Join Order Benchmark on Snowflake and Postgres

Frank Chen
Data 516, Fall 2019
kfrankc@uw.edu

1. INTRODUCTION

After being introduced to Snowflake DB in class, I am intrigued by Snowflake's query optimization engine. It looks like Snowflake should be able to perform very well in warm cache runtime scenarios. For my project, I analyzed the IMDB Join Order Benchmark on Snowflake, and comparing the results to Postgres running on my laptop. I found that Snowflake consistently outperforms Postgres in terms of both cold and warm cache query time, but that is also dependent on the cluster and machine setup (more on that in the results section). I also found that as the number of joins in a query increase, the cold cache runtime steadily increases, but the warm cache time remained steady, further showcasing Snowflake's query optimization.

2. EVALUATED SYSTEM(S)

For this study, I am using Snowflake and Postgres. I am using the DATA 516 Snowflake instance, which has a DEMO_WH warehouse with size Small, and 1 server per cluster. I am also running Postgres on my 2015 MacBook Pro, with 2.7 GHz Intel Core i5, and 8 GB 1867 MHz DDR3.

2.1. Snowflake

Snowflake Inc, founded in 2012, offers Snowflake Elastic Data Warehouse (Snowflake for short), a cloud-based data warehouse solution, generally termed as 'data warehouse-as-a-service'. Snowflake has a multi-clustered, shared-data architecture [1], and is multi-tenant, transactional, secure, highly scalable and elastic system with support for both semi-structured and schema-less data.

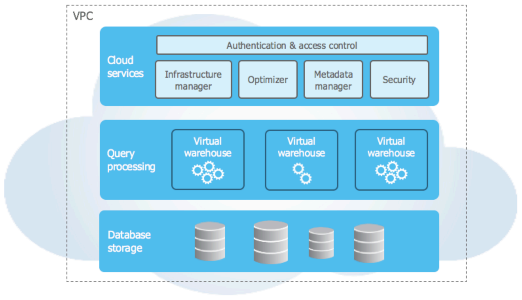


Figure 1: Snowflake Data Warehouse Architecture [2]

2.2. Postgres

PostgreSQL (Postgres for short) is a free and open-source relational database management system (RDBMS). It features transactions with ACID properties, updatable views, foreign keys, and stored procedures, optimal for a range of workloads, from single machines to data warehouses with concurrent users [3].

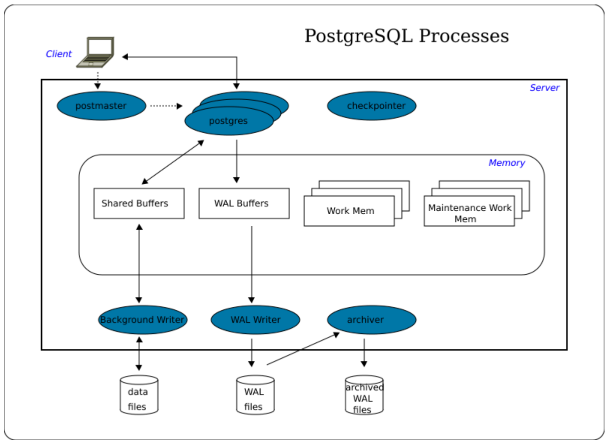


Figure 2: Postgres Architecture [4]

3. PROBLEM STATEMENT & METHOD

The follow questions outline the goals of my study:

1. How much does Snowflake DB optimize runtime when it uses warm caching?
2. How does Snowflake DB's query runtime vary as the # of joins increase or decrease?
3. How does Snowflake DB's query runtime compare with other database systems, such as Postgres?

I will be performing analysis to try and answer these questions by running the Join Order Benchmark (JOB) on the IMDB dataset. This benchmark tests the quality of cardinality estimators [5], and the queries are a good measure for analyzing a SQL data warehouse solution's runtime when it comes to large numbers of joins.

3.1. The Data

The IMDB dataset used in JOB has the following data profile:

- Size: 1.26 GB gzipped
- 21 tables (details in section 3.1 on data ingestion)

3.2. Join Order Benchmark Queries

I am using the Join Order Benchmark (JOB) queries to perform my analysis. The queries are publicly available at this Github Repo: https://github.com/gregrahn/join-order-benchmark. There are 114 queries in JOB. In order to perform my benchmark analysis on Snowflake, I first need to upload the IMDB dataset into the Snowflake, then run the JOB queries. I downloaded the IMDB dataset, and created tables with defined schemas in my Snowflake database. I then created staging for each of my tables using snowsql, and loaded the data in.

3.3. Ingesting Data

After running the ingestion commands in snowsql, I have the following rows loaded in. Note: some of the data from IMDB were not formatted correctly, so there were errors in loading them in. I've recorded which tables had those errors, and how many rows were affected. It is unclear if the original paper also had those errors.

- aka_name: 676 rows error, 900662 rows success
- aka_title: 3 rows error, 361376 rows success
- cast_info: 118792 rows error, 36124530 rows success
- char_name: 3412 rows error, 3136382 rows success
- company_name: 172 rows error, 234825 rows success
- company_type: 4 rows success
- complete_cast: 135086 rows success
- comp_cast_type: 4 rows success
- info_type: 113 rows success
- keyword: 60 rows error, 134110 rows success
- kind_type: 7 rows success
- link_type: 18 rows success
- movie_companies: 5062 rows error, 2604067 rows success
- movie_info: 355462 rows error, 14355706 rows success
- movie_info_idx: 1380035 rows success
- movie_keyword: 4523930 rows success
- movie_link: 29997 rows success
- name: 38 rows error, 4167453 rows success
- person_info: 802506 rows error, 2024951 rows success
- row_type: 12 rows success
- title: 170 rows error, 2527799 rows success

Next, I did a similar exercise for ingesting data into Postgres. Unfortunately, due to data size constraints and my personal computer's space limitations, I could only load in parts of the dataset, and run a subset of the queries. I ran the JOB queries on both Snowflake and Postgres, and recorded both warm and cold cache runtime, as well as the # of joins in each query.

4. RESULTS

4.1. Question 1 Analysis

The IMDB JOB Benchmark has a total of 114 queries, with some queries containing more than 10 joins. After running each query on Snowflake multiple times, I recorded both the warm and cold cache runtime (first query runtime is cold, second is warm), and plotted a graph to show the difference between warm and cold caching performance in terms of runtime.

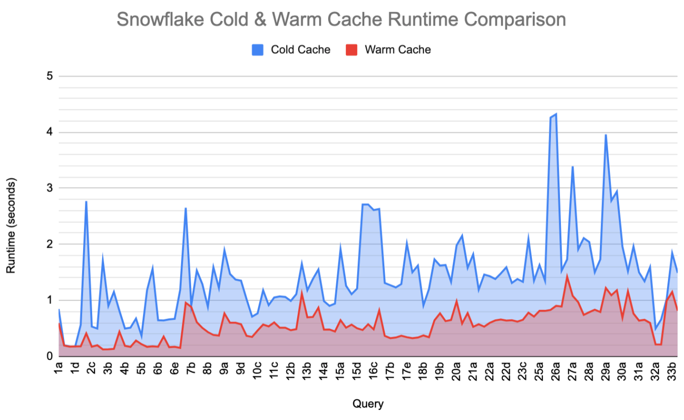
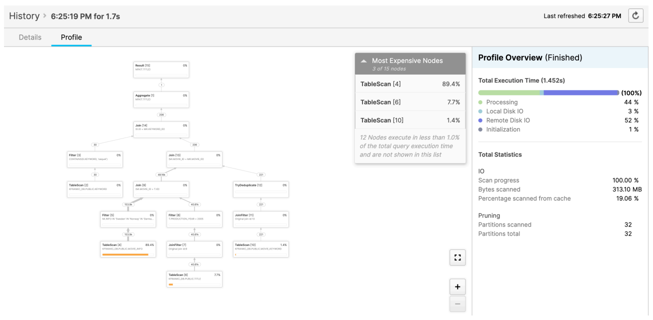


Figure 3: Difference between cold and warm cache runtime based on # of joins on Snowflake

It is clear that Snowflake's warm caching capability optimizes the reduction of runtime, with some queries such as query 3a achieving over 80 % reduction in runtime. Using the Snowflake query profiler, I determined that the majority of the cold cache runtime was due to the large tablescan on the movie_info table, consisting of more than 80 % of the total runtime. The 3a query is below:



Most Expensive Nodes
3 of 15 nodes

TableScan [4]	89.4%
TableScan [6]	7.7%
TableScan [10]	1.4%

12 Nodes execute in less than 1.0% of the total query execution time and are not shown in this list

Profile Overview (Finished)

Total Execution Time (1.452s) (100%)

● Processing	44 %
● Local Disk IO	3 %
● Remote Disk IO	52 %
● Initialization	1 %

Total Statistics

IO	
Scan progress	100.00 %
Bytes scanned	313.10 MB
Percentage scanned from cache	19.06 %

Pruning	
Partitions scanned	32
Partitions total	32

Figure 4: Query 3a profile on Snowflake, showing the tablescan

```
SELECT MIN(t.title) AS movie_title
FROM keyword AS k,
     movie_info AS mi,
     movie_keyword AS mk,
     title AS t
WHERE k.keyword LIKE '%sequel%'
  AND mi.info IN ('Sweden',
                  'Norway',
                  'Germany',
                  'Denmark',
                  'Swedish',
                  'Denish',
                  'Norwegian',
                  'German')
  AND t.production_year > 2005
  AND t.id = mi.movie_id
  AND t.id = mk.movie_id
  AND mk.movie_id = mi.movie_id
  AND k.id = mk.keyword_id;
```

4.2. Question 2 Analysis

Next, I analyzed how runtime varies with the number of joins. The general pattern indicates that as the number of joins increase, the cold cache runtime increases faster than the warm cache runtime. This is to be expected, as Snowflake's query optimizer is able to load the warm cache query plan faster than upon seeing a query for the first time.

In addition, I observed that the line of best fit for cold caching runtime has a higher slope than that of the warm cache runtime. This indicates that warm cache runtime has a slower rate of increase as the number of joins increase, but this has some assumptions in the limited query samples we used as part of JOB, and additional confounding factors should be taken into account.

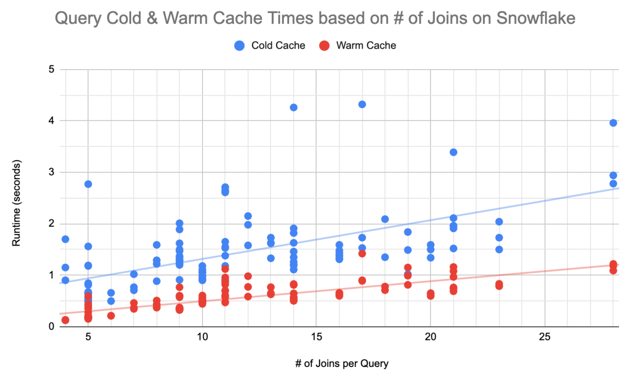


Figure 5: Snowflake cold and warm cache runtime comparison across the 113 queries

Another interesting observation is that while some queries in JOB have a large number of joins (query 29b, for example, has 28), the majority of the runtime is spent on only one or two joins that take substantial time.

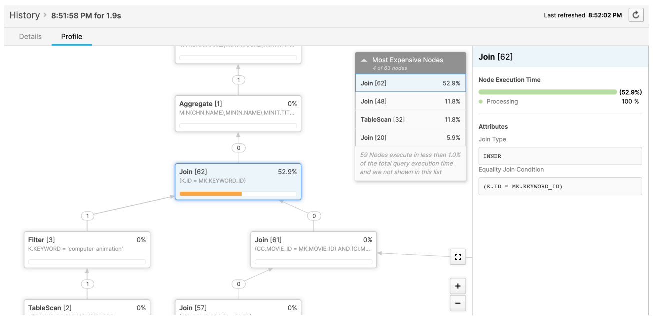


Figure 6: Query 29b spends more than half its runtime on a join between keyword table and movie_keyword table

4.3. Question 3 Analysis

Lastly, I analyzed the warm and cold cache performance of Postgres with Snowflake as the # of joins varied. I ran the JOB queries on Postgres, and recorded its warm and cold cache (first query runtime is cold, second is warm). I then created a bar chart comparing the two solutions (see Figure 7).

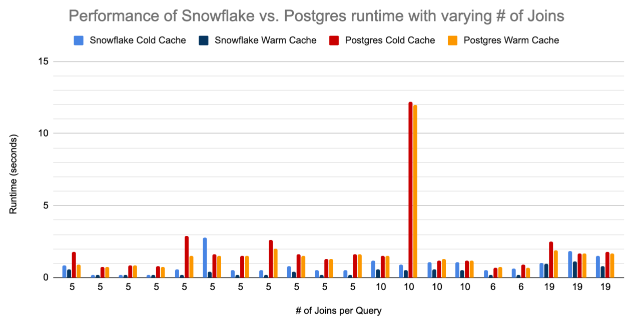


Figure 7: Snowflake vs. Postgres in warm and cold cache as the number of joins increased

It is clear that Snowflake has in general faster runtime in both cold and warm cache compared to the Postgres database set up on my personal computer. Query 11b shows high Postgres runtime in both warm and cold cache runtime. While I am unsure about the exact cause of this anomaly, it could be due to the skewness of the particular joins in this query. Both Snowflake and Postgres query plan shows the tablescan on the title table as the operation that took the most time in this query. It is interesting to note the remainder of the queries did not show substantial runtime differences between Postgres and Snowflake compared to the anomaly.

5. CONCLUSION

Snowflake generally has significantly faster runtime when comparing its warm and cold cache, with some queries achieving an 80 percent decrease in runtime from cold to warm. As the number of joins increased, the Snowflake cold cache runtime tends to increase at a higher rate than warm cache, which had a much smaller slope coefficient. Lastly, Snowflake outperformed Postgres in both warm and cold cache runtime across the queries I was able to run in Postgres, but this could also be due to my machine's limitations.

6. REFERENCES

1. Dageville, B., Cruanes, T., Zukowski, M., Antonov, V.A., Avanes, A., Bock, J., Claybaugh, J.H., Engovatov, D., Hentschel, M., Huang, J., Lee, A.W., Motivala, A., Munir, A., Pelley, S., Povinec, P., Rahn, G., Triantafyllis, S., & Unterbrunner, P. (2016). The Snowflake Elastic Data Warehouse. SIGMOD '16.
2. (n.d.). Retrieved from https://docs.snowflake.net/manuals/user-guide/intro-key-concepts.html
3. Stonebraker, M., & Rowe, L.A. (1986). The design of POSTGRES. SIGMOD '86.
4. PostgreSQL/Architecture. (n.d.). Retrieved December 10, 2019, from https://en.wikibooks.org/wiki/PostgreSQL/Architecture
5. Leis, V., Gubichev, A., Mirchev, A., Boncz, P.A., Kemper, A., & Neumann, T. (2015). How Good Are Query Optimizers, Really? *PVLDB, 9*, 204-215.