Hadoop 集群（第 13 期）

——Hive 简介及安装

1、Hive简介

Hive 是一个构建在 **Hadoop** 上的数据仓库平台，其**设计目标**是使 **Hadoop** 上的**数据操作**与**传统 SQL 结合**，让熟悉 SQL 编程的开发人员能够轻松向 Hadoop 平台转移。Hive 是 Facebook 的信息平台的重要组成部分，Facebook 在 **2008** 年将其贡献给 Apache，现已成为 Apache 旗下的一个独立子项目。

Hive 是基于 **Hadoop 文件系统**上的数据仓库架构。它为**数据仓库**的**管理**提供了**许多功能：数据 ETL（抽取，转换和加载）工具**，**数据存储管理**和**大型数据集的查询与分析能力**。同时 Hive 还定义了类 SQL 的语言——**HiveQL**，HiveQL 允许用户进行和 SQL **相似**的操作。HiveQL 还允许开发人员方便地使用 mapper 和 reducer 操作，这样对 MapReduce 框架是一个强有力的支持。

由于 **Hadoop** 是**批量处理**系统，**任务**是**高延迟性**的，所以在**任务提交**和**处理过程**中**会消耗**一些**时间成本**。同样，即使 **Hive** 处理的**数据集非常小**（比如几百 MB），在执行时也会出现**延迟现象**。这样，Hive 的**性能**就**不可能**很好地和**传统**的 **Oracle 数据库**进行**比较**了。**Hive 不能提供数据排序**和**查询 cache 功能**，也**不提供在线事务处理**，**不提供实时**的**查询功能**和**记录级的更新**，但 **Hive** 能**更好地处理不变**的**大规模数据集**（例如网络日志）上的**批量任务**。所以，**Hive 最大的价值**是**可扩展性**（基于 Hadoop 平台，可以自动适应机器数目和数据量的动态变化）、**可延展性**（结合 MapReduce 和用户定义的函数库），并且拥有**良好**的**容错性**和**低约束**的**数据输入格式**。

2、Hive体系结构

Hive 本身建立在 Hadoop 的体系架构上，提供了一个 SQL 解析过程，并从外部接口中获取命令，以对用户指令进行解析。**Hive** 可将**外部命令解析**成一个 **MapReduce** 可**执行计划**，并按照该计划生产 MapReduce 任务后交给 Hadoop 集群处理，Hive 的体系结构如图 2-1、图 2-2、图 2-3 以及图 2-4 所示。

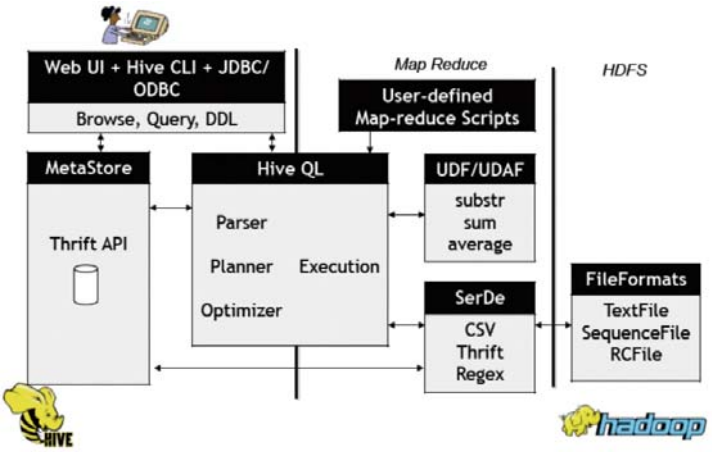


图 2-1 Hive 的体系结构（1）

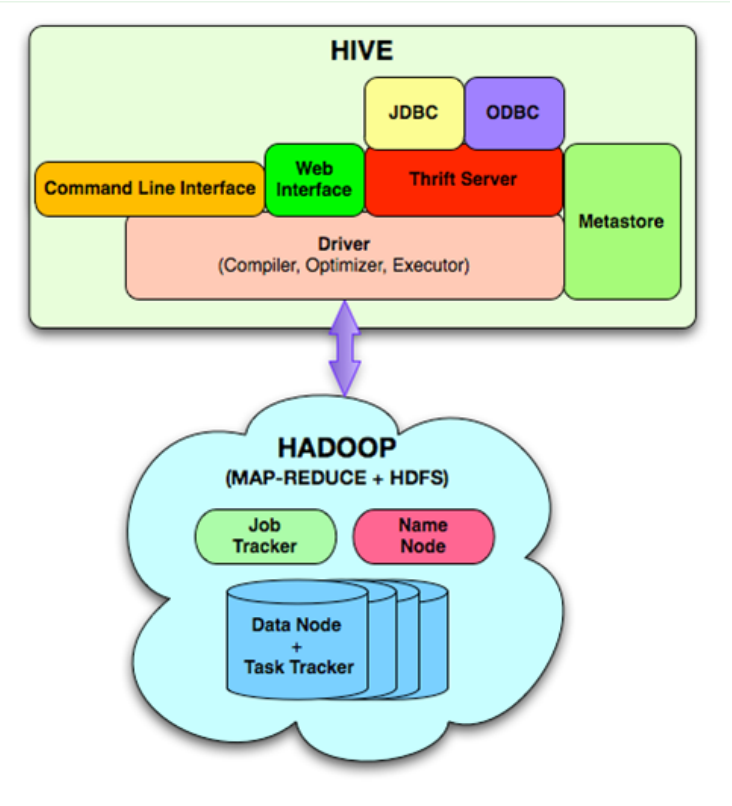


图 2-2 Hive 的体系结构（2）

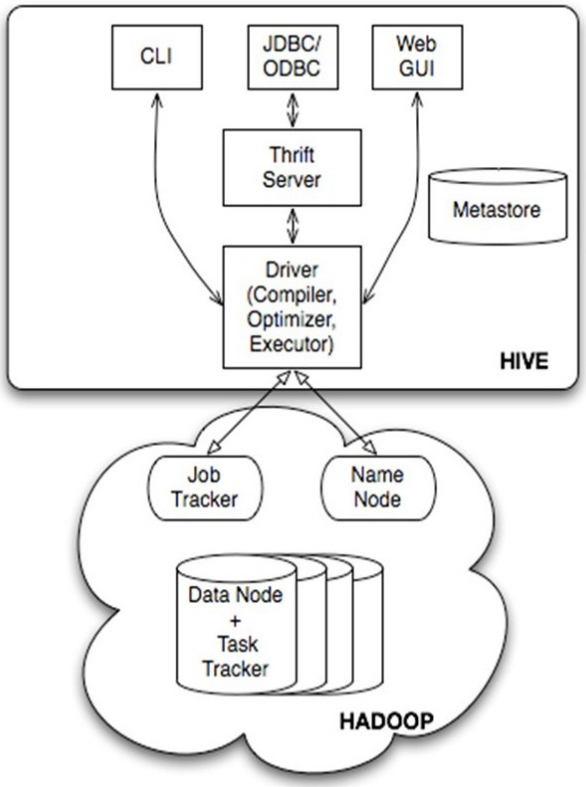


图 2-3 Hive 的体系结构（3）

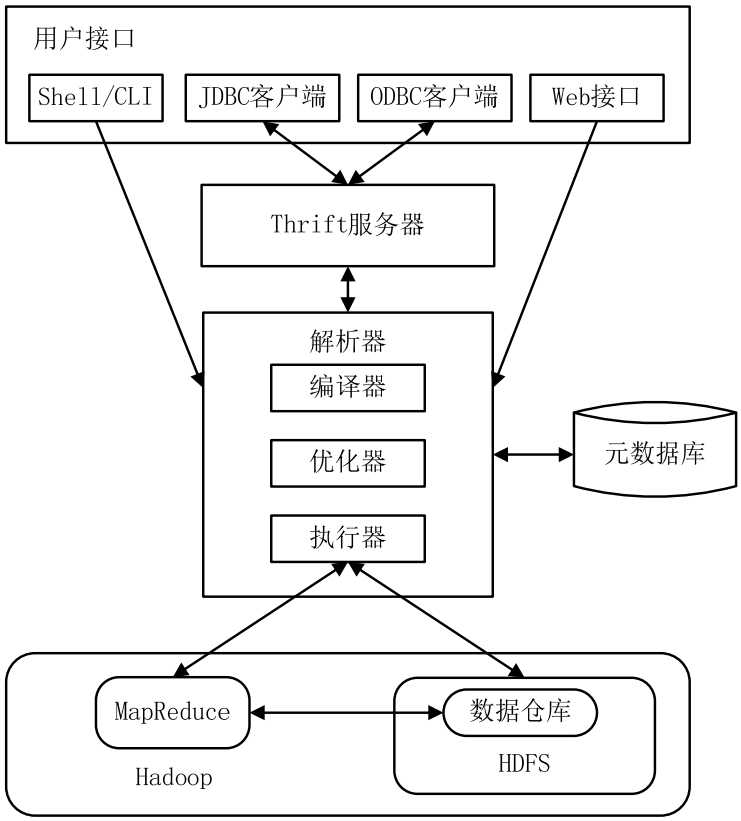


图 2-4 Hive 的体系结构（4）

主要分为以下几个部分：

- **用户接口**

UI 用户提交查询和其他的操作。包含类似命令行接口 CLI（Command Line Interface），Web UI 以及 JDBC/ODBC 驱动。其中最常用的是 CLI，CLI 启动的时候，会同时启动一个 Hive 副本。Client 是 Hive 的客户端，用户连接至 Hive Server。在启动 Client 模式的时候，需要指出 Hive Server 所在节点，并且在该节点启动 Hive Server。 WUI 是通过浏览器访问 Hive。

备注：命令行接口 CLI 又称为 Hive Shell。

- **Thrift 服务器**

当 Hive 以**服务器模式**运行时，可以作为 Thrift 服务器，供客户连接。

- **元数据存储（Metastore）**

元数据存储此组件存储数据仓库里所有的各种表与分区的结构化信息，包括**列**与**列类型**信息，**序列化**器与**反序列化**器，从而能够读写 hdfs 中的数据，**通常**是**存储**在**关系数据库**如 **MySQL**，**Derby** 中。

- **解析器**

解释器、编译器、优化器完成 HQL 查询语句从词法分析、语法分析、编译、优化以及查询计划的生成。生成的查询计划存储在 HDFS 中，并在随后有 MapReduce 调用执行。

- **Hadoop**

Hive 的数据存储在 HDFS 中，利用 MapReduce 进行计算。

2.1 Hive的存储结构

基于 MapReduce 的 Hive 数据仓库在超大规模数据分析中扮演了重要角色，对于典型的 Web 服务供应商，这些分析有助于它们快速理解动态的用户行为及变化的用户需求。数据存储结构是影响数据仓库性能的关键因素之一。Hadoop 系统中常用的文件存储格式有支持文本的 TextFile 和支持二进制的 SequenceFile 等，它们都属于行存储方式。Facebook 工程师发表的 RCFile: A Fast and Spaceefficient Data Placement Structure in MapReducebased Warehouse Systems 一文，介绍了一种高效的数据存储结构——RCFile（Record Columnar File），并将其应用于 Facebook 的数据仓库 Hive 中。与传统数据库的数据存储结构相比，RCFile 更有效地满足了基于 MapReduce 的数据仓库的四个关键需求，即 Fast data loading、Fast query processing、Highly efficient storage space utilization 和 Strong adaptivity to highly dynamic workload patterns。

1）数据仓库的需求

基于 Facebook 系统特征和用户数据的分析，在 MapReduce 计算环境下，数据仓库对于数据存储结构有四个关键需求。

- **Fast data loading**

对于 Facebook 的产品数据仓库而言，快速加载数据（写数据）是非常关键的。每天大约有超过 20TB 的数据上传到 Facebook 的数据仓库，由于数据加载期间网络和磁盘流量会干扰正常的查询执行，因此缩短数据加载时间是非常必要的。

- **Fast query processing**

为了满足实时性的网站请求和支持高并发用户提交查询的大量读负载，查询响应时间是非常关键的，这要求底层存储结构能够随着查询数量的增加而保持高速的查询处理。

- **Highly efficient storage space utilization**

高速增长的用户活动总是需要可扩展的存储容量和计算能力，有限的磁盘空间需要合理管理海量数据的存储。实际上，该问题的解决方案就是最大化磁盘空间利用率。

- **Strong adaptivity to highly dynamic workload patterns**

同一份数据集会供给不同应用的用户，通过各种方式来分析。某些数据分析是例行过程，按照某种固定模式周期性执行；而另一些则是从中间平台发起的查询。大多数负载不遵循任何规则模式，这需要底层系统在存储空间有限的前提下，对数据处理中不可预知的动态数据具备高度的适应性，而不是专注于某种特殊的负载模式。

2）MapReduce 存储策略

要想设计并实现一种基于 MapReduce 数据仓库的高效数据存储结构，关键挑战是在 MapReduce 计算环境中满足上述四个需求。在传统数据库系统中，三种数据存储结构被广泛研究，分别是行存储结构、列存储结构和 PAX 混合存储结构。上面这三种结构都有其自身特点，不过简单移植这些数据库导向的存储结构到基于 MapReduce 的数据仓库系统并不能很好地满足所有需求。

- **行存储**

如图 2.1-1 所示，基于 Hadoop 系统行存储结构的优点在于快速数据加载和动态负载的高适应能力，这是因为行存储保证了相同记录的所有域都在同一个集群节点，即同一个 HDFS 块。不过，行存储的缺点也是显而易见的，例如它不能支持快速查询处理，因为当查询仅仅针对多列表中的少数几列时，它不能跳过不必要的列读取；此外，由于混合着不同数据值的列，行存储不易获得一个极高的压缩比，即空间利用率不易大幅提高。尽管通过熵编码和利用列相关性能够获得一个较好的压缩比，但是复杂数据存储实现会导致解压开销增

大。

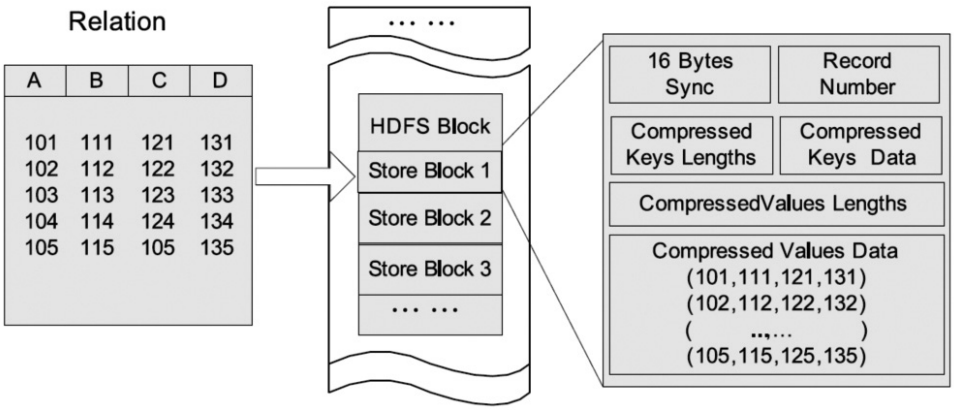


图 2.1-1 HDFS 块内行存储的例子

- **列存储**

图 2.1-2 显示了在 **HDFS** 上按照**列组存储表格**的例子。在这个例子中，列 A 和列 B 存储在同一列组，而列 C 和列 D 分别存储在单独的列组。查询时列存储能够避免读不必要的列，并且压缩一个列中的相似数据能够达到较高的压缩比。然而，由于元组重构的较高开销，它并不能提供基于 Hadoop 系统的快速查询处理。列存储不能保证同一记录的所有域都存储在同一集群节点，例如图 2.1-2 的例子中，记录的 4 个域存储在位于不同节点的 3 个 HDFS 块中。因此，记录的重构将导致通过集群节点网络的大量数据传输。尽管预先分组后，多个列在一起能够减少开销，但是对于高度动态的负载模式，它并不具备很好的适应性。除非所有列组根据可能的查询预先创建，否则对于一个查询需要一个不可预知的列组合，一个记录的重构或许需要 2 个或多个列组。再者由于多个组之间的列交叠，列组可能会创建多余的列数据存储，这导致存储利用率的降低。

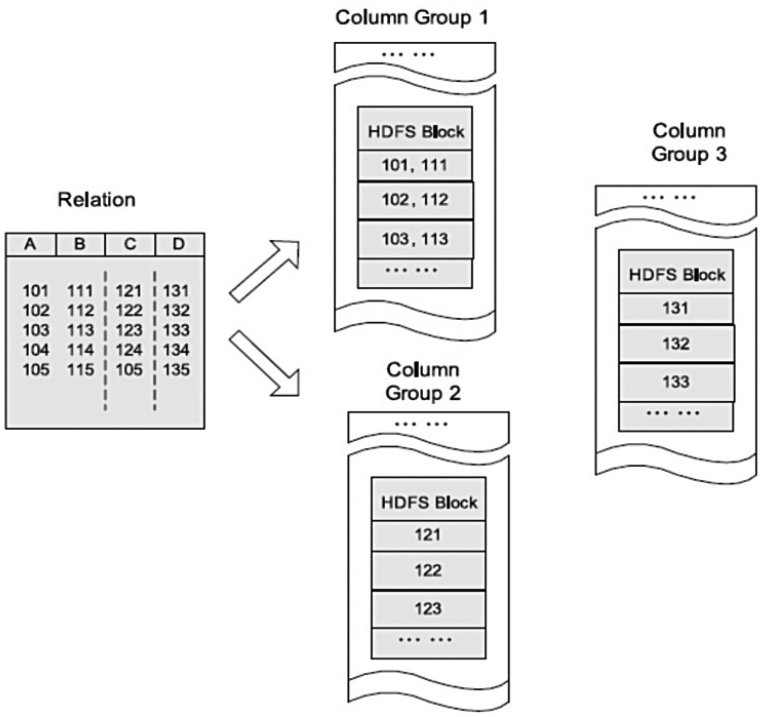


图 2.1-2 HDFS 块内列存储的例子

- **PAX 混合存储**

PAX 存储模型（用于 Data Morphing 存储技术）使用混合存储方式，目的在于提升 CPU Cache 性能。对于记录中来自不同列的多个域，PAX 将它们放在一个磁盘页中。在每个磁盘页中，PAX 使用一个迷你页来存储属于每个列的所有域，并使用 一个页头来存储迷你页的指针。类似于行存储，PAX 对多种动态查询有很强的适应能力。然而，它并不能满足大型分布式系统对于高存储空间利用率和快速查询处理的需求，原因在于：首先，PAX 没有数据压缩的相关工作，这部分与 Cache 优化关系不大，但对于大规模数据处理系统是非常关键的，它提供了列维度数据压缩的可能性；其次，PAX 不能提升 I/O 性能，因为它不能改变实际的页内容，该限制使得大规模数据扫描时不易实现快速查询处理；再次，PAX 用固定的页 作为数据组织的基本单位，按照这个大小，在海量数据处理系统中，PAX 将不会有效存储不同大小类型的数据域。

本小节介绍的是 **RCFile** 数据存储结构在 **Hadoop** 系统上的实现。该结构强调：第一，RCFile 存储的表是水平划分的，分为多个行组， 每个行组再被垂直划分， 以便每列单独存储；第二，RCFile 在每个行组中利用一个列维度的数据压缩，并提供一种 Lazy 解压（decompression）技术来在查询执行时避免不必要的列解压；第三，RCFile 支持弹性的行组大小，行组大小需要权衡数据压缩性能和查询性能两方面。

3）RCFile 的设计与实现

RCFile（Record Columnar File）存储结构遵循的是“先水平划分，再垂直划分”的设计理念，这个想法来源于 PAX。它结合了行存储和列存储的优点：首先，RCFile 保证同一行的数据位于同一节点，因此元组重构的开销很低；其次，像列存储一样，RCFile 能够利用列维度的数据压缩，并且能跳过不必要的列读取。图 2.1-3 是一个 HDFS 块内 RCFile 方式存储的例子。

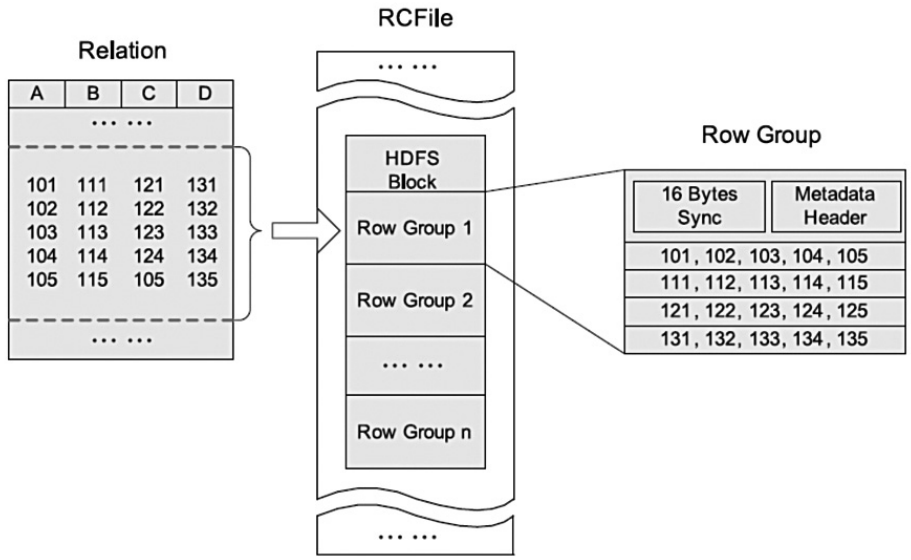


图 2.1-3 HDFS 块内 RCFile 方式存储的例子

- **数据格式**

RCFile 在 HDFS 分布式文件系统之上设计并实现，如图 2.1-3 所示，RCFile 按照下面的数据格式来存储一张表。

RCFile 基于 HDFS 架构，表格占用多个 HDFS 块。

每个 HDFS 块中，RCFile 以行组为基本单位来组织记录。也就是说，存储在一个 HDFS

块中的所有记录被划分为多个行组。对于一张表，所有行组大小都相同。一个 HDFS 块会有一个或多个行组。

一个行组包括三个部分。第一部分是行组头部的同步标识，主要用于分隔 HDFS 块中的两个连续行组；第二部分是行组的元数据头部，用于存储行组单元的信息，包括行组中的记录数、每个列的字节数、列中每个域的字节数；第三部分是表格数据段，即实际的列存储数据。在该部分中，同一列的所有域顺序存储。从图 2.1-3 可以看出，首先存储了列 A 的所有域，然后存储列 B 的所有域等。

- 压缩方式

RCFile 的每个行组中，元数据头部和表格数据段分别进行压缩。

对于所有元数据头部，RCFile 使用 **RLE**（Run Length Encoding）算法来压缩数据。由于同一列中所有域的长度值都顺序存储在该部分，RLE 算法能够找到重复值的长序列，尤其对于固定的域长度。

表格数据段不会作为整个单元来压缩；相反每个列被独立压缩，使用 **Gzip** 压缩算法。RCFile 使用重量级的 Gzip 压缩算法，是为了获得较好的压缩比，而不使用 RLE 算法的原因在于此时列数据非排序。此外，由于 **Lazy** 压缩策略，当处理一个行组时，RCFile 不需要解压所有列。因此，相对较高的 Gzip 解压开销可以减少。

尽管 RCFile 对表格数据的所有列使用同样的压缩算法，不过如果使用不同的算法来压缩不同列或许效果会更好。RCFile 将来的工作之一可能就是根据每列的数据类型和数据分布来自适应选择最好的压缩算法。

- 数据追加

RCFile 不支持任意方式的数据写操作，仅提供一种追加接口，这是因为底层的 HDFS 当前仅仅支持数据追加写文件尾部。数据追加方法描述如下。

RCFile 为每列创建并维护一个内存 column holder，当记录追加时，所有域被分发，每个域追加到其对应的 column holder。此外，RCFile 在元数据头部中记录每个域对应的元数据。

RCFile 提供两个参数来控制在刷写到磁盘之前，内存中缓存多少个记录。一个参数是记录数的限制，另一个是内存缓存的大小限制。

RCFile 首先压缩元数据头部并写到磁盘，然后分别压缩每个 column holder，并将压缩后的 column holder 刷写到底层文件系统中的一个行组中。

- 数据读取和 Lazy 解压

在 MapReduce 框架中，mapper 将顺序处理 HDFS 块中的每个行组。当处理一个行组时，RCFile 无需全部读取行组的全部内容到内存。

相反，它仅仅读元数据头部和给定查询需要的列。因此，它可以跳过不必要的列以获得列存储的 I/O 优势。例如，表 tbl(c1, c2, c3, c4)有 4 个列，做一次查询“SELECT c1 FROM tbl WHERE c4 = 1”，对每个行组，RCFile 仅仅读取 c1 和 c4 列的内容。在元数据头部和需要的列数据加载到内存中后，它们需要解压。元数据头部总会解压并在内存中维 护直到 RCFile 处理下一个行组。然而，RCFile 不会解压所有加载的列，相反，它使用一种 Lazy 解压技术。Lazy 解压意味着列将不会在内存解压，直到 RCFile 决定列中数据真正对查询执行有用。由于查询使用各种 WHERE 条件，Lazy 解压非常有 用。如果一个 WHERE 条件不能被行组中的所有记录满足，那么 RCFile 将不会解压 WHERE 条件中不满足的列。例如，在上述查询中，所有行组中的列 c4 都解压了。然而，对于一个行组，如果列 c4 中没有值为 1 的域，那么就无需解压列 c1。

- 行组大小

I/O 性能是 RCFile 关注的重点，因此 RCFile 需要行组够大并且大小可变。行组大小和

下面几个因素相关。

行组大的话，数据压缩效率会比行组小时更有效。根据对 Facebook 日常应用的观察，当行组大小达到一个阈值后，增加行组大小并不能进一步增加 Gzip 算法下的压缩比。

行组变大能够提升数据压缩效率并减少存储量。因此，如果对缩减存储空间方面有强烈需求，则不建议选择使用小行组。需要注意的是，当行组的大小超过 4MB，数据的压缩比将趋于一致。

尽管行组变大有助于减少表格的存储规模，但是可能会损害数据的读性能，因为这样减少了 Lazy 解压带来的性能提升。而且行组变大会占用更多的内存， 这会影响并发执行的其他 MapReduce 作业。考虑到存储空间和查询效率两个方面，Facebook 选择 4MB 作为默认的行组大小，当然也允许用户自行选择参数进行配置。

2.2 Hive的数据存储

Hive 的**存储**是**建立**在 **Hadoop** **文件系统**之上的。**Hive** **本身**没有**专门**的**数据存储格式**，也**不能**为数据**建立索引**，用户可以非常**自由**地**组织** **Hive** 中的**表**，**只需要**在**创建表**的时候告诉 **Hive** 数据中的**列分隔符**和**行分隔符**就可以解析数据了。

Hive 中主要包括**四类数据模型**：**表**（Table）、**外部表**（External Table）、**分区**（Partition）和**桶**（Bucket）。

- **表（Table）**

Hive 中的**表**和**数据库**中的**表**在概念上是**类似**的，**每个表**在 **Hive** 中**都有一个对应**的**存储目录**。

例如，一个表 htable 在 HDFS 中的路径为“/datawarehouse/htable”，其中“/datawarehouse”是 **hive-site.xml** 配置文件中由“${hive.metastore.warehouse.dir}”指定的数据仓库的目录，所有的表数据（**除了外部表**）都保存在这个目录中。

- **分区（Partition）**

Hive 中**每个分区**都**对应**数据库中**相应分区列**的**一个索引**，但是**分区**的**组织方式**和**传统关系型数据库不同**。在 **Hive** 中，表中的一个**分区**对应表下的一个**目录**，所有分区的**数据**都**存储**在对应的**目录**中。

例如，htable 表中包含的 ds 和 city 两个分区，分别对应两个目录。

对应于 ds=20100301，city=Beijing 的 HDFS 子目录为：

“/datawarehouse/htable/ ds=20100301/city=**Beijing**”

对应与 ds=20100301，city=Shanghai 的 HDFS 子目录为：

“/datawarehouse/htable/ ds=20100301/city=**Shanghai**”

- **桶（Bucket）**

桶对于**指定列**进行**哈希**（hash）计算时，根据**哈希值切分数据**，**每个桶对应**一个**文件。**

例如，将属性列 user 列分散到 32 个桶中，首先要对 user 列的值进行 hash 计算。

对应哈希值为 0 的桶写 HDFS 的目录为：

“/datawarehouse/htable/ ds=20100301/city=Beijing/part-**00000**”

对应哈希值为 10 的 HDFS 目录为：

“/datawarehouse/htable/ ds=20100301/city=Beijing/part-**00010**”

后面的目录依此类推。

- **外部表（External Table）**

外部表指向已经在 **HDFS** 中存在的数据，也可以创建分区。它和表在元数据的组织上是相同的，也可以创建分区。它和表在元数据的组织上是相同的，而实际数据的存储则存在较大差异，主要表现在一下两点上。

1）创建表的操作（Create Table）包含两个步骤：表创建过程和数据加载步骤（这两个过程可以在同一语句中完成）。在数据加载过程中，实际数据会移动到数据仓库目录中。之后的数据访问将会直接在数据仓库目录中完成。删除表时，表中的数据和元数据将会被同时删除。

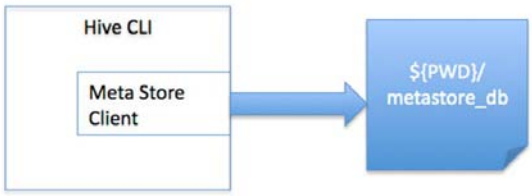
2）外部表的创建只有一个步骤，加载数据和创建表同时完成，实际数据存储在创建语句 LOCATION 指定的 HDFS 路径中，并不会移动到数据仓库目录中。如果删除一个外部表，仅会删除元数据，表中数据不会被删除。

2.3 Hive的元数据存储

由于 **Hive** 的元数据可能要面临不断的更新、修改和读取，所以它显然不适合使用 Hadoop 文件系统进行存储。目前 **Hive** 将元数据存储存储在 **RDBMS** 中，比如 MySQL、Derby 中，Hive 有三种模式可以连接到 Derby 数据库：

- **Single User Mode**

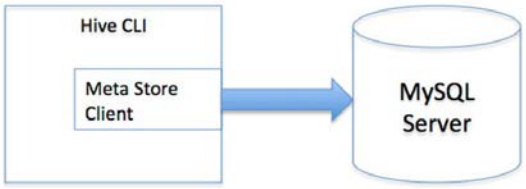
此模式连接到一个 In-memory（内存）数据库 **Derby**，一般用于单元测试。



Parameter	Description	Example
javax.jdo.option.ConnectionURL	JDBC connection URL along with database name containing metadata	jdbc:derby:;databaseName=metastore_db;create=true
javax.jdo.option.ConnectionDriverName	JDBC driver name. Embedded Derby for Single user mode.	org.apache.derby.jdbc.EmbeddedDriver
javax.jdo.option.ConnectionUserName	User name for Derby database	APP
javax.jdo.option.ConnectionPassword	Password	mine

- **Multi User Mode**

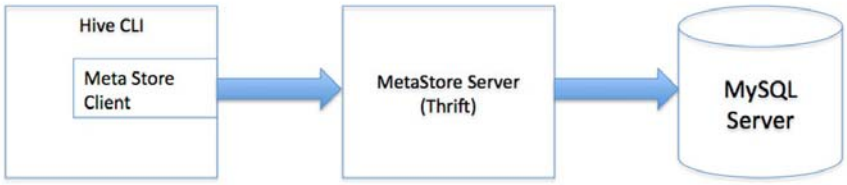
通过网络连接到一个数据库中，是最常用使用的模式。



Parameter	Description	Example
javax.jdo.option.ConnectionURL	JDBC connection URL along with database name containing metadata	jdbc:mysql://<host name>/<database name>?createDatabaseIfNotExist=true
javax.jdo.option.ConnectionDriverName	Any JDO supported JDBC driver.	com.mysql.jdbc.Driver
javax.jdo.option.ConnectionUserName	User name	
javax.jdo.option.ConnectionPassword	Password	

- **Remote Server Mode**

用于“**非 Java 客户端**”访问**元数据库**，在**服务器**启动一个 **MetaStoreServer**，客户端利用 Thrift 协议通过 MetaStoreServer 访问元数据库。



- Server Configuration same as multi user mode client config (prev slide). To run server

```
$JAVA_HOME/bin/java -Xmx1024m -Dlog4j.configuration=file://$HIVE_HOME/conf/hms-log4j.properties
    -Djava.library.path=$HADOOP_HOME/lib/native/Linux-amd64-64/ -cp $CLASSPATH
    org.apache.hadoop.hive.metastore.HiveMetaStore
```

- Client Configuration

Parameter	Description	Example
hive.metastore.uris	Location of the metastore server	thrift://<host_name>:9083
hive.metastore.local		false

3、Hive分布式安装

Hive 是一个**客户端**工具，需要在哪台机器上运行就将其安装在哪台机器上，根据“元数据存储（Metastore）”的位置可以将 Hive 的运行模式分为三种：内嵌模式、本地模式、远程模式。下面分别介绍三种模式的安装过程。

3.1 先决条件

- **Hive 的稳定版本**

目前的稳定版本是“**hive-0.8.1**”，下面是它支持的 Hadoop 以及修正的 Bug 信息。

5 February, 2012: release 0.8.1 available
This release is the latest release of Hive and it works with **Hadoop 0.20.1** and **0.20.2**
You can look at the complete JIRA change log for this release.
[HIVE-2616] - Passing user identity from metastore client to server in non-secure mode
[HIVE-2629] - Make a single Hive binary work with both 0.20.x and 0.23.0
[HIVE-2631] - **Make Hive work with Hadoop 1.0.0**
[HIVE-2868] - Insert into table wipes out table content
[HIVE-2893] - Hive MetaStore is not changing from Derby to MySQL

从上面得知目前 Hive 支持的 Hadoop 平台有：hadoop0.20.1、hadoop0.20.2 和 hadoop1.0.0。

下载地址：http://www.apache.org/dyn/closer.cgi/hive/

- **元数据库**

Hive 的默认元数据库是内嵌的 **Derby**，但只能允许一个会话连接，如果要支持多用户多会话，则需要一个独立的元数据库，目前比较流行的是使用 **MySQL**。

- **Hadoop 安装**

必选与选用的 Hive 版本兼容，并设置了 HADOOP_HOME 环境变量，本实验采用的 Hadoop 版本是目前的稳定版 hadoop-1.0.0，与 hive-0.8.1 兼容良好。Hadoop 的安装及配置，可以参考“**Hadoop 集群_第 5 期_Hadoop 安装配置**”。

3.2 集群环境

下面为当前 Hadoop 集群的环境情况。

- **Java** 版本：jdk-6u31-linux-i586
- **Linux** 版本：CentOS-6.0-i386
- **Hive** 版本：hbase-0.8.1
- **MySQL** 版本：MySQL-5.5.21-1.linux2.6.i386
- **Hadoop** 版本：hadoop-1.0.0
- **Hadoop** 集群：

表 3.2-1 Hadoop 集群信息

机器名称	IP 地址	守护进程
Master.Hadoop	192.168.1.2	NameNode、SecondaryNameNode、JobTracker
Salve1.Hadoop	192.168.1.3	DataNode、TaskTracker
Salve2.Hadoop	192.168.1.4	DataNode、TaskTracker
Salve3.Hadoop	192.168.1.5	DataNode、TaskTracker

下面所示为即将安装的 **Hive** 数据仓库的安装部署。

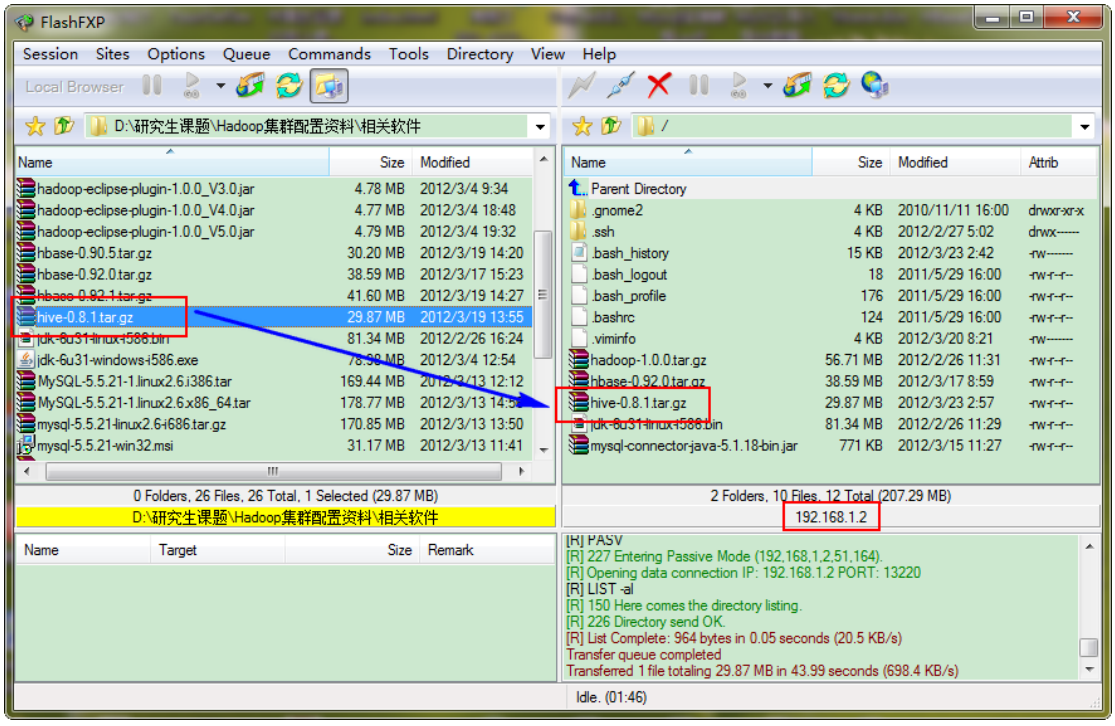
表 3.2-2 Hadoop 安装部署

机器名称	IP 地址	安装程序
Master.Hadoop	192.168.1.2	Hive、MySQL Client
Salve1.Hadoop	192.168.1.3	MySQL Client、MySQL Server

3.2 内嵌模式安装

第一步：FTP 上传 Hive 安装文件。

用“FlashFXP”把 Hive 安装文件上传到“Master.Hadoop”机器上。



用 SecureCRT 进行查看结果如下：

```
[hadoop@Master ~]$ pwd
/home/hadoop
[hadoop@Master ~]$ ll
总用量 212252
-rw-r--r--. 1 hadoop hadoop 59468784  2月 27 03:31 hadoop-1.0.0.tar.gz
-rw-r--r--. 1 hadoop hadoop 40461930  3月 18 00:59 hbase-0.92.0.tar.gz
-rw-r--r--. 1 hadoop hadoop 31325840  3月 23 18:57 hive-0.8.1.tar.gz
-rw-r--r--. 1 hadoop hadoop 85292206  2月 27 03:29 jdk-6u31-linux-i586.bin
-rw-r--r--. 1 hadoop hadoop   789885  3月 16 03:27 mysql-connector-java-5.1.18-bin.jar
[hadoop@Master ~]$
```

第二步：安装 Hive 数据仓库

首先切换到“**root**”用户下，我们把 **Hive** 安装在“**/usr**”目录下面。

```
[hadoop@Master ~]$ su -
密码:
[root@Master ~]#
```

然后把“**hive-0.8.1.tar.gz**”复制到“**/usr**”下面。

```
cp　/home/hadoop/hive-0.8.1.tar.gz　/usr
```

```
[root@Master ~]# ll /home/hadoop
总用量 212252
-rw-r--r--. 1 hadoop hadoop 59468784  2月 27 03:31 hadoop-1.0.0.tar.gz
-rw-r--r--. 1 hadoop hadoop 40461930  3月 18 00:59 hbase-0.92.0.tar.gz
-rw-r--r--. 1 hadoop hadoop 31325840  3月 23 18:57 hive-0.8.1.tar.gz
-rw-r--r--. 1 hadoop hadoop 85292206  2月 27 03:29 jdk-6u31-linux-i586.bin
-rw-r--r--. 1 hadoop hadoop   789885  3月 16 03:27 mysql-connector-java-5.1.18-bin.jar
[root@Master ~]# cp /home/hadoop/hive-0.8.1.tar.gz /usr
[root@Master ~]# ll /usr
总用量 30700
dr-xr-xr-x.  2 root   root      24576  2月 24 05:34 bin
drwxr-xr-x.  2 root   root       4096 11月 11 2010 etc
drwxr-xr-x.  2 root   root       4096 11月 11 2010 games
drwxr-xr-x. 16 hadoop hadoop     4096  2月 29 20:50 hadoop
drwxr-xr-x.  9 hadoop hadoop     4096  3月 18 06:26 hbase
-rw-r--r--.  1 root   root   31325840  3月 23 19:08 hive-0.8.1.tar.gz
drwxr-xr-x. 33 root   root       4096  2月 24 04:40 include
drwxr-xr-x.  3 root   root       4096  2月 28 04:06 java
dr-xr-xr-x. 53 root   root      32768  2月 24 05:34 lib
drwxr-xr-x. 14 root   root       4096  2月 24 05:34 libexec
drwxr-xr-x. 11 root   root       4096  2月 24 04:37 local
dr-xr-xr-x.  2 root   root       4096  2月 24 05:34 sbin
drwxr-xr-x. 95 root   root       4096  2月 24 04:41 share
drwxr-xr-x.  4 root   root       4096  2月 24 04:37 src
lrwxrwxrwx.  1 root   root         10  2月 24 04:37 tmp -> ../var/tmp
[root@Master ~]#
```

接着进入“**/usr**”目录下，用下面命令把“**hive-0.8.1.tar.gz**”进行解压，并将其命名为“**hive**”，把该文件夹的权限分配给普通用户 **hadoop**，然后删除“**hive-0.8.1.tar.gz**”安装包。

```
cd　/usr                              #进入“/usr”目录
tar　–zxvf　hbase-0.8.1.tar.gz        #解压“hive-0.8.1.tar.gz”安装包
mv　hive-0.8.1　hive                  #将“hive-0.8.1”文件夹重命名“hive”
chown　–R　hadoop:hadoop　hive        #将文件夹“hive”权限分配给 hadoop 用户
rm　–rf　hive-0.8.1.tar.gz            #删除“hive-0.8.1.tar.gz”安装包
```

进入“**/usr**”目录，然后解压“**hive-0.8.1.tar.gz**”安装包。

```
[root@Master ~]# cd /usr
[root@Master usr]# ll | grep hive*
-rw-r--r--.  1 root   root   31325840  3月 23 19:08 hive-0.8.1.tar.gz
[root@Master usr]# tar -zxvf hive-0.8.1.tar.gz
```

解压之后，然后重命名为“**hive**”。

```
[root@Master usr]# ll
总用量 30704
dr-xr-xr-x.   2 root   root      24576  2月 24 05:34 bin
drwxr-xr-x.   2 root   root       4096 11月 11 2010 etc
drwxr-xr-x.   2 root   root       4096 11月 11 2010 games
drwxr-xr-x.  16 hadoop hadoop     4096  2月 29 20:50 hadoop
drwxr-xr-x.   9 hadoop hadoop     4096  3月 18 06:06 hbase
drwxr-xr-x.   9 root   root       4096  3月 23 19:17 hive-0.8.1
-rw-r--r--.   1 root   root   31325840  3月 23 19:08 hive-0.8.1.tar.gz
drwxr-xr-x.  33 root   root       4096  2月 24 04:40 include
drwxr-xr-x.   3 root   root       4096  2月 28 04:06 java
dr-xr-xr-x.  53 root   root      32768  2月 24 05:34 lib
drwxr-xr-x.  14 root   root       4096  2月 24 05:34 libexec
drwxr-xr-x.  11 root   root       4096  2月 24 04:37 local
dr-xr-xr-x.   2 root   root       4096  2月 24 05:34 sbin
drwxr-xr-x.  95 root   root       4096  2月 24 04:41 share
drwxr-xr-x.   4 root   root       4096  2月 24 04:37 src
lrwxrwxrwx.   1 root   root         10  2月 24 04:37 tmp -> ../var/tmp
[root@Master usr]# mv hive-0.8.1 hive
[root@Master usr]# ll
总用量 30704
dr-xr-xr-x.   2 root   root      24576  2月 24 05:34 bin
drwxr-xr-x.   2 root   root       4096 11月 11 2010 etc
drwxr-xr-x.   2 root   root       4096 11月 11 2010 games
drwxr-xr-x.  16 hadoop hadoop     4096  2月 29 20:50 hadoop
drwxr-xr-x.   9 hadoop hadoop     4096  3月 18 06:06 hbase
drwxr-xr-x.   9 root   root       4096  3月 23 19:17 hive
-rw-r--r--.   1 root   root   31325840  3月 23 19:08 hive-0.8.1.tar.gz
drwxr-xr-x.  33 root   root       4096  2月 24 04:40 include
drwxr-xr-x.   3 root   root       4096  2月 28 04:06 java
dr-xr-xr-x.  53 root   root      32768  2月 24 05:34 lib
drwxr-xr-x.  14 root   root       4096  2月 24 05:34 libexec
```

把“**/usr/hive**”的**权限**分别给 **hadoop** 用户。（**非常重要**）

```
[root@Master usr]# chown -R hadoop:hadoop hive
[root@Master usr]# ll
总用量 30704
dr-xr-xr-x.   2 root   root      24576  2月 24 05:34 bin
drwxr-xr-x.   2 root   root       4096 11月 11 2010 etc
drwxr-xr-x.   2 root   root       4096 11月 11 2010 games
drwxr-xr-x.  16 hadoop hadoop     4096  2月 29 20:50 hadoop
drwxr-xr-x.   9 hadoop hadoop     4096  3月 18 06:06 hbase
drwxr-xr-x.   9 hadoop hadoop     4096  3月 23 19:17 hive
-rw-r--r--.   1 root   root   31325840  3月 23 19:08 hive-0.8.1.tar.gz
drwxr-xr-x.  33 root   root       4096  2月 24 04:40 include
drwxr-xr-x.   3 root   root       4096  2月 28 04:06 java
dr-xr-xr-x.  53 root   root      32768  2月 24 05:34 lib
drwxr-xr-x.  14 root   root       4096  2月 24 05:34 libexec
drwxr-xr-x.  11 root   root       4096  2月 24 04:37 local
dr-xr-xr-x.   2 root   root       4096  2月 24 05:34 sbin
drwxr-xr-x.  95 root   root       4096  2月 24 04:41 share
drwxr-xr-x.   4 root   root       4096  2月 24 04:37 src
lrwxrwxrwx.   1 root   root         10  2月 24 04:37 tmp -> ../var/tmp
[root@Master usr]#
```

删除“**hive-0.8.1.tar.gz**”安装包。

```
[root@Master usr]# rm -rf hive-0.8.1.tar.gz
[root@Master usr]#
```

第三步：添加 **Hive** 环境变量

在“/etc/profile”文件尾部添加以下内容，并使其有效（source /etc/profile）：

```
# set hive environment
export HIVE_HOME=/usr/hive
export PATH=$PATH:$HIVE_HOME/bin
export CLASSPATH=$CLASSPATH:$HIVE_HOME/lib
```

```
# set java environment
export JAVA_HOME=/usr/java/jdk1.6.0_31
export CLASSPATH=.:$CLASSPATH:$JAVA_HOME/lib:$JAVA_HOME/jre/lib
export PATH=$PATH:$JAVA_HOME/bin:$JAVA_HOME/jre/bin

# set hadoop environment
export HADOOP_HOME=/usr/hadoop
export HADOOP_HOME_WARN_SUPPRESS=1
export PATH=$PATH:$HADOOP_HOME/bin

# set hbase environment
export HBASE_HOME=/usr/hbase
export PATH=$PATH:$HBASE_HOME/bin

# set hive environment
export HIVE_HOME=/usr/hive
export PATH=$PATH:$HIVE_HOME/bin
export CLASSPATH=$CLASSPATH:$HIVE_HOME/lib
"/etc/profile" 87L, 1980C 已写入
[root@Master ~]#
```

备注：在 **PATH** 中保证有 **HADOOP_HOME** 的配置，便于 Hive 寻找 **fs.default.name** 和 **job.mapred.tracker** 属性，当然也可以在 **Hive 根目录**下的 **conf** 子目录中新建 **hive-site.xml**，在该文件中加入相关属性和值。

第四步：配置 Hive 配置文件

1）配置 hive-conf.sh

在“**/usr/hive/bin**”目录下，“**hive-conf.sh**”，然后在里面添加下面内容。

```
# set hive environment
export    HADOOP_HOME=/usr/hadoop
export    HIVE_HOME=/usr/hive
```

```
[root@Master bin]# pwd
/usr/hive/bin
[root@Master bin]# ll
总用量 16
drwxr-xr-x. 3 hadoop hadoop 4096  1月 26 08:16 ext
-rwxr-xr-x. 1 hadoop hadoop 5667  1月 26 08:16 hive
-rwxr-xr-x. 1 hadoop hadoop 1926  3月 23 21:38 hive-config.sh
[root@Master bin]#
```

进入“**/usr/hive/bin**”目录中，添加结果如下：

```
# Default to use 256MB
export HADOOP_HEAPSIZE=${HADOOP_HEAPSIZE:-256}

export HIVE_HOME=/usr/hive
export HADOOP_HOME=/usr/hadoop
"hive-config.sh" 71L, 1926C 已写入
[root@Master bin]#
```

2）配置 hive-default.xml 和 hive-site.xml

在“**/usr/hive/conf**”目录下，**没有**这两个文件，**只有**一个“**hive-default.xml.template**”，所以我们要**复制两**个“**hive-default.xml.template**”，并分别**命名**为“**hive-default.xml**”和“**hive-site.xml**”。因为我们当前是 **root** 用户，所以还要把两个的文件的授权给 **hadoop** 用户。

```
[root@Master conf]# cp hive-default.xml.template hive-default.xml
[root@Master conf]# cp hive-default.xml.template hive-site.xml
[root@Master conf]# ll | grep "xml$"
-rw-r--r--. 1 root   root   46817  3月 23 22:31 hive-default.xml
-rw-r--r--. 1 root   root   46817  3月 23 22:31 hive-site.xml
[root@Master conf]# chown -R hadoop:hadoop hive-default.xml
[root@Master conf]# chown -R hadoop:hadoop hive-site.xml
[root@Master conf]# ll | grep "xml$"
-rw-r--r--. 1 hadoop hadoop 46817  3月 23 22:31 hive-default.xml
-rw-r--r--. 1 hadoop hadoop 46817  3月 23 22:31 hive-site.xml
[root@Master conf]#
```

备注：“hive-default.xml”用于保留默认配置，“hive-site.xml”用于个性化配置，可覆盖默认配置。

第五步：启动 Hive

此时切换用户至 **hadoop** 用户，在命令行输入“hive”命令进行测试。

第一次运行出现如下**错误**：

```
Exception in thread "main" java.lang.NoClassDefFoundError:org/apache/hadoop/hive/conf/HiveConf
```

原因是我们在前面设置 **HBase** 时，再 **Hadoop** 的“**/usr/hadoop/conf/hadoop-env.sh**”添加“**HADOOP_CLASSPATH**”变量配置有问题。

错误结果：

```
[hadoop@Master ~]$ hive
Exception in thread "main" java.lang.NoClassDefFoundError: org/apache/hadoop/hive/conf/HiveConf
        at java.lang.Class.forName0(Native Method)
        at java.lang.Class.forName(Class.java:247)
        at org.apache.hadoop.util.RunJar.main(RunJar.java:149)
Caused by: java.lang.ClassNotFoundException: org.apache.hadoop.hive.conf.HiveConf
        at java.net.URLClassLoader$1.run(URLClassLoader.java:202)
        at java.security.AccessController.doPrivileged(Native Method)
        at java.net.URLClassLoader.findClass(URLClassLoader.java:190)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:306)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:247)
        ... 3 more
[hadoop@Master ~]$
```

“**hadoop-env.sh**”文件修改**前**

```
export JAVA_HOME=/usr/java/jdk1.6.0_31

# set hbase environment
export HBASE_HOME=/usr/hbase
export HADOOP_CLASSPATH=$HBASE_HOME/hbase-0.92.0.jar:$HBASE_HOME/hbase-0.92.0-tests.jar
export HADOOP_CLASSPATH=$HADOOP_CLASSPATH:$HBASE_HOME/conf:$HBASE_HOME/lib/zookeeper-3.4.2.jar
"hadoop-env.sh" 62L, 2537C 已写入
```

“**hadoop-env.sh**”文件修改**后**

```
# set java environment
export JAVA_HOME=/usr/java/jdk1.6.0_31

# set hbase environment
export HBASE_HOME=/usr/hbase
export HADOOP_CLASSPATH=$HADOOP_CLASSPATH:$HBASE_HOME/hbase-0.92.0.jar:$HBASE_HOME/hbase-0.92.0-tests.jar
export HADOOP_CLASSPATH=$HADOOP_CLASSPATH:$HBASE_HOME/conf:$HBASE_HOME/lib/zookeeper-3.4.2.jar
"hadoop-env.sh" 62L, 2555C 已写入
[hadoop@Master conf]$
```

第二次运行出现如下**警告**：

```
[hadoop@Master ~]$ hive
WARNING: org.apache.hadoop.metrics.jvm.EventCounter is deprecated. Please use org.apache.
hadoop.log.metrics.EventCounter in all the log4j.properties files.
Logging initialized using configuration in jar:file:/usr/hive/lib/hive-common-0.8.1.jar!/
hive-log4j.properties
Hive history file=/tmp/hadoop/hive_job_log_hadoop_201203232252_1046231841.txt
hive>
```

解决的办法就是在 hive-log4j.properties 中将 log4j.appender.EventCounter 的值修改为 org.apache.hadoop.log.metrics.EventCounter，这样就不会报。该文件在“**/usr/hive/conf**”下面。

第三次运行表现**正常**：

```
"hive-log4j.properties" 67L, 2043C 已写入
[hadoop@Master conf]$ hive
Logging initialized using configuration in file:/usr/hive/conf/hive-log4j.properties
Hive history file=/tmp/hadoop/hive_job_log_hadoop_201203232258_743842805.txt
hive>
```

3.3 独立模式安装

我们前面按照“内嵌模式安装”，这时元数据保存在内嵌的 Derby 数据库中，只能允许一个会话连接，只适合简单的测试。为了支持多用户多会话，则需要一个独立的元数据库，我们使用 MySQL 作为元数据库，Hive 内部对 MySQL 提供了很好的支持，配置一个独立的元数据库需要增加以下几步骤：

第一步：安装 MySQL 服务器端和 MySQL 客户端，并启动 mysql 服务。

我们在“Hadoop 集群_第 10 期_MySQL 关系数据库”中已经对 MySQL 在 Linux 下安装进行了详细讲解，不会的可以参考一下。按照前面 3.2 小节的 Hive 仓库规划，我们将把 MySQL 安装在 Slave1.Hadoop 机器上。

第二步：为 Hive 建立相应的 MySQL 账户，并赋予足够的权限，执行命令如下：

```
CREATE USER ‘hive’ IDENTIFIED BY ‘hadoop’
GRANT ALL PRIVILEGES ON *.* TO ‘hive’@’%’ WITH GRANT OPTION;
```

```
[hadoop@Slave1 ~]$ mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 11
Server version: 5.5.21-log MySQL Community Server (GPL)

Copyright (c) 2000, 2011, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> create user 'hive' identified by 'hadoop';
Query OK, 0 rows affected (0.00 sec)

mysql> grant all privileges on *.* to 'hive'@'%' with grant option;
Query OK, 0 rows affected (0.00 sec)

mysql> flush privileges;
Query OK, 0 rows affected (0.00 sec)

mysql>
```

第三步：建立 Hive 专用的元数据库，记得创建时用刚才创建的“**hive**”账号登陆。

```
mysql> create database hive;
Query OK, 1 row affected (0.01 sec)

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| hive               |
| mysql              |
| performance_schema |
| test               |
+--------------------+
5 rows in set (0.00 sec)

mysql>
```

第四步：在“Master.Hadoop”上面安装 MySQL 客户端。

先把 MySQL 的客户端上传到“Master.Hadoop”上，然后在进行安装，记得把机器旧版本的 MySQL 软件卸载掉。

```
[root@Master hadoop]# ll
总用量 228852
-rw-rw-r--. 1 hadoop hadoop      343  3月 23 22:16 derby.log
-rw-r--r--. 1 hadoop hadoop 59468784  2月 27 03:31 hadoop-1.0.0.tar.gz
-rw-r--r--. 1 hadoop hadoop 40461930  3月 18 00:59 hbase-0.92.0.tar.gz
-rw-r--r--. 1 hadoop hadoop 31325840  3月 23 18:57 hive-0.8.1.tar.gz
-rw-r--r--. 1 hadoop hadoop 85292206  2月 27 03:29 jdk-6u31-linux-i586.bin
drwxrwxr-x. 5 hadoop hadoop     4096  3月 23 22:16 metastore_db
-rw-r--r--. 1 hadoop hadoop 16988103  3月 24 03:57 MySQL-client-5.5.21-1.linux2.6.i386.rpm
-rw-r--r--. 1 hadoop hadoop   789885  3月 16 03:27 mysql-connector-java-5.1.18-bin.jar
[root@Master hadoop]# rpm -ivh MySQL-client-5.5.21-1.linux2.6.i386.rpm
Preparing...                ########################################### [100%]
   1:MySQL-client           ########################################### [100%]
[root@Master hadoop]#
```

第五步：在 Hive 的 **conf** 目录下的文件“**hive-site.xml**”中增加如下配置：

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>
    <property>
        <name>hive.metastore.local</name>
        <value>true</value>
    </property>
    <property>
        <name>javax.jdo.option.ConnectionURL</name>
        <value>jdbc:mysql://192.168.1.3:3306/hive? characterEncoding=UTF-8</value>
    </property>
    <property>
        <name>javax.jdo.option.ConnectionDriverName</name>
        <value>com.mysql.jdbc.Driver</value>
    </property>
    <property>
        <name>javax.jdo.option.ConnectionUserName</name>
        <value>hive</value>
    </property>
    <property>
        <name>javax.jdo.option.ConnectionPassword</name>
        <value>hadoop</value>
    </property>
</configuration>
```

从前面我们知道我们的“**hive-site.xml**”是一个“hive-default.xml.**template**”的一个拷贝，里面的配置参数非常之多，但是并不是我们都需要的，我们知道，Hive 系统会加载两个配置文件一个默认配置文件“hive-**default**.xml”，另一个就是用户自定义文件“hive-**site**.xml”。

当“hive-site.xml”中的配置参数的值与“hive-default.xml”文件中不一致时，以用户自定义的为准。所以我们就把我们不需要的参数都删除掉，只留下上面所示的内容。

```
[hadoop@Master conf]$ vim hive-site.xml

<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>
        <property>
          <name>hive.metastore.local</name>
          <value>true</value>
        </property>
        <property>
          <name>javax.jdo.option.ConnectionURL</name>
          <value>jdbc:mysql://192.168.1.3:3306/hive?characterEncoding=UTF-8</value>
        </property>
        <property>
          <name>javax.jdo.option.ConnectionDriverName</name>
          <value>com.mysql.jdbc.Driver</value>
        </property>
        <property>
          <name>javax.jdo.option.ConnectionUserName</name>
          <value>hive</value>
        </property>
        <property>
          <name>javax.jdo.option.ConnectionPassword</name>
          <value>hadoop</value>
        </property>
</configuration>
```

备注：其实修改这里的配置文件如果在 Linux 下面进行则非常麻烦，尽然我们都建立了 FTP 了，为何不用，所以把 Master.Hadoop 上面的这个配置文件下载下来，按照要求进行修改，这样的文件在 Windows 进行操作是非常方便的，弄好之后在上传上去，覆盖原来的即可。

第六步：把 MySQL 的 JDBC 驱动包复制到 Hive 的 lib 目录下。

JDBC 驱动包的版本：mysql-connector-java-**5.1.18**-bin.jar

细心的你，应该发现我们在上传 MySQL 客户端时，就已经把这个 Jar 上传上去了，所以现在就是通过下面命令把这个 Jar 复制到“**/usr/hive/lib**”下面去。

```
[hadoop@Master ~]$ ll
总用量 228852
-rw-rw-r--. 1 hadoop hadoop       343  3月 23 22:16 derby.log
-rw-r--r--. 1 hadoop hadoop 59468784  2月 27 03:31 hadoop-1.0.0.tar.gz
-rw-r--r--. 1 hadoop hadoop 40461930  3月 18 00:59 hbase-0.92.0.tar.gz
-rw-r--r--. 1 hadoop hadoop 31325840  3月 23 18:57 hive-0.8.1.tar.gz
-rw-r--r--. 1 hadoop hadoop 85292206  2月 27 03:29 jdk-6u31-linux-i586.bin
drwxrwxr-x. 5 hadoop hadoop     4096  3月 23 22:16 metastore_db
-rw-r--r--. 1 hadoop hadoop 16988103  3月 24 03:57 MySQL-client-5.5.21-1.linux2.6.i386.rpm
-rw-r--r--. 1 hadoop hadoop   789885  3月 16 03:27 mysql-connector-java-5.1.18-bin.jar
[hadoop@Master ~]$ cp mysql-connector-java-5.1.18-bin.jar /usr/hive/lib
[hadoop@Master ~]$ ll /usr/hive/lib | grep mysql*
-rw-r--r--. 1 hadoop hadoop  789885  3月 24 04:44 mysql-connector-java-5.1.18-bin.jar
[hadoop@Master ~]$
```

第七步：启动 Hive Shell，执行“show tables;”命令，如果不报错，表明基于独立元数

据库的 Hive 已经安装成功了。

```
[hadoop@Master conf]$ hive
Logging initialized using configuration in file:/usr/hive/conf/hive-log4j.properties
Hive history file=/tmp/hadoop/hive_job_log_hadoop_201203240541_1286421383.txt
hive> show tables;
FAILED: Error in metadata: MetaException(message:Got exception: javax.jdo.JDODataStoreException An exception was th
rown while adding/validating class(es) : Specified key was too long; max key length is 767 bytes
com.mysql.jdbc.exceptions.jdbc4.MySQLSyntaxErrorException: Specified key was too long; max key length is 767 bytes
        at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
```

当 mysql 的字符集设置成 utf8 的时候使用 hive 会有问题，所以当 hive 使用 mysql 作为元数据库的时候 mysql 的字符集要设置成 latin1。

```
[client]
default-character-set=latin1
```

```
# The following options will be passed to all MySQL clients
[client]
#password       = your_password
port            = 3306
#socket         = /var/lib/mysql/mysql.sock
socket          = /mysql_data/mysql/mysql.sock
default-character-set=latin1

# Here follows entries for some specific programs

# The MySQL server
[mysqld]
port            = 3306
socket          = /mysql_data/mysql/mysql.sock
datadir         = /mysql_data/mysql
#character-set-server=utf8
lower_case_table_names=1
skip-external-locking
key_buffer_size = 16M
```

```
mysql> show variables like 'character%';
+--------------------------+-----------------------------+
| Variable_name            | Value                       |
+--------------------------+-----------------------------+
| character_set_client     | latin1                      |
| character_set_connection | latin1                      |
| character_set_database   | latin1                      |
| character_set_filesystem | binary                      |
| character_set_results    | latin1                      |
| character_set_server     | latin1                      |
| character_set_system     | utf8                        |
| character_sets_dir       | /usr/share/mysql/charsets/ |
+--------------------------+-----------------------------+
8 rows in set (0.00 sec)
```

为了保存那些 utf8 的中文，要将 mysql 中存储注释的那几个字段的字符集单独修改为 utf8。

- **修改字段注释字符集**

```
alter table COLUMNS modify column COMMENT varchar(256) character set utf8;
```

- **修改表注释字符集**

```
alter table TABL_PARAMS modify column PARAM_VALUE varchar(4000) character set utf8;
```

设置完编码之后，从新执行“show tables;”命令结果如下：

```
[hadoop@Master conf]$ hive
Logging initialized using configuration in file:/usr/hive/conf/hive-log4j.properties
Hive history file=/tmp/hadoop/hive_job_log_hadoop_201203240613_393110033.txt
hive> show tables;
OK
Time taken: 8.116 seconds
hive>
```

第八步：验证 Hive 配置无误，进入 Hive 的 shell 新建表，在 MySQL 的 Hive 数据库中可以看到相应的元数据库信息。

1）在 Hive 上建立数据表

```
CREATE TABLE xp(id INT,name string) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t';
```

```
[hadoop@Master ~]$ hive
Logging initialized using configuration in file:/usr/hive/conf/hive-log4j.properties
Hive history file=/tmp/hadoop/hive_job_log_hadoop_201203250353_1740213339.txt
hive> CREATE TABLE xp(id INT,name string) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t';
OK
Time taken: 2.784 seconds
hive> show tables;
OK
xp
Time taken: 0.114 seconds
hive>
```

备注：当出现如下错误时，是因为 Hadoop 处于安全模式，退出即可。

```
FAILED: Error in metadata: MetaException(message:Got exception: org.apache.hadoop.ipc.RemoteException org.apache.hadoop.hdfs.server.namenode.
SafeModeException: Cannot create directory /user/hive/warehouse/xp. Name node is in safe mode.
The ratio of reported blocks 0.9811 has not reached the threshold 0.9990. Safe mode will be turned off automatically.
        at org.apache.hadoop.hdfs.server.namenode.FSNamesystem.mkdirsInternal(FSNamesystem.java:2053)
        at org.apache.hadoop.hdfs.server.namenode.FSNamesystem.mkdirs(FSNamesystem.java:2027)
        at org.apache.hadoop.hdfs.server.namenode.NameNode.mkdirs(NameNode.java:817)
```

用下面语句即可解决。

```
[hadoop@Master ~]$ hadoop dfsadmin -safemode leave
Safe mode is OFF
```

2）从 MySQL 数据库上查看元数据信息

```
mysql> use hive;
Database changed
mysql> show tables;
+------------------+
| Tables_in_hive   |
+------------------+
| bucketing_cols   |
| cds              |
| columns_v2       |
| database_params  |
| dbs              |
| partition_keys   |
| sd_params        |
| sds              |
| sequence_table   |
| serde_params     |
| serdes           |
| sort_cols        |
| table_params     |
| tbls             |
+------------------+
14 rows in set (0.00 sec)
```

用到的 SQL 语句：

- use hive; //使用 hive 数据库库
- show tables;//显示 hive 数据库中的数据表
- select * from tbls;//查看 hive 的元数据信息

从作图中我们可以看出里面已经存在很多数据表了。至于是什么，我们现在和不得而知，估计随着研究的深入会了解，Hive 的独立模式也是最常用的模式。下面我在简单的讲解一下远程模式的安装。

```
mysql> select * from tbls;
+--------+-------------+-------+------------------+--------+-----------+--------+----------+---------------+--------------------+--------------------+
| TBL_ID | CREATE_TIME | DB_ID | LAST_ACCESS_TIME | OWNER  | RETENTION | SD_ID  | TBL_NAME | TBL_TYPE      | VIEW_EXPANDED_TEXT | VIEW_ORIGINAL_TEXT |
+--------+-------------+-------+------------------+--------+-----------+--------+----------+---------------+--------------------+--------------------+
|      1 |  1332618821 |     1 |                0 | hadoop |         0 |      1 | xp       | MANAGED_TABLE | NULL               | NULL               |
+--------+-------------+-------+------------------+--------+-----------+--------+----------+---------------+--------------------+--------------------+
1 row in set (0.00 sec)
```

在 TBLS 中可以看到 Hive 表的元数据。

3.4 远程模式安装

远程模式安装是把 **metastore** 配置到远程机器上，可以配置多个。在独立模式的基础上需要在 hive-site.xml 文件中增加的配置项如下：

```
<property>
  <name>hive.metastore.local</name>
  <value>local</value>
</property>
<property>
  <name>hive.metastore.uris</name>
  <value>uri1,uri2,... </value>//可配置多个 uri
  <description>JDBC connect string for a JDBC metastore</description>
</property>
```

参考文献

感谢以下文章的编写作者，没有你们的铺路，我或许会走得很艰难，参考不分先后，贡献同等珍贵。

【1】Hadoop 实战——陆嘉恒——机械工业出版社

【2】实战 Hadoop——刘鹏——电子工业出版社

【3】hive 安装过程

地址：http://blog.csdn.net/gudaoqianfu/article/details/7319287

【4】Hive 安装及问题解决

地址：http://blog.csdn.net/lengzijian/article/details/7042280

【5】hive 配置

地址：http://blog.csdn.net/wf1982/article/details/6641827

【6】hive 的 Specified key was too long; max key length is 767 bytes 问题解决

地址：http://blog.csdn.net/lengzijian/article/details/7045538

【7】hive 元数据存储在 mysql 字符集 utf8 修改

地址：http://blog.csdn.net/tylgoodluck/article/details/7009952

【8】配置 hadoop HIVE 元数据保存在 mysql 中

地址：http://space.itpub.net/?uid-22418990-action-viewspace-itemid-718885

【9】Hive 结构

地址：http://www.tbdata.org/archives/499

【10】HIVE RCFile 高效存储结构

地址：http://blog.csdn.net/wh62592855/article/details/6409680

【11】Hive 安装配置详细

地址：http://yymmiinngg.iteye.com/blog/708230