Hive简介

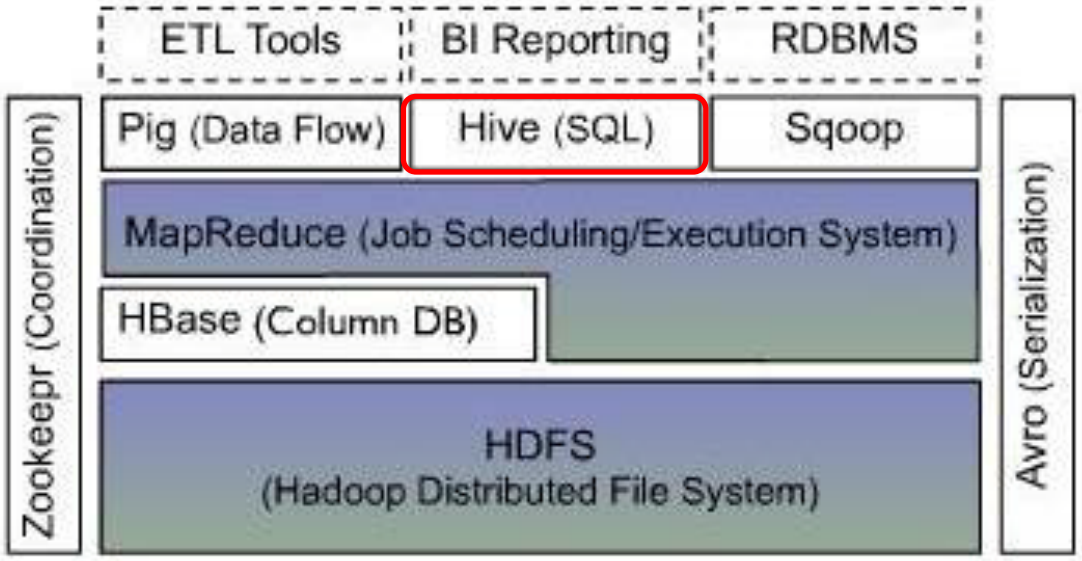
摘要

- Hive基本原理
- Hive基本操作
- Hive程序设计

摘要

- Hive基本原理
- Hive基本操作
- Hive程序设计

Hadoop生态



本次讲义针对的是Hive 2.x版本，3.x版本有较大变化

使用Hadoop进行数据分析

- 通过前面的课程知道，很多分析任务可以通过Hadoop集群进行，即可以通过Hadoop集群将任务分布到数百甚至上千个节点中进行分析，通过并行执行缩短分析的时间
- Hadoop通过MapReduce的并行化方式进行并行处理，能够充分利用数目庞大的服务器节点
- 但是，MapReduce是一个底层的编程接口，对于数据分析人员来说，这个编程接口并不是十分友好，还需要进行大量的编程以及调试工作

在Hadoop上加入数据分析功能

- 显然，为了能够支持一个类似于SQL相关的数据查询语言，Hadoop还需要加入一些额外的模块才能够方便数据分析人员的使用，这些额外的模块包括：
  - 数据查询语言本身的定义与构造，这是与终端用户进行交互的接口，最简单的可以通过命令行接口的方式展开用户与系统的交互
  - 构造数据查询语言的执行引擎，即将上述的查询语言进行编译，并通过分布式的执行引擎完成查询，在Hive中，执行引擎会将查询语言翻译为多个MapReduce的任务序列，交给MapReduce程序去执行
  - 数据查询语言本身需要定义一套数据组织的格式

Hive简介

- Hive可以被认为是一种数据仓库，包括数据的存储以及查询
- Hive包括一个高层语言的执行引擎，类似于SQL的执行引擎
- Hive建立在Hadoop的其它组成部分之上，包括Hive依赖于HDFS进行数据保存，依赖于MapReduce完成查询操作
- Hive最初的开发由Facebook推动，在Facebook内部每天会搜集大量的数据，并需要在这些数据上进行大量分析
- 最初的分析是通过手工的python脚本形式进行
- 数据分析的数量十分巨大，在2006年每天需要分析数十个GB左右的数据，在2007年增长到大约TB的量级，现在数据分析的数量可能是这个数量的10倍

Hive是什么

- Hive是一个基于Apache Hadoop的数据仓库。对于数据存储与处理，Hadoop提供了主要的扩展和容错能力。
- Hive设计的初衷：对于大量的数据，使得数据汇总、查询和分析更加简单。它提供了SQL，允许用户简单地进行查询、汇总和数据分析。
- Hive的SQL给予了用户多种方式来集成自己的功能，然后做定制化的查询，例如用户自定义函数。

Hive优点

- 可扩展性：横向扩展，Hive 可以自由的扩展集群的规模，一般情况下不需要重启服务。
- 延展性：Hive 支持自定义函数，用户可以根据自己的需求来实现自己的函数。
- 良好的容错性：可以保障即使有节点出现问题，SQL 语句仍可完成执行。

Hive缺点

- Hive 不支持记录级别的增删改操作，但是用户可以通过查询生成新表或者将查询结 果导入到文件中（新版本支持记录级别的插入操作）
- Hive 的查询延时很严重，因为 MapReduce Job 的启动过程消耗很长时间，所以不能用在交互查询系统中。
- Hive不是为在线事务处理而设计，所以主要用来做 OLAP（联机分析处理），而 不是 OLTP（联机事务处理）。它最适合用于传统的数据仓库任务。

RDBMS vs Hive

- Hive和关系数据库存储文件的系统不同，Hive使用的是Hadoop的HDFS，关系数据库则是服务器本地的文件系统；
- Hive使用的计算模型是MapReduce，而关系数据库则是自己设计的计算模型；
- 关系数据库都是为实时查询的业务进行设计的，而Hive则是为海量数据做数据挖掘设计的，实时性很差；实时性的区别导致Hive的应用场景和关系数据库有很大的不同；
- Hive很容易扩展自己的存储能力和计算能力，这个是继承Hadoop的，而关系数据库在这个方面要比数据库差很多。

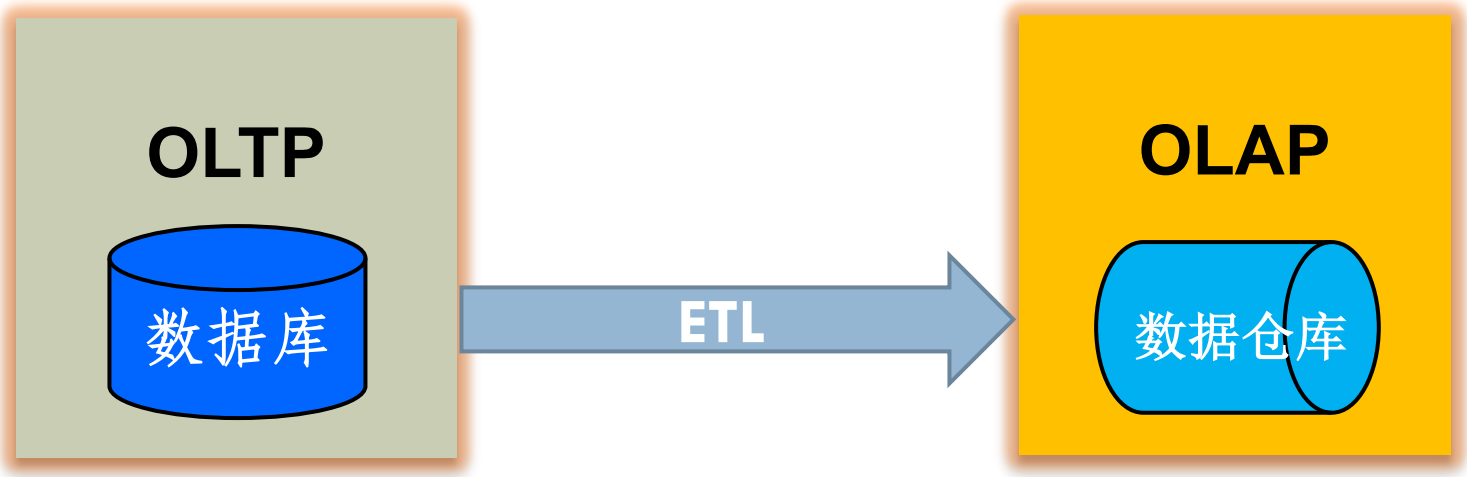
RDBMS vs Hive

对比项	Hive	RDBMS
查询语言	*HQL*	*SQL*
数据存储	*HDFS*	*Raw Device or Local FS*
执行器	*MapReduce*	*Executor*
数据插入	*支持批量导入/单条插入*	*支持单条或者批量导入*
数据操作	*覆盖追加*	*行级更新删除*
处理数据规模	*大*	*小*
执行延迟	*高*	*低*
分区	*支持*	*支持*
索引	*0.8 版本之后加入简单索引*	*支持复杂的索引*
扩展性	*高（好）*	*有限（差）*
数据加载模式	*读时模式（快）*	*写时模式（慢）*
应用场景	*海量数据查询*	*实时查询*

Hive 具有 SQL 数据库的外表，但应用场景完全不同，**Hive 只适合用来做海量离线数据统计分析，也就是数据仓库。**

HBase vs Hive

- HBase和Hive在大数据架构中处在不同位置，HBase主要解决实时数据查询问题，Hive主要解决数据处理和计算问题，一般是配合使用。
  - HBase主要针对的是OLTP应用
  - Hive主要针对的是OLAP应用



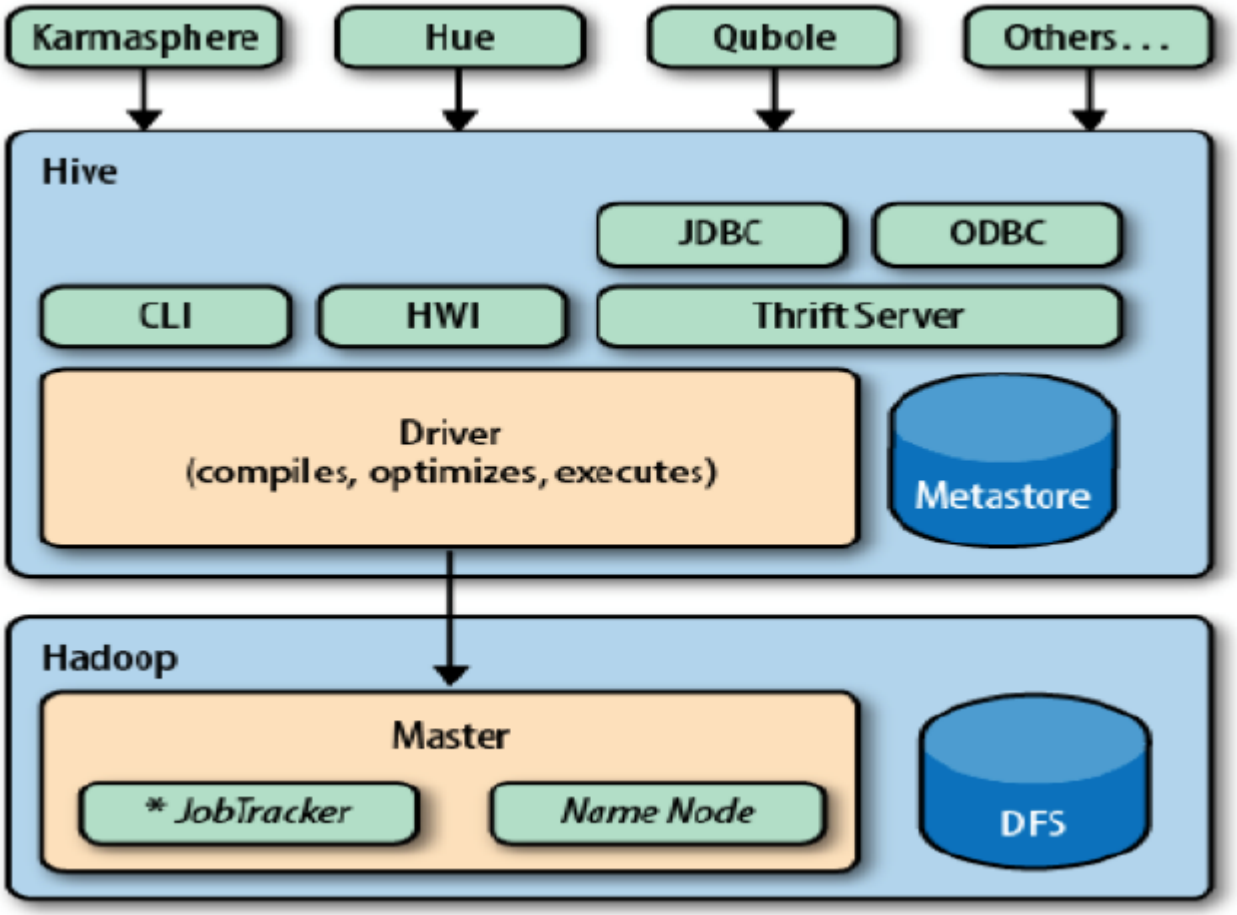
HBase vs Hive

- 区别：
  - HBase： 基于Hadoop的NoSQL数据库，主要适用于海量明细数据（十亿、百亿）的随机实时查询，如日志明细、交易清单、轨迹行为等
  - Hive：基于Hadoop的数据仓库，严格来说，不是数据库，主要是让开发人员能够通过SQL来计算和处理HDFS上的结构化数据，适用于离线的批量数据计算。

Hive的应用范围举例

- 日志分析：在互联网公司中，每天会产生大量的日志数据，分析这些日志数据是每天都需要进行的重要工作；日志分析可以优化系统，可以获知用户行为，也可以获知数据的统计信息
- 数据挖掘：通过对结构化数据的挖掘，能够获得原先使用者没有意识到的信息
- 文档索引：可以对一系列文档进行分析，并形成文档的索引结构，不一定是完整的倒排表，可能是关联信息的索引
- 商业智能信息处理：可以对商业信息进行查询分析，从中可以获得一些智能决策的信息
- 即时查询以及数据验证：数据分析人员可能临时需要验证数据的特性，需要查询引擎迅速进行数据统计分析

Hive的体系结构


Karmasphere
Hue
Qubole
Others...
Hive
JDBC
ODBC
CLI
HWI
Thrift Server
Driver
(compiles, optimizes, executes)
Metastore
Hadoop
Master
* JobTracker
Name Node
DFS

Hive的组成模块(1)

- **Driver组件**：核心组件，整个Hive的核心。该组件包括Complier、Optimizer和Executor，它的作用是将我们写的HQL语句进行解析、编译优化，生成执行计划，然后调用底层的MapReduce计算框架。
  - Compiler：Hive需要一个编译器，将HiveQL语言编译成中间表示，包括对于HiveQL语言的分析，执行计划的生成等工作
  - Optimizer：进行优化
  - Executor：执行引擎，在Driver的驱动下，具体完成执行操作，包括MapReduce执行，或者HDFS操作，或者元数据操作

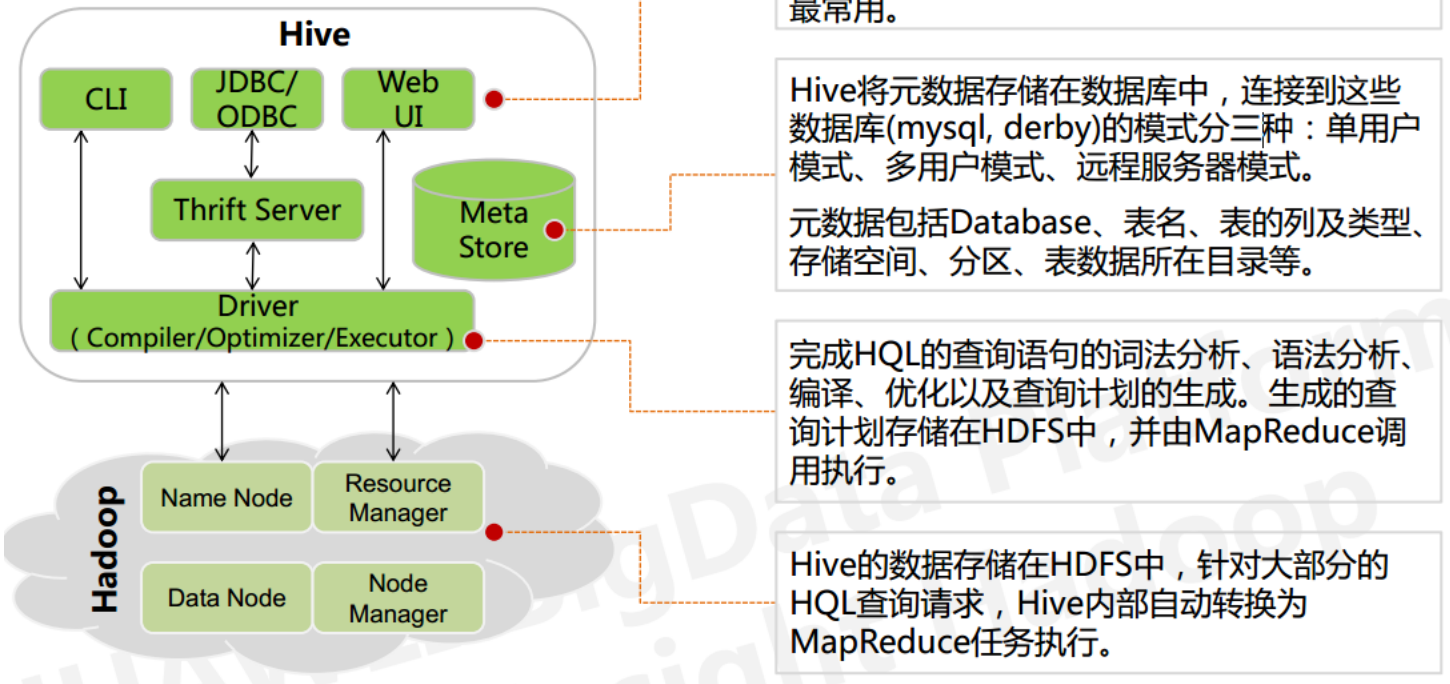
Hive的组成模块(2)

- **Metastore组件**：数据服务组件，用以存储Hive的元数据：存储操作的数据对象的格式信息，在HDFS中的存储位置的信息以及其他的用于数据转换的信息SerDe等。Hive的元数据存储在关系数据库里，Hive支持的关系数据库有derby、mysql。
- **CLI组件**：Command Line Interface，命令行接口。
- **ThriftServers**：提供JDBC和ODBC接入的能力，它用来进行可扩展且跨语言的服务的开发，Hive集成了该服务，能让不同的编程语言调用Hive的接口。
- **Hive WEB Interface（HWI）**：Hive客户端提供了一种通过网页的方式访问hive所提供的服务。这个接口对应Hive的HWI组件（hive web interface）

Hive的组成模块(3)

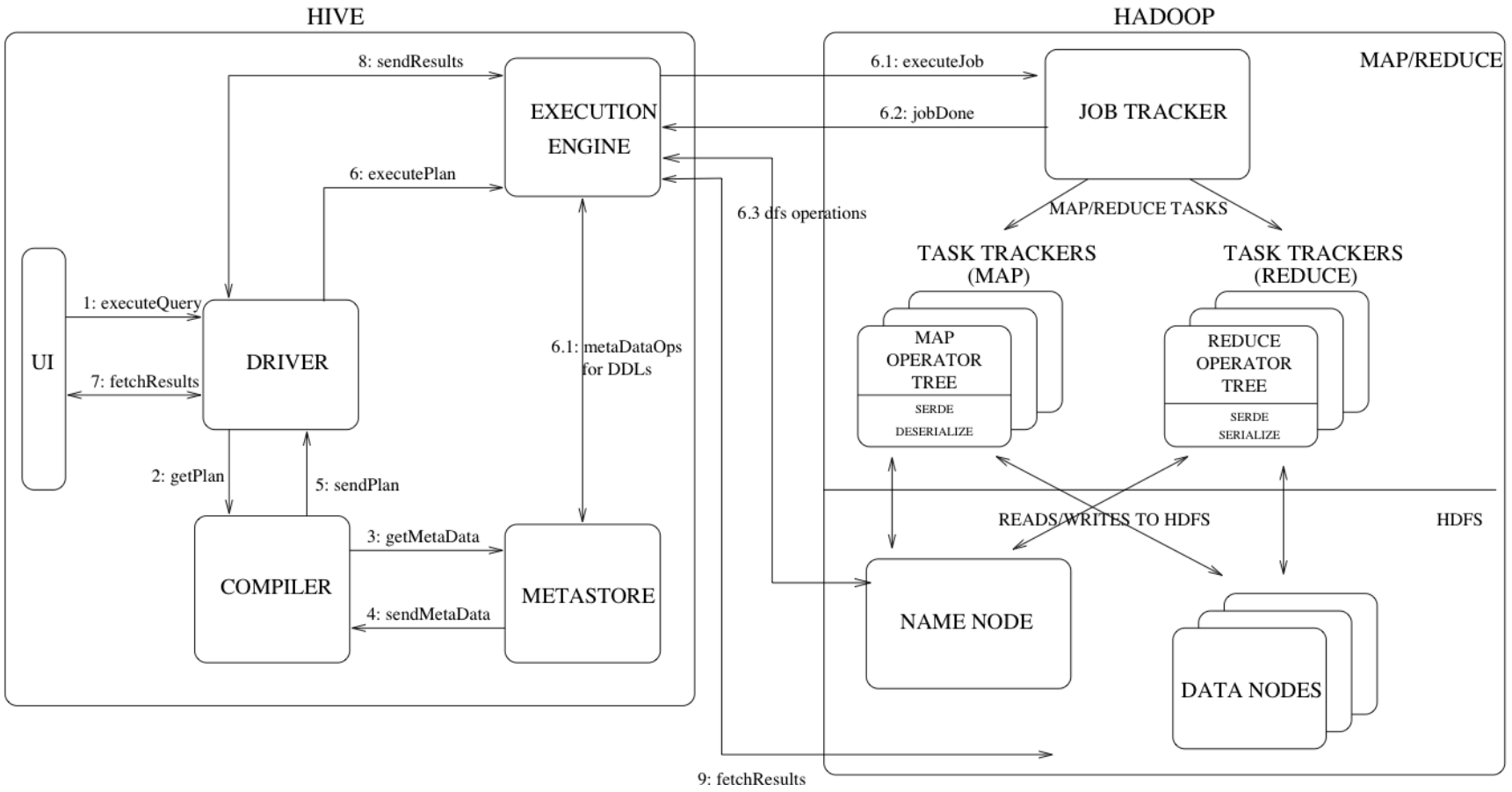
- HiveQL：这是Hive的数据查询语言，与SQL非常类似。Hive提供了这个数据查询语言与用户的接口，包括一个shell的接口，可以进行用户的交互以及网络接口与JDBC接口。JDBC接口可以用于编程，与传统的数据库编程类似，使得程序可以直接使用Hive功能而无需更改。

Hive的执行流程简单示意图



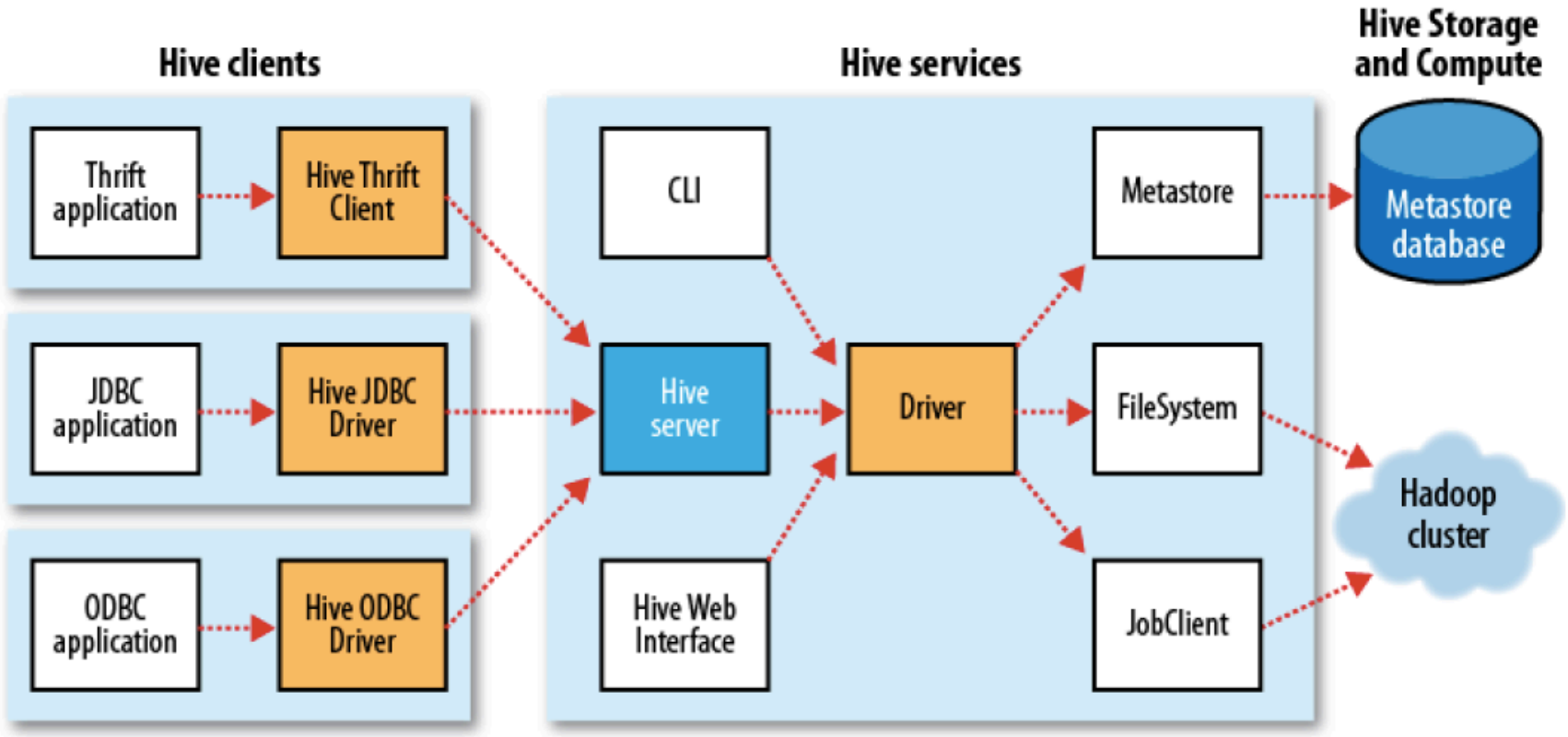
Hive 将通过CLI接入，JDBC/ODBC接入，或者HWI接入的相关查询，通过Driver(Complier、Optimizer和Executor)，进行编译，分析优化，最后变成可执行的MapReduce。

Hive的系统结构



https://cwiki.apache.org/confluence/display/Hive/Design#Design-HiveArchitecture

Hive客户端



通过将Hive运行为服务器hive --service hiveservice可以启动一个服务，而后可以通过不同的客户端连接到这个服务去访问Hive提供的功能。

Hive客户端

- Thrift客户端：被绑定在多种语言中，包括C++，Java，PHP，Python以及Ruby等
- JDBC驱动：提供纯Java的JDBC驱动，连接字符串类似于jdbc:hive://host:port/dbname，除非使用了嵌入的模式，否则JDBC模式的驱动访问了Thrift服务器
- ODBC驱动：与JDBC类似，但尚未成熟

Hive的数据模型

- 每一个类似于数据库的系统都首先需要定义一个数据模型，然后才是在这个数据模型之上的各种操作
- Tables：Hive的数据模型由数据表组成。数据表中的列是有类型的（int, float, string, data, boolean）也可以是复合的类型，如list, map （类似于JSON形式的数据）
- Partitions：数据表可以按照一定的规则进行划分Partition。例如，通过日期的方式将数据表进行划分
- Buckets：数据存储的桶。在一定范围内的数据按照Hash的方式进行划分（这对于数据的抽样以及对于join的优化很有意义）

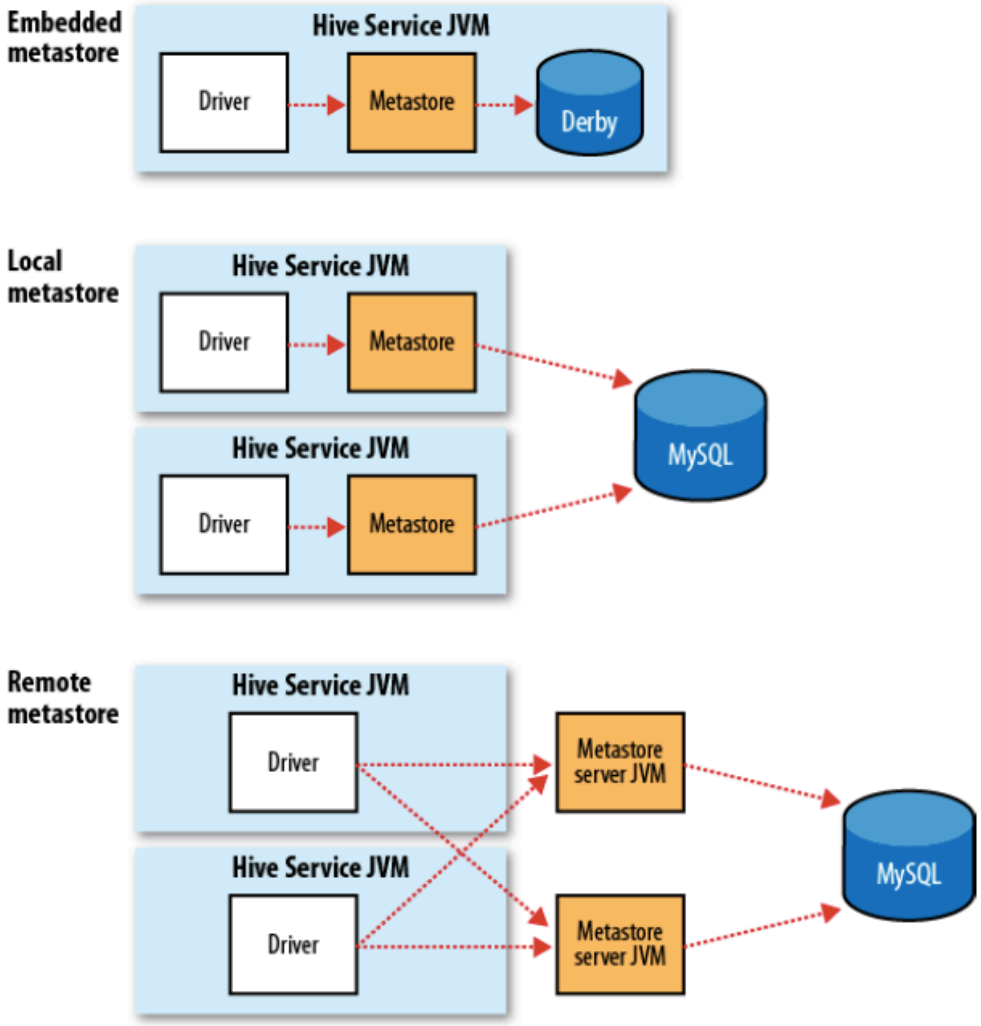
元数据存储：Metastore

- 在Hive中由一系列的数据表格组成一个命名空间，关于这个命名空间的描述信息会保存在Metastore的空间中
- 元数据使用SQL的形式存储在传统的关系数据库中，因此可以使用任意一种关系数据库，例如Derby（Apache的关系数据库实现），MySQL以及其他的多种关系数据库存储方法
- 在数据库中，保存最重要的信息是有关数据库中的数据表格描述，包括每一个表的格式定义，列的类型，物理的分布情况，数据划分情况等

metastore

- metastore包括两个部分：服务和后台的数据存储
  - Embedded metastore: 默认使用内嵌的Derby数据库实例，每次只能打开一个Hive会话
  - Local metastore: 可以使用运行在一个进程中的metastore进程来访问独立的数据库，可通过JDBC进行设置具体的数据库访问
  - Remote metastore: 一个或者多个metastore服务器和Hive服务运行在不同的进程内

metastore的配置情况



数据的物理分布情况

- Hive在HDFS中有固定的位置，通常被放置在HDFS的如下目录中：/home/hive/warehouse
- 每个数据表被存放在warehouse的子目录中。进一步来说，数据划分Partition、数据桶Buckets形成了数据表的子目录
- 数据可能以任意一种形式存储，例如：
  - 使用分隔符的文本文件，或者是SequenceFile
  - 使用用户自定义的SerDe，则可以定义任意格式的文件

Hive系统的配置

- 要想Hive使用HDFS进行数据仓库的存储，使用MapReduce进行HQL语言的执行，需要进行相应的配置。
  - hive-env.sh
    - Set HADOOP_HOME
  - hive-site.xml (hive-default.xml.template)
    - 元数据仓库

```
<property>
   <name>javax.jdo.option.ConnectionURL</name>
   <value>jdbc:derby:;databaseName=metastore_db;create=true</value>
</property>
```

    - 元数据文件目录

```
<property>
   <name>hive.metastore.warehouse.dir</name>
   <value>/user/hive/warehouse</value>
</property>
```

摘要

- Hive基本原理
- Hive基本操作
- Hive程序设计

启动Hive的命令行界面shell

- 完成Hive系统的合适配置之后，打开任意一个命令行界面，执行下面的命令就可以启动hive的界面
- $cd /hive/install/directory进入hive的安装目录
- $bin /hive 如果已经将hive加入到执行路径，则可以直接执行hive即可
- 执行成功的话，我们就可以看到hive的主界面 hive>
- 等待用户输入查询命令

Hive QL

- Hive主要支持以下几类操作：
  - DDL：数据定义语句，包括CREATE, ALTER, SHOW, DESCRIBE, DROP等
  - DML：数据操作语句，包括LOAD DATA, INSERT。Hive的设计中没有考虑UPDATE操作。
  - QUERY：数据查询语句，主要是SELECT语句。

创建数据表的命令

- 显示所有的数据表
  - hive> show tables;
- 创建一个表，这个表包括两列，分别是整数类型以及字符串类型，使用文本文件表达
  - hive> CREATE TABLE pokes (foo INT, bar STRING);
  - hive> CREATE TABLE invites (foo INT, bar STRING) PARTITIONED BY (ds STRING);
  - hive>CREATE TABLE Shakespeare (freq int, word string)

    row format delimited fields terminated by '\t'

    stored as textfile; //定义这张表使用的数据文件格式，这里指定为txt类型

描述数据表的命令

- 显示所创建的数据表的描述，即创建时候对于数据表的定义
  - hive> DESCRIBE invites;
- 修改表的语句
  - hive> ALTER TABLE pokes ADD COLUMNS (new_col INT);
  - hive> ALTER TABLE invites ADD COLUMNS (new_col2 INT COMMENT 'a comment‘);
  - hive> ALTER TABLE invites REPLACE COLUMNS (foo INT, bar STRING, baz INT COMMENT 'baz replaces new_col2');

装入数据

- 数据装入到Hive中
  - hive> LOAD DATA LOCAL INPATH './examples/files/kv1.txt' OVERWRITE INTO TABLE pokes;
- NOTES
  - If the 'OVERWRITE' keyword is omitted, data files are appended to existing data sets.
  - NO verification of data against the schema is performed by the load command.
  - If the file is in hdfs, it is moved into the Hive-controlled file system namespace.

装入数据

- 分区
  - hive> LOAD DATA LOCAL INPATH './examples/files/kv2.txt' OVERWRITE INTO TABLE invites PARTITION (ds='2008-08-15');
  - hive> LOAD DATA LOCAL INPATH './examples/files/kv3.txt' OVERWRITE INTO TABLE invites PARTITION (ds='2008-08-08');
- 从HDFS装入数据
  - hive> LOAD DATA INPATH '/user/myname/kv2.txt' OVERWRITE INTO TABLE invites PARTITION (ds='2008-08-15');

SELECTS and FILTERS

- hive> SELECT a.foo FROM invites a WHERE a.ds='2008-08-15';
- hive> SELECT a.foo FROM invites a WHERE a.ds='2008-08-15' sort by a.foo asc limit 10;
- hive> INSERT OVERWRITE DIRECTORY '/tmp/hdfs_out'

  SELECT a.* FROM invites a WHERE a.ds='2008-08-15';
- hive> INSERT OVERWRITE LOCAL DIRECTORY '/tmp/local_out'

  SELECT a.* FROM pokes a;

Group By

- hive> INSERT OVERWRITE TABLE events
  SELECT a.bar, count(*)
  FROM invites a
  WHERE a.foo > 0 GROUP BY a.bar;

Join

- hive> SELECT t1.bar, t1.foo, t2.foo
  FROM pokes t1 JOIN invites t2 ON
  t1.bar = t2.bar;

Hive: Example

- Hive looks similar to an SQL database
- Relational join on two tables:
  - Table of word counts from Shakespeare collection
  - Table of word counts from the bible

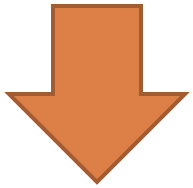
```
SELECT s.word, s.freq, k.freq FROM shakespeare s
  JOIN bible k ON (s.word = k.word) WHERE s.freq >= 1 AND k.freq >= 1
  ORDER BY s.freq DESC LIMIT 10;
```

the	25848	62394
I	23031	8854
and	19671	38985
to	18038	13526
of	16700	34654
a	14170	8057
you	12702	2720
my	11297	4135
in	10797	12445
is	8882	6884

Source: Material drawn from Cloudera training VM

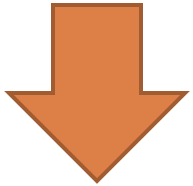
Hive: Behind the Scenes

```
SELECT s.word, s.freq, k.freq FROM shakespeare s
  JOIN bible k ON (s.word = k.word) WHERE s.freq >= 1 AND k.freq >= 1
  ORDER BY s.freq DESC LIMIT 10;
```



(Abstract Syntax Tree)

```
(TOK_QUERY (TOK_FROM (TOK_JOIN (TOK_TABREF shakespeare s) (TOK_TABREF bible k) (= (. (TOK_TABLE_OR_COL s)
word) (. (TOK_TABLE_OR_COL k) word)))) (TOK_INSERT (TOK_DESTINATION (TOK_DIR TOK_TMP_FILE)) (TOK_SELECT
(TOK_SELEXPR (. (TOK_TABLE_OR_COL s) word)) (TOK_SELEXPR (. (TOK_TABLE_OR_COL s) freq)) (TOK_SELEXPR (.
(TOK_TABLE_OR_COL k) freq))) (TOK_WHERE (AND (>= (. (TOK_TABLE_OR_COL s) freq) 1) (>= (. (TOK_TABLE_OR_COL k)
freq) 1))) (TOK_ORDERBY (TOK_TABSORTCOLNAMEDESC (. (TOK_TABLE_OR_COL s) freq))) (TOK_LIMIT 10)))
```



(one or more of MapReduce jobs)

Hive: Behind the Scenes

```
STAGE DEPENDENCIES:
  Stage-1 is a root stage
  Stage-2 depends on stages: Stage-1
  Stage-0 is a root stage

STAGE PLANS:
  Stage: Stage-1
    Map Reduce
      Alias -> Map Operator Tree:
        s
          TableScan
            alias: s
            Filter Operator
              predicate:
                  expr: (freq >= 1)
                  type: boolean
              Reduce Output Operator
                key expressions:
                      expr: word
                      type: string
                sort order: +
                Map-reduce partition columns:
                      expr: word
                      type: string
                tag: 0
                value expressions:
                      expr: freq
                      type: int
                      expr: word
                      type: string
        k
          TableScan
            alias: k
            Filter Operator
              predicate:
                  expr: (freq >= 1)
                  type: boolean
              Reduce Output Operator
                key expressions:
                      expr: word
                      type: string
                sort order: +
                Map-reduce partition columns:
                      expr: word
                      type: string
                tag: 1
                value expressions:
                      expr: freq
                      type: int
      Reduce Operator Tree:
        Join Operator
          condition map:
               Inner Join 0 to 1
          condition expressions:
            0 {VALUE._col0} {VALUE._col1}
            1 {VALUE._col0}
          outputColumnNames: _col0, _col1, _col2
          Filter Operator
            predicate:
                expr: ((_col0 >= 1) and (_col2 >= 1))
                type: boolean
            Select Operator
              expressions:
                    expr: _col1
                    type: string
                    expr: _col0
                    type: int
                    expr: _col2
                    type: int
              outputColumnNames: _col0, _col1, _col2
              File Output Operator
                compressed: false
                GlobalTableId: 0
                table:
                    input format: org.apache.hadoop.mapred.SequenceFileInputFormat
                    output format: org.apache.hadoop.hive.ql.io.HiveSequenceFileOutputFormat

  Stage: Stage-2
    Map Reduce
      Alias -> Map Operator Tree:
        hdfs://localhost:8022/tmp/hive-training/364214370/10002
            Reduce Output Operator
              key expressions:
                    expr: _col1
                    type: int
              sort order: -
              tag: -1
              value expressions:
                    expr: _col0
                    type: string
                    expr: _col1
                    type: int
                    expr: _col2
                    type: int
      Reduce Operator Tree:
        Extract
          Limit
            File Output Operator
              compressed: false
              GlobalTableId: 0
              table:
                  input format: org.apache.hadoop.mapred.TextInputFormat
                  output format: org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat


  Stage: Stage-0
    Fetch Operator
      limit: 10
```

Hive: Example

- 数据文件

1 Rod 18 study-game-driver std_addr:Beijing-work_addr:shanghai

2 Tom 21 study-game-driver std_addr:Beijing-work_addr:Beijing

3 Jerry 33 study-game-football std_addr:Beijing-work_addr:Shenzhen

4 Bob 23 study-game-music std_addr:Beijing-work_addr:Nanjing

- 创建Hive表
  - hive> CREATE TABLE person (id INT, name STRING, age INT, fav ARRAY<STRING>, addr MAP<STRING, STRING>)

    COMMENT 'This is the person table'

    ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'

    COLLECTION ITEMS TERMINATED BY '-'

    MAP KEYS TERMINATED BY ':'

    STORED AS TEXTFILE;
- 导入数据
  - hive> LOAD DATA LOCAL INPATH person.txt OVERWRITE INTO TABLE person;

表的分区

- 可以把数据依照单个或多个列进行分区，通常按照时间、地域进行分区。为了达到性能表现的一致性，对不同列的划分应该让数据尽可能均匀分布。分区应当在建表时设置。
  - hive> CREATE TABLE person (id INT, name STRING, age INT, fav ARRAY<STRING>, addr MAP<STRING, STRING>)

    COMMENT 'This is the person table'

    PARTITIONED BY (dt STRING)

    ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'

    COLLECTION ITEMS TERMINATED BY '-'

    MAP KEYS TERMINATED BY ':'

    STORED AS TEXTFILE;
  - hive> LOAD DATA LOCAL INPATH person.txt OVERWRITE INTO TABLE person partition (dt='20180315');
  - hive> SELECT addr['work_addr'] from person WHERE dt='20180315';

表的分桶

- 分桶是相对于分区进行更细粒度的划分。在分区数量过于庞大以至于可能导致文件系统崩溃时，就需要使用分桶来解决问题。
- 分桶将整个数据内容按照某列属性值的hash值进行区分。分桶同样应当在建表时就建立。
  - hive> set hive.strict.checks.bucketing=false;
  - hive> CREATE TABLE person (id INT, name STRING, age INT, fav ARRAY<STRING>, addr MAP<STRING, STRING>)

    PARTITIONED BY (ds STRING)

    CLUSTERED BY (id) into 3 buckets

    ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'

    COLLECTION ITEMS TERMINATED BY '-'

    MAP KEYS TERMINATED BY ':'

    STORED AS TEXTFILE;

表的分桶

- 导入数据
  - hive> LOAD DATA LOCAL INPATH person.txt OVERWRITE INTO TABLE person partition (dt='20180315');
- 查询某个桶里的数据
  - hive> SELECT * from person tablesample (bucket 1 out of 3 on id);

内部表和外部表

- 内部表：在创建的时候会把数据移动到数据仓库所指向的位置；在删除的时候会将元数据和数据一起删除。如果仅仅是Hive内部使用，可以使用内部表。
- 外部表：仅仅记录数据所在的位置；在删除的时候仅仅删除元数据，真正的数据不会删除。
  - 创建空表（通过external关键词表明创建的表是外部表），然后导入数据；
  - 创建表的时候指定数据文件的位置，用external关键词创建外部表，使用location关键词指定数据文件的位置；

外部表

- hive> CREATE external TABLE person (id INT, name STRING, age INT, fav ARRAY<STRING>, addr MAP<STRING, STRING>)

  COMMENT 'This is the person table'

  ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'

  COLLECTION ITEMS TERMINATED BY '-'

  MAP KEYS TERMINATED BY ':'

  LOCATION 'hdfs://localhost:9000/root/ '

  STORED AS TEXTFILE;
- hive> dfs –ls /user/hive/warehouse/
- hive> dfs –ls /user/hive/warehouse/person

自定义函数

- User Defined Function (UDF)
  - UDF：作用于单条数据，并且输出一个数据行，如字符串函数、日期函数。
  - UDAF：可接受多个数据输入，同时输出一个数据行，如COUNT、MIN、MAX等聚集函数。
  - UDTF：作用于单个数据行，同时产生多个输出数据行。

自定义函数

- 编写Java代码
  - public class Maximum extends UDAF {}
- 导出Jar包，并将Jar包添加到Hive中
  - hive> add jar Maximum.jar
- 用Jar包生成函数
  - hive> create function maximumtest as ‘com.firsthigh.udafMaximum’;
- 运行函数并检查结果
  - hive> select maximumtest(price) from record_dimension;

SQL和HiveQL比较

特性	SQL	HiveQL
更新	UPDATE, INSERT, DELETE	insert OVERWRITE\INTO TABLE
事务	支持	有限支持
模式	写模式	读模式
索引	支持	支持
延迟	亚秒级	分钟级
数据类型	整数、浮点数、定点数、文本和二机制串、时间	布尔型、整数、浮点数、文本和二进制串、时间戳、数组、映射、结构
函数	数百个内置函数	数百个内置函数
多表插入	不支持	支持
子查询	完全支持	只能在FROM、WHERE或HAVING子句中
视图	可更新	只读
可扩展性	低	高

Hive优化

- 数据倾斜问题
  - Key分布不均匀
    - 随机值，打散key
  - 业务数据本身的原因
  - 建表考虑不周
  - SQL本身就有数据倾斜
    - 选用join key分布最均匀的表作为驱动表
    - 大小表join的时候，让维度较小的表先进内存
    - 大表join的时候，把空值的key变成一个字符串加上一个随机数，把倾斜的数据分到不同的reduce上
    - count distinct大量相同特殊值。

Hive优化

- MapReduce优化
  - 尽量避免大量的job
- 配置优化
  - 列裁剪：忽略不需要的列
  - 分区裁剪：减少不必要的分区
  - Hive参数调节：（1）hive.map.aggr = true；（2）hive.groupby.skewindata = true。
- 程序优化
  - Join操作：将数目少的表或者子查询放在join操作符的左边（小表放前原则）。
  - GROUP BY操作：（1）Map端部分聚合；（2）在有数据倾斜时进行负载均衡。

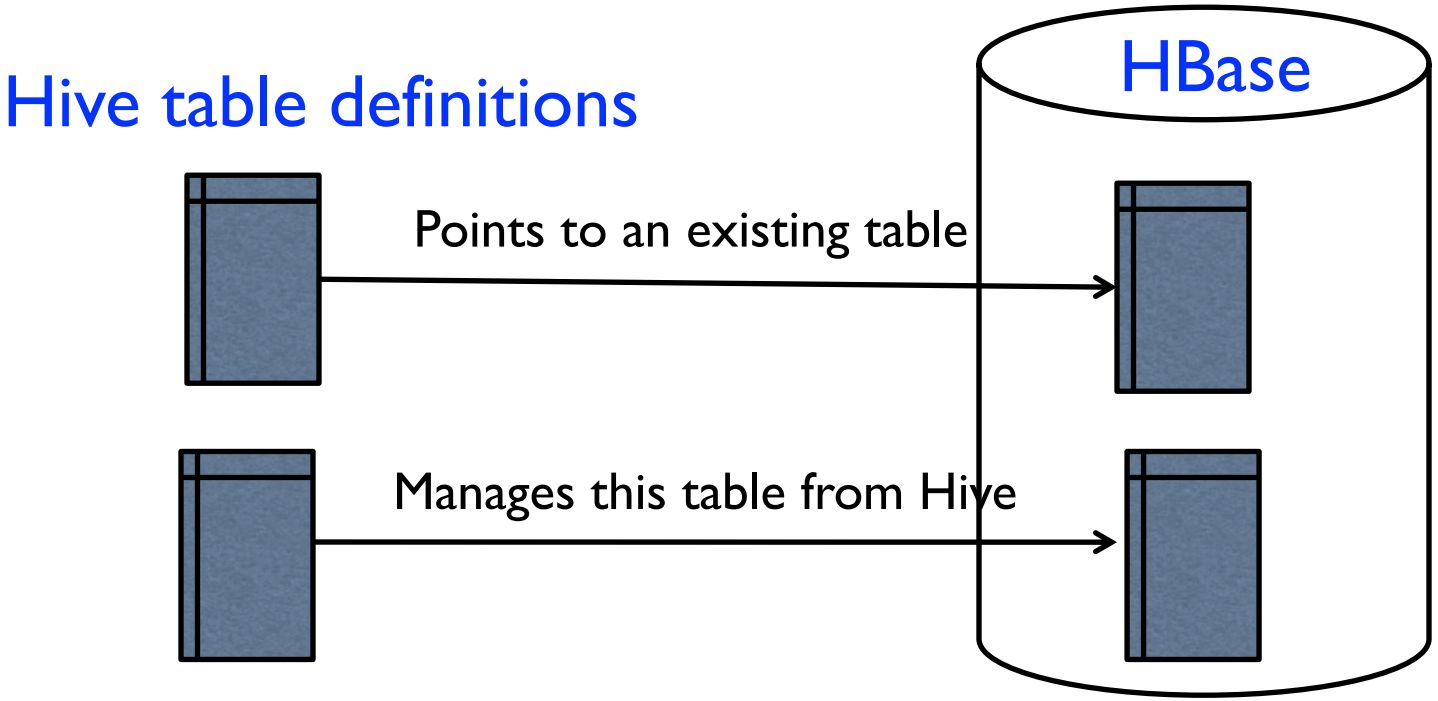
Hive + HBase?

- Reasons to use Hive on HBase:
  - A lot of data sitting in HBase due to its usage in a real-time environment, but never used for analysis
  - Give access to data in HBase usually only queried through MapReduce to people that don't code (business analysts)
  - When needing a more flexible storage solution, so that rows can be updated live by either a Hive job or an application and can be seen immediately to the other

- Reasons not to do it:
  - Run SQL queries on HBase to answer live user requests (it's still a MR job)
  - Hoping to see interoperability with other SQL analytics systems

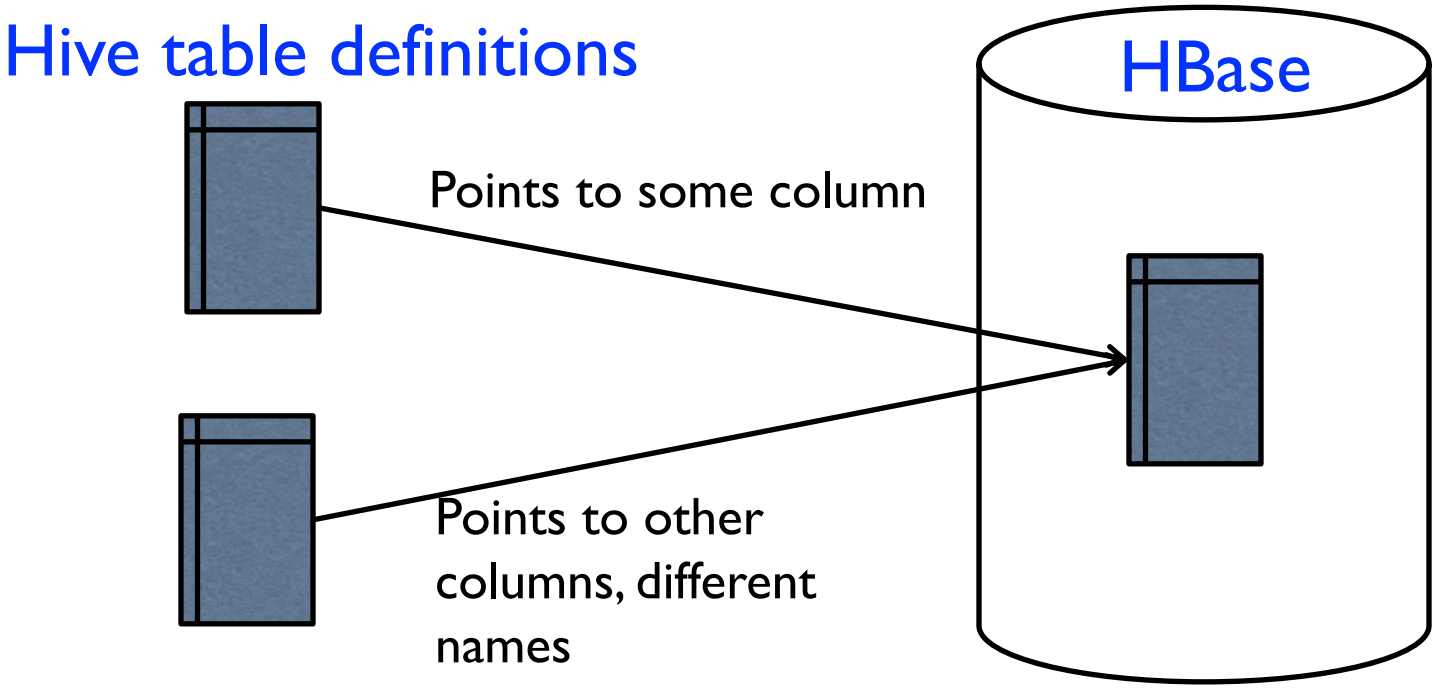
Integration

- How it works:
  - Hive can use tables that already exist in HBase or manage its own ones, but they still all reside in the same HBase instance



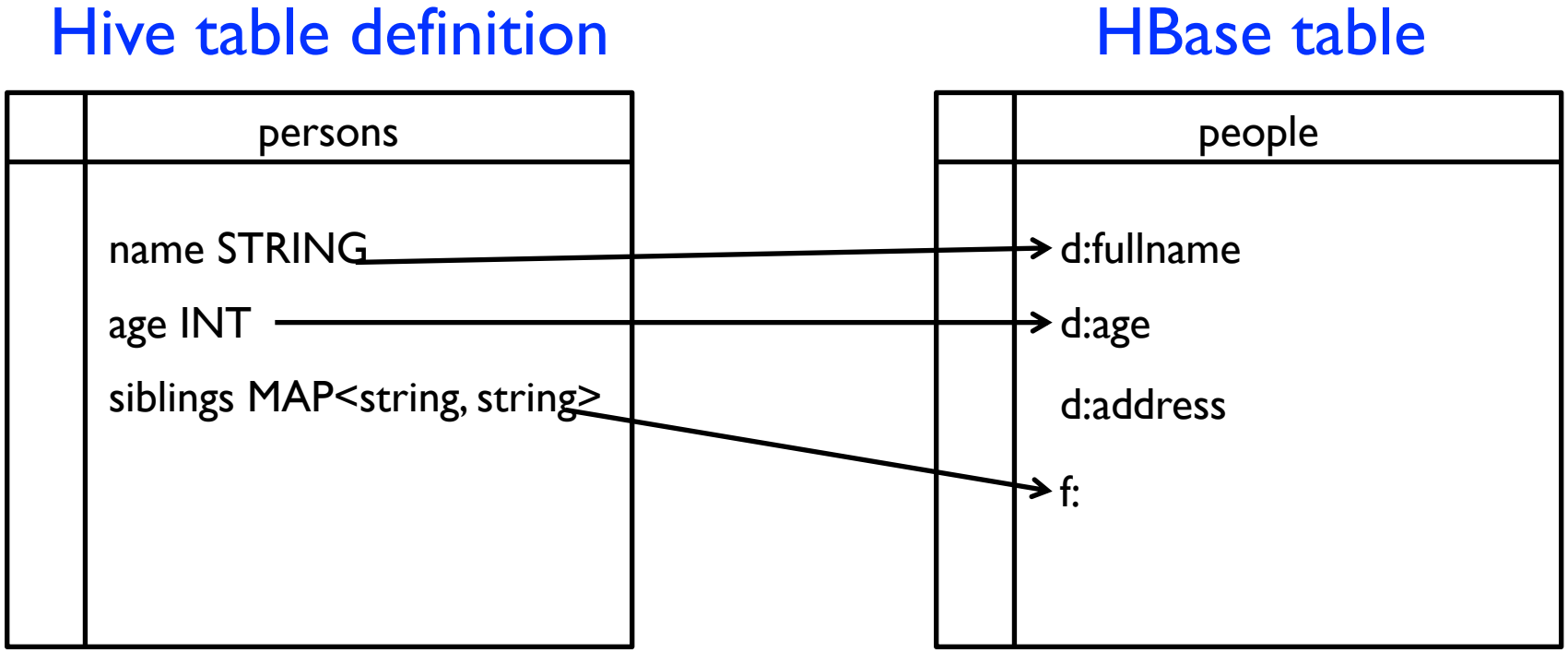
Integration

- How it works:
  - When using an already existing table, defined as EXTERNAL, you can create multiple Hive tables that point to it.



Integration

- How it works:
  - Columns are mapped however you want, changing names and giving types



Hive总结

- Hive提供了一种类似于SQL的查询语言，使得其能够用于用户的交互查询
- 与传统的数据库类似，Hive提供了多个数据表之间的联合查询，能够完成高效的多个数据表之间的查询
- 通过底层执行引擎的工作，Hive将SQL语言扩展到很大的查询规模