Custom Mapper

Custom Mapper:
Create weekday_mapper.py:

```python
import sys
import datetime

for line in sys.stdin:
    line = line.strip()
    userid, movieid, rating, unixtime = line.split('	')
    weekday =
datetime.datetime.fromtimestamp(float(unixtime)).isoweekday()
print '	'.join([userid, movieid, rating, str(weekday)])
```

Example Contd

/* get the top 10 meme per school */
REDUCE subq2.school, subq2.meme, subq2.cnt USING 'top10.py'
AS (school,meme,cnt)
FROM (SELECT subq1.school, subq1.meme, COUNT(t) AS cnt
FROM (MAP b.school, a.status USING 'meme-extractor.py'
     AS (school,meme)
     FROM status, updates a JOIN profiles b ON (a.userid = b.userid)
   ) subq1
GROUP BY subq1.school, subq1.meme
DISTRIBUTE BY school, meme
SORT BY school, meme, cnt desc
) subq2
Hive - Open Source Data Warehousing Solution for BI

Build on top of hadoop, used at facebook
supports queries in SQL like interface called HiveQL

CSCI- 572 Summer 201 - Karan Singh
Applications

- aggregation of click count
- complex measures of user engagement
- application API usage patterns
Related Software

Google - Sazwall
Yahoo - Pig
IBM Research - JAQL
Microsoft - SCOPE
Architecture

- CLI, WEBUI, Thrift
- MetaStore the system catalog
- Driver maintain life cycle of Hiveql program, on receiving the request from thrift server it creates a session handle used to keep track of statistics
- The Compiler is invoked by the driver upon receiving a HiveQL statement. The compiler translates this statement into a plan which consists of a DAG of map-reduce jobs.
- The driver submits the individual map-reduce jobs from the DAG to the Execution Engine in a topological order. Hive currently uses Hadoop as its execution engine.
- DML (Data Manipulation Language) to load and insert data and for data manipulation
- DDL to create tables with SeDer(serialization format) partition and buckets
- select, project, join, aggregate, union all
- support for custom map reduce scripts
- sql compiled into map reduce jobs
- updating and deleting rows in tables now supported
- User defined column transformations and aggregations (UDF/UDAF) in java supported
- custom map reduce scripts via streaming supported (has overhead of converting to strings)
- rank aggregation not supported
- to execute queries from command line: bin/hive -e 'quoted query string'
- add custom mapper, reducer via transform keyword
Hive MetaStore

- system catalog information
- specified during table creation
- distinguishes hive from traditional solutions like pig and scope
- Database - namespace for tables - the database default is used for tables with no user supplied database name
- Table - Metadata for table contains list of columns and their types, owner, storage and SerDe information. It can also contain any user supplied key and value data; this facility can be used to store table statistics in the future.
- Storage information includes location of the table’s data in the underlying file system, data formats and bucketing information.
- SerDe metadata includes the implementation class of serializer and deserializer methods and any supporting information required by that implementation. All this information can be provided during the creation of table.
- Partition - Each partition can have its own columns and SerDe and storage information.
- HDFS not suitable as is designed for sequential scans and not random access (uses store like MySql, NFS) etc
- MetaStore (trunk/metastore) - This component implements the metadata server which is used to hold all the information about tables and partitions that are in the warehouse.
Tables

- have corresponding HDFS directory
- data serialized and stores in files HDFS directory
- serialization format of table stored in system catalog
- external tables also supported over NFS, HDFS
- have partitions which determine distribution of data within sub directories of the table directory
- If T is partitioned on columns ds and ctry, then data with a particular ds value 20090101 and ctry value US, will be stored is files within the directory/wh/T/ds=20090101/ctry=US.
- partition can be further divided into buckets
- data types supported (integer, float, strings, dates, booleans, array, map)
- users can also define custom types
Query Processor (trunk/ql)

- This component implements the processing framework for converting SQL to a graph of map/reduce jobs and also the execution time framework to run those jobs in the order of dependencies.
- Parse and SemanticAnalysis (ql/parse) - This component contains the code for parsing SQL, converting it into Abstract Syntax Trees, converting the Abstract Syntax Trees into Operator Plans and finally converting the operator plans into a directed graph of tasks which are executed by Driver.java.
- Optimizer (ql/optimizer) - This component contains some simple rule based optimizations like pruning non referenced columns from table scans (column pruning) that the Hive Query Processor does while converting SQL to a series of map/reduce tasks.
- Plan Components (ql/plan) - This component contains the classes (which are called descriptors), that are used by the compiler (Parser, SemanticAnalysis and Optimizer) to pass the information to operator trees that is used by the execution code.
- MetaData Layer (ql/metadata) - This component is used by the query processor to interface with the MetaStore in order to retrieve information about tables, partitions and the columns of the table. This information is used by the compiler to compile SQL to a series of map/reduce tasks.
- Map/Reduce Execution Engine (ql/exec) - This component contains all the query operators and the framework that is used to invoke those operators from within the map/reduces tasks.
- Hadoop Record Readers, Input and Output Formatters for Hive (ql/io) - This component contains the record readers and the input, output formatters that Hive registers with a Hadoop Job.
- Sessions (ql/session) - A rudimentary session implementation for Hive.
- Type interfaces (ql/typeinfo) - This component provides all the type information for table columns that is retrieved from the MetaStore and the SerDes.
- Hive Function Framework (ql/udf) - Framework and implementation of Hive operators, Functions and Aggregate Functions. This component also contains the interfaces that a user can implement to create user defined functions.
- Tools (ql/tools) - Some simple tools provided by the query processing framework. Currently, this component contains the implementation of the lineage tool that can parse the query and show the source and destination tables of the query.
Compiler

- generates Metadata operations for DDL
- HDFS statements for LOAD
- map reduce for insert statements
- parser transforms query string into parse tree
- Semantic query analyzer that transforms parse tree to block based internal query and verifies it using type data store in metadata
- logical plan generator generates a logical plan from the parse tree
- physical plan actually converts the original plan into a physical plan DAG of map reduce jobs
Optimizer

- converts multiple joins into multiway joins
- adds repartition operators that mark the boundary between map and reduce
- prunes columns and pushes predictions close to reduce data transferred
- prunes partitions not needed by query
- for sampling queries prunes buckets not needed
loads status updates for facebook which is log rotated for analysis in a table
status updates(userid int,status string,ds string)
profiles(userid int,school string,gender int)

LOAD DATA LOCAL INPATH "/logs/status_updates"
INTO TABLE status_updates PARTITION (ds='2009-03-20')

/*Count the number of statuses per gender grouped by school*/
school summary(school string,cnt int,ds string)
gender summary(gender int,cnt int,ds string)

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
Example Contd

/* get the top 10 meme per school */
REDUCE subq2.school, subq2.meme, subq2.cnt USING ‘top10.py’
AS (school, meme, cnt)

FROM (SELECT subq1.school, subq1.meme, COUNT(1) AS cnt
     FROM (MAP b.school, a.status USING ‘meme-extractor.py’
           AS (school, meme)
     FROM status_updates a JOIN profiles b
     ON (a.userid = b.userid)
   ) subq1
GROUP BY subq1.school, subq1.meme
DISTRIBUTE BY school, meme
SORT BY school, meme, cnt desc
) subq2
Custom Mapper:
Create weekday_mapper.py:

```python
import sys
import datetime

for line in sys.stdin:
    line = line.strip()
    userid, movieid, rating, unixtime = line.split('\t')
    weekday =
    datetime.datetime.fromtimestamp(float(unixtime)).isoweekday()
    print '\t'.join([userid, movieid, rating, str(weekday)])
```
Custom Mapper Example contd

CREATE TABLE u_data_new ( 
  userid INT, 
  movieid INT, 
  rating INT, 
  weekday INT) 
ROW FORMAT DELIMITED 
FIELDS TERMINATED BY '	';

add FILE weekday_mapper.py;

INSERT OVERWRITE TABLE u_data_new 
SELECT 
TRANSFORM (userid, movieid, rating, unixtime) 
USING 'python weekday_mapper.py' 
AS (userid, movieid, rating, weekday) 
FROM u_data;

SELECT weekday, COUNT(*) 
FROM u_data_new 
GROUP BY weekday;
ToDo

- supports only a subset of SQL
- optimizer is a naive based can add cost based and adaptive based support
- need to improve scan performance by looking at columnar storage
- using text instead or string improved performance
- adding more support for JDBC and ODBC
- multiquery optimization and n-way joins