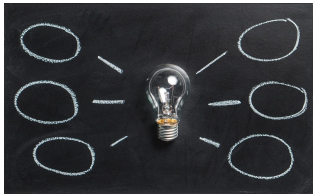
Julian Kunkel

Big Data SQL using Hive



Outline

Learning Objectives

- Compare the execution model of SQL in an RDBMS with Hive
- Justify the features of the ORC format
- Apply a bloom filter on example data
- Describe how tables are generally mapped to the file system hierarchy and optimizations
- Describe how data sampling can be optimizing via the mapping of tables on HDFS
- Sketch the mapping of a (simple) SQL query to a MapReduce job

Hive Overview

- Hive: Data warehouse functionality on top of Hadoop/HDFS
  - Compute engines: Map/Reduce, Tez, Spark
  - Storage formats: Text, ORC, HBASE, RCFile, Avro
  - Manages metadata (schemes) in RDBMS (or HBase)
- Access via: SQL-like query language HiveQL
  - Similar to SQL-92 but several features are missing
  - Limited transactions, subquery and views
- Query latency: 10s of seconds to minutes (new versions: sub-seconds)

Features

- Basic data indexing (compaction and bitmaps)
- User-defined functions to manipulate data and support data-mining
- Interactive shell: `hive`
- Hive Web interface (simple GUI data access and query execution)
- WebHCat API (RESTful interface)

Data Model [22]

Data types

- Primitive types (int, float, strings, dates, boolean)
- Bags (arrays), dictionaries
- Derived data types (structs) can be defined by users

Data organization

- Table: Like in relational databases with a schema
  - The Hive data definition language (DDL) manages tables
  - **Data is stored in files on HDFS**
- Partitions: table key determining the mapping to directories
  - Reduces the amount of data to be accessed in filters
  - Example key: /ds=<date> for table T
  - Predicate T.ds='2017-09-01' searches for files in /ds=2017-09-01/ directory
- Buckets/Clusters: Data of partitions are mapped into files
  - Hash value of a column determines partition

Managing Schemas with HCatalog

- Schemas are stored in Hive's Metastore
- HCatalog is a layer providing a relational data abstraction [29]
  - Tables incl. metadata can be accessed regardless of file format
  - Exposes schemas and allows data access by other apps
    - Can be used by Pig, Spark, MapReduce jobs
    - Streaming API for Flume and Storm
  - ⇒ Create a table in HCatalog (pointing to files) and use the schema elsewhere
- Can send a Java Message upon integration of new data (for workflows)
- Provides the REST API WebHCat
- Command line interface: `hcat`

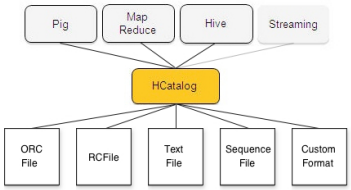


Figure: Source [29]

Accessing Schemas and Data [45]

hive

- Interactive SQL shell
- Allows execution from the command line

```
hive -S -e <SQL COMMAND>
Example: hive -S -e 'SELECT * from stud'

hive -f <SQL SCRIPT>
```

hcat

- Executes Hives Data Definition Language (DDL)

```
hcat -e <COMMAND>
Example: hcat -e 'CREATE TABLE tbl(a INT);'

hcat -f <script.hcatalog> -- runs the script of DDL commands
```

Outline

Execution of Hive Queries with Hadoop

1. A user submits a query to a user interface
2. The driver receives the SQL query
3. The compiler parses the SQL query
   - Case 1: It changes the data definition or is very simple
   - **Case 2**: It requires to actually process/ingest data (typical case)
4. Information about the data types (incl. de/serializer) and (directory) structure of the tables is queried from the metastore (HCatalog)
5. The query plan generator translates the query into a execution plan
   - Creates DAG (of MapReduce jobs or Tez graph)
   - Optimizes the execution, e.g., via cost-based evaluation
   - May use intermediate files on HDFS to process the query
   - Goal: reduce intermediate files
6. The execution engine runs the DAG and handles dependencies
7. The result is stored on HDFS and read by the UI

Hive Architecture and Query Execution in Hadoop

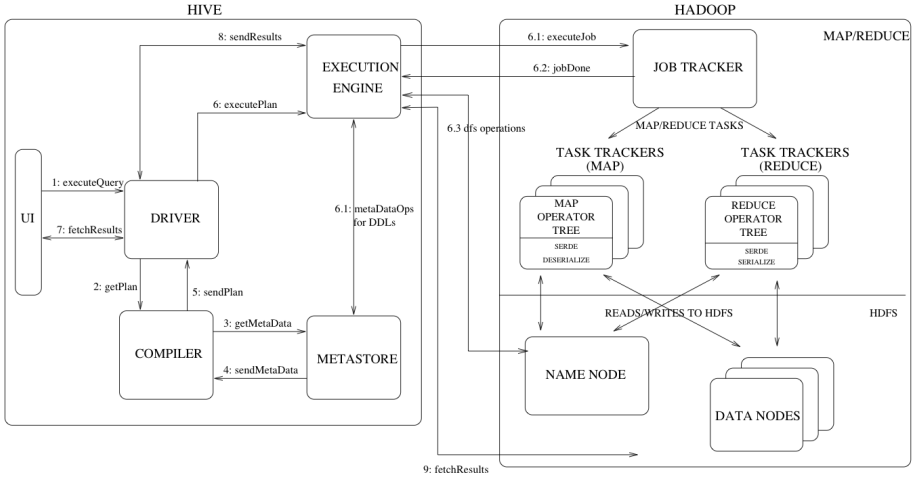


Figure: Architecture. Source: Design – Apache Hive [22]

Mapping Queries to "MapReduce" jobs

- The SQL query must be translated to MapReduce (or Tez) jobs
- Consider t1 to be stored in a single file

SQL: Count the number of keys which values are above 100

```
1 SELECT key, count(*) FROM t1 WHERE t1.value > 100 GROUP BY key;
```

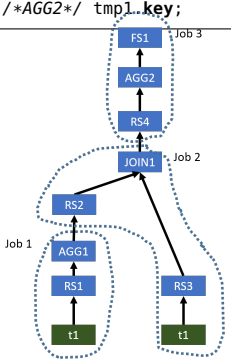
Solution

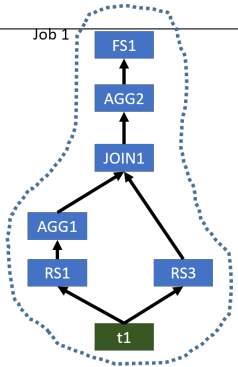
- Map: filter rows with value > 100, emit tuple (key, 1)
- Reduce: sum (key, <counts>) and emit key, sum
- A single MapReduce Job will do here... but not in general
  - How about subqueries, for example?
- The mapping to files determines efficiency

Example Query Execution and Optimization [23]

SQL: Count the number of keys which values are above the average

```
1 SELECT tmp1.key, count(*)
2 FROM (SELECT key, avg(value) AS avg /* Read select 1 = RS1 */
3       FROM t1
4       GROUP BY /*AGG1*/ key) tmp1
5   JOIN /*JOIN1*/ t1 ON (tmp1.key = t1.key)
6   WHERE t1.value > tmp1.avg
7   GROUP BY /*AGG2*/ tmp1.key;
```





Mapping Queries to "MapReduce" jobs

Solution for optimized query

- This optimization is actually tricky but easy to express in procedural language...
- Map: emit tuple (key, value)
- Reduce (key, values)

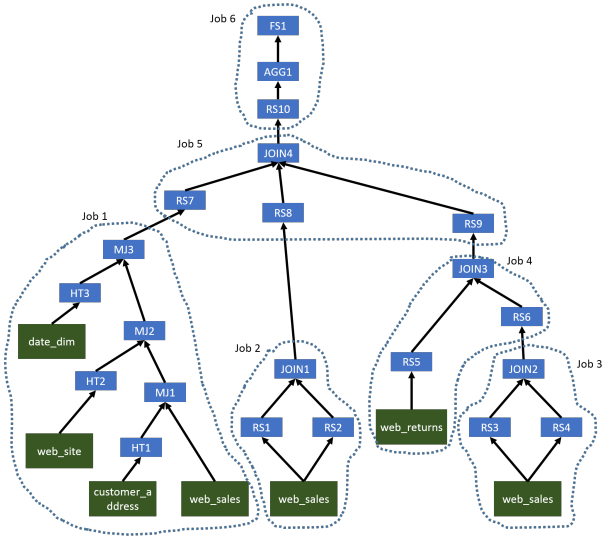
```
1 mean = sum(values)/count(values)
2 aboveAvg = 0
3 for each value in values:
4   if value > mean:
5       aboveAvg++
6 emit(key, aboveAvg)
```

Example Query Execution and Optimization [23]

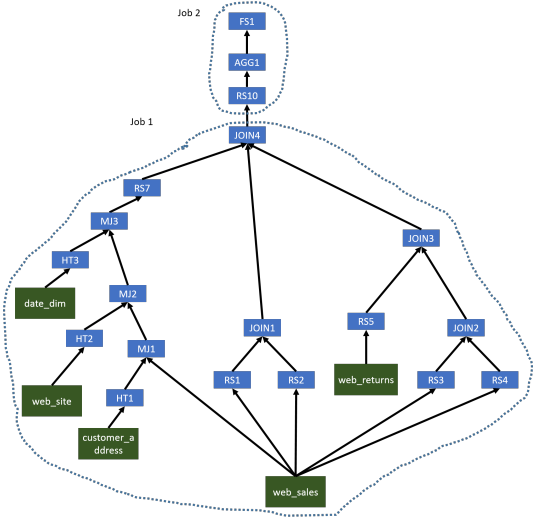
Count orders, their profit and costs for a given company and state

```
1  SELECT count(distinct ws1.ws_order_number) as order_count,
2         sum(ws1.ws_ext_ship_cost) as total_shipping_cost,
3         sum(ws1.ws_net_profit) as total_net_profit
4  FROM web_sales ws1
5  JOIN /*MJ1*/ customer_address ca ON (ws1.ws_ship_addr_sk = ca.ca_address_sk)
6  JOIN /*MJ2*/ web_site s ON (ws1.ws_web_site_sk = s.web_site_sk)
7  JOIN /*MJ3*/ date_dim d ON (ws1.ws_ship_date_sk = d.d_date_sk)
8  LEFT SEMI JOIN /*JOIN4*/ (SELECT ws2.ws_order_number as ws_order_number
9       FROM web_sales ws2 JOIN /*JOIN1*/ web_sales ws3
10      ON (ws2.ws_order_number = ws3.ws_order_number)
11      WHERE ws2.ws_warehouse_sk <> ws3.ws_warehouse_sk) ws_wh1
12 ON (ws1.ws_order_number = ws_wh1.ws_order_number)
13 LEFT SEMI JOIN /*JOIN4*/ (SELECT wr_order_number
14     FROM web_returns wr
15     JOIN /*JOIN3*/ (SELECT ws4.ws_order_number as ws_order_number
16                     FROM web_sales ws4 JOIN /*JOIN2*/ web_sales ws5
17                     ON (ws4.ws_order_number = ws5.ws_order_number)
18                     WHERE ws4.ws_warehouse_sk <> ws5.ws_warehouse_sk) ws_wh2
19     ON (wr.wr_order_number = ws_wh2.ws_order_number)) tmp1
20 ON (ws1.ws_order_number = tmp1.wr_order_number)
21 WHERE d.d_date >= '2001-05-01' and d.d_date <= '2001-06-30' and
22       ca.ca_state = 'NC' and s.web_company_name = 'pri';
```

Example Query: Original Execution Plan



Example Query: Optimized Execution Plan



Hive DAG Execution on TEZ

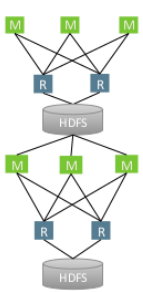
```
SELECT ss.ss_item_sk, ss.ss_quantity, avg_price, inv.inv_quantity_on_hand
FROM (select avg(ss_sold_price) as avg_price, ss_item_sk, ss_quantity_sk from store_sales
group by ss_item_sk) ss
JOIN inventory inv
ON (inv.inv_item_sk = ss.ss_item_sk);
```

Hive : Broadcast Join

Hive – MR

Store Sales scan. Group by and aggregation.

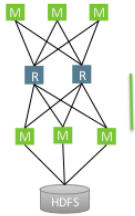
Inventory and Store Sales (aggr.) output scan and shuffle join.



Hive – Tez

Store Sales scan. Group by and aggregation reduce size of this input.

Inventory scan and Join



Broadcast edge

Beyond MapReduce – LLAP: Live Long and Process [38, 39]

Features identified to be necessary to improve performance [38]

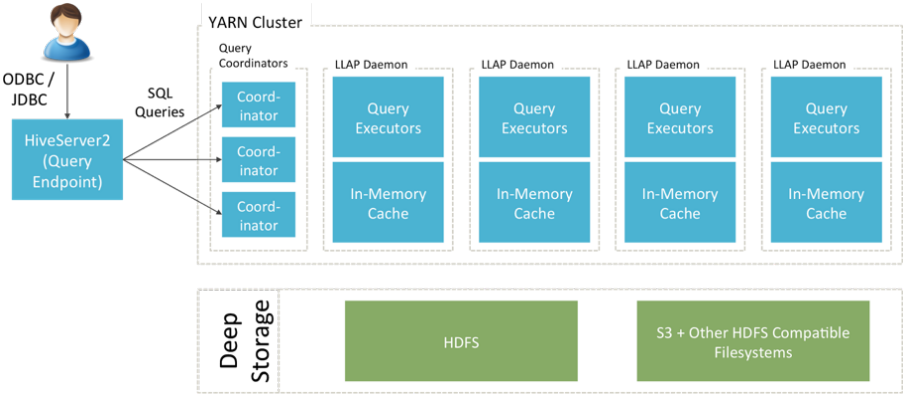
- Asynchronous spindle-aware IO
- Pre-fetching, caching of column chunks
- Multi-threaded JIT-friendly operator pipelines

Strategy

- Optional (LLAP) long-living daemon can be run on worker nodes
- LLAP daemon receives tasks (instead of HDFS) and query fragments
  - A processing engine that forwards I/O requests to HDFS
  - It survives single queries, processes data and caches
  - Multiple queries can be processed at the same time
- Compress cached data
- LLAP is integrated into YARN and can use Slider for fast deployment
  - Supports container delegation, i.e., workload depending resources

LLAP Benefits and Architecture [38,39]

- Significantly reduced startup time (sub second for some queries)
- Scalable architecture
- Caching of compressed data reduces memory footprint
- Other frameworks such as Pig can use the daemon
- Together with Apache Ranger, it enables column-level access control



Outline

File Formats: Overview

- Hive supports typical HDFS file types
  - SequenceFile, CSV, JSON, AVRO
  - Can access data stored in HBASE via HBaseStorageHandler [30]
- A format is not necessarily well suited for queries
  - A query may require a full scan of all content!
- Requirements of Hive lead to ORC file format
- To see table properties/file format: `describe extended` $< table >$

ORC: Optimized Row Columnar [25]

Features

- Light-weight index stored within the file
- Compression based on data type
- Concurrent reads of the same file
- Split files without scanning for markers
- Support for adding/removal of fields
- Partial support of table updates
- Partial ACID support (if requested by users)

ORC Files [25]

- Stripe: group of row data
- Postcript: contains file metadata
  - Compression parameters
  - Size of the file footer
- Index data (per stripe & row group)
  - Min and max values
  - Bloom filter (to pre-filter matches)
  - Row position
- Compression of blocks: RLE, ZLIB, SNAPPY, LZO
- Tool to output ORC files:
  `hive -orcfiledump`

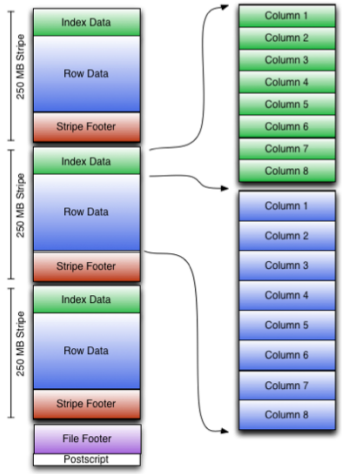


Figure: Source: [25]

Row groups are by default 10k rows of one column

Group Work

- Think about potential reasons behind the file layout
- Particularly, how will this be stored in HDFS and processed with MapReduce?
- Time: 10 min
- Organization: breakout groups - please use your mic or chat

Solution

Data Layout

- Stripe: alignment to HDFS blocks, each stripe is one HDFS block
- Footer: metadata necessary to understand
- Index: speedup lookups, i.e., do we have to read any data
- Columnwise, because we often just need parts of the data
  - Reading data you do not need is waste
  - Reading large blocks of data triggers efficient I/O
- Compression reduces amount of data that must be read
- Blocks: allow to add arbitrary amounts of rows/tuples

MapReduce

- Each block is processed by a RecordReader
- For each row invoke Map()
- Hive will build and compile a MapReduce job on the fly with suitable signature

ORC Table Creation [25]

Example Creation

```
create table Addresses (
  name string,
  street string,
  city string,
  state string,
  zip int
) stored as orc tblproperties ("orc.compress"="SNAPPY");
```

Table properties

key	default	notes
orc.compress	ZLIB	high level compression (one of NONE, ZLIB, SNAPPY)
orc.compress.size	262,144	number of bytes in each compression chunk
orc.stripe.size	268435456	number of bytes in each stripe
orc.row.index.stride	10,000	number of rows between index entries (must be >= 1000)
orc.create.index	true	whether to create row indexes
orc.bloom.filter.columns	""	comma separated list of column names for bloom filters
orc.bloom.filter.fpp	0.05	false positive probability for bloom filter (must >0.0 and <1.0)

Bloom Filters [10]

Characteristics

- Goal: Identify if an element is a member of a set with n elements
- Space-efficient probabilistic data structure
  - Probabilistic: Allow false positives but not false negatives
  - Space-efficient: fixed bit array to test on arbitrary sized n

Data structure

- Allocate array B of m bits, 1 indicates a key is present
- Provide k hash functions $h_i(key) \rightarrow 1, ..., m$
- Example with 14 bits

0	0	0	0	1	1	0	0	0	0	1	1	1	0

Bloom Filter Algorithms [10]

Insertion Algorithm

- Insert a member e to the set B with (n-1) elements
  - Apply all k hash functions and set $B(h_i(e)) = 1$

Test for (likely) presence

- Apply all k hash functions and check all $B(h_i(e))$
- If one is 0, we know the element is not part of the set

Space-efficiency

- For a fixed m, the false positive rate ϵ increases with n (number of set elements)
- Classic number of bits per element is $m = 1.44 \cdot log_2(1/\epsilon)$
  - e.g., 9.6 bits for $\epsilon = 1\%$ and 14.4 for 0.1%
- Optimal number of hash functions: $k = (m/n) \cdot ln(2)$
  - e.g., 6.6 = 7 hash functions (for 9.6 bits per element, $m = 9.6 \cdot n$)

Updates and Transactions [37]

- Typically individual rows are read-ONLY
- ORC file format allows ACID on row level
  - Requires several configuration parameters for Hive to be set [37]
- Limited (single query) multirow transactions on bucketed tables
  - Requires in HiveQL: SET `transactional=true`
- Use cases for updates/transactions
  - Ingesting streaming data
  - Extending a slow-pacing dimension, e.g., adding a row with a new product
  - Correcting wrong data, e.g., deletion of customers
- Design
  - Updates/deletes are stored in delta files
  - Delta files are applied upon read
  - Periodic file compaction in background
  - Lock manager for transactions

Outline

HiveQL

- HiveQL = SQL alike declarative language
- Mostly SQL-92 compliance, some features from SQL 2003, 2011
- Extensions for big data processing with HDFS

SQL extensions

- Importing/Exporting of data from HDFS
  - Identical semantics on all supported file formats
  - Declaration of compression
- Control for parallelism (map/reduce, file striping)
- Support for complex data types
  - Compound data types
  - Semi-structured column formats, e.g., JSON
- Efficient sampling (picking of subsets) of data
- Integration of external tools for streaming (map/reduce)

Example HiveQL Statements Beyond SQL (see [21])

Creating a table

```
1 CREATE TABLE student (name STRING, matrikel INT, birthday date)
2 COMMENT 'This is a table with all students'
3 ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' COLLECTION ITEMS TERMINATED BY ':' MAP KEYS TERMINATED
      ↪ BY '#' LINES TERMINATED BY '\n'
4 STORED AS TEXTFILE;
```

With optimizations: sorted order, clustering and compression

```
1 CREATE TABLE studentp (name STRING, matrikel INT)
2 PARTITIONED by (birthday date) -- remember: each partition receives its own directory
3 CLUSTERED BY(matrikel) SORTED BY(name) INTO 32 BUCKETS
4 STORED AS SEQUENCEFILE LOCATION '/wr/kunkel/testfile';
```

Example SQL query: For each student identify all lectures attended

```
1 SELECT s.name, s.matrikel, collect_set(l.name) as lecs FROM studlec sl JOIN student s ON sl.matrikel=s.matrikel JOIN lectures l ON sl.lid=l.id
      ↪ GROUP BY s.name, s.matrikel
```

Importing Data and HDFS Structure

Loading data from HDFS or the local file system (this moves the data!)

```
1 LOAD DATA LOCAL INPATH './student.txt' OVERWRITE INTO TABLE student;
```

File student.txt

```
1 "kunkel",22222,2021-06-08
2 "hpda",22223,2021-06-08
```

Importing data into the partioned table

```
1 INSERT OVERWRITE TABLE studentp PARTITION(birthday) SELECT name, matrikel, birthday FROM student;
```

Checking the file structure in HDFS

```
1  $ hadoop fs -ls $FS/wr/kunkel/testfile -- the table is a directory
2  drwxr-xr-x   - hdfs hdfs 0 2021-09-10 16:13 hdfs://cluster/kunkel/testfile/birthday=2021-06-08
3  $ hadoop fs -get $FS/kunkel/testfile/birthday=2021-06-08 -- the partitions
4  $ ls -lah birthday\=2008-06-08/    -- shows the buckets
5  -rw-r--r-- 1 kunkel wr  87 Sep 10 16:17 000000_0
6  ...
7  -rw-r--r-- 1 kunkel wr  87 Sep 10 16:17 000016_0
8  -rw-r--r-- 1 kunkel wr 112 Sep 10 16:17 000017_0 -- here is one student
9  -rw-r--r-- 1 kunkel wr 114 Sep 10 16:17 000018_0 -- here the other
10 -rw-r--r-- 1 kunkel wr  87 Sep 10 16:17 000019_0
11 ...
12 -rw-r--r-- 1 kunkel wr  87 Sep 10 16:17 000032_0
```

Exporting Data

Exporting tables to HDFS [21]

```
INSERT OVERWRITE [LOCAL] DIRECTORY directory
  [ROW FORMAT row_format] [STORED AS file_format]
  SELECT ...
```

Example: Outputing data to HDFS via DIRECTORY

```
INSERT OVERWRITE DIRECTORY 'myDir' STORED AS TEXTFILE
SELECT * FROM tbl;
```

Example: Outputing data to a local file via DIRECTORY

```
INSERT OVERWRITE LOCAL DIRECTORY 'myDir'
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n'
STORED AS TEXTFILE
SELECT * FROM tbl;
```

Datatypes and Conversion Between Formats

- Hive supports complex data structures
- Data conversion can be done (theoretically) easily

Creating a complex table and converting data [24]

```
CREATE TABLE as_avro(string1 STRING,
  int1 INT,
  tinyint1 TINYINT,
  smallint1 SMALLINT,
  bigint1 BIGINT,
  boolean1 BOOLEAN,
  float1 FLOAT,
  double1 DOUBLE,
  list1 ARRAY<STRING>,
  map1 MAP<STRING,INT>,
  struct1 STRUCT<sint:INT,sboolean:BOOLEAN,sstring:STRING>,
  union1 uniontype<FLOAT, BOOLEAN, STRING>,
  enum1 STRING,
  nullableint INT,
  bytes1 BINARY)
STORED AS AVRO; -- The AVRO file format is useful for exchanging data

-- importing data from old_table into the new table
```

Processing External Data From TextFiles

- An external „table“ allows specifying a schema for an existing file
- Multiple schemes can be created to the same file
- The `location` keyword points to the parent **directory**
  - Files in this directory are considered
  - Remember: a MapReduce job creates multiple files in a directory
- `External` keyword allows to keep files when the table is dropped

Creating an external table with a Regex

```
1 -- Here we use a regex to define the columns
2 CREATE EXTERNAL TABLE wiki(text STRING)
3   ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe' with
4   SERDEPROPERTIES ("input.regex" = "^([^,]*)$") LOCATION "/user/bigdata/wiki-clean";
5 DROP table wiki; -- does not delete the table (directory)
```

Using Partitions for Importing Data

- Partitions are convenient for importing new data
- Create an external table with partitions
- Copy data to the location

Example: convert data from a staging area [26]

```
hive> CREATE EXTERNAL TABLE page_view_staging(viewTime INT,
  userid BIGINT, page_url STRING, referrer_url STRING,
  ip STRING COMMENT 'IP Address of the User', c STRING COMMENT 'country of
      ↪ origination') COMMENT 'This is the staging page view table'
ROW FORMAT DELIMITED FIELDS TERMINATED BY '44' LINES TERMINATED BY '12'
STORED AS TEXTFILE LOCATION '/user/staging/page_v';


$ hadoop dfs -put /tmp/pv_us-2008-06-08.txt /user/staging/page_v
-- Import data by converting it
hive> FROM page_view_staging pvs
INSERT OVERWRITE TABLE page_view PARTITION(dt='2008-06-08', c='US')
SELECT pvs.viewTime, pvs.userid, pvs.page_url, pvs.referrer_url, pvs.ip
WHERE pvs.c = 'US';
```

Interesting Collection Functions [31]

Functions are available to process collections

- collect_set(col): Bag all elements together in a set, eliminates duplicates
- collect_list(col): Bag elements together in a list
- Statistics: percentile(), corr(), covar()
- ntile(n): Divide the ordered partition into n groups and assign the bucket number to each row in the partition
- histogram_numeric(col, b): Compute a histogram with b bins

Users can provide additional functions

Further functions are available at `https://github.com/klout/brickhouse`.
Basically development of this stopped around 2016.

Built-in Table-Generating Functions (UDTF) [21]

- Table generating functions create multiple rows from one value
  - explode(Array): One row per array element
  - explode(Map): One row with key/value pair for each pair
  - inline($Array < Struct >$): explode array of structures
  - stack($n, v_1, v_2, ..., v_k$): explode v_i into n rows of k/n elements
  - json_tuple(json string, k1, k2, ...): maps the values of the keys to columns
  - ...
- Custom functions are possible

Examples

```
1 SELECT explode(xMap) AS (mapKey, mapValue) FROM mapTable;
2
3 -- key1   value1
4 -- key2   value2
5 -- key3   value3
```

Lateral View Create Virtual Tables [21]

Lateral view form a virtual table joining rows with existing columns

- `LATERAL VIEW udtf(expression) tableAlias AS columnAlias`

Example: Student table

```
1 CREATE TABLE student (name STRING, matrikel INT, birthday date, attends array<int>) ROW FORMAT
      ↪ DELIMITED FIELDS TERMINATED BY ',' collection items terminated by '|' STORED AS TEXTFILE;
2 LOAD DATA LOCAL INPATH './source/student-array.txt' OVERWRITE INTO TABLE student;
3
4 SELECT name, lecture FROM student LATERAL VIEW explode(attends) tblX AS lecture;
5 -- "kunkel" 1
6 -- "kunkel" 2
7 -- "kunkel" 3
8 -- "hans" 2
```

student-array.txt

```
1 "kunkel",22222,2008-06-08,1|2|3
2 "hans",22223,2008-06-08,2
```

Acessing Semi-Structured JSON Data

- To cope for semi-structured data JSON is supported

```
1 -- Create a table with one column containing rows of JSON
2 CREATE EXTERNAL TABLE mstudent ( value STRING ) LOCATION  '/user/kunkel/student-table';
3 LOAD DATA LOCAL INPATH 'stud.json' OVERWRITE INTO TABLE mstudent;
4
5 -- Create a lateral view, i.e., a virtual table from the unpacked data using json_tuple
6 SELECT b.matrikel, b.name FROM mstudent s LATERAL VIEW json_tuple(s.value, 'matrikel', 'name') b as
      ↪ matrikel, name;
7 -- 22 Fritz Musterman M.
8 -- 23 Nina Musterfrau F.
```

stud.json

```
1 { "matrikel":22,"name":"Fritz Musterman M.","female":false,"birth":"2000-01-01","private":{"urls":["http://xy", "http://z"]},
      ↪ "other":{"mykey":"val"} }
2 { "matrikel":23,"name":"Nina Musterfrau F.","female":true,"birth":"2000-01-01","private":{},"other":{} }
```

Acessing Column-Based JSON Data

- A JSON serializer allows for accessing JSON columns directly
- No LATERAL VIEW needed

```
1  -- Access to column based JSON
2  DROP TABLE mstudent;
3  -- A new serializer
4  ADD JAR /home/kunkel/bigdata/hive/json-serde-1.3-jar-with-dependencies.jar;
5
6  CREATE EXTERNAL TABLE mstudent ( matrikel INT, name STRING, female BOOLEAN, birth STRING, private struct<urls:array<STRING>>, other
       ↪ MAP<STRING,STRING>) ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe' LOCATION  '/user/kunkel/student-table-json';
7  LOAD DATA LOCAL INPATH 'stud.json' OVERWRITE INTO TABLE mstudent;
8  select * from mstudent;
9  -- 22 Fritz Musterman M.  false 2000-01-01  {"urls":["http://xy","http://z"]}}
10 -- 23 Nina Musterfrau F.  true  2000-01-01  {"urls":null}
11
12 -- Access array from the struct and the map
13 SELECT private.urls[1], other["mykey"] FROM mstudent;
14 -- http://z val
15 -- NULL -1
```

Sampling of Data

- Sampling allows execution of queries to a subset of (random) rows
- e.g., pick 5 random rows using basic SQL:

```
1 select * from TABLE order by rand() limit 5;
```

  - Requires full table scan, thus, is slow
- Hive variants: Table sampling and block sampling
  - They use the internal structures of buckets and input splits
  - To change the random seed: `SET hive.sample.seednumber=<INTEGER>`

Tablesampling

- Partition data using bucket sampling on the column matrikel

```
1 SELECT * FROM student TABLESAMPLE(BUCKET 1 OUT OF 2 ON matrikel) tblAlias;
```

  - This will only scan a subset of the partitions, if the column is clustered
- Use all columns for the sampling by using “ON rand()”

```
1 SELECT * FROM student TABLESAMPLE(BUCKET 2 OUT OF 2 ON rand()) s;
```

Sampling of Data (2)

Block sampling

- Sample on physical HDFS blocks/splits, either pick data block or not
- At least X percent or bytes

```
1 SELECT * FROM student TABLESAMPLE(0.1 PERCENT) tblAlias;
2 SELECT * FROM student TABLESAMPLE(10M) tblAlias;
```

- Take first N rows from each input split

```
1 SELECT * FROM student TABLESAMPLE(1 ROWS) s;
```

Compression in Hive

- Support transparent reading of commpressed Text files
- File extension must be, e.g., .gz or .bz2
- Compression of tables, intermediate stages
  - Controlled via properties [34]

Benefit of compression

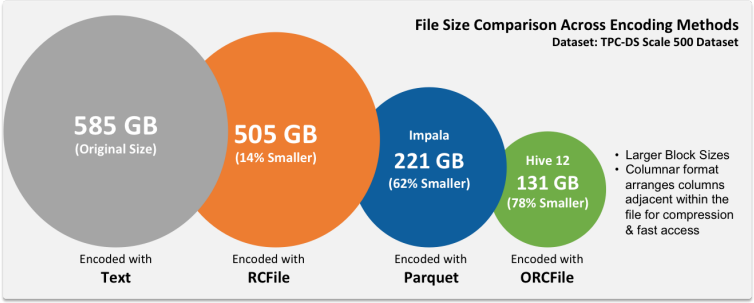


Figure: Source: [32]

Embedding Hive with External Tools [36]

- Users can integrate custom mappers and reducers in the data stream
- TRANSFORM clause integrates scripts
  - Alternatively MAP, REDUCE clauses, but prefer TRANSFORM
- Script output can be automatically casted (otherwise type: string)
- Number of mappers/reducers can be controlled
- Mapping to reducers is controlled by DISTRIBUTE BY clause

Basic Example

```
1 SELECT TRANSFORM(name,matrikel) USING '/bin/cat' AS (name STRING, anonymized_matrikel STRING) from
      ↪ student
```

- Input: name, matrikel, Output: name and anonymized_matrikel

Embedding Hive with External Tools (2)

- Using an external script requires to add it to the cluster cache
  1. Add [*FILE*|*JAR*|*ARCHIVE*] < *FILE* >: add to the cluster cache for later use
  2. list [*FILE*|*JAR*|*ARCHIVE*]: show cluster cache
- Input and output formats can be redefined (default tab)

Example with a bash script

- Input: name, matrikel
- Output: user-defined format with line count, word count, character count

```
1 ADD file wc.sh;
2 SELECT TRANSFORM(name,matrikel) USING 'wc.sh' AS (lines INT, words INT, chars INT) ROW FORMAT
      ↪ DELIMITED FIELDS TERMINATED BY '\t' LINES TERMINATED BY '\n' from student
3 -- 2  4 28
```

wc.sh

```
1 #!/bin/bash
2 # Run wc to count words, then remove sparse spaces and replace with tab
3 /usr/bin/wc|sed "s/ [ ]*/\t/g"| sed "s/^\t//"
```

Embedding Hive with External Tools (3)

Example with R [33]

```
ADD FILE script.r;
SET mapred.reduce.tasks=2;
FROM (SELECT state, city FROM cities DISTRIBUTE BY state) t1
 INSERT OVERWRITE DIRECTORY 'R_example'
 REDUCE state,city USING 'script.r' AS state,count;
```

```
#!/usr/bin/env RScript
f <- file("stdin") ## read the contents of stdin
open(f) ## open the handle on stdin
my_data = read.table(f) ## read stdin as a table
# count the number of occurrences in column 1
my_data_count=table(my_data$V1)
# format the output so we see column1<tab>count
write.table(my_data_count,quote = FALSE,row.names = FALSE,col.names = FALSE,sep = " ")
```

Debugging

- Use EXPLAIN clause
- Show table properties: `DESCRIBE EXTENDED` $<$ *table* $>$
- Show generic properties and environment vars: SET

```
EXPLAIN SELECT * from student;
OK
Plan not optimized by CBO. # Cost-based optimizer!

Stage-0
   Fetch Operator
      limit:-1
      Select Operator [SEL_1]
         outputColumnNames:["_col0","_col1","_col2","_col3"]
         TableScan [TS_0]
            alias:student
```

Summary

- Hive provides an SQL interface to files in Hadoop
  - Idea: avoid time consuming data ingestion (often: simply move data)
  - Create schemas on demand based on need
  - It uses MapReduce or Tez for data processing
  - Processing may require a full scan of files
- Original Hive was a HDFS client + metadata service, LLAP is server based
- HCatalog offers a relational abstraction to many file formats
  - Create a schema once, use it everywhere
  - Data can be organized in partitions and buckets/clusters
  - ORC files are optimized for Hive
  - Import of data not necessary
- HiveQL provides an extended SQL syntax
  - Schema enable complex data types
  - Processing of JSON possible
  - MapReduce jobs can be embedded

Bibliography

10 Wikipedia

21 Hive Language Manual. https://cwiki.apache.org/confluence/display/Hive/LanguageManual

22 Design – Apache Hive. https://cwiki.apache.org/confluence/display/Hive/Design

23 https://cwiki.apache.org/confluence/display/Hive/Correlation+Optimizer

24 https://cwiki.apache.org/confluence/display/Hive/AvroSerDe

25 https://cwiki.apache.org/confluence/display/Hive/LanguageManual+ORC

26 https://cwiki.apache.org/confluence/display/Hive/Tutorial

27 http://www.adaltas.com/blog/2012/03/13/hdfs-hive-storage-format-compression/

28 http://hortonworks.com/blog/hive-cheat-sheet-for-sql-users/

29 https://cwiki.apache.org/confluence/display/Hive/HCatalog+UsingHCat

30 https://cwiki.apache.org/confluence/display/Hive/HBaseIntegration

31 https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF

32 http://hortonworks.com/blog/orcfile-in-hdp-2-better-compression-better-performance/

33 http://hortonworks.com/blog/using-r-and-other-non-java-languages-in-mapreduce-and-hive/

34 https://cwiki.apache.org/confluence/display/Hive/Configuration+Properties

36 https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Transform

37 https://cwiki.apache.org/confluence/display/Hive/Hive+Transactions

38 https://cwiki.apache.org/confluence/display/Hive/LLAP

39 http://hortonworks.com/blog/llap-enables-sub-second-sql-hadoop/

45 https://cwiki.apache.org/confluence/display/Hive/HCatalog+CLI

Enabling Compression

Commands on the shell to enable compression

```
-- Compress output data (e.g., when inserting into a new table)
set hive.exec.compress.output = true;
-- Should we compress intermediate data?
set hive.exec.compress.intermediate = true;

-- For ORC: compression strategy, speed or compression (rate)
set hive.exec.orc.compression.strategy = "speed"

-- None, BLOCK-wise or individual RECORD-level compression
-- For sequential files:
set io.seqfile.compression.type = BLOCK;
set mapred.output.compression.codec = org.apache.hadoop.io.compress.SnappyCodec;

-- On storage, ORC tables are compressed blocks, but they require a schema
CREATE TABLE t (
...
) STORED AS orc tblproperties ("orc.compress"="SNAPPY","orc.row.index.stride"="1000");
```

Compressing a File on the Fly

Converting a text file to a compressed SequenceFile

```
1 CREATE EXTERNAL TABLE wiki(text string) LOCATION "/user/bigdata/wiki-clean";
2 CREATE TABLE wiki_c (line STRING) STORED AS SEQUENCEFILE LOCATION
      ↪ "/user/bigdata/wiki-clean-seq.gz";
3
4 SET hive.exec.compress.output=true;
5 SET io.seqfile.compression.type=BLOCK;
6 INSERT OVERWRITE TABLE wiki_c SELECT * FROM wiki;
```