Comparing Query Performance between Hive, Pig, and SQlite

Alejandro Hernández Segura

CSC 555 Mining Big Data

Introduction

For this project, we will be assessing query performance between Hive, Pig, and SQlite. At the outset, it seemed to us that SQlite might have the advantage since we will be working with structured data. However, it is possible, through the parallelization aspect of Hadoop, that some Hive or Pig queries will outperform those of SQlite. It is our hope that the results of this report will demonstrate markedly different runtimes depending on the type of query and size of the data, thus highlighting the strengths of the different platforms (RDBMS style Sqlite vs. Hadoop map reduce Hive and Pig).

The Data

For our analysis, we will be using the MovieLens 20M data provided by GroupLens Research. The data is split into three files: ratings.csv, which contains the user id, movie id, rating, and timestamp; movies.csv, which contains the movie id, movie title and year, and genres; and tags.csv, which contains the user id, movie id, tags, and timestamp. In total, there are 20 million movie ratings and 465 thousand tags applied by 138 thousand users on 27 thousand movies. Although this sounds like a sizeable dataset, it only takes up about half a gigabyte of space. Thus, we replicated the data four times, giving us a more significant amount of data (2GB) to work with.

The task of duplicating the data was rather trivial in the Hadoop environment, since keys do not matter to Hive and a single line of code can duplicate the data four times into another file. But this was not the case for the data in SQlite. We simply could not reload the data again into the database because doing so would violate the integrity constraint (a primary key needs to be unique). Thus, when we loaded the data for the

second time into the database, we had to add the maximum value of movie id and the maximum value of user id of the first database to the ingoing values. As an example, take the movie “Toy Story”, which has movie id 1 in our movie.csv file. When we loaded the data the second time, we added the max movie id value of our first database, which was id number 131262. So “Toy Story” should now appear twice in our movies table with two different movie ids. We can check that this is indeed the case with the following query:

```
SELECT * FROM movies WHERE movie_title = "Toy Story"
(1, 'Toy Story', 1995)
(131263, 'Toy Story', 1995)
```

We repeated this method until we re-loaded the data a total of four times.

```
SELECT * FROM movies WHERE movie_title = "Toy Story"
(1, 'Toy Story', 1995)
(131263, 'Toy Story', 1995)
(262525, 'Toy Story', 1995)
(393787, 'Toy Story', 1995)
```

The tables themselves were also structured differently in SQlite compared to Hive and Pig. In Hive and Pig, the tables are structured in the same way they are in the csv files: movies(movie_id, movie_title and year, genres), tags(user_id, movie_id, tags, timestamp), and ratings(user_id, movie_id, rating, timestamp). In SQlite, however, the movies table was normalized into 3NF which resulted in the following tables: movies(movie_id, movie_title, year), movie_categories(movie_id, category_id), and categories(category_id, genre_desc). We probably did not have to do this, but it is good practice, and if we really wanted to compare Hive and Pig to SQlite, we should set up the tables as they would be implemented in relational database.

Cluster Setup

For both the one node cluster and the three node cluster, we are running the t2.small instances with 1 vCPU, 2 GiB ram, and 8 GiB hard disks. The laptop which will run SQlite has an intel i5 processor, 4 GB ram, and 128 GB SSD. As far as setting up the clusters, creating the one-node cluster is relatively straightforward compared to setting up the three node cluster—setting up the three node cluster was a frustrating and frightening experience on the command line. The whole time, we hoped that we were not making a mistake, but really we were not sure that we were doing things right either. However, this experience did help us realize become more comfortable and knowledgeable about working though a shell, and we are grateful for that.

Performance

Now that we have covered the details of our table and cluster setups, we are ready to compare the performance of the Hive and Pig queries (in both the one node and three node clusters) to those of a traditional relational database queries in SQlite. We will cover three sets of queries, and in each set we include: the queries we ran, the number of records returned, and the runtimes. If applicable, we also discuss discrepancies between the results. For Pig, since no runtime is displayed, we calculated the runtime (to the nearest second) by taking the difference between the StartedAt time and the FinishedAt time ended that are displayed in the summary after a query runs. The SQlite queries were times using the Timeit library. Also, note that the screenshots are not provided on all the queries. We actually did not include any at first because we wanted

all of the output on our summary tables, but then we saw the email from Prof. Rasin that suggested we include some output. So we reran some queries and added screen captured numbers into the tables as best as we could.

I. Query Set One

For the first set of queries, we will be verifying that the tables contain the same number of entries:

SQlite/Hive:

```
SELECT COUNT(*) FROM movies;
```

Pig:

```
movies = LOAD '/user/hive/warehouse/movies/BIG_movies.csv'
USING PigStorage(',') AS (movies:INT, title_year:CHARARRAY,
genres:CHARARRAY);

moviesG = GROUP movies ALL;
Count = FOREACH moviesG GENERATE COUNT(movies);
DUMP Count;
```

SQlite results: (109112,)

Hive results: 109116

Pig results: (109112)

```
HadoopVersion   PigVersion      UserId  StartedAt       FinishedAt      Features
0.20.205.1      0.9.2   ec2-user        2015-06-10 14:21:21     2015-06-10 14:21
:59     GROUP_BY
```

	1 Node Cluster		3 Node Cluster		
	Hive	Pig	Hive	Pig	Sqlite
Time (in seconds)	35.478	44	42.948	38	0.10169953101757008

Hive performed around the same speed on both the 3 node and 1 node clusters, and similar for Pig on both the 3 node and 1 node cluster. SQlite performed 350X better than the second fastest query (Hive on the 1 node).

SQlite/Hive:

```
SELECT COUNT(*) FROM tags;
```

Pig:

```
tags = LOAD '/user/hive/warehouse/tags/BIG_tags.csv' USING
PigStorage(',') AS (user_id:INT, movie_id:INT,
tag:CHARARRAY, timestamp:CHARARRAY);

tagsG = GROUP tags ALL;
Count = FOREACH tagsG GENERATE COUNT(tags);
DUMP Count;
```

SQlite results: (1862256,)

Hive results: 1862260

Pig results: (1862256)

	1 Node Cluster		3 Node Cluster		
	Hive	Pig	Hive	Pig	Sqlite
Time (in seconds)	33.309	49	36.769	47	0.8732905816230596

Similar results as before: hive performed around the same on both the 3 node and 1 node clusters, and so did Pig. Sqlite performed around 40X faster than the second fastest query (Hive on the 1 node cluster).

SQlite/Hive:

```
SELECT COUNT(*) FROM ratings;
```

Pig:

```
ratings = LOAD
'/user/hive/warehouse/ratings/BIG_ratings.csv' USING
PigStorage(',') AS (user_id:INT, movie_id:INT, rating:INT,
timestamp:CHARARRAY);

ratingsG = GROUP ratings ALL;
Count = FOREACH ratingsG GENERATE COUNT(ratings);
DUMP Count;
```

SQlite results: (80001052,)

Hive results: 80001056

Pig results: (80001052)

```
HadoopVersion   PigVersion      UserId  StartedAt       FinishedAt      Features
0.20.205.1      0.9.2   ec2-user        2015-06-10 14:26:06     2015-06-10 14:32
:38     GROUP_BY
```

	1 Node Cluster		3 Node Cluster		
	Hive	Pig	Hive	Pig	Sqlite
Time (in seconds)	100.914	1023	68.861	392	228.9873222841237

In this query, Hive on the three node cluster was the fastest (about 1.5 faster than Hive on the 1 node, 5X faster than Pig on the 3 node, and 3.3X faster than SQlite). Pig on the three node cluster also performed 2.6X faster than Pig on the one node. This is also the first really large table (about 2 GB), the others were less than 64MB each. The small size of the other tables is probably why the queries on the 3 node cluster were slightly slower than those of the one node (for Hive); the tables were on one block but the master node still had to communicate to all of the nodes.

Also note that the counts of SQlite and Pig are different from those of Hive (they are off by four). This is due to the headers not being removed from the csv files, and then being duplicated four times to form our larger dataset. Pig, as far as we can tell, did not include the headers in the data because the data types did not match the column data types specified in the LOAD statement.

II. Query Set Two

For the next set, we will be running single table queries. Note that from here on out, we did not run the Pig queries on the single node cluster (the Pig queries were the last we attempted and so we ran them solely on the three node cluster due to time). For the first query, we extract all films from the movies table that were released in 1995;

```
SQlite: SELECT * FROM movies WHERE year = 1995;

Hive:   SELECT * FROM movies WHERE title_year LIKE "%1995%";

Pig:    X = FILTER movies BY (title_year MATCHES '.*1995.*');
        DUMP X;

SQlite number of records returned: (1452,)
  Hive number of records returned:  1472
   Pig number of records returned:  1472
```

	1 Node Cluster		3 Node Cluster		
	Hive	Pig	Hive	Pig	Sqlite
Time (in seconds)	21.605	–	21.154	32	0.802558670700983

Here, we are dealing with one of the small tables again, so SQlite did the best, Hive performed about the same on both clusters, and Pig was the slowest. What is interesting here is that the queries took less than 30 seconds to run. We wondered why until we noticed in the output that no reducer was ran in this query, which makes some sense. We only want certain records, and the mapper can filter these out and output them if they meet the criteria.

Note also that there is a discrepancy of 20 records (SQlite returns 1452 records, while Hive and Pig return 1472). What we believe this means is that there are 5 movies (we divide 20/4 because we duplicated the data four times) that have the

year in the title but were not actually released in that year, and that is what is giving us extra records in the Hive and Pig queries.

For the next query, we look for tags in the tags table that begin with “e”:

```
SQlite/Hive: SELECT * FROM tags WHERE tag LIKE "e%";

Pig: Y = FILTER tags BY (tag MATCHES 'e.+');
     DUMP Y;

   tagsyg = GROUP tagsY ALL;
   County = FOREACH tagsyg GENERATE COUNT(tagsY);
   DUMP County;


SQlite number of records returned: (51160,)
  Hive number of records returned:  31160
   Pig number of records returned:  31656
```

	1 Node Cluster		3 Node Cluster		
	Hive	Pig	Hive	Pig	Sqlite
Time (in seconds)	40.38	–	25.267 seconds	53	0.802558670700983

SQlite again did the best, followed by Hive on the three node, Hive on the one node, and Pig on the three node. Hive on the three node is only about 1.6X faster than Hive on the one node, but it is about 2X faster than Pig on the three node.

What is more concerning, however, is the significant difference in records returned between the Hive/Pig and SQlite queries (a difference of 20 thousand records). To try to see why the difference is so large, we try another query:

```
SQlite/Hive: SELECT * FROM tags WHERE tag = "excellent";

Pig: Z = FILTER tags BY (tag == 'excellent');
```

```
DUMP Z;
```

```
SQlite number of records returned: (116,)
  Hive number of records returned:  116
   Pig Total records written : 116
```

```
HadoopVersion   PigVersion      UserId  StartedAt       FinishedAt      Features
0.20.205.1      0.9.2   ec2-user        2015-06-10 14:37:48     2015-06-10 14:38
:20     FILTER
```

	1 Node Cluster		3 Node Cluster		
	Hive	Pig	Hive	Pig	Sqlite
Time (in seconds)	28.745	–	18.467 seconds	32	2.078897076281885

SQlite is the fastest (about 9X faster than Hive on the 3 node). Hive on the three node is still about 1.5X faster than Hive on the one node and about 2X faster than Pig on the three node.

More importantly though, this time we get the same number of records on the three queries. It seems to us that in SQlite “e%” returns all strings that start with “e” (including those with multiple words), but Hive and Pig, we think, only return the one word strings that start with “e”.

Next, we try to find number times a rating lower than 2 was given:

```
SQlite/Hive: SELECT COUNT(*) FROM ratings WHERE rating < 2;
```

```
Pig: R = FILTER ratings BY (rating < 2);
     rG = GROUP R ALL;
     Count = FOREACH rG GENERATE COUNT(R);
     DUMP Count;
```

```
SQlite number of records returned: (4796436,)
  Hive number of records returned:  4796436
   Pig number of records returned: (4796436)
```

	1 Node Cluster		3 Node Cluster		
	Hive	Pig	Hive	Pig	Sqlite
Time (in seconds)	121.174	–	73.941	267	377.56723746494947

This time Hive on the 3 node was the fastest (1.6X faster than Hive on the 1 node, 3X faster than pig, and 5X faster than SQlite). What is strange, so far, is that we have not gotten close to the theoretical 3X faster comparative performance between the 3 node cluster and the 1 node cluster in the Hive queries.

III. Query Set Three

For the last set of queries we will be joining tables. Here, we are concerned that the duplicate keys in the Hive and Pig tables will give us strange results. The first query joins the movies and tags tables and returns the records where the tag starts with "e".

```
SQlite/Hive: SELECT * FROM tags INNER JOIN movies
             ON tags.movie_id = movies.movie_id WHERE tag
             LIKE "e%";

Pig: J = JOIN movies BY movies, tags BY movie_id;
     jf = FILTER J BY (tag MATCHES 'e.+');
     DUMP jf;


SQlite number of records returned: (51160,)
Hive   number of records returned:  126640
Pig    number of records returned:  126624
```

	1 Node Cluster		3 Node Cluster		
	Hive	Pig	Hive	Pig	Sqlite
Time (in seconds)	61.668	–	47.263 seconds	79	5.355257269775061

Here, SQlite was the fastest again. Hive on the three node was only about 1.3X faster than Hive on the one node, but 1.6X faster than Pig on the three node cluster.

And again, the counts do not match, but we believe the numbers returned make sense. Remember that when we ran the similar query in the one table queries section looking for tags that started with “e”, we got 51160 for SQlite, 31160 for Hive, and 31656 for Pig. In this case, we get the same results for SQlite, but 4 times as many results for Hive and Pig. This tells us that the duplicated keys in the Pig/Hive tables caused the tables to join on the same key four times. We can try to see if this occurs again by joining on the tags and movies tables again, extracting the records with the “excellent” tag:

```
SQlite/Hive: SELECT * FROM tags INNER JOIN movies
             ON tags.movie_id = movies.movie_id
             WHERE tag = "excellent";

Pig: jf2 = FILTER mtags BY (tag == 'excellent');

SQlite number of records returned:  116
  Hive number of records returned:  464
   Pig Total records written : 464
```

```
HadoopVersion   PigVersion      UserId  StartedAt       FinishedAt      Features
0.20.205.1      0.9.2   ec2-user        2015-06-10 14:41:12     2015-06-10 14:41
:49     HASH_JOIN,FILTER
```

	1 Node Cluster		3 Node Cluster		
	Hive	Pig	Hive	Pig	Sqlite
Time (in seconds)	34.22	–	31.615 seconds	37	2.1501155350707273

SQlite was the fastest (about 16X faster that the second fastest). Here we also get a similar performance between Hive and Pig on the different clusters.

As far as the query results themselves, we see a similar pattern as the last query (four times the output on the Hive/Pig queries), so we believe our hypothesis is correct; since keys were duplicated four times, the tables join on the same key four times causing four times the output. For the next query, we join the movies and ratings tables, and extract the films that got a higher rating than 4.

```
SQlite: SELECT DISTINCT movie_title FROM movies
        INNER JOIN ratings
        ON movies.movie_id = ratings.movie_id
        WHERE rating > 4;

Hive:   SELECT DISTINCT title_year FROM movies
        INNER JOIN ratings
        ON movies.movie_id = ratings.movie_id
        WHERE rating > 4;

Pig: mr = JOIN movies BY movies, ratings BY movie_id;
     mr2 = FILTER mr BY rating > 4;
    mrf = DISTINCT mr2;

    mrG = GROUP mrf ALL;
    Count = FOREACH mrG GENERATE COUNT(mrf);
    DUMP Count;

SQlite number of records returned: (16592,)
  Hive number of records returned:  16900
   Pig number of records returned: (2898660)
```

```
HadoopVersion   PigVersion      UserId  StartedAt       FinishedAt      Features
0.20.205.1      0.9.2   ec2-user        2015-06-11 18:00:14     2015-06-11 18:20
:48     HASH_JOIN,GROUP_BY,DISTINCT,FILTER
```

	1 Node Cluster		3 Node Cluster		
	Hive	Pig	Hive	Pig	Sqlite
Time (in seconds)	279.92	–	153.846	1234	321.558929044292

Here, Hive on the three node cluster was the fastest (1.8X faster than Hive on the one node, 3.8 times faster than Pig on the three node, and 2X faster than SQlite). Again, it is strange that Hive on the three node cluster has not performed 3X faster than Hive on the 1 node.

Again, here number of records between SQlite/Hive vary significantly from Pig. So it appears we are not correctly filtering the tables in the Pig query, but we are unsure as to how to change the Pig statements to get similar results as those of Hive and SQlite.

Conclusion

For the smaller tables (in this case, that means tables of about 64MB), SQlite had the advantage in runtime performance. The start-up cost of running map reduce in the Hive/Pig queries meant that every query would need about 30 seconds to run (unless no Reducer needed to run, then the query could run in less time, but it was still not as fast as SQlite). However, once we moved to larger tables (in this case, the ratings table which is about 2GB), Hive performed the best. Distributing the workload among the nodes helped cut runtimes considerably. Pig, on the other hand, did not do as well (especially on table joins), and we would like to know why that is. In class, a classmate asked if a table is loaded each time a Pig query runs, and we are inclined to believe that it does. It would certainly explain why Pig on the three node cluster ran consistently slower than Hive on a one-node cluster. We also wonder why the Hive queries did not run as 3X faster on the three node cluster compared to the one node. At

best they ran about 2X faster only. The best explanation we have is that the blocks of data are not evenly distributed among the nodes, but we are still puzzled as to why when we ran the balancer. Still, from these results, it is clear that the parallelization offers a huge advantage when working with large amounts of data.