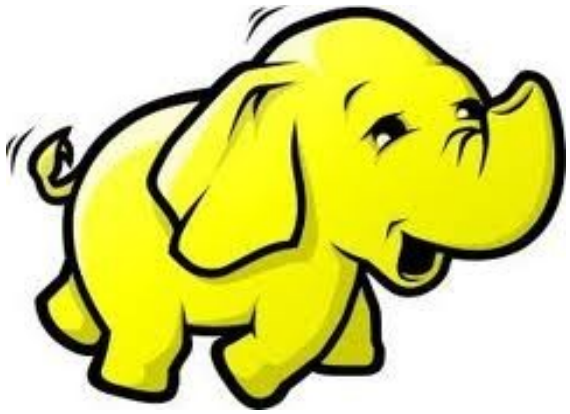
Facebook-Hive POC



For online Hadoop training, send mail to neeraj.ymca.2k6@gmail.com

Agenda

Hive usage @ Facebook
Sample click_history data
Create table in Hive
Static Partitioning
Dynamic Partitioning
Mixed Partitioning
UDF(User Defined Function)
Temporary UDF
Permanent UDF
Use case example
Hive scripts

Hive usage @ Facebook

Users click on various advertisements on Facebook

Details are captured in the form of text file.

These text file can be processed using Hadoop/Hive

Hadoop/Hive uses Map-Reduce framework to process large amout of data

Facebook is extensively using Hadoop/Hive for business intelligence

Facebook Click history

Facebook records click history of Users

The details captured are

User-id
Day
Time
Location of user (Country)
Advertisement category

Sample click history data

Sandeep	Friday	5:30 pm	INDIA	Games
Ekta	Friday	4:23 am	BRAZIL	Technology
Shravan	Tuesday	9:51 am	INDIA	Life Insurance
Sandeep	Monday	9:24 pm	USA	Garments
Sonia	Monday	7:19 am	USA	Travel
Santosh	Tuesday	8:58 pm	DUBAI	Games
Shravan	Tuesday	6:16 am	INDIA	Games
John	Sunday	1:12 pm	DUBAI	Life Insurance
Geeta	Monday	0:24 am	UK	Games
Sonia	Monday	5:32 pm	SPAIN	Travel
Geeta	Monday	7:02 pm	UK	Technology
Sandeep	Sunday	4:31 pm	BRAZIL	Travel
Sonia	Friday	0:50 am	INDIA	Games

Create table in Hive

Table creation

```
CREATE TABLE click_history (
userid STRING,
day STRING,
time STRING,
country STRING,
ad_category STRING
) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
STORED AS TEXTFILE;
```

Load data into Table

```
LOAD DATA LOCAL INPATH
“/home/neeraj/PDF/Hive/Hive_POC_Sample_Data.txt”
INTO TABLE click_history;
```

Partitioning

Behind every Hive table, there is a text file stored in HDFS.

Hive queries are converted into MR jobs which deals with text files.

We need to scan the complete file to find out the result.

We can use **Partitioning** to increase the performance of Hive queries.

There are two types of partitioning in Hive

Static Partitioning

Dynamic Partitioning

Static Partitioning

Create a partitioned table

```
CREATE TABLE click_history_part (
userid STRING,
day STRING,
time STRING,
country STRING
) PARTITIONED BY ( ad_category STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
STORED AS TEXTFILE;
```

[This will create a table with 1 partition column]

Load data into partitioned table

```
INSERT INTO TABLE click_history_part PARTITION(ad_category='Games')
SELECT userid, day, time, country FROM click_history
WHERE ad_category='Games';
```

[This will load the selected data into the specified partition i.e. click_history_part/Games directory]

Dynamic Partitioning

By default, dynamic partitioning is disabled in Hive

We need to set these property in Hive shell in order to enable Dynamic Partitioning

```
set hive.exec.dynamic.partition=true;
set hive.exec.dynamic.partition.mode=nonstrict;
```

Load data into partitioned table using dynamic partitioning

```
INSERT INTO TABLE click_history_part
PARTITION( ad_category )
SELECT * FROM click_history ;
```

[This will load the data into all possible partitions depending upon unique values of ad-category]

Mixed Partition

Create a table with 2 partitioned column

```
CREATE TABLE click_history_part2 (
userid STRING,
day STRING,
time STRING
)  PARTITIONED BY ( country STRING, ad_category STRING )
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
STORED AS TEXTFILE;
```

[This will create a table with 2 partitioned column]

Load data into partitioned table

```
INSERT INTO TABLE click_history_part2
PARTITION( country='INDIA', ad_category )
SELECT userid, day, time, ad_catagory FROM click_history
WHERE country='INDIA' ;
```

[This will load the data into all possible partitions depending upon the combination of “INDIA” & unique values of ad_category]

UDF (User Defined Function)

SELECT fname from employee;

Neeraj
Shravan
MurugaN

SELECT UPPER(fname) from employee;

NEERAJ
SHRAVAN
MURUGAN

There are 2 types of UDF

Temporary User defined Function

Permanent User defined Function

GetCurrency UDF

```
package udf;

import java.util.HashMap;
import java.util.Map;
import org.apache.hadoop.hive.ql.exec.Description;
import org.apache.hadoop.hive.ql.exec.UDF;

@Description(name = "getCurrency", value = "This function can be used to find currency
                        for the given country")

public class GetCurrency extends UDF
{
     static Map<String, String> country_currency = new HashMap<String, String>();

     static
     {

          Country_currency.put("INDIA", "INR");
          country_currency.put("USA", "USD");
          country_currency.put("UK", "GBP");
          country_currency.put("SPAIN", "EUR");
          country_currency.put("DUBAI", "AED");
          country_currency.put("BRAZIL", "BRL");
          country_currency.put("CANADA", "CAD");
     }

     public String evaluate(String country)
     {
          return country_currency.get(country);
     }

}
```

Temporary UDF

Steps to use Temporary UDF's.

1. Create a JAR file of your UDF.

2. Run the command on Hive CLI..

```
Hive> add jar /home/neeraj/PDF/Hive/udf_example.jar ;
```

3. Run the command on Hive CLI..

```
Hive> CREATE TEMPORARY FUNCTION getCurrency as 'udf.GetCurrency';
```

Note: You need to run the above 2 commands in each Hive session]

```
SELCT country, getCurrency( country ) from click_history LIMIT 5;
```

```
INDIA    INR
SPAIN    EUR
USA      USD
DUBAI    AED
BRAZIL   BRL
```

Permanent UDF

We can also create permanent UDF which can be used in any Hive session without executing those 2 commands.

Below are the steps to use permanent UDF.

1. Extract the content of hive-exec-0.8.1.jar from HIVE_HOME/lib.
2. Modify the java file **FunctionRegistry.java**.
3. Add the line **registerUDF("getCurrency", udf.GetCurrency.class);**
4. Add the class files of your UDF with proper directory structure.
5. Create the new jar with Modified FunctionRegistry.java & your UDF classes
6. Put the updated jar hive-exec-0.8.1.jar into HIVE_HOME/lib.

Use case Example

Find out the number of facebook users from each country (along with country's currency) who clicked on Technology advertisement

```
SELECT country, getCurrency(country), count(*)
FROM click_history
WHERE ad_category='Technology'
GROUP BY country;
```

Terminal

```
hive> SELECT country, getCurrency(country), count(*)
    > FROM click_history
    > WHERE ad_category='Technology'
    > GROUP BY country;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapred.reduce.tasks=<number>
Starting Job = job_201309211610_0002, Tracking URL = http://myubuntu:50030/jobdetails.jsp?jobid=job_201309211
610_0002
Kill Command = /home/neeraj/local_cluster_home/hadoop-1.2.1/libexec/../bin/hadoop job  -Dmapred.job.tracker=m
yubuntu:9001 -kill job_201309211610_0002
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2013-09-21 16:36:32,273 Stage-1 map = 0%,  reduce = 0%
```

Use case output

```
2013-09-21 16:36:32,273 Stage-1 map = 0%,  reduce = 0%
2013-09-21 16:36:39,341 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.22 sec
2013-09-21 16:36:40,349 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.22 sec
2013-09-21 16:36:41,361 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.22 sec
2013-09-21 16:36:42,371 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.22 sec
2013-09-21 16:36:43,454 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.22 sec
2013-09-21 16:36:44,461 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.22 sec
2013-09-21 16:36:45,471 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.22 sec
2013-09-21 16:36:46,480 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.22 sec
2013-09-21 16:36:47,489 Stage-1 map = 100%,  reduce = 33%, Cumulative CPU 2.22 sec
2013-09-21 16:36:48,496 Stage-1 map = 100%,  reduce = 33%, Cumulative CPU 2.22 sec
2013-09-21 16:36:49,515 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 4.74 sec
2013-09-21 16:36:50,523 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 4.74 sec
2013-09-21 16:36:51,531 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 4.74 sec
2013-09-21 16:36:52,547 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 4.74 sec
MapReduce Total cumulative CPU time: 4 seconds 740 msec
Ended Job = job_201309211610_0002
MapReduce Jobs Launched:
Job 0: Map: 1  Reduce: 1   Accumulative CPU: 4.74 sec   HDFS Read: 1865653 HDFS Write
Total MapReduce CPU Time Spent: 4 seconds 740 msec
OK
BRAZIL  BRL     1444
DUBAI   AED     1301
INDIA   INR     1421
SPAIN   EUR     1452
UK      GBP     1329
USA     USD     1369
Time taken: 34.55 seconds
hive>
```

Hive Scripts

```
./hive -e "SELECT * FROM employee WHERE fname='Neeraj'"
```

[It will run the quoted command without opening Hive CLI (terminal)]

```
./hive -e "SELECT * FROM employee WHERE fname='Neeraj'" >
/home/neeraj/local_cluster_home/hive_queries_op/usecase1_op.txt
```

[It will run the quoted command & save the output of query in the specified file]

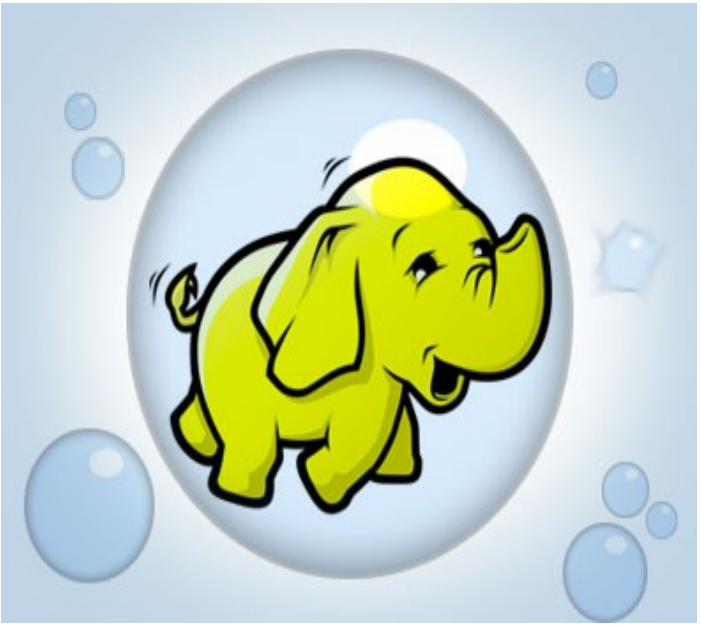
```
./hive -f /home/neeraj/local_cluster_home/hive_queries/usecase1.sql
```

[This will run the commands written in usecase1.sql]

```
./hive -f /home/neeraj/local_cluster_home/hive_queries/usecase1.sql  >
/home/neeraj/local_cluster_home/hive_queries_op/usecase1_op.txt
```

[This will run the commands written in usecase1.sql & save the output of query in the specified file]

...Thanks...



For online Hadoop training, send mail to neeraj.ymca.2k6@gmail.com