hive与其他部分的关系：

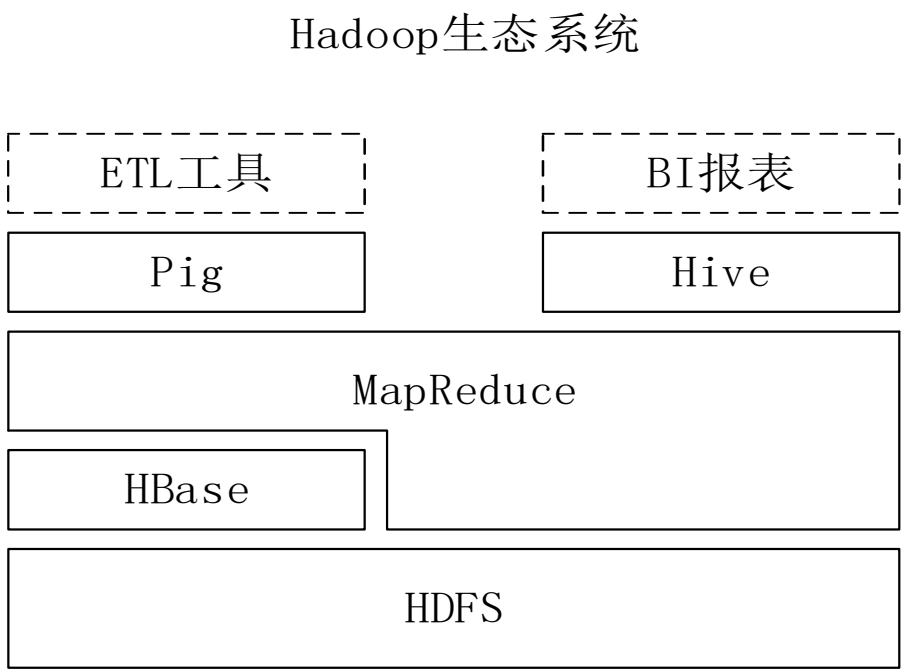


图9-2 Hive 在Hadoop生态系统的位置

■ 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。

9.1.4 Hive与传统数据库的对比分析

- Hive在很多方面和传统的关系数据库类似，但是它的底层依赖的是HDFS和MapReduce，所以在很多方面又有别于传统数据库

表9-1 Hive与传统数据库的对比

对比项目	Hive	传统数据库
数据插入	支持批量导入	支持单条和批量导入
数据更新	不支持	支持
索引	支持	支持
分区	支持	支持
执行延迟	高	低
扩展性	好	有限

Hive与传统数据库的区别主要体现在以下几个方面：

（1）**数据插入**：在传统数据库中同时支持导入单条数据和批量数据，而Hive中仅支持批量导入数据，因为Hive主要用来支持大规模数据集上的数据仓库应用程序的运行，常见操作是全表扫描，所以单条插入功能对Hive并不实用

（2）**数据更新**：更新是传统数据库中很重要的特性，Hive不支持数据更新。Hive是一个数据仓库工具，而数据仓库中存放的是静态数据，所以Hive不支持对数据进行更新。

（3）**索引**：索引也是传统数据库中很重要的特性，Hive在hive 0.7版本后已经可以支持索引了。但Hive不像传统的关系型数据库那样有键的概念，它只提供有限的索引功能，使用户可以在某些列上创建索引来加速一些查询操作，Hive中给一个表创建的索引数据被保存在另外的表中。

（4）**分区**：传统的数据库提供分区功能来改善大型表以及具有各种访问模式的表的可伸缩性，可管理性和提高数据库效率。Hive也支持分区功能，Hive表组织成分区的形式，根据分区列的值对表进行粗略的划分，使用分区可以加快数据的查询速度。

（5）**执行延迟**：因为Hive构建于HDFS与MapReduce上，所以对比传统数据库来说Hive的延迟比较高，传统的SQL语句的延迟少于一秒，而HiveQL语句的延迟会达到分钟级。

（6）**扩展性**：传统关系数据库很难横向扩展，纵向扩展的空间也很有限。相反Hive的开发环境是基于集群的，所以具有较好的可扩展性。

9.1.5 Hive在企业中的部署和应用

1. Hive在企业大数据分析平台中的应用

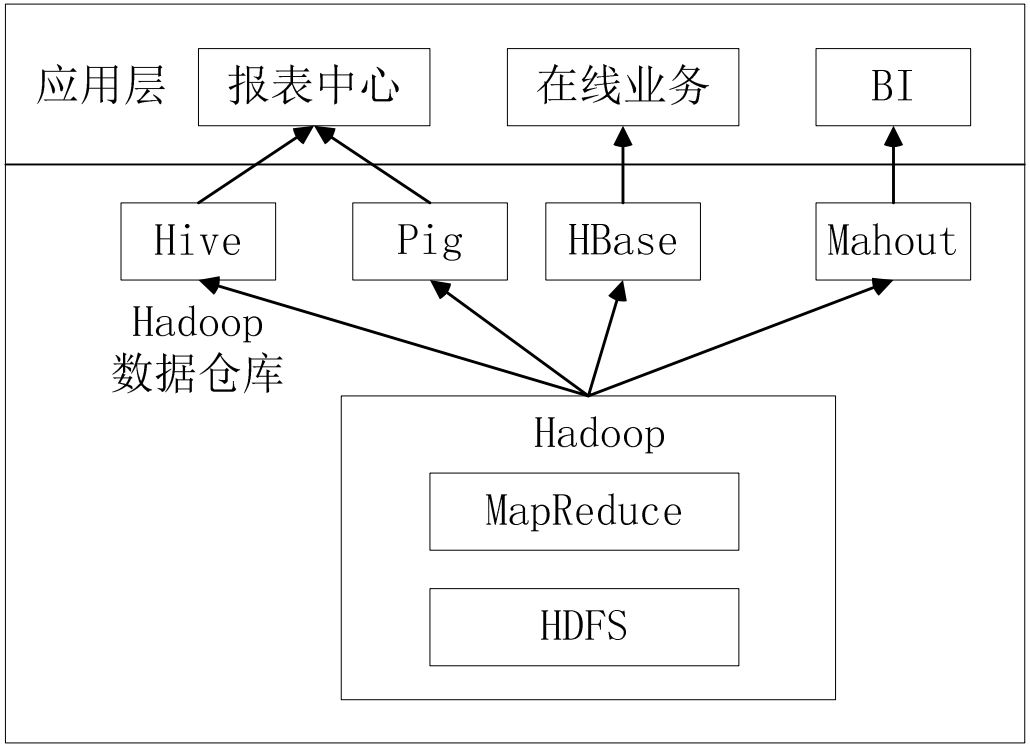


图 9-3 企业中一种常见的大数据分析平台部署框架

2.Hive在Facebook公司中的应用

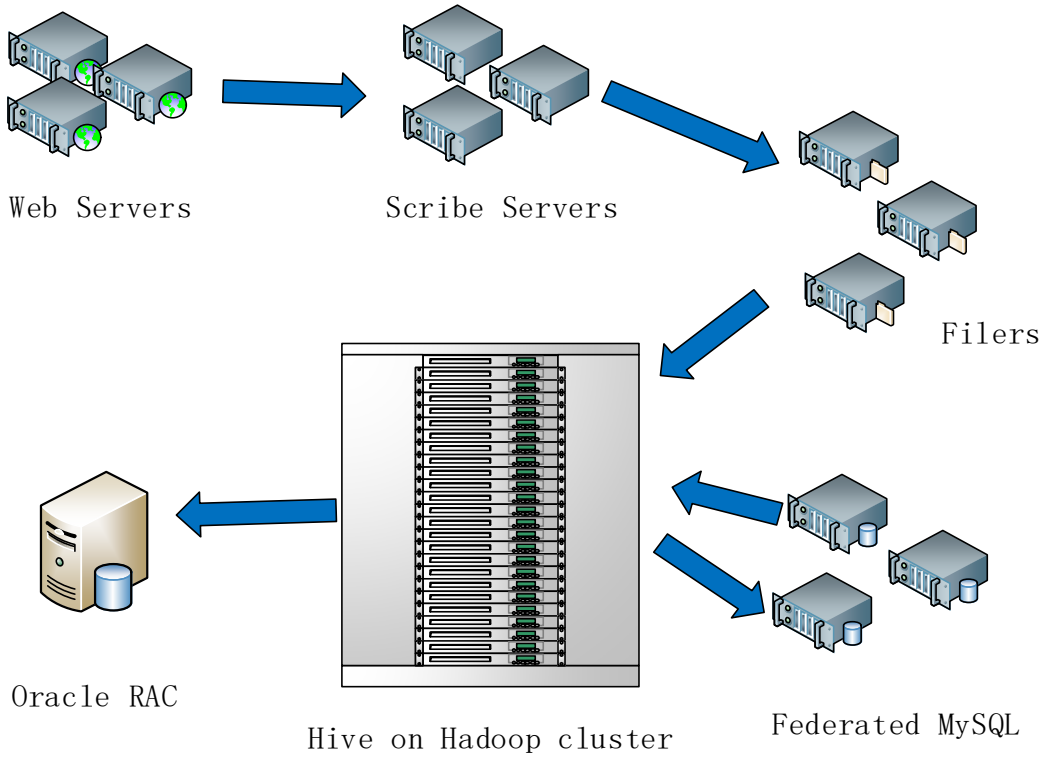


图9-4 Facebook的数据仓库架构

- 随着Facebook网站使用量的增加， Facebook公司开始把数据仓库构建在Hadoop集群上，此时的数据处理过程描述如下：

  - Web服务器及内部服务（如搜索后台）产生日志数据
  - Scribe服务器把几百个甚至上千个日志数据集存放在几个甚至几十个Filers（网络文件服务器）上
  - 网络文件服务器上的大部分日志文件被复制存放在HDFS系统中。并且维度数据也会每天从内部的MySQL数据库上复制到这个HDFS系统中
  - Hive为HDFS收集所有数据创建一个数据仓库，用户可以通过编写HiveQL语言创建各种概要信息和报表以及数据执行的历史分析，同时内部的MySQL数据库也可以从中获取处理后的数据。
  - 把需要实时联机访问的数据存放在Oracle RAC上。

9.2 Hive系统架构

- 9.2.1 Hive系统架构
- 9.2.2 Hive组成模块
- 9.2.3 Hive工作原理
- 9.2.4 从外部访问Hive的典型方式

9.2.1 Hive系统架构

Hive系统架构如图9-5所示：

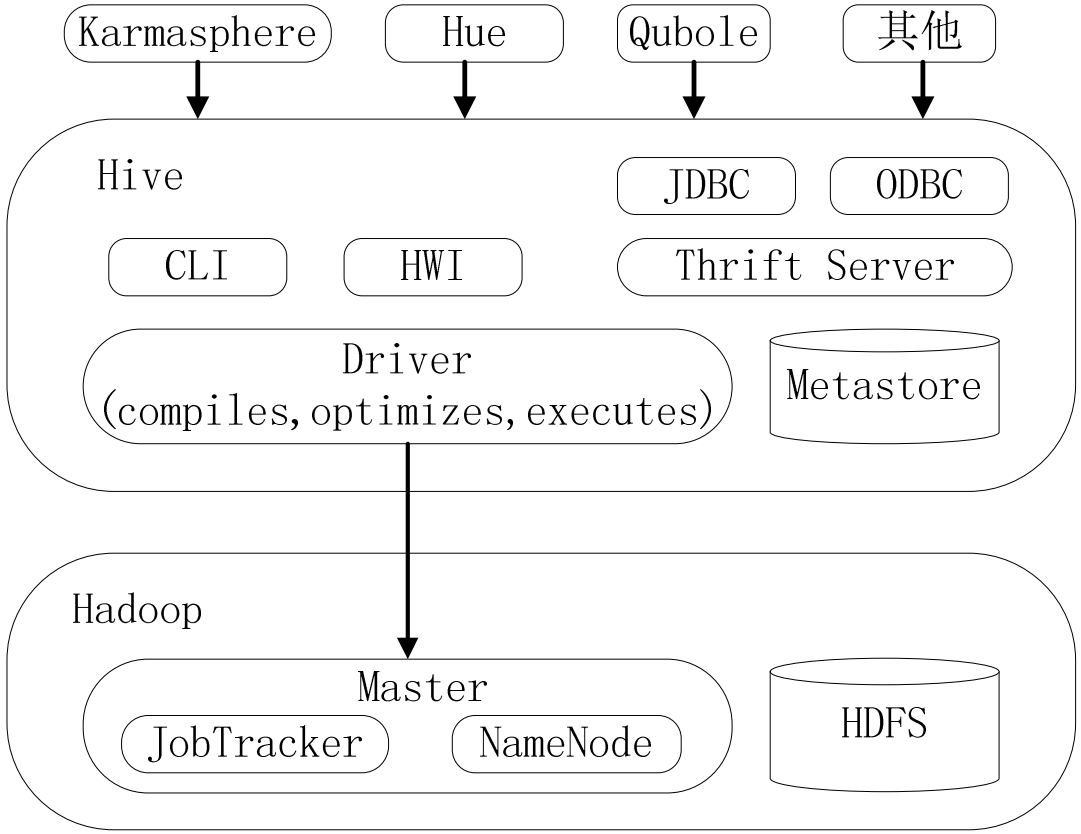


图9-5 Hive系统架构

9.2.2 Hive组成模块

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

  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，负责把HiveSQL语句转换成一系列MapReduce作业**

  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（自带derby数据库，或MySQL数据库）**

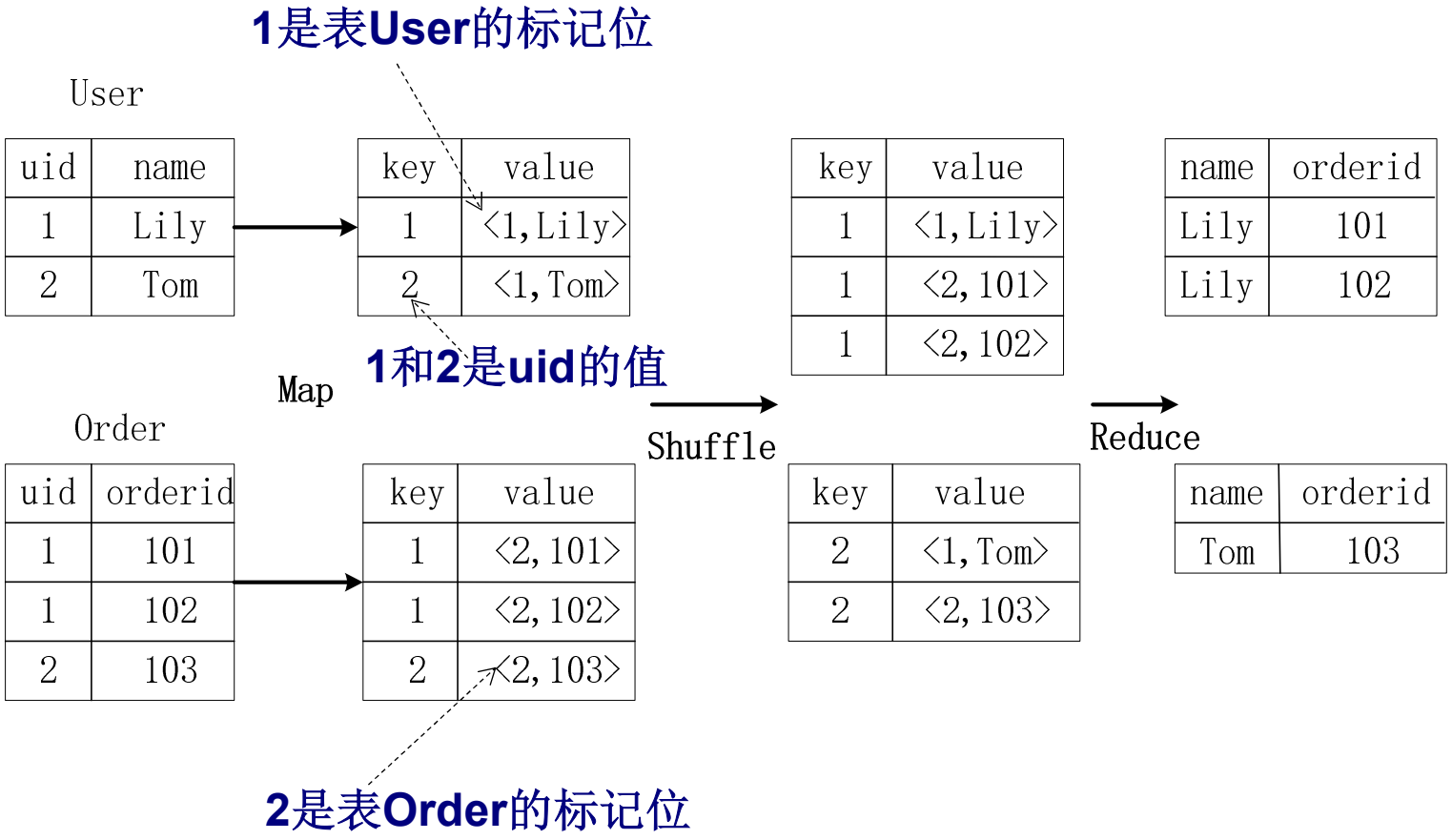
  通常是与MySQL数据库连接后创建的一个MySQL实例，也可以是Hive自带的derby数据库实例。元数据存储模块中主要保存表模式和其他系统元数据，如表的名称、表的列及其属性、表的分区及其属性、表的属性、表中数据所在位置信息等。

9.2.3 Hive工作原理

- 9.2.3.1 SQL语句转换成MapReduce作业的基本原理
- 9.2.3.1 Hive中SQL查询转换成MapReduce作业的过程

9.2.3.1 SQL语句转换成MapReduce作业的基本原理

1. join的实现原理



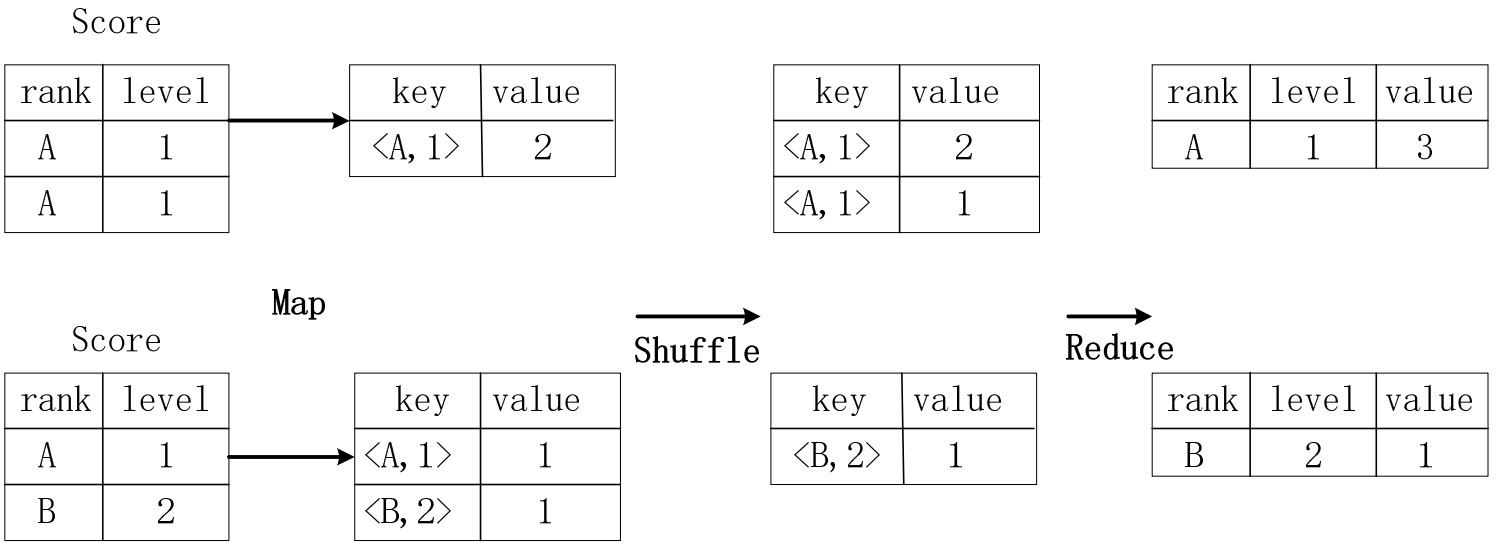
join转化为MapReduce任务的具体过程:

- 在Map阶段，表user中记录(uid,name)映射为键值对(uid,<1,name>)，表order中记录(uid, orderid)映射为键值对(uid,<2, orderid >)
  - 1,2是表user和order的标记位
  - (1,Lily)⇒(1,<1,Lily>)， (1,101)⇒(1,<2,101>)
- 在Shuffle、Sort阶段， (uid,<1,name>)和(uid,<2, orderid >)按键uid的值进行哈希，然后传送给对应的Reduce机器执行，并在该机器上按表的标记位对这些键值对进行排序
  - (1,<1,Lily>)、(1,<2,101>)和 (1,<2,102>)传送到同一台Reduce机器上，并按该顺序排序
  - (2,<1,Tom>)和(2,<2,103>)传送到同一台Reduce机器上，并按该顺序排序
- 在Reduce阶段，对同一台Reduce机器上的键值对，根据表标记位对来自不同表的数据进行笛卡尔积连接操作，以生成最终的连接结果。
  - (1,<1,Lily>)∞(1,<2,101>) ⇒(Lily, 101)
  - (1,<1,Lily>)∞ (1,<2,102>)⇒(Lily, 102)
  - (2,<1,Tom>) ∞(2,<2,103>)⇒(Tom, 103)

2. group by的实现原理

存在一个分组（**Group By**）操作，其功能是把表**Score**的不同片段按照**rank**和**level**的组合值进行合并，计算不同**rank**和**level**的组合值分别有几条记录：

select rank, level ,count(*) as value from score group by rank, level



group by转化为MapReduce任务的具体过程:

- 在Map阶段，表score中记录(rank,level)映射为键值对(<rank,level> , count(rank,level))
  - score表的第一片段中有两条记录(A,1)，(A,1) ➡(<A,1> ,2)
  - score表的第二片段中有一条记录(A,1)，(A,1) ➡(<A,1> ,1)
- 在Shuffle、Sort阶段， (<rank,level> ,count(rank,level))按键<rank,level>的值进行哈希，然后传送给对应的Reduce机器执行，并在该机器上按<rank,level>的值对这些键值对进行排序
  - (<A,1>,2)和 (<A,1>,1)传送到同一台Reduce机器上，按到达顺序排序
  - (<B,2>,1>)传送到另一台Reduce机器上
- 在Reduce阶段，对Reduce机器上的这些键值对，把具有相同<rank,level>键的所有count(rank,level)值进行累加，生成最终结果。
  - (<A,1>,2>)+ (<A,1>,1>)➡(A,1,3)
  - (<B,2>,1)➡ (B,2,1)

9.2.3.2 Hive中SQL查询转换成MapReduce作业的过程

- 当用户向Hive输入一段命令或查询时，Hive需要与Hadoop交互工作来完成该操作：

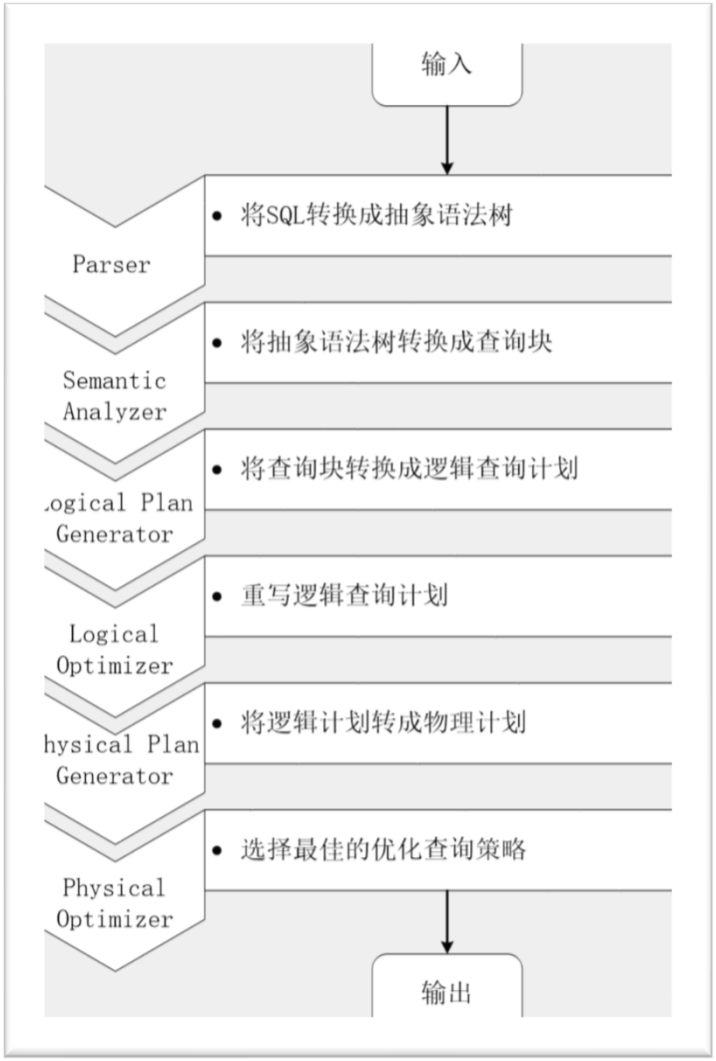
（1）驱动模块接收该命令或查询编译器，对该命令或查询进行解析编译

（2）由优化器对该命令或查询进行优化计算

（3）该命令或查询通过执行器进行执行

执行器通常的任务是启动一个或多个MapReduce任务，有时也不需要启动MapReduce任务，像执行包含*的操作（如select * from 表）时。

Hive中SQL查询的MapReduce作业转化过程：



Hive查询的具体执行过程：

第0步： 用户通过命令行CLI或其他Hive访问工具，向Hive输入一段命令或查询

第1步：由Hive驱动模块中的编译器——Antlr语言识别工具，对用户输入的SQL语言进行词法和语法解析，将SQL语句转化为抽象语法树（AST Tree）的形式。

第2步：对该抽象语法树进行遍历，进一步转化成QueryBlock查询单元。因为抽象语法树的结构仍很复杂，不方便直接翻译为MapReduce算法程序，所以Hive把抽象语法树进一步转化为QueryBlock，其中QueryBlock是一条最基本的SQL语法组成单元，包括输入源、计算过程和输出三部分。

第3步：再对QueryBlock进行遍历，生成执行操作树（OperatorTree）。其中，OperatorTree由很多逻辑操作符组成，如TableScanOperator，SelectOperator，FilterOperator，JoinOperator，GroupByOperator，ReduceSinkOperator等。这些逻辑操作符可以在Map阶段和Reduce阶段完成某一特定操作。

第4步：通过Hive驱动模块中的逻辑优化器对OperatorTree进行优化。变换OperatorTree的形式，来合并多余的操作符，已减少MapReduce任务数量以及Shuffle阶段的数据量。

第5步：对优化后的OperatorTree进行遍历，根据OperatorTree中的逻辑操作符生成需要执行的MapReduce任务。

- 第6步：启动Hive驱动模块中的物理优化器，对生成MapReduce任务进行优化，生成最终的MapReduce任务执行计划。

第7步：最后，由Hive驱动模块中的执行器，对最终的MapReduce任务进行执行。

几点说明：

- 当启动MapReduce程序时，Hive本身是不会生成MapReduce算法程序的
- 需要通过一个表示“Job执行计划”的XML文件驱动执行内置的、原生的Mapper和Reducer模块
- Hive通过和JobTracker通信来初始化MapReduce任务，不必直接部署在JobTracker所在的管理节点上执行
- 通常在大型集群上，会有专门的网关机来部署Hive工具。网关机的作用主要是远程操作和管理节点上的JobTracekr通信来执行任务
- 数据文件通常存储在HDFS上，HDFS由名称节点管理

9.2.4 从外部访问Hive 的典型方式

除了用CLI和HWI工具来访问Hive外，还可以采用以下几种外部访问工具：

（1）Karmasphere是由Karmasphere公司发布的一个商业产品

- 可以直接访问Hadoop里面结构化和非结构化的数据，可以运用SQL及其他语言，可以用于Ad Hoc查询和进一步的分析；
- 还为开发人员提供了一种图形化环境，可以在里面开发自定义算法，为应用程序和可重复的生产流程创建实用的数据集。

（2）Hue是由Cloudera公司提供的一个开源项目

- 是运营和开发Hadoop应用的图形化用户界面；
- Hue程序被整合到一个类似桌面的环境，以web程序的形式发布，对于单独的用户来说不需要额外的安装。

（3）Qubole公司提供了“Hive即服务”的方式

- 托管在AWS平台，这样用户在分析存储在亚马逊S3云中的数据集时，就无需了解Hadoop系统管理；
- 提供的Hadoop服务能够根据用户的工作负载动态调整服务器资源配置，实现随需计算。

9.3 Hive的应用

构建于Hadoop上的数据仓库，除了依赖于Hadoop的基本组件HDFS和MapReduce外，还结合使用了Hive、Pig、HBase与Mahout，如图9-6所示。

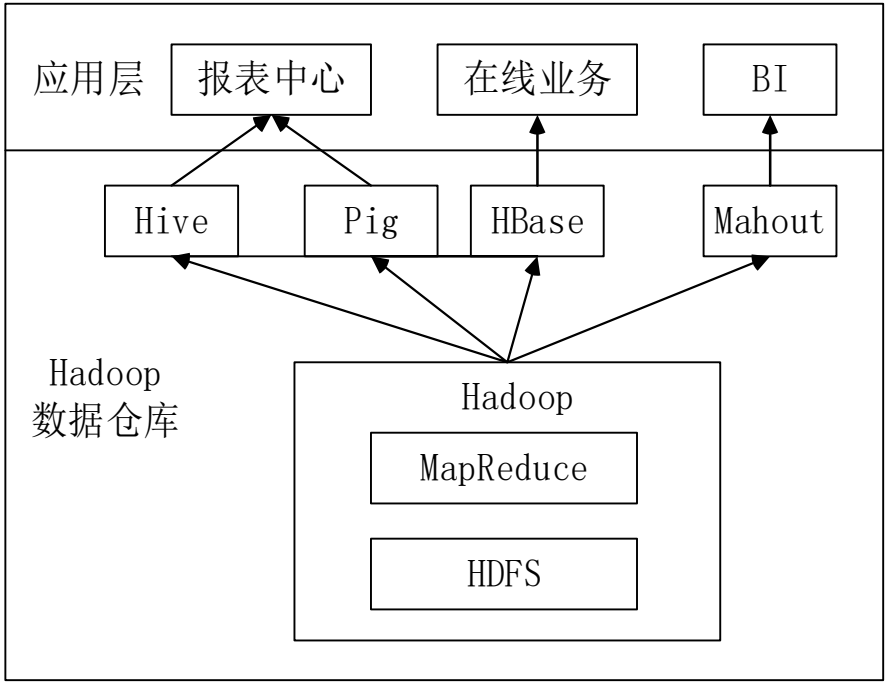


图9-6 Hadoop 数据仓库框架

Hadoop 数据仓库框架各部分的作用：

- 在Hadoop数据仓库中， Hive和Pig主要应用在报表中心上，Hive主要用于报表分析，Pig主要用于报表中数据的转换工作；
- HBase主要用于在线业务，因为HDFS缺乏随机读写操作，而HBase正是为此开发的，支持实时访问数据。
- Mahout常用于BI（商务智能），Mahout提供一些可扩展的机器学习领域的经典算法的实现，旨在帮助开发人员更加方便快捷地创建智能应用程序。

Hive的应用如图9-7所示：

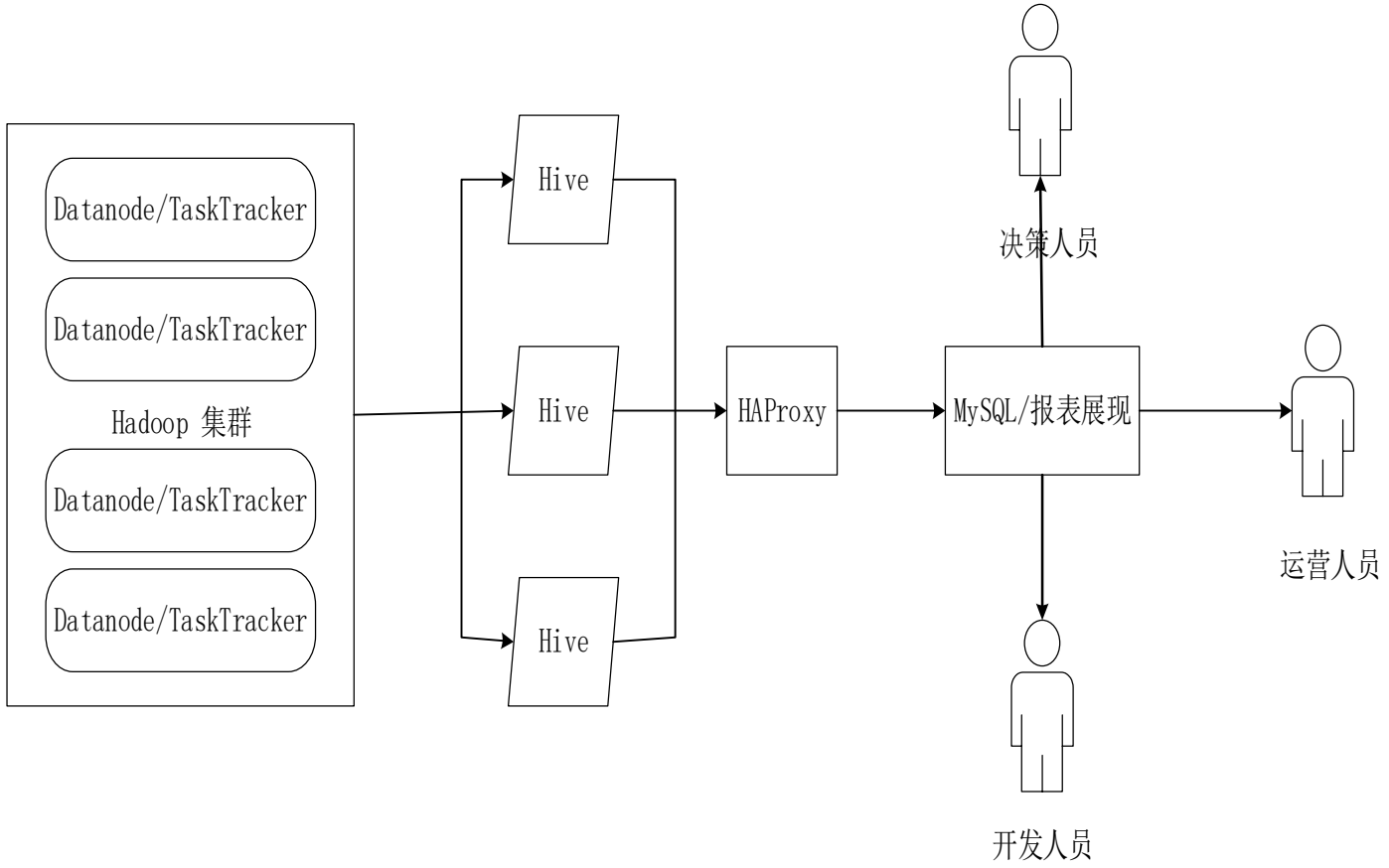


图9-7. Hive的应用

Hive在报表中心的应用流程：

- 在Hadoop集群上构建的数据仓库由多个Hive进行管理
- 由HAProxy提供一个接口，对Hive实例进行访问
- 由Hive处理后得到的各种数据信息，或存放在MySQL数据库中，或直接以报表的形式展现
- 不同人员会根据所需数据进行相应工作

其中，**HAProxy是Hive HA的实现。**

Hive HA原理：

- 将若干个Hive 实例纳入一个资源池，然后对外提供一个唯一的接口，进行proxy relay。
- 对于程序开发人员，就把它认为是一台超强“Hive"。每次它接收到一个Hive查询连接后，都会轮询资源池里可用的Hive 资源。

9.4 Impala

- 9.4.1 Impala简介
- 9.4.2 Impala系统架构
- 9.4.3 Impala查询执行过程
- 9.4.4 Impala与Hive的比较

9.4.1 Impala简介

- 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。

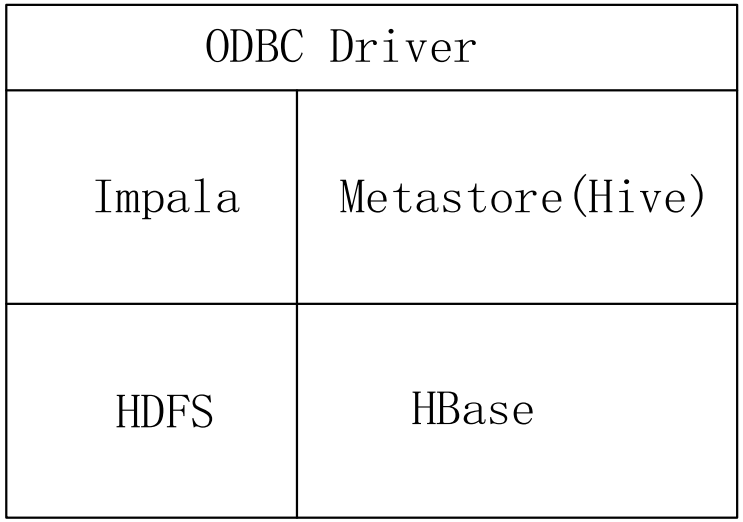


图9-8 Impala与其他组件关系

- 与Hive类似，Impala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。

9.4.2 Impala系统架构

Impala系统架构如图9-9所示：

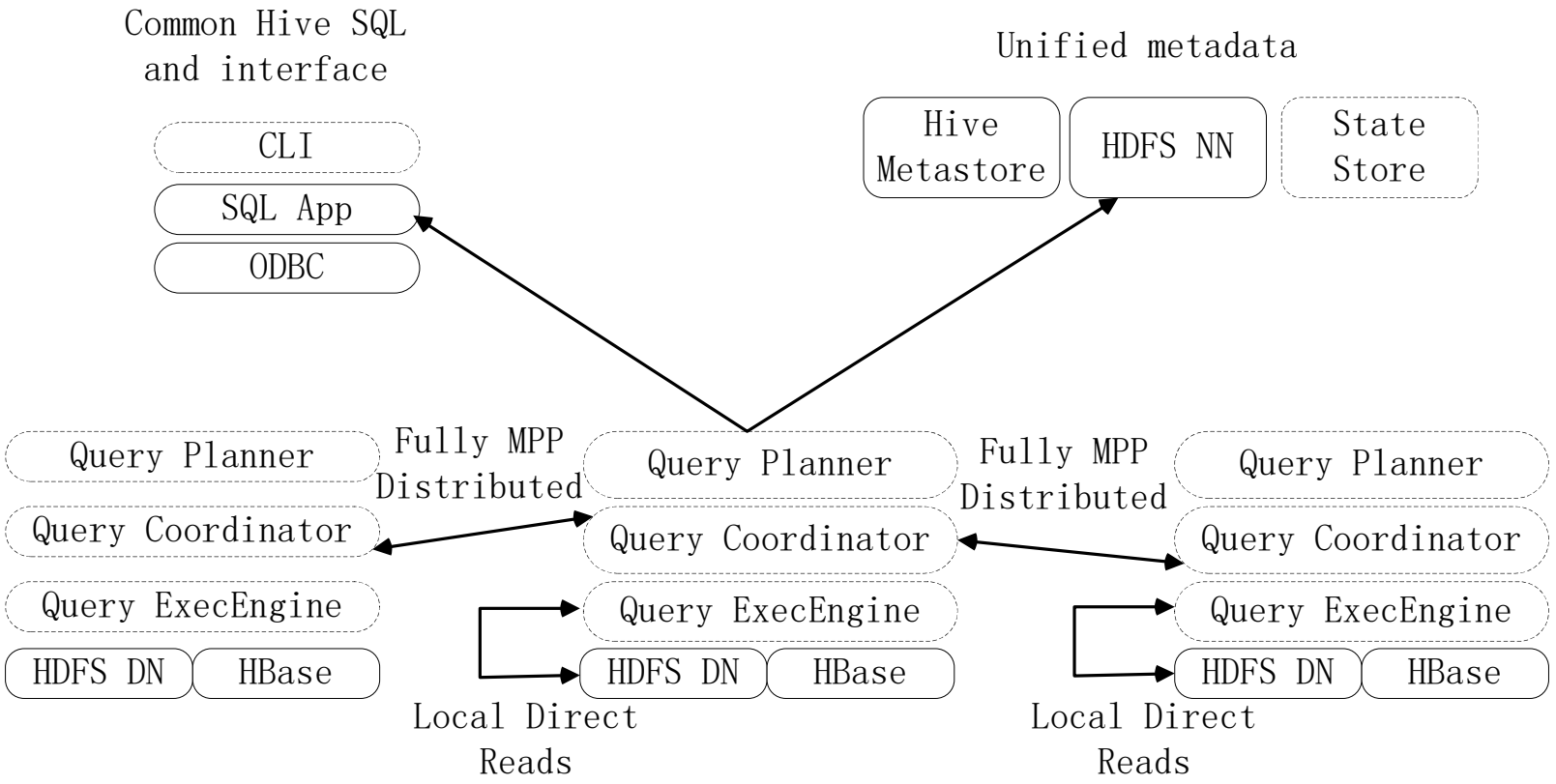


图9-9 Impala系统架构

Impala主要由Impalad，State Store和CLI三部分组成：

1. Impalad是Impala一个进程

- 负责协调客户端提交的查询的执行，给其他impalad分配任务以及收集其他impalad的执行结果进行汇总
- 执行其他impalad给其分配的任务，主要就是对本地HDFS和HBase里的部分数据进行操作

2. State Store

- 跟踪集群中的Impalad的健康状态及位置信息
- 创建多个线程来处理Impalad的注册订阅和与各类Impalad保持心跳连接

当State Store离线后，Impalad一旦发现State Store处于离线时，就会进入recovery模式，并进行反复注册；

当State Store重新加入集群后，自动恢复正常，更新缓存数据。

3. CLI给用户提供查询使用的命令行工具

- 提供了Hue、JDBC及ODBC的使用接口
- Impala采用与Hive相同的元数据、SQL语法、ODBC驱动程序和用户接口，这样做的主要原因是在使用同一公司Hadoop产品时，批处理和实时查询的平台是统一的
- Impala中表的元数据存储采用的是Hive的元数据存储方式
- State Store负责收集分布在集群中各个impalad进程的资源信息，用于查询的调度
- HDFS名称节点（HDFS NN）记录了每个文件中各个块所在的数据节点的位置信息。
- impalad进程主要包含Query Planner、Query Coordinator和Query Exec Engine三个模块与HDFS的数据节点（HDFS DN）运行在同一节点上完全分布运行在MPP（大规模并行处理系统）架构。

9.4.3 Impala查询执行过程

Impala查询过程如图9-10所示：

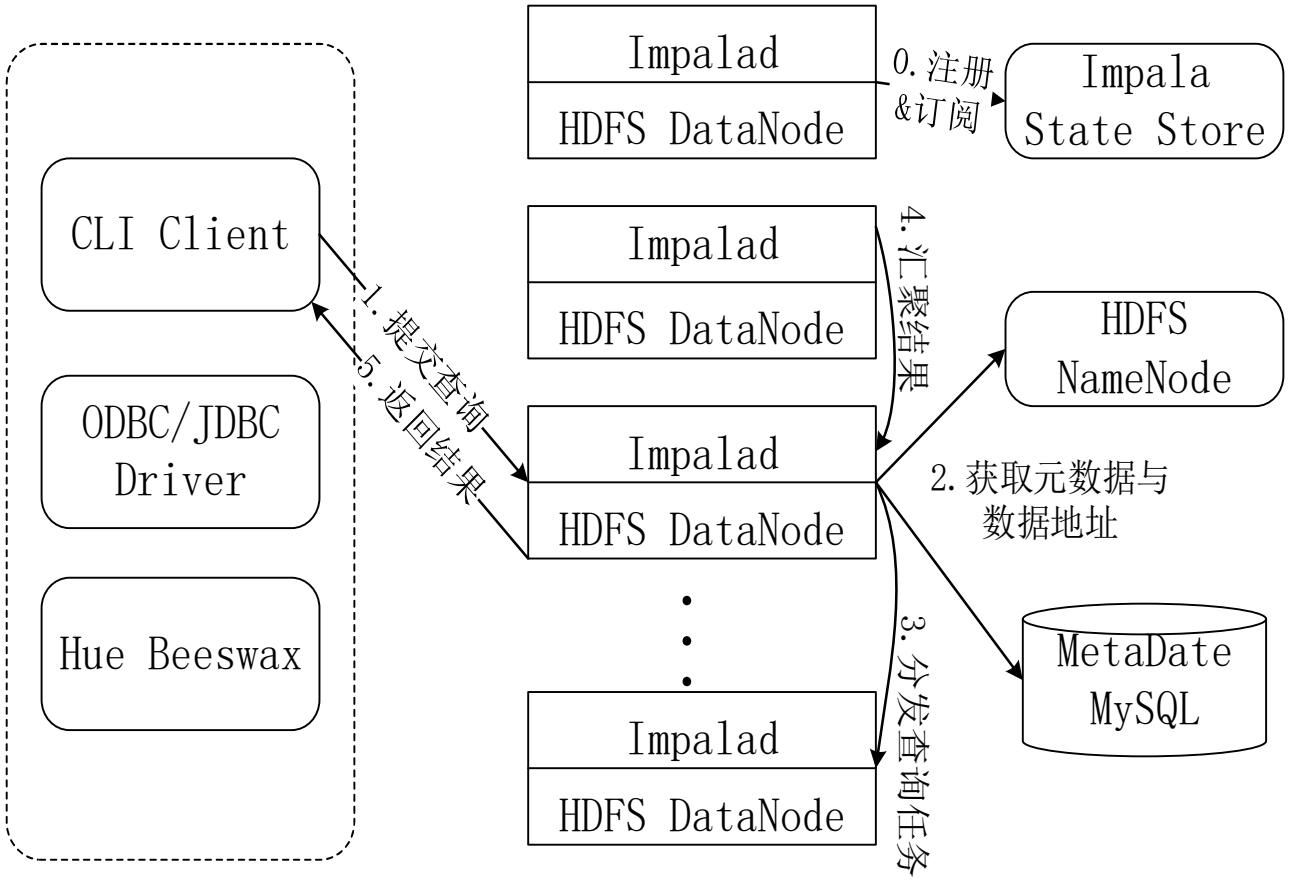


图9-10 Impala查询过程图

Impala执行查询的具体过程：

- 第0步，当用户提交查询前，Impala先创建一个负责协调客户端提交的查询的Impalad进程，该进程会向Impala State Store提交注册订阅信息，State Store会创建一个statestored进程，statestored进程通过创建多个线程来处理Impalad的注册订阅信息。
- 第1步，用户通过CLI客户端提交一个查询到impalad进程，Impalad的Query Planner对SQL语句进行解析，生成解析树；然后，Planner把这个查询的解析树变成若干PlanFragment，发送到Query Coordinator，其中，PlanFragment由PlanNode组成的，能被分发到单独的节点上原子执行，每个PlanNode表示一个关系操作和对其执行优化需要的信息。

- 第2步，Coordinator通过从MySQL元数据库中获取元数据，从HDFS的名称节点中获取数据地址，以得到存储这个查询相关数据的所有数据节点。
- 第3步，Coordinator初始化相应impalad上的任务执行，即把查询任务分配给所有存储这个查询相关数据的数据节点。
- 第4步，Query Executor通过流式交换中间输出，并由Query Coordinator汇聚来自各个impalad的结果。
- 第5步，Coordinator把汇总后的结果返回给CLI客户端。

9.4.4 Impala与Hive的比较

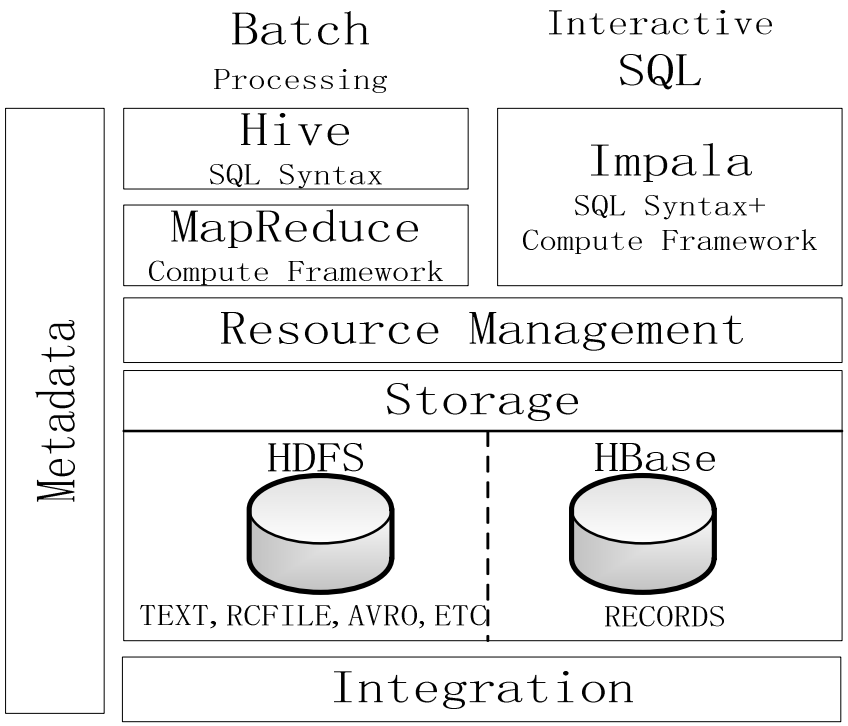


图9-11 Impala与Hive的对比

Hive与Impala的不同点总结如下：

1. Hive适合于长时间的批处理查询分析，而Impala适合于实时交互式SQL查询
2. Hive依赖于MapReduce计算框架，Impala把执行计划表现为一棵完整的执行计划树，直接分发执行计划到各个Impalad执行查询
3. Hive在执行过程中，如果内存放不下所有数据，则会使用外存，以保证查询能顺序执行完成，而Impala在遇到内存放不下数据时，不会利用外存，所以Impala目前处理查询时会受到一定的限制

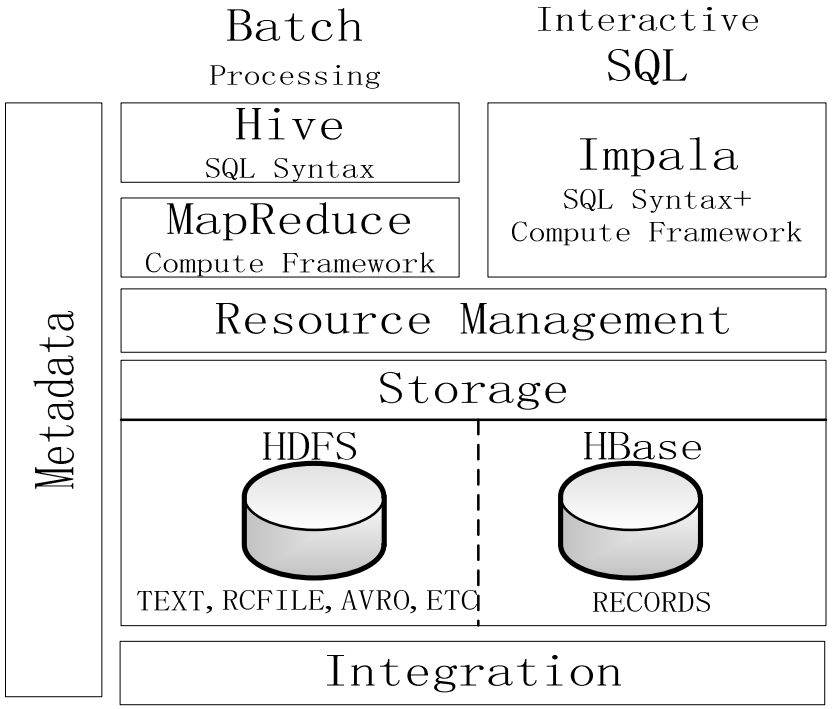


图 9-11 Impala与Hive的对比

Hive与Impala的相同点总结如下：

1. Hive与Impala使用相同的存储数据池，都支持把数据存储于HDFS和HBase中，其中HDFS支持存储TEXT、RCFILE、PARQUET、AVRO、ETC格式数据，HBase存储表中记录。
2. Hive与Impala使用相同的元数据
3. Hive与Impala中对SQL的解释处理比较相似，都是通过词法分析生成执行计划

总结:

- Impala的目的不在于替换现有的MapReduce工具
- 把Hive与Impala配合使用效果最佳
- 可以先使用Hive进行数据转换处理，之后再使用Impala在Hive处理后的结果数据集上进行快速的数据分析

9.6 Hive编程实践

- 9.6.1 Hive安装与配置
- 9.6.2 Hive数据类型
- 9.6.3 Hive基本操作
- 9.6.4 Hive应用实例：WordCount
- 9.6.5 Hive编程的优势

9.6.1 Hive的安装与配置

1. Hive安装

安装Hive之前需要安装jdk1.6以上版本以及启动Hadoop

- 下载安装包apache-hive-3.2.1-bin.tar.gz
- 下载地址：http://www.apache.org/dyn/closer.cgi/hive/
- 解压安装包 apache-hive-3.2.1-bin.tar.gz至路径 /usr/local
- 配置系统环境，将hive下的bin目录添加到系统的path中

2. Hive配置

Hive有三种运行模式，单机模式、伪分布式模式、分布式模式，均是通过修改hive-site.xml文件实现，如果 hive-site.xml文件不存在，我们可以参考$HIVE_HOME/conf目录下的hive-default.xml.template文件新建。

9.6.2 Hive数据类型

表 9-2 Hive基本数据类型

类型	描述	示例
TINYINT	1个字节（8位）有符号整数	1
SMALLINT	2个字节（16位）有符号整数	1
INT	4个字节（32位）有符号整数	1
BIGINT	8个字节（64位）有符号整数	1
FLOAT	4个字节（32位）单精度浮点数	1.0
DOUBLE	8个字节（64位）双精度浮点数	1.0
BOOLEAN	布尔类型，true/false	true
STRING	字符串，可以指定字符集	“xmu”
TIMESTAMP	整数、浮点数或者字符串	1327882394（Unix新纪元秒）
BINARY	字节数组	[0,1,0,1,0,1,0,1]

表9-3 Hive集合数据类型

类型	描述	示例
ARRAY	一组有序字段，字段的类型必须相同	Array(1,2)
MAP	一组无序的键/值对，键的类型必须是原子的，值可以是任何数据类型，同一个映射的键和值的类型必须相同	Map(‘a’,1,’b’,2)
STRUCT	一组命名的字段，字段类型可以不同	Struct(‘a’,1,1,0)

9.6.3 Hive基本操作

1. create: 创建数据库、表、视图

- 创建数据库

①创建数据库hive

```
hive> create database hive;
```

②创建数据库hive。因为hive已经存在，所以会抛出异常，加上if not exists关键字，则不会抛出异常

```
hive> create database if not exists hive;
```

创建表

①在hive数据库中，创建表usr，含三个属性id，name，age

```
hive> use hive;
hive>create table if not exists usr(id bigint,name string,age int);
```

②在hive数据库中，创建表usr，含三个属性id，name，age，存储路径为“/usr/local/hive/warehouse/hive/usr”

```
hive>create table if not exists hive.usr(id bigint,name string,age int)
    >location ‘/usr/local/hive/warehouse/hive/usr’;
```

③在hive数据库中，创建外部表usr，含三个属性id，name，age，可以读取路径“/usr/local/data”下以“，”分隔的数据。

```
hive>create external table if not exists hive.usr(id bigint,name string,age
int)
    >row format delimited fields terminated by ','
     location ‘/usr/local/data’;
```

④在hive数据库中，创建分区表usr，含三个属性id，name，age，还存在分区字段sex。

```
hive>create table hive.usr(id bigint,name string,age int) partition by(sex boolean);
```

⑤在hive数据库中，创建分区表usr1，它通过复制表usr得到。

```
hive> use hive;
hive>create table if not exists usr1 like usr;
```

■创建视图

①创建视图little_usr，只包含usr表中id，age属性

```
hive>create view little_usr as select id,age from usr;
```

2. drop：删除数据库、表、视图

- 删除数据库

①删除数据库hive，如果不存在会出现警告

```
hive> drop database hive;
```

②删除数据库hive，因为有if exists关键字，即使不存在也不会抛出异常

```
hive>drop database if not exists hive;
```

③删除数据库hive，加上cascade关键字，可以删除当前数据库和该数据库中的表

```
hive> drop database if not exists hive cascade;
```

删除表

①删除表**usr**，如果是内部表，元数据和实际数据都会被删除；
如果是外部表，只删除元数据，不删除实际数据

```
hive> drop table if exists usr;
```

删除视图

①删除视图 **little_usr**

```
hive> drop view if exists little_usr;
```

3. alter：修改数据库、表、视图

■ 修改数据库

①为hive数据库设置dbproperties键值对属性值来描述数据库属性信息

```
hive> alter database hive set dbproperties('edited-by'='lily');
```

■ 修改表

①重命名表usr为user

```
hive> alter table usr rename touser
```

②为表usr增加新分区

```
hive> alter table usr add if not exists partition(age=10);
```

③删除表usr中分区

```
hive> alter table usr drop if exists partition(age=10);
```

④把表usr中列名name修改为username，并把该列置于age列后

```
hive>alter table usr change name username string after age;
```

⑤在对表usr分区字段之前，增加一个新列sex

```
hive>alter table usr add columns(sex boolean);
```

⑥删除表usr中所有字段并重新指定新字段newid，newname，newage

```
hive>alter table usr replace columns(newid bigint,newname string,newage int);
```

⑥为usr表设置tblproperties键值对属性值来描述表的属性信息

```
hive> alter table usr set tabproperties('notes'='the columns in usr may be null except id');
```

■ 修改视图

①修改little_usr视图元数据中的tblproperties属性信息

```
hive> alter view little_usr set tabproperties('create_at'='refer to timestamp');
```

4. show：查看数据库、表、视图

- 查看数据库

①查看Hive中包含的所有数据库

```
hive> show databases;
```

②查看Hive中以h开头的所有数据库

```
hive>show databases like ‘h.*’;
```

- 查看表和视图

①查看数据库hive中所有表和视图

```
hive> use hive;
hive> show tables;
```

②查看数据库hive中以u开头的所有表和视图

```
hive> show tables in hive like ‘u.*’;
```

5. describe：描述数据库、表、视图

- 描述数据库

①查看数据库**hive**的基本信息，包括数据库中文件位置信息等

```
hive> describe database hive;
```

②查看数据库**hive**的详细信息，包括数据库的基本信息及属性信息等

```
hive>describe database extended hive;
```

- 描述表和视图

①查看表**usr**和视图**little_usr**的基本信息，包括列信息等

```
hive> describe hive.usr/ hive.little_usr;
```

②查看表**usr**和视图**little_usr**的详细信息，包括列信息、位置信息、属性信息等

```
hive> describe extended hive.usr/ hive.little_usr;
```

③查看表**usr**中列**id**的信息

```
hive> describe extended hive.usr.id;
```

6. load：向表中装载数据

①把目录'/usr/local/data'下的数据文件中的数据装载进usr表并覆盖原有数据

```
hive> load data local inpath '/usr/local/data' overwrite into table usr;
```

②把目录'/usr/local/data'下的数据文件中的数据装载进usr表不覆盖原有数据

```
hive> load data local inpath '/usr/local/data' into table usr;
```

③把分布式文件系统目录'hdfs://master_server/usr/local/data'下的数据文件数据装载进usr表并覆盖原有数据

```
hive> load data inpath 'hdfs://master_server/usr/local/data'
    >overwrite into table usr;
```

7. insert：向表中插入数据或从表中导出数据

①向表usr1中插入来自usr表的数据并覆盖原有数据

```
hive> insert overwrite table usr1
    > select * from usr where age=10;
```

②向表usr1中插入来自usr表的数据并追加在原有数据后

```
hive> insert into table usr1
    > select * from usr
    > where age=10;
```

9.6.4 Hive应用实例：WordCount

现在我们通过一个实例——词频统计，来深入学习一下Hive的具体使用。首先，需要创建一个需要分析的输入数据文件，然后编写HiveQL语句实现WordCount算法，在Linux下实现步骤如下：

（1）创建input目录，其中input为输入目录。命令如下：

```
$ cd /usr/local/hadoop
$ mkdir input
```

（2）在input文件夹中创建两个测试文件file1.txt和file2.txt，命令如下：

```
$ cd  /usr/local/hadoop/input
$ echo "hello world" > file1.txt
$ echo "hello hadoop" > file2.txt
```

（3）进入hive命令行界面，编写HiveQL语句实现WordCount算法，命令如下：

```
$ hive
hive> create table docs(line string);
hive> load data inpath 'input' overwrite into table docs;
hive>create table word_count as
    select word, count(1) as count from
    (select explode(split(line,' '))as word from docs) w
    group by word
    order by word;
```

执行完成后，用select语句查看运行结果如下：

```
OK
Time taken: 2.662 seconds
hive> select * from word_count;
OK
hadoop  1
hello   2
world   1
Time taken: 0.043 seconds, Fetched: 3 row(s)
```

9.6.5 Hive的编程优势

WordCount算法在MapReduce中的编程实现和Hive中编程实现的主要不同点：

1. 采用Hive实现WordCount算法需要编写较少的代码量

- 在**MapReduce**中，**WordCount**类由**63**行**Java**代码编写而成

代码位置：

%HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.1.jar；

- 在**Hive**中只需要编写**7**行代码

2. 在MapReduce的实现中，需要进行编译生成jar文件来执行算法，而在Hive中不需要.

- **HiveQL**语句的最终实现需要转换为**MapReduce**任务来执行，这都是由**Hive**框架自动完成的，用户不需要了解具体实现细节.

本章小结

- 本章详细介绍了Hive的基本知识。Hive是一个构建于Hadoop顶层的数据仓库工具，主要用于对存储在 Hadoop 文件中的数据集进行数据整理、特殊查询和分析处理。Hive在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HDFS存储数据，依赖MapReduce处理数据。
- Hive支持使用自身提供的命令行CLI、简单网页HWI访问方式，及通过Karmasphere、Hue、Qubole等工具的外部访问。
- Hive在数据仓库中的具体应用中，主要用于报表中心的报表分析统计上。在Hadoop集群上构建的数据仓库由多个Hive进行管理，具体实现采用Hive HA原理的方式，实现一台超强“hive"。
- Impala作为新一代开源大数据分析引擎，支持实时计算，并在性能上比Hive高出3~30倍，甚至在将来的某一天可能会超过Hive的使用率而成为Hadoop上最流行的实时计算平台。
- 本章最后以单词统计为例，详细介绍了如何使用Hive进行简单编程。