A Benchmark for Hive, PIG and Hadoop[1]

Yuntao Jia, Zheng Shao

July 12th, 2009

1 Introduction

We are proposing a simple benchmark to test the performances of Hive [1], PIG [2] and Hadoop [3] based on the data and queries from a SIGMOD 09 paper by Pavlo et al. [4]. The goals are to:

- Provide a simple framework to compare performances of several hadoop based data processing systems, including Hive and PIG.
- Identify performance bottlenecks in Hive to further identify optimizations that need to be performed in Hive.

2 Benchmark Environment

2.1 Test Systems

We used Hadoop version 0.18.3[2], PIG trunk version 786346, and Hive trunk version 786346[3]. We stored all the input data and query results in the Hadoop Distributed File System (HDFS). We used the UNIX command line interfaces to run the benchmarked systems. Aside from the basic settings, we configured Hive, PIG and Hadoop to use the LZO compression library [7] for compressing the intermediate map output data. Each task tracker runs up to four map and four reduce tasks simultaneously. Combiners were enabled in PIG. All the configuration files can be found in the Hive benchmark package [6].

2.2 Node Configuration

We performed the benchmark on an eleven node cluster. One node is used as the master/name node/job tracker. Ten others are used as slave nodes which run the data nodes and task trackers. All of them are in the same rack. Their hardware and software information are listed in Table 1.

Table 1 Cluster hardware and software information

CPU	2 Dual-Core AMD Opteron(tm) 280 Processors, 2.4 GHz, cache 1 MB
Memory	8 GB
Disk	4 hard drives, total 1.6 T
Ethernet	1 Gbps
Linux	2.6.12-1.1398_FC4smp
Lzo lib	2.02

2.3 Test Dataset and Queries

We adopt the test data and queries from the experiments by Pavlo et al. which are available on their website [5]. We wanted to speed up the tests and we determined that running queries for 30 minutes gave

[1] If you have any questions or suggestions about this benchmark, please email to Yuntao Jia (yjia@facebook.com) or comment at the Hive JIRA at https://issues.apache.org/jira/browse/HIVE-396.

[2] PIG currently does not work with later versions of Hadoop.

[3] Latest as of June 2009.

stable timing results. So, we scaled down the data sizes to make sure the queries would not take more than 30 minutes. We also changed one of the queries (see join query below) to compensate for the scaled data. See Table 2 for more details about the data set. We tested the first four of the five queries, which include two select queries, one aggregation query and one join query[4]. We have reproduced the queries in Table 3. Details on how to generate the data can be found in the Hive benchmark package [6].

Table 2 Test Dataset

grep(key VARCHAR(10), field VARCHAR(90))	2 columns, 500 million rows, 50GB.
rankings(pageRank INT, pageURL VARCHAR(100), avgDuration INT).	3 columns, 56.3 million rows, 3.3GB.
uservisits(sourceIP VARCHAR(16), destURL VARCHAR(100), visitDate DATE, adRevenue FLOAT, userAgent VARCHAR(64), countryCode VARCHAR(3), languageCode VARCHAR(6), searchWord VARCHAR(32), duration INT).	9 columns, 465 million rows, 60GB (scaled down from 200GB).

Table 3 Test Queries

Select query 1	`SELECT * FROM grep WHERE field like '%XYZ%';`
Select query 2	`SELECT pageRank, pageURL FROM rankings WHERE pageRank > 10;`
Aggregation query	`SELECT sourceIP, SUM(adRevenue)` `FROM uservisits GROUP BY sourceIP;`

Join query	`SELECT INTO Temp sourceIP,`
	`AVG(pageRank) as avgPageRank,`
	`SUM(adRevenue) as totalRevenue`
	`FROM rankings AS R, userVisits AS UV`
	`WHERE R.pageURL = UV.destURL`
	`AND UV.visitDate BETWEEN Date(`1999-01-01') AND Date(`2000-01-01')`
	`GROUP BY UV.sourceIP;`
	
	`SELECT sourceIP, totalRevenue, avgPageRank`
	`FROM Temp ORDER BY totalRevenue DESC LIMIT 1;`

3 Benchmark Results

All data are pre-loaded into Hadoop Distributed File System (HDFS) before running the benchmarks. We do not consider loading time as part of the benchmark results.

For any particular query, all of Hadoop, Hive and PIG are configured to use the same number of mappers and reducers if possible. Details about the queries results, including numbers of jobs, mappers and reducers, are described in Table 4, Table 5, Table 6 and Table 7 for each of the four queries, respectively.

The timings are averaged over 3 runs. In each run, we record the time of the entire command line call, which includes the time taken for compiling the query, manipulating the metadata, running the actual compiled map-reduce job and storing the query results. We have also run the benchmark with NO compression of the intermediate map output data. Figure 1 shows the results with compression and Figure 2 shows the results without compression. Comparing the two results, we see that LZO compression does not affect the query times significantly.

[4] We are still working on benchmarking the fifth query.

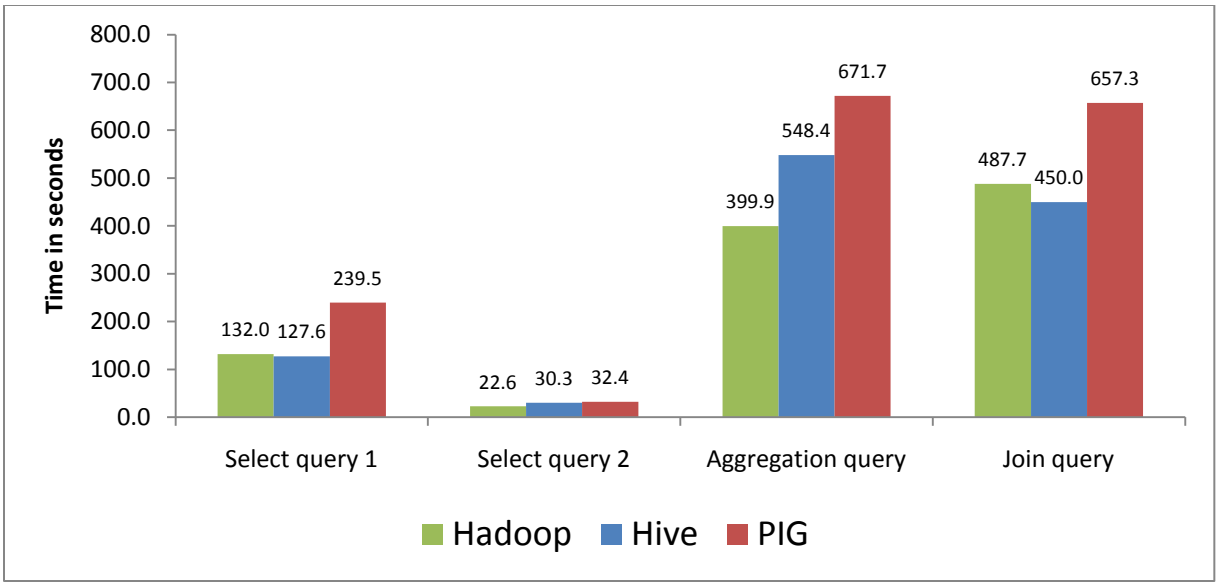

Figure 1 Hadoop, Hive and PIG query times with LZO compression of intermediate map output data

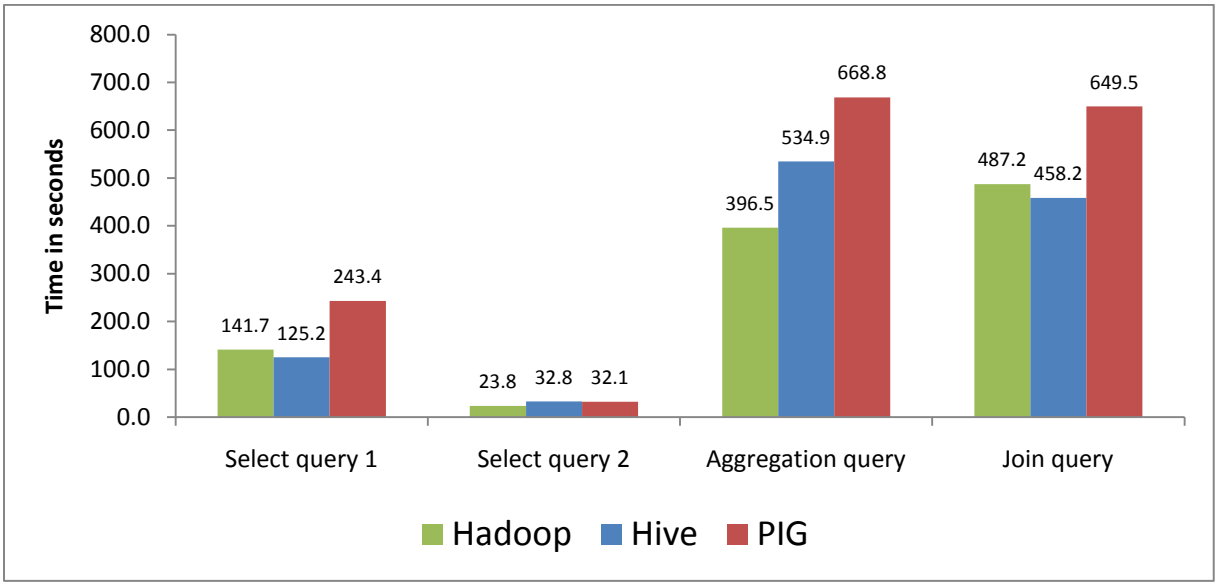

Figure 2 Hadoop, Hive and PIG query times without compression of intermediate map output data

Table 4 Select query 1

#Jobs **#Mappers** **#Reducers**	The query is finished in a single job with 380 mappers and 0 reducers on all of Hive, PIG and Hadoop.
Timing	Hadoop took 132.0 seconds. Hive took 127.6 seconds. PIG took 239.5 seconds. Hive took 3.4% less time than Hadoop. PIG took 81.4% more time than Hadoop and 87.6% more time than hive

Table 5 Select query 2

#Jobs **#Mappers** **#Reducers**	The query is finished in a single job with 30 mappers and 0 reducers on all of Hive, PIG and Hadoop.
Timing	Hadoop took 22.6 seconds. Hive took 30.3 seconds. PIG took 32.4 seconds. Hive took 34.0% more time than Hadoop. PIG took 43.0% more time than Hadoop and 6.7% more time than hive.

Table 6 Aggregation query

#Jobs **#Mappers** **#Reducers**	The query is finished in a single job with 450 mappers and 60 reducers on all of Hive, PIG and Hadoop.
Timing	Hadoop took 399.9 seconds. Hive took 548.4 seconds. PIG took 671.7 seconds. Hive took 37.1% more time than Hadoop. PIG took 67.9% more time than Hadoop and 22.5% more time than hive.

Table 7 Join query

#Jobs **#Mappers** **#Reducers**	The query is finished in three jobs on Hadoop, four jobs on Hive and five jobs on PIG. The first job, which takes the majority of the time, has 480 mappers and 60 reducers on all Hive, PIG and Hadoop. Other jobs on Hadoop have 60/60 and 60/1 mappers/reducers. Other jobs on Hive have 60/60, 60/1 and 1/1 mappers/reducers. Other jobs on PIG have 60/60, 60/1, 60/36 and 36/1 mappers/reducers.
Timing	Hadoop took 487.7 seconds. Hive took 450.0 seconds. PIG took 657.3 seconds. Hive took 7.7% less time than Hadoop. PIG took 34.8% more time than Hadoop and 46.1% more time than hive.

4 Acknowledgements

We thank Alan Gates from Yahoo for helping with the optimization of all the PIG queries. We thank the data infrastructure team in facebook for their valuable discussions. Particularly, we thank Raghu Murthy for revising the report. We thank Pavlo et al. for sharing their source code.

5 Bibliography

[1] Hive, http://wiki.apache.org/hadoop/Hive
[2] PIG, http://hadoop.apache.org/pig
[3] Hadoop, http://hadoop.apache.org/core
[4] A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. Dewitt, S. Madden, and M. Stonebraker, "A Comparison of Approaches to Large-Scale Data Analysis," in SIGMOD '09: Proceedings of the 2009 ACM SIGMOD International Conference, 2009
[5] A Comparison of Approaches to Large-Scale Data Analysis, http://database.cs.brown.edu/projects/mapreduce-vs-dbms
[6] Hive Benchmark Package, https://issues.apache.org/jira/secure/attachment/12413334/hive_benchmark_2009-07-12.tar.gz
[7] LZO Compression Library, http://www.oberhumer.com/opensource/lzo

6 Appendix

Here we describe the Hive and PIG query strings for all the four queries. They can also be found in the Hive benchmark package [6]. For Hadoop queries, we used the source code provided by Pavlo et al. on their website [5]. We did make some changes to the source code of the last Hadoop query so that it uses certain numbers of mappers and reducers in different phases of that query. Those changes are also included in the Hive benchmark package.

6.1 Select Query 1

- The Hive query is:

```
DROP TABLE grep;
DROP TABLE grep_select;
CREATE EXTERNAL TABLE grep ( key STRING, field STRING )
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' STORED AS TEXTFILE
LOCATION '/data/grep';
CREATE TABLE grep_select ( key STRING, field STRING );
INSERT OVERWRITE TABLE grep_select
SELECT * FROM grep WHERE field LIKE '%XYZ%';
```

- The PIG query is:

```
rmf output/PIG_bench/grep_select;
a = load '/data/grep/*' using PigStorage as ( key, field );
b = filter a by field matches '.*XYZ.*';
store b into 'output/PIG_bench/grep_select';
```

6.2 Select Query 2

- The Hive query is:

```
DROP TABLE rankings_select;
DROP TABLE rankings;
CREATE EXTERNAL TABLE rankings ( pageRank INT, pageURL STRING, avgDuration INT )
ROW FORMAT DELIMITED FIELDS TERMINATED BY '|' STORED AS TEXTFILE
LOCATION '/data/rankings';
CREATE TABLE rankings_select ( pageRank INT, pageURL STRING );
INSERT OVERWRITE TABLE rankings_select
SELECT pageRank, pageURL FROM rankings WHERE pageRank > 10;
```

- The PIG query is:

```
rmf output/PIG_bench/rankings_select;
a = load '/data/rankings/*' using PigStorage('|') as ( pagerank, pageurl,
aveduration );
b = filter a by pagerank > 10;
store b into 'output/PIG_bench/rankings_select';
```

6.3 Aggregation Query

- The Hive query is:

```
DROP TABLE uservisits;
DROP TABLE uservisits_aggre;
set mapred.reduce.tasks=60;
CREATE EXTERNAL TABLE uservisits ( sourceIP STRING, destURL STRING, visitDate STRING,
adRevenue DOUBLE, userAgent STRING, countryCode STRING, languageCode STRING,
searchWord STRING, duration INT )
ROW FORMAT DELIMITED FIELDS TERMINATED BY '|' STORED AS TEXTFILE
LOCATION '/data/uservisits';
CREATE TABLE uservisits_aggre ( sourceIP STRING, sumAdRevenue DOUBLE );
INSERT OVERWRITE TABLE uservisits_aggre
SELECT sourceIP, SUM(adRevenue) FROM uservisits GROUP BY sourceIP;
```

- The PIG query is:

```
rmf output/PIG_bench/uservisits_aggre;
a  = load '/data/uservisits/*' using PigStorage('|') as (sourceIP, destURL, visitDate,
adRevenue, userAgent, countryCode, languageCode, searchWord, duration);
a1 = foreach a generate sourceIP, adRevenue;
```

```
b  = group a1 by sourceIP parallel 60;
c  = FOREACH b GENERATE group, SUM(a1. adRevenue);
store c into 'output/PIG_bench/uservisits_aggre';
```

6.4 Join Query

- The Hive query is:

```
DROP TABLE rankings;
DROP TABLE uservisits;
DROP TABLE rankings_uservisits_join;
set mapred.reduce.tasks=60;
CREATE EXTERNAL TABLE rankings (pageRank INT, pageURL STRING, avgDuration INT)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '|' STORED AS TEXTFILE
LOCATION '/data/rankings';
CREATE EXTERNAL TABLE uservisits (sourceIP STRING, destURL STRING, visitDate STRING,
adRevenue DOUBLE, userAgent STRING, countryCode STRING, languageCode STRING,
searchWord STRING, duration INT )
ROW FORMAT DELIMITED FIELDS TERMINATED BY '|' STORED AS TEXTFILE
LOCATION '/data/uservisits/';
CREATE TABLE rankings_uservisits_join ( sourceIP STRING, avgPageRank DOUBLE,
totalRevenue DOUBLE);
INSERT OVERWRITE TABLE rankings_uservisits_join
SELECT sourceIP, avg(pageRank), sum(adRevenue) as totalRevenue FROM rankings R JOIN
(SELECT sourceIP, destURL, adRevenue FROM uservisits UV WHERE UV.visitDate > '1999-01-
01' AND UV.visitDate < '2000-01-01') NUV ON (R.pageURL = NUV.destURL)
GROUP BY sourceIP ORDER BY totalRevenue DESC LIMIT 1;
```

- The PIG query is:

```
rmf output/PIG_bench/html_join;
a = load '/data/uservisits/*' using PigStorage('|') as (sourceIP, destURL, visitDate,
adRevenue, userAgent, countryCode, languageCode, searchWord, duration);
b = load '/data/rankings/*' using PigStorage('|') as (pagerank:int, pageurl,
aveduration);
b1 = foreach b generate pagerank, pageurl;
c = filter a by visitDate > '1999-01-01' AND visitDate < '2000-01-01';
c1 = foreach c generate sourceIP, destURL, adRevenue;
d = JOIN c1 by destURL, b1 by pageurl parallel 60;
d1 = foreach d generate sourceIP, pagerank, adRevenue;
e = group d1 by sourceIP parallel 60;
f = FOREACH e GENERATE group, AVG(d1.pagerank), SUM(d1.adRevenue);
g = order f by $2 desc;
h = limit g 1;
store h into 'output/PIG_bench/html_join';
```