分布式数据仓库——Hive

贝毅君

beiyj@zju.edu.cn

统计系统使用演进


MySQL
Cluster


ORACLE
10g
Real Application Cluster


HIVE


MySQL


ORACLE
DATABASE


hadoop

关系型数据库 vs MapReduce

	传统关系型数据库	MapReduce
数据大小	GB	PB
访问	交互型和批处理	批处理
更新	多次读写	一次写多次读
结构	静态模式	动态模式
集成度	高	低
伸缩性	非线性	线性

Hive概述

- Hive是基于Hadoop的一个开源数据仓库系统，可以将结构化的数据文件映射为一张数据库表，并提供完整的sql查询功能，可以将sql语句转换为MapReduce任务进行运行支持超大规模的数据查询分析。
- 其优点是学习成本低，可以通过类SQL语句快速实现简单的MapReduce统计，不必开发专门的MapReduce应用，十分适合数据仓库的统计分析。

Hive定义

- Hive是建立在 Hadoop 上的数据仓库基础构架。它提供了一系列的工具，可以用来进行数据提取转化加载（ETL），可以存储、查询和分析存储在 Hadoop 中的大规模数据。
- Hive 定义了简单的类 SQL 查询语言，称为 HQL，它允许熟悉 SQL 的用户查询数据。
- 这个语言也允许熟悉 MapReduce 开发者的开发自定义的 mapper 和 reducer， 来处理内建的 mapper 和 reducer 无法完成的复杂的分析工作。

Hive特性

- Hive 构建在基于静态批处理的Hadoop 之上，Hadoop 通常都有较高的延迟并且在作业提交和调度的时候需要大量的开销。
- Hive 并不适合那些需要低延迟的应用，例如，联机事务处理（OLTP）
  - Hive 在几百MB 的数据集上执行查询一般有分钟级的时间延迟。
  - Hive 不提供实时查询和基于行级的数据更新操作。
- Hive 的最佳使用场合是大数据集的批处理作业
  - 网络日志分析。

Hive举例（1）

#创建表人信息表 person(String name, int age)
hive> create table person(name STRING,age INT)ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' ESCAPED BY '\\' STORED AS TEXTFILE;
OK
Time taken: 0.541 seconds

#创建表票价信息表 ticket(int age, float price)
hive> create table ticket(age INT,price FLOAT)ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' ESCAPED BY '\\' STORED AS TEXTFILE;
OK
Time taken: 0.154 seconds

#创建本地数据文件
-rw-rw-r-- 1 hadoop hadoop 40 Feb 6 13:28 person.txt
-rw-rw-r-- 1 hadoop hadoop 45 Feb 6 13:28 ticket.txt

Hive举例（2）

#将本地的数据文件load到hive数据仓库中，数据路径下/user/hive/warehouse
hive> LOAD DATA LOCAL INPATH '/home/hadoop/hfxdoc/person.txt' OVERWRITE INTO TABLE person;
Copying data from file:/home/hadoop/hfxdoc/person.txt
Copying file: file:/home/hadoop/hfxdoc/person.txt
Loading data to table default.person
Deleted hdfs://10.15.107.155:8000/user/hive/warehouse/person
OK
Time taken: 0.419 seconds

hive> LOAD DATA LOCAL INPATH '/home/hadoop/hfxdoc/ticket.txt' OVERWRITE INTO TABLE ticket;
Copying data from file:/home/hadoop/hfxdoc/ticket.txt
Copying file: file:/home/hadoop/hfxdoc/ticket.txt
Loading data to table default.ticket
Deleted hdfs://10.15.107.155:8000/user/hive/warehouse/ticket
OK
Time taken: 0.25 seconds

Hive举例（3）

```
hive> show tables;
hive> describe person
hive> select * from person;
 OK
 huang   26
 lili    25
 dongdong        13
 wangxiao        5
 Time taken: 0.092 seconds
hive>
```

#注意select *语句是不会编译成MapReduce程序的，所以很快。

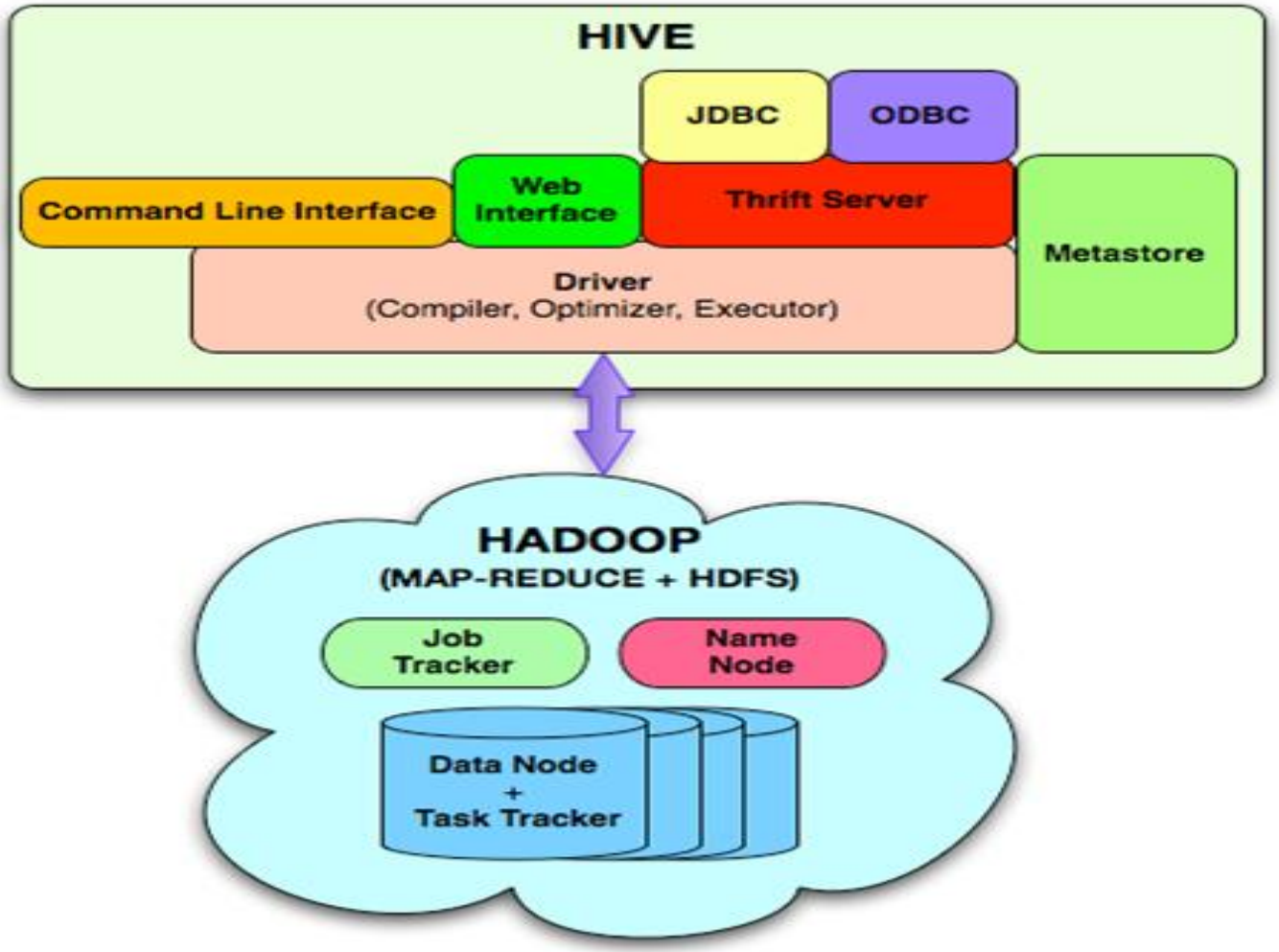
Hive举例（4）

#稍作复杂点的join查询

```
hive> select * from person join ticket on person.age = ticket.age;
MapReduce Total cumulative CPU time: 5 seconds 510 msec
Ended Job = job_201301211420_0011
MapReduce Jobs Launched:
Job 0: Map: 2  Reduce: 1   Cumulative CPU: 5.51 sec   HDFS Read:
519 HDFS Write: 71 SUCCESS
Total MapReduce CPU Time Spent: 5 seconds 510 msec
OK
wangxiao        5       5       10.0
dongdong        13      13      20.0
lili    25      25      30.0
huang   26      26      30.0
Time taken: 32.465 seconds
```

#这里查询语句被编译成MapReduce程序，在hadoop上执行

Hive体系架构


HIVE
JDBC
ODBC
Command Line Interface
Web Interface
Thrift Server
Metastore
Driver
(Compiler, Optimizer, Executor)
HADOOP
(MAP-REDUCE + HDFS)
Job Tracker
Name Node
Data Node
+
Task Tracker

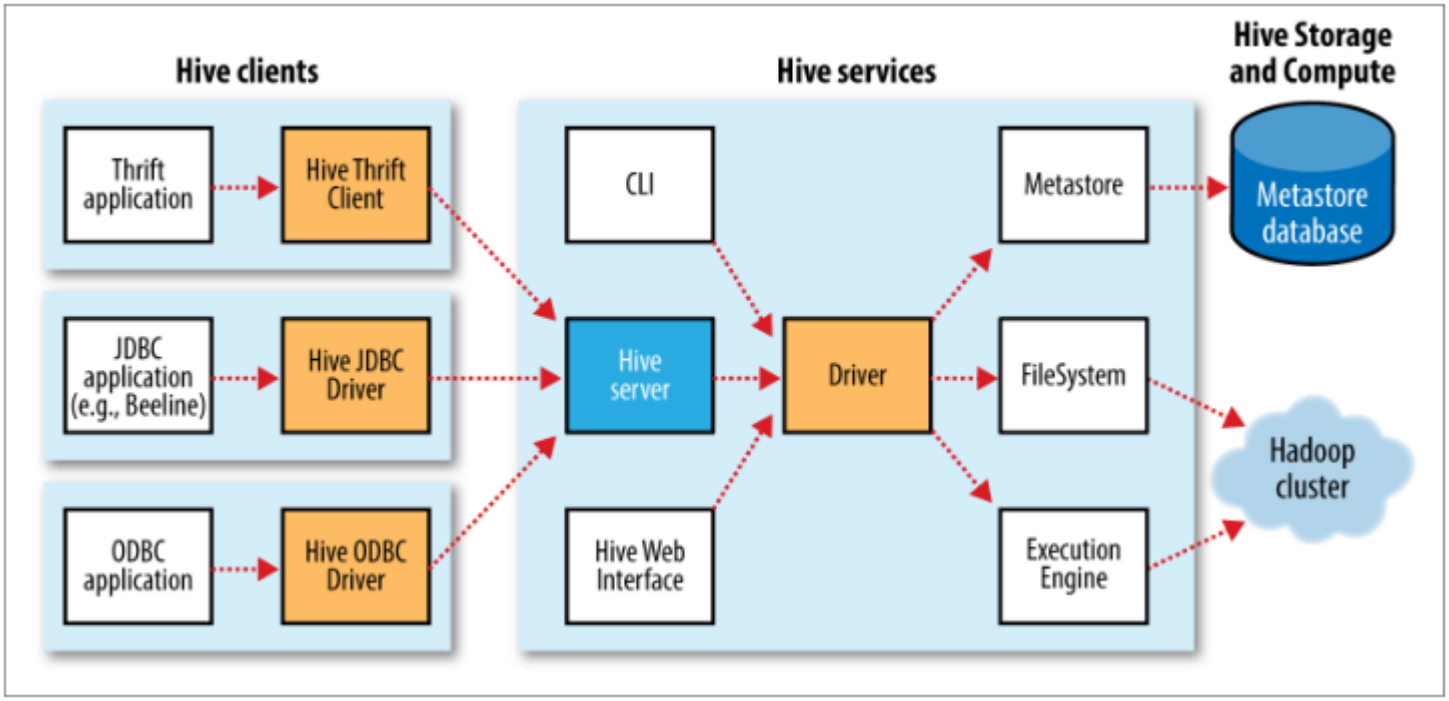
Hive组件（1）

- 用户接口
  - 包括 CLI，Client，WUI。
  - 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。
- 元数据存储
  - 通常是存储在关系数据库如mysql, derby 中。
  - Hive 中的元数据包括表的名字，表的列和分区及其属性，表的属性（是否为外部表等），表的数据所在目录等。

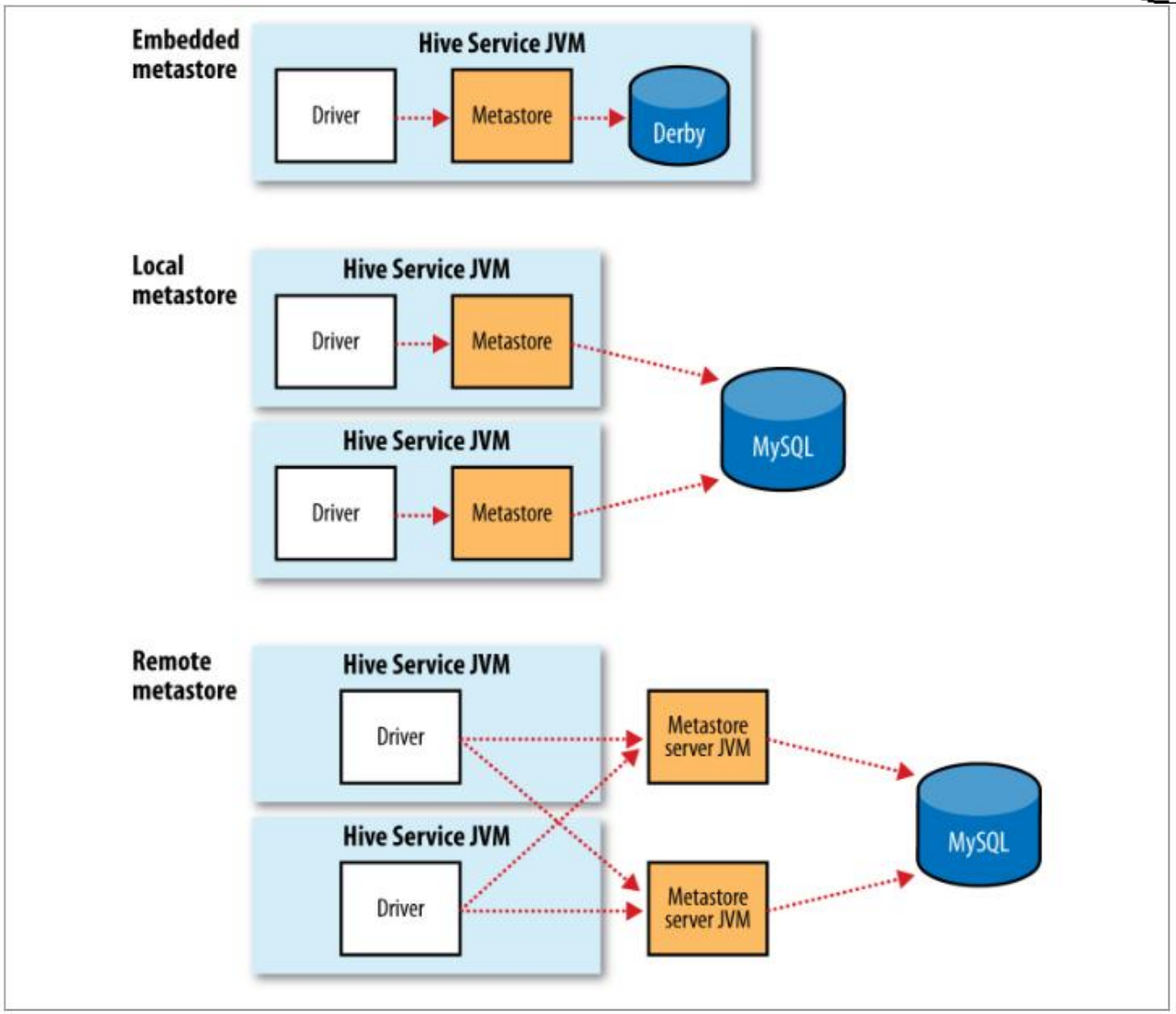
Hive组件（2）

- 驱动
  - 包括解释器、编译器、优化器、执行器
  - 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：用 HDFS 进行存储，利用 MapReduce 进行计算。
  - Hive 的数据存储在 HDFS 中，大部分的查询由 MapReduce 完成（包含 * 的查询，比如 select * from tbl 不会生成 MapRedcue 任务）。

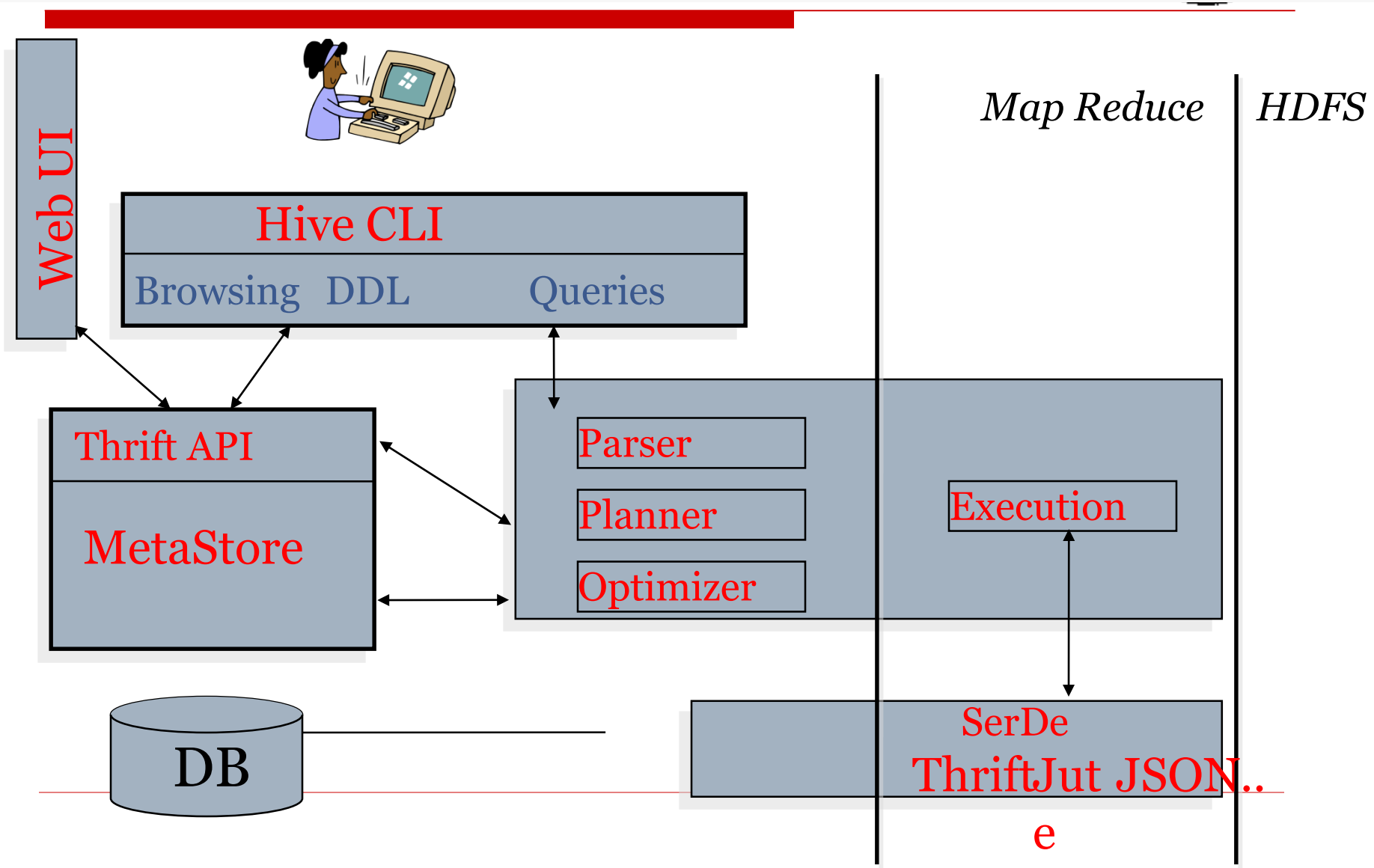
Hive访问方式


Hive clients
Hive services
Hive Storage and Compute
Thrift application
Hive Thrift Client
JDBC application (e.g., Beeline)
Hive JDBC Driver
ODBC application
Hive ODBC Driver
CLI
Hive server
Hive Web Interface
Driver
Metastore
FileSystem
Execution Engine
Metastore database
Hadoop cluster

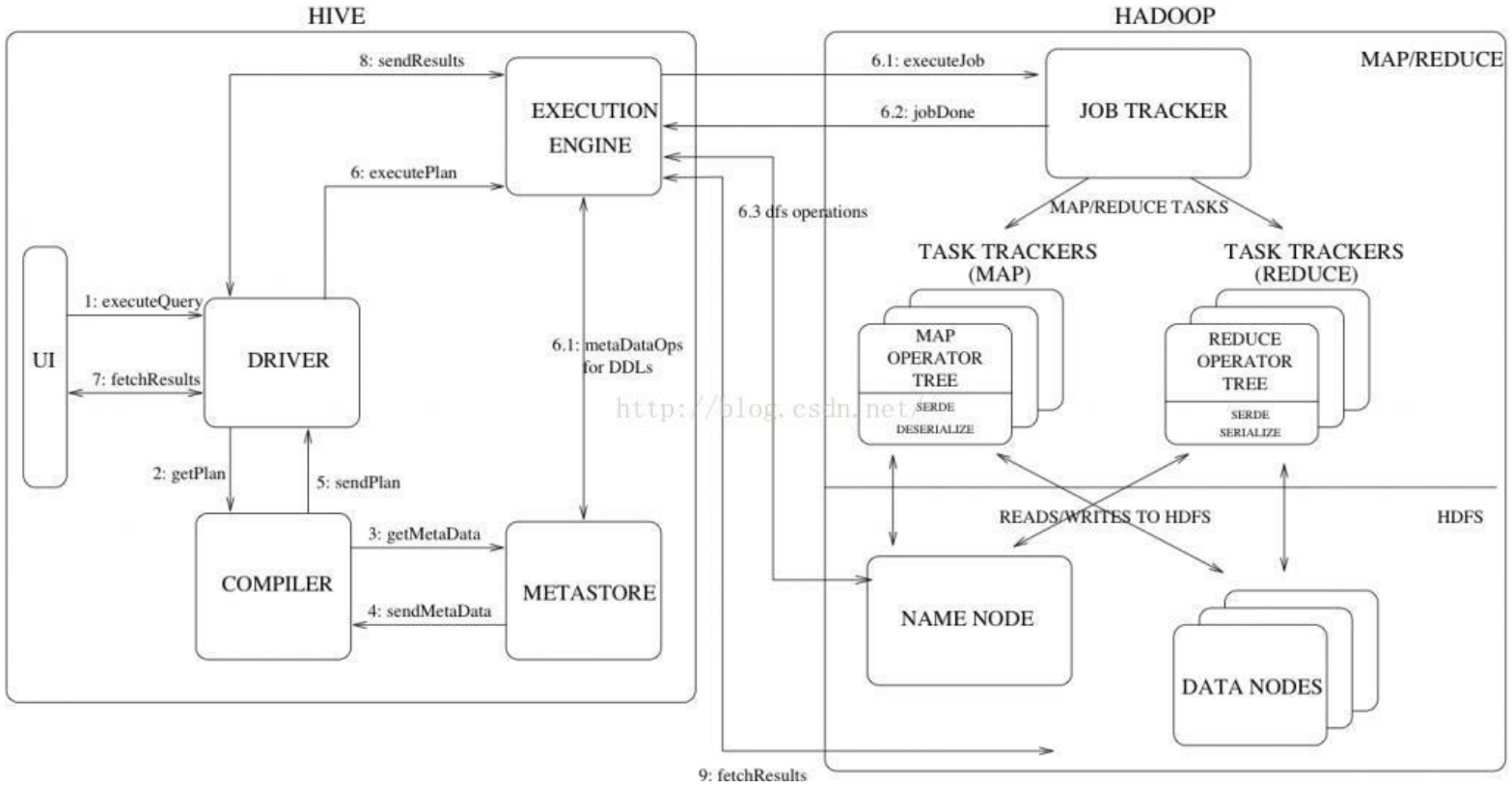
Hive Metastore



Hive层次结构


Web UI
Hive CLI
Browsing
DDL
Queries
Thrift API
MetaStore
Parser
Planner
Optimizer
Execution
Map Reduce
HDFS
DB
SerDe
ThriftJut
JSON..
e

Hive执行过程


HIVE
HADOOP
MAP/REDUCE
8: sendResults
6: executePlan
EXECUTION ENGINE
6.1: executeJob
6.2: jobDone
JOB TRACKER
6.3 dfs operations
MAP/REDUCE TASKS
TASK TRACKERS (MAP)
TASK TRACKERS (REDUCE)
UI
1: executeQuery
7: fetchResults
DRIVER
6.1: metaDataOps for DDLs
MAP OPERATOR TREE
SERDE
DESERIALIZE
REDUCE OPERATOR TREE
SERDE
SERIALIZE
2: getPlan
5: sendPlan
READS/WRITES TO HDFS
HDFS
COMPILER
3: getMetaData
4: sendMetaData
METASTORE
NAME NODE
DATA NODES
9: fetchResults

Hive和关系数据库的比较

查询语言	HQL	SQL
数据存储位置	HDFS	Raw Device 或者 Local FS
数据格式	用户定义	系统决定
数据更新	支持	不支持
索引	无	有
执行	MapRedcue	Executor
执行延迟	高	低
可扩展性	高	低
数据规模	大	小

数据模型

- Hive没有专门的数据存储格式，也没有为数据建立索引。
- 用户可以非常自由的组织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，Hive就可以解析数据。
- Hive中所有的数据都存储在 HDFS 中，Hive中包含以下数据模型：表(Table)，外部表(External Table)，分区(Partition)，桶(Bucket)。

数据模型——表

- 表 Table
  - 隶属数据库的表
  - 每个表在HDFS上有相应的目录
  - 例如
    - 网页浏览表：pvs
    - HDFS 目录
      - **/user/hive/warehouse/pvs**

```
CREATE TABLE managed_table (dummy STRING);
LOAD DATA INPATH '/user/tom/data.txt' INTO table managed_table;
```

数据模型——分区

- 分区 Partitions
  - 类似于列上的索引
  - 对应列值的分区组合是HDFS的一组嵌套子目录
  - 例如
    - Partition columns: ds, ctry
    - HDFS 子目录对应 ds = 20090801, ctry = US
      - `/wh/pvs/ds=20090801/ctry=US`
    - HDFS 子目录对应 ds = 20090801, ctry = CA
      - `/wh/pvs/ds=20090801/ctry=CA`


```
CREATE TABLE logs (ts BIGINT, line STRING)
PARTITIONED BY (dt STRING, country STRING);
```

```
LOAD DATA LOCAL INPATH 'input/hive/partitions/file1'
INTO TABLE logs
PARTITION (dt='2001-01-01', country='GB');
```

```
/user/hive/warehouse/logs
├── dt=2001-01-01/
│   ├── country=GB/
│   │   ├── file1
│   │   └── file2
│   └── country=US/
│       └── file3
└── dt=2001-01-02/
    ├── country=GB/
    │   └── file4
    └── country=US/
        ├── file5
        └── file6
```

```
hive> SHOW PARTITIONS logs;
dt=2001-01-01/country=GB
dt=2001-01-01/country=US
dt=2001-01-02/country=GB
dt=2001-01-02/country=US
```

```
SELECT ts, dt, line
FROM logs
WHERE country='GB';
```

数据模型——桶

- 桶 Buckets
  - 对列基于哈希做拆分 – 并行
  - 在分区子目录里每个桶一个hdfs文件
  - 例如
    - Bucket column: user into 32 buckets
    - user hash bucket 0 的 HDFS 文件
      - `/wh/pvs/ds=20090801/ctry=US/part-00000`
    - user hash bucket 20 的 HDFS 文件
      - `/wh/pvs/ds=20090801/ctry=US/part-00020`


```
CREATE TABLE bucketed_users (id INT, name STRING)
CLUSTERED BY (id) INTO 4 BUCKETS;
```

```
CREATE TABLE bucketed_users (id INT, name STRING)
CLUSTERED BY (id) SORTED BY (id ASC) INTO 4 BUCKETS;
```

数据模型——外部表

- 外部表 external table
  - 指向已存的HDFS数据目录
  - 可创建表和分区 – 分区列就是对应外部目录的标记
  - 例如: 创建带分区的外部表

```
CREATE EXTERNAL TABLE pvs(userid int, pageid int, ds string, ctry string)
PARTITIONED ON (ds string, ctry string)
STORED AS textfile
LOCATION ‘/path/to/existing/table’
```

  - 例如: 为外部表增加分区

```
ALTER TABLE pvs
ADD PARTITION (ds= ‘20090801’ , ctry= ‘US’ )
LOCATION ‘/path/to/existing/partition’
```

表和外部表的区别

- Table 的创建过程和数据加载过程（这两个过程可以在同一个语句中完成），在加载数据的过程中，实际数据会被移动到数据仓库目录中；之后对数据对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。
- External Table 只有一个过程，加载数据和创建表同时完成（CREATE EXTERNAL TABLE ……LOCATION），实际数据是存储在 LOCATION 后面指定的 HDFS 路径中，并不会移动到数据仓库目录中。当删除一个 External Table 时，仅删除元数据，表中的数据不会真正被删除。

数据类型

- 原始类型 Primitive Types
  - integer types, float, string, date, boolean
- 集合嵌套 Nestable Collections
  - array<any-type> 如：arr[0]
  - map<primitive-type, any-type> 如：map[‘key’]
- 用户自定义类型 User-defined types
  - 包含属性的结构体 如:struct

Hive 查询语言

- SQL
  - From 子句里可包含子查询
  - 等值连接 Equi-joins
    - Inner
    - Left, Right, full Outer
  - 多表插入 Multi-table Insert
  - 同时多维度分组 Multi-group-by
- 抽样采样 Sampling

Hive 查询语言

- 扩展性
  - 可插入的并行脚本 Map-reduce scripts
  - 可插入的用户自定义函数 User Defined Functions
  - 可插入的用户自定义类型 User Defined Types
    - 复杂对象类型: List of Maps
  - 可插入的数据格式 Data Formats
    - 阿帕奇日志 Apache Log Format
    - 列存储格式 Columnar Storage Format

应用范例

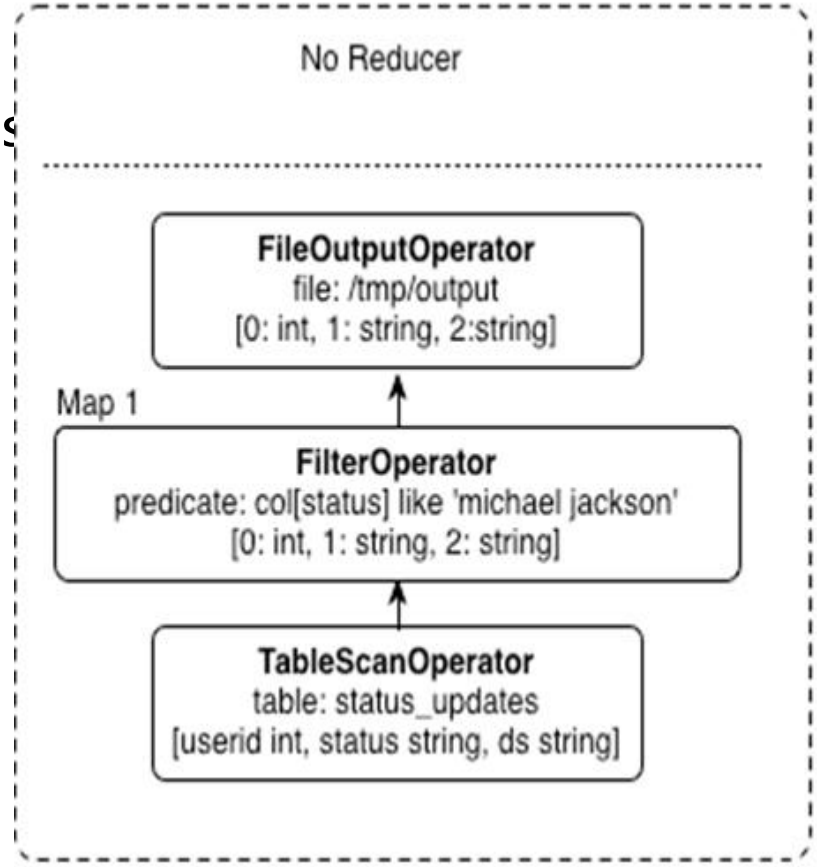
- **Status updates table:**
  - status_updates(userid int, status string, ds string)
- **Load the data from log files:**
  - LOAD DATA LOCAL INPATH ‘/logs/status_updates’ INTO TABLE status_updates PARTITION (ds=’2009-03-20’)
- **User profile table**
  - profiles(userid int, school string, gender int)
- **Load the data from MySQL udb potentially using sqoop***

查询例子 (Filter)

- 过滤出 status_updates 表中包含 ‘michael jackson’的记录
  - SELECT * FROM status_updates

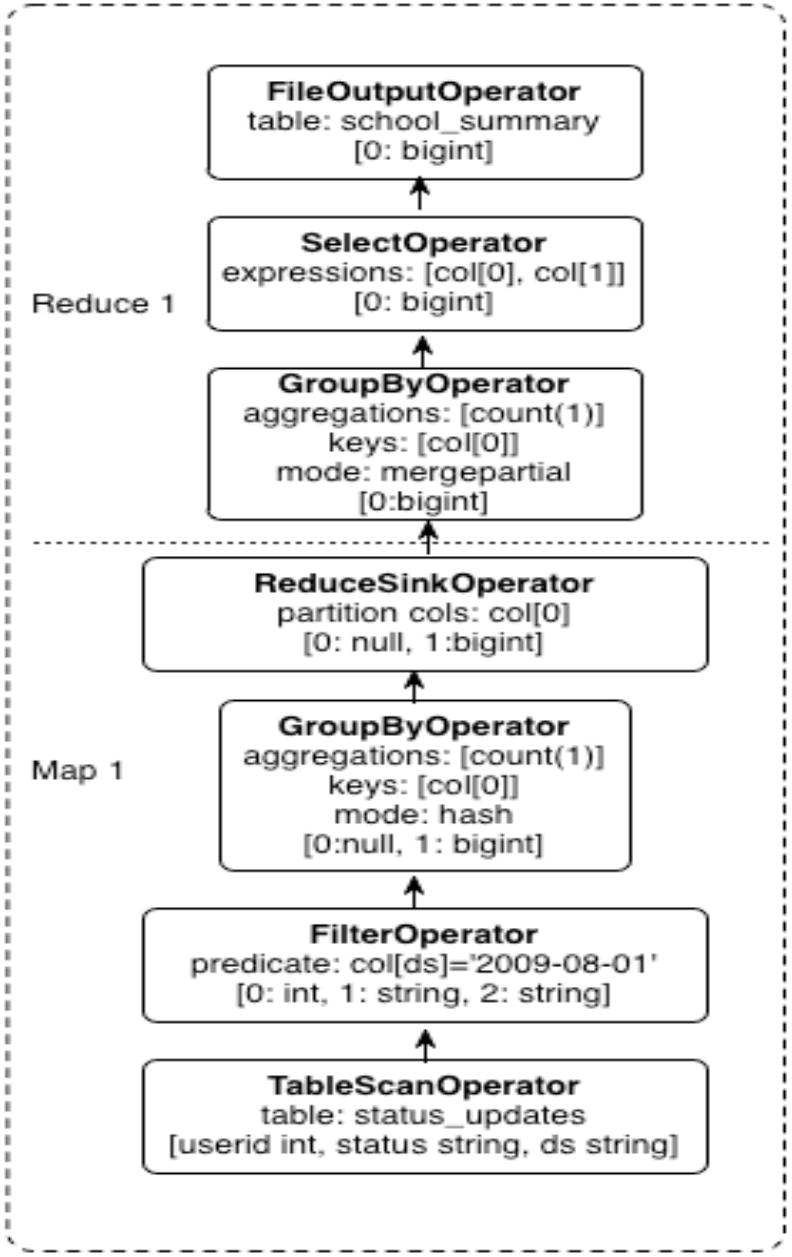
WHERE status LIKE

‘michael jackson’



查询例子(Aggregation)

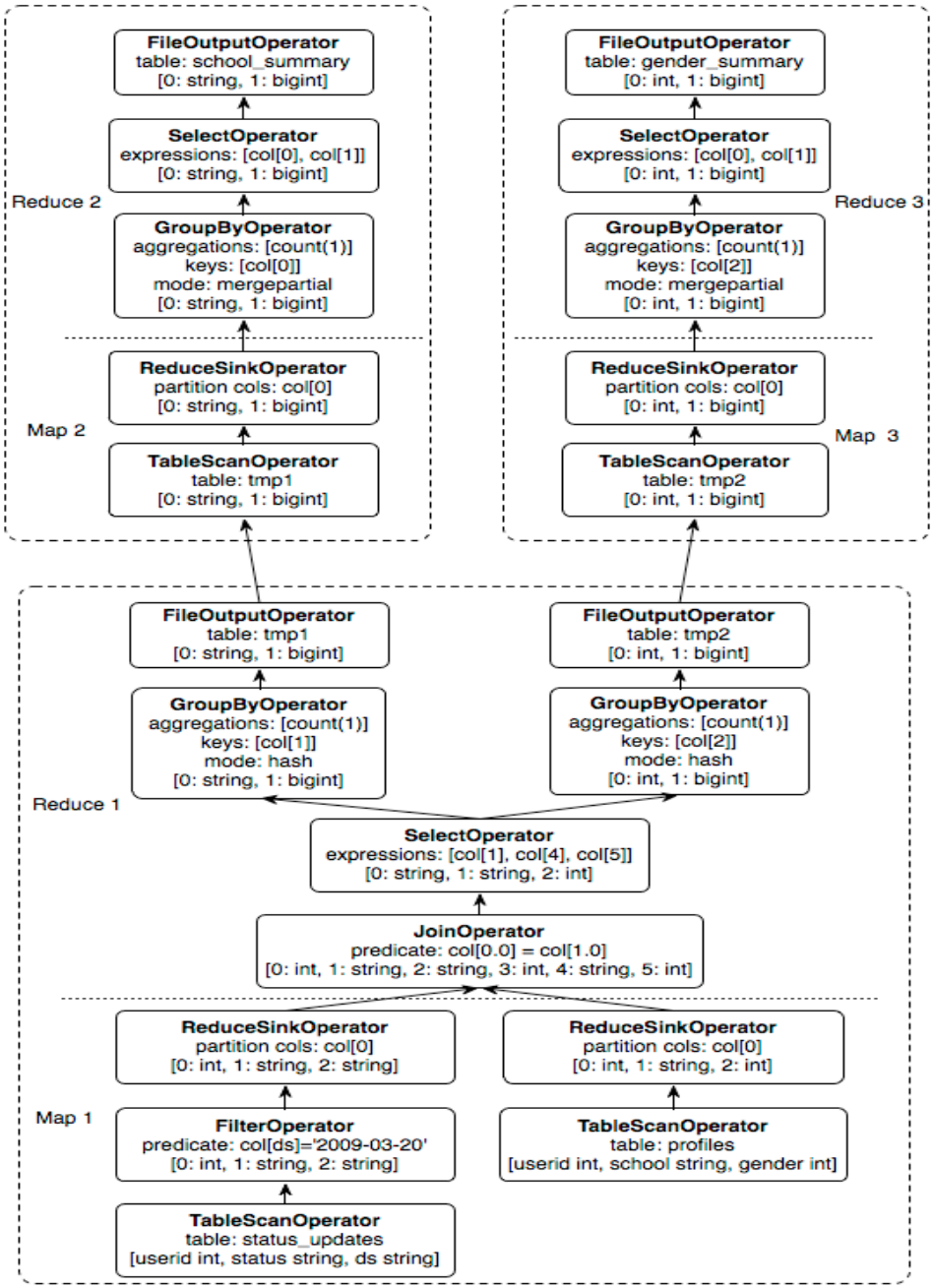
- 计算status_updates 表中指定日期的记录数
  - SELECT COUNT(1) FROM status_updates WHERE ds = ’ 2009-08-01’


FileOutputOperator
table: school_summary
[0: bigint]
SelectOperator
expressions: [col[0], col[1]]
[0: bigint]
Reduce 1
GroupByOperator
aggregations: [count(1)]
keys: [col[0]]
mode: mergepartial
[0:bigint]
ReduceSinkOperator
partition cols: col[0]
[0: null, 1:bigint]
GroupByOperator
aggregations: [count(1)]
keys: [col[0]]
mode: hash
[0:null, 1: bigint]
Map 1
FilterOperator
predicate: col[ds]='2009-08-01'
[0: int, 1: string, 2: string]
TableScanOperator
table: status_updates
[userid int, status string, ds string]

- 下面的查询例子包括了multi-group-by, joins 和 multi-table inserts.

查询例子(multi-group-by)


```
FROM (SELECT a.status, b.school, b.gender
      FROM status_updates a JOIN profiles b
           ON (a.userid = b.userid and
               a.ds='2009-03-20' )
      ) subq1
INSERT OVERWRITE TABLE gender_summary
                       PARTITION(ds='2009-03-20')
SELECT subq1.gender, COUNT(1)
GROUP BY subq1.gender
INSERT OVERWRITE TABLE school_summary
                             PARTITION(ds='2009-03-20')
SELECT subq1.school, COUNT(1)
GROUP BY subq1.school
```


FileOutputOperator
table: school_summary
[0: string, 1: bigint]
SelectOperator
expressions: [col[0], col[1]]
[0: string, 1: bigint]
Reduce 2
GroupByOperator
aggregations: [count(1)]
keys: [col[0]]
mode: mergepartial
[0: string, 1: bigint]
ReduceSinkOperator
partition cols: col[0]
[0: string, 1: bigint]
Map 2
TableScanOperator
table: tmp1
[0: string, 1: bigint]
FileOutputOperator
table: gender_summary
[0: int, 1: bigint]
SelectOperator
expressions: [col[0], col[1]]
[0: int, 1: bigint]
Reduce 3
GroupByOperator
aggregations: [count(1)]
keys: [col[2]]
mode: mergepartial
[0: int, 1: bigint]
ReduceSinkOperator
partition cols: col[0]
[0: int, 1: bigint]
Map 3
TableScanOperator
table: tmp2
[0: int, 1: bigint]
FileOutputOperator
table: tmp1
[0: string, 1: bigint]
FileOutputOperator
table: tmp2
[0: int, 1: bigint]
GroupByOperator
aggregations: [count(1)]
keys: [col[1]]
mode: hash
[0: string, 1: bigint]
GroupByOperator
aggregations: [count(1)]
keys: [col[2]]
mode: hash
[0: int, 1: bigint]
Reduce 1
SelectOperator
expressions: [col[1], col[4], col[5]]
[0: string, 1: string, 2: int]
JoinOperator
predicate: col[0.0] = col[1.0]
[0: int, 1: string, 2: string, 3: int, 4: string, 5: int]
ReduceSinkOperator
partition cols: col[0]
[0: int, 1: string, 2: string]
ReduceSinkOperator
partition cols: col[0]
[0: int, 1: string, 2: int]
Map 1
FilterOperator
predicate: col[ds]='2009-03-20'
[0: int, 1: string, 2: string]
TableScanOperator
table: profiles
[userid int, school string, gender int]
TableScanOperator
table: status_updates
[userid int, status string, ds string]

Hive 是一个开放的系统（1）

- 支持多种数据文件格式
  - Text File, Sequence File, …
- 多种 in-memory 数据类型
  - Java Integer/String, Hadoop IntWritable/Text …
- 用户提供的并行脚本 map/reduce scripts
  - 对于任何语言，采用 stdin/stdout 来转换数据

Hive 是一个开放的系统（2）

- 用户自定义函数 UDF
  - Substr, Trim, From_unixtime …
- 用户自定义聚合函数
  - Sum, Average …
- 用户自定义表转换函数
  - explode_map …
- 用户自定义窗口分析函数*
  - Sum, Average …

文件格式例子


- ```
  CREATE TABLE mylog (
    user_id BIGINT,
    page_url STRING,
    unix_time INT)
  STORED AS TEXTFILE;
  ```
- ```
  LOAD DATA INPATH '/user/myname/log.txt' INTO
  TABLE mylog;
  ```

内置的文件格式

	TEXTFILE	SEQUENCEFILE	RCFILE
Data type	**text only**	**text/binary**	**text/binary**
Internal Storage order	**Row-based**	**Row-based**	**Column-based**
Compression	**File-based**	**Block-based**	**Block-based**
Splitable*	**YES**	**YES**	**YES**
Splitable* after compression	**NO**	**YES**	**YES**

*** Splitable: Capable of splitting the file so that a single huge file can be processed by multiple mappers in parallel.**

什么情况下需要增加新的 File Format

- 特殊文件格式，不想在导入hive之前做转换
- 更有效的存储方式

怎样增加一个新的 File Format

- 参考例子
  contrib/src/java/org/apache/hadoop/hive/contrib/fileformat/base64
- **Base64TextFileFormat 支持二进制存储到文本, 通过 base64 encoding/decoding 同步计算实现.**
- ```
  CREATE TABLE base64_test(col1 STRING, col2 STRING)
    STORED AS
      INPUTFORMAT
  'org.apache.hadoop.hive.contrib.fileformat.base64.Base64TextInputFormat'
     OUTPUTFORMAT
  'org.apache.hadoop.hive.contrib.fileformat.base64.Base64TextOutputFormat';
  ```
  ```

SerDe 例子


```
CREATE TABLE mylog (
    user_id BIGINT,
    page_url STRING,
    unix_time INT)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t';
```

```
CREATE table mylog_rc (
    user_id BIGINT,
    page_url STRING,
    unix_time INT)
ROW FORMAT SERDE

  'org.apache.hadoop.hive.serde2.columnar.ColumnarSerDe
  '
STORED AS RCFILE;
```

SerDe

- SerDe 是 serialization/deserialization的缩写. 它控制行的格式.
- 序列化格式 Serialized format:
  - 带分隔符的格式 Delimited format (tab, comma, ctrl-a …)
  - Thrift Protocols
  - ProtocolBuffer*
- 反序列化格式 Deserialized (in-memory) format:
  - Java Integer/String/ArrayList/HashMap
  - Hadoop Writable classes
  - 用户自定义Java 类 (Thrift, ProtocolBuffer*)

- * ProtocolBuffer 尚未支持.

内置的 SerDes

	LazySimpleSerDe	LazyBinarySerDe (HIVE-640)	BinarySortable SerDe
serialized format	**delimited**	**proprietary binary**	**proprietary binary sortable***
deserialized format	**LazyObjects***	**LazyBinaryObjects***	**Writable**
	ThriftSerDe (HIVE-706)	RegexSerDe	ColumnarSerDe
serialized format	**Depends on the Thrift Protocol**	**Regex formatted**	**proprietary column-based**
deserialized format	**User-defined Classes, Java Primitive Objects**	**ArrayList<String>**	**LazyObjects***

*** LazyObjects: deserialize the columns only when accessed.**

*** Binary Sortable: binary format preserving the sort order.**

什么情况下需要增加一个新的 SerDe

- 特殊序列化格式，并且用户不想在加载到hive之前做转换
- 有更有效率的序列化方式存储数据

如何为文本数据增加新的SerDe

- 参考例子
  contrib/src/java/org/apache/hadoop/hive/contrib/serde2/RegexSerDe.java
- **RegexSerDe** 使用用户定义正则表达式反序列化数据.
- CREATE TABLE apache_log(host STRING,
  identity STRING, user STRING, time STRING, request STRING,
  status STRING, size STRING, referer STRING, agent STRING)
  ROW FORMAT SERDE
  'org.apache.hadoop.hive.contrib.serde2.RegexSerDe'
  WITH SERDEPROPERTIES (
  "input.regex" = "([^]*) ([^]*) ([^]*) (-|\\[[^\\]]*\\])
  ([^ \"]*|\"[^\"]*\") (-|[0-9]*) (-|[0-9]*)(?: ([^
  \"]*|\"[^\"]*\") ([^ \"]*|\"[^\"]*\"))?",
  "output.format.string" = "%1$s %2$s %3$s %4$s %5$s %6$s
  %7$s %8$s %9$s")
  STORED AS TEXTFILE;

如何为二进制数据提供SerDe

- **参考例子**

  contrib/src/java/org/apache/hadoop/hive/contrib/serde2/thrift (HIVE-706)

  serde/src/java/org/apache/hadoop/hive/serde2/binarysortable

- **CREATE TABLE mythrift_table**

```
ROW FORMAT SERDE
  'org.apache.hadoop.hive.contrib.serde2.thrift.ThriftSerDe'
WITH SERDEPROPERTIES (
 "serialization.class" = "com.facebook.serde.tprofiles.full",
 "serialization.format" =
  "com.facebook.thrift.protocol.TBinaryProtocol");
```

- **标注：列信息通过 SerDe 类提供.**

UDF例子

- add jar build/ql/test/test-udfs.jar;
- CREATE TEMPORARY FUNCTION testlength AS 'org.apache.hadoop.hive.ql.udf.UDFTestLength';
- SELECT testlength(src.value) FROM src;
- DROP TEMPORARY FUNCTION testlength;
- UDFTestLength.java:

```
package org.apache.hadoop.hive.ql.udf;
public class UDFTestLength extends UDF {
  public Integer evaluate(String s) {
    if (s == null) {
      return null;
    }
    return s.length();
  }}
```

更有效率的UDF

- 避免创建新对象查看器
- 避免转码 UTF-8 encoding/decoding
- UDFTestLength.java:

```
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
public class UDFTestLength extends UDF {
    IntWritable result = new IntWritable();
    public IntWritable evaluate(Text s) {
        if (s == null) {
            return null;
        }
        result.set(countUDF8Characters(s));
        return result;
    }
}
```

装载UDF

- add jar build/contrib/hive_contrib.jar;
- CREATE TEMPORARY FUNCTION example_add AS
  'org.apache.hadoop.hive.contrib.udf.example.UDFExampleAdd';
- SELECT example_add(1, 2) FROM src;
- SELECT example_add(1.1, 2.2) FROM src;
- UDFExampleAdd.java:

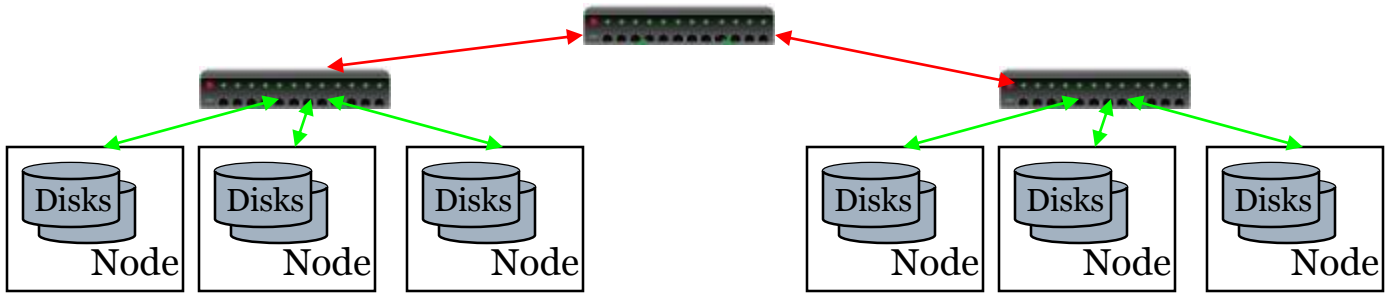
```
public class UDFExampleAdd extends UDF {
  public Integer evaluate(Integer a, Integer b) {
    if (a == null || b == null) return null;
    return a + b;
  }
  public Double evaluate(Double a, Double b) {
    if (a == null || b == null) return null;
    return a + b;
  }
}
```

Hive可扩展和优化修改

- 现有接口扩展
  - UDF 如url_to_map
  - UDAF
  - UDTF
  - UDWF*
  - Row Fomat (SerDe)
  - File Format (hadoop Stored) 如:thrift /protocol buffers
- 内核（query language）修改
  - 命令扩展
  - SQL词义分析、语义分析、语法解析扩展 （antlr v3）分区等
  - 编译执行任务操作修改(query plan/task/work/operator)
  - 优化修改
  - 权限修改*
- 客户端/工具支持
  - 客户端调用 (thrift c/s)/ thrift 日志服务器存储到hadoop
  - Sql生成页面工具*
  - 调度支持 入库/执行hql/数据导出至oracle

Hive在facebook的应用

- 应用范围
  - 报表
  - 在线分析
  - 数据挖掘
  - 。。。
- 系统规模
  - 640节点
  - 2000T容量
  - 每节点8核、4TB存储

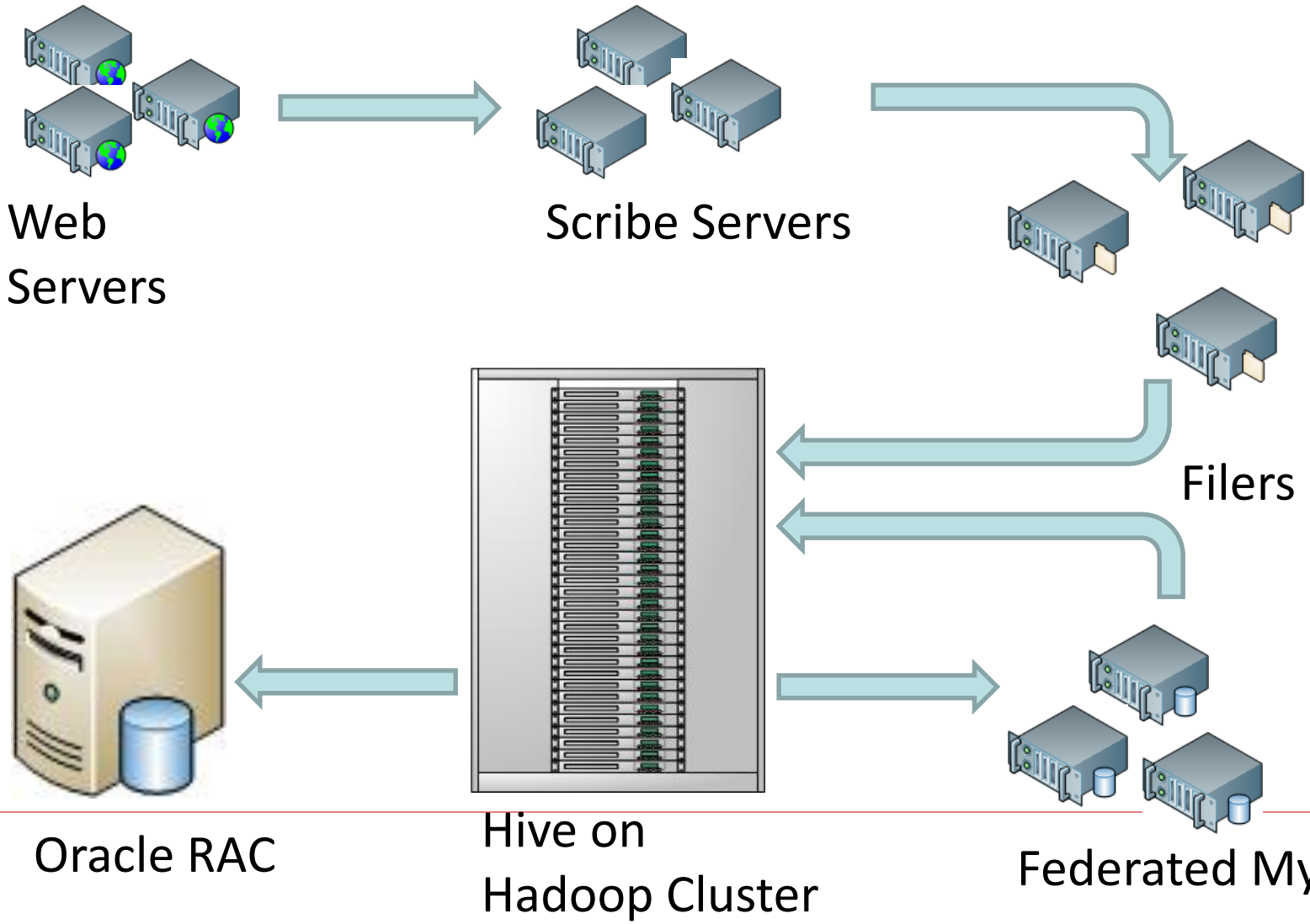


Node
=
DataNode
+
Map-
Reduce

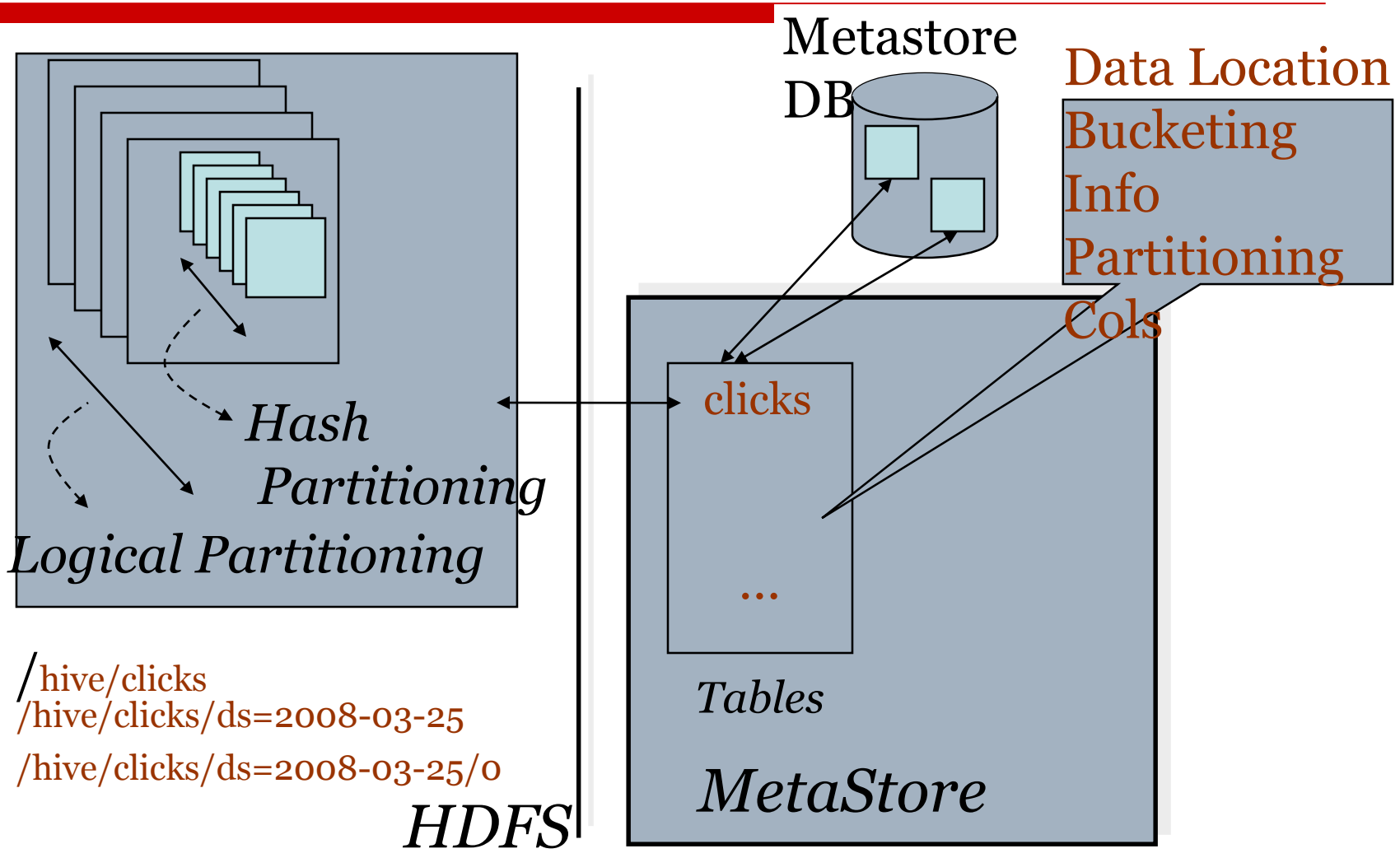
1 Gigabit

4-8 Gigabit

Facebook数据仓库



Facebook数据模型



使用情况

数据量统计：

- 系统总容量：~2.4PB
- 每日新增数据量：~15TB
  - 6TB 未压缩原始日志
  - 4TB 未压缩的多维数据（每天重新加载）
- 压缩率：~5x (gzip, more with bzip)

使用情况统计：

- 3200 jobs/day with 800K tasks(map-reduce tasks)/day
- 55TB of compressed data scanned daily
- 15TB of compressed output data written to hdfs
- 80 compute minutes/day

谢谢！