JOB Comparison on Distributed Systems

Comparing Join Order Benchmark results on RedShift and Snowflake

Riley Waters
MS Data Science
University of Washington
Seattle, Washington
rdwaters at uw.edu

1 Introduction

Database Management Systems (DBMS) use query optimization to construct an optimal execution plan for a query. In the case of complex, multi-table joins, there could be thousands or millions of possible join orders to choose from. Most systems construct a cost model for subset of possible join orders and choose the cheapest one. However, these cost models are found with cardinality estimations built on assumptions that are not always true in real-world data.

In "How Good Are Query Optimizers, Really?" [1], A new benchmark called the Join Order Benchmark (JOB) was introduced to better test the cost models and cardinality estimation of different DBMSs. The benchmark uses the publicly available IMDB 2013 dataset [2], a 3.9 GB set with 21 tables about movies, actors, companies, and more. The data has many correlations and non-uniform data distributions which can complicate cardinality estimation. The benchmark includes 113 queries, each with a high number of possible join orders. The authors used JOB to assess optimizer performance for PostgreSQL, HyPer, and three unnamed DBMSs.

In this project I tested JOB on different distributed database systems. My goal was to see how performance differs across platforms and configurations.

I found that the queries tested a wide variety of joins. During my initial testing, some took several minutes to complete. On Redshift there was a relatively consistent pattern of small runtime decreases when adding more nodes, but nothing close to linear speedup. Some queries barely responded to the additional nodes, likely due to the heavily skewed nature of the dataset.

I found that Snowflake was much faster than the 2 node RedShift cluster. This could be due to differences in node speeds or the way Snowflake computes joins. Further investigation and additional resources will be needed to determine larger patterns in how these systems calculate complex multi-table joins.

2 Evaluated Systems

I chose 2 distributed systems to benchmark: Amazon Redshift and Snowflake.

Redshift has at least two compute nodes specified by the user upon creation of a cluster. They are coordinated through a leader node which prepares the query execution plans and distributes the execution code to the nodes. The compute nodes each have slices that contain some portion of its memory and disk. Slices work in parallel on different assigned loads to get the result.[3]

The Redshift Query Optimizer generates plans with massive parallel processing in mind, taking advantage of columnar data storage.

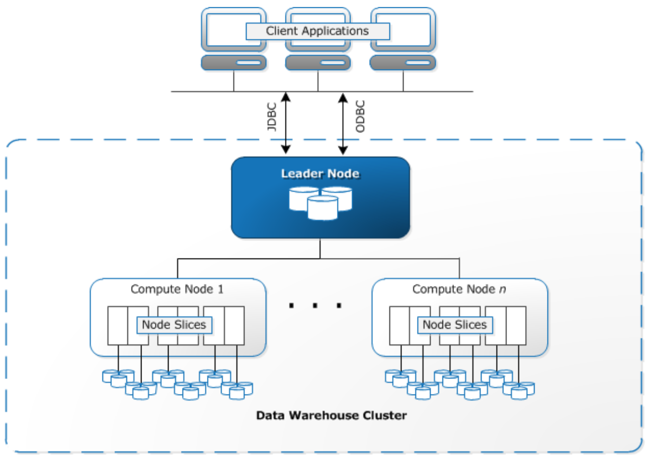


Figure 1: Amazon Redshift Architecture (credit: https://docs.aws.amazon.com/redshift/latest/dg/c_high_level_system_architecture.html)

Snowflake uses a central data repository that is accessible from all compute nodes, backed by Amazon S3. It allows the user to create "virtual warehouses" (VW) which are compute clusters that can load data or run queries. These VWs come in "t-shirt' sizes and may be scaled up or down on demand. The cloud services layer handles encryption, authentication, and other services in Snowflake including the query optimization.[4]

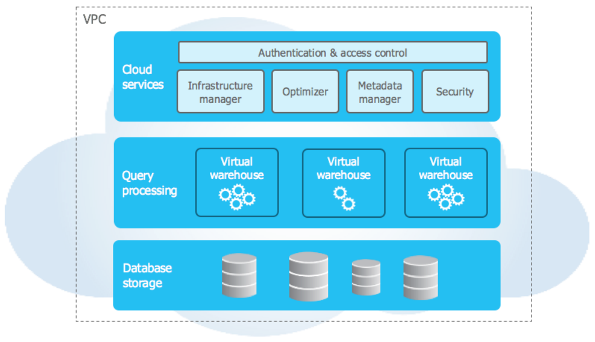


Figure 2: Snowflake Architecture. (credit: https://docs.snowflake.net/manuals/user-guide/intro-key-concepts.html)

Both of these systems use a Massive Parallel Processing, Columnar store designed for high-speed analytic queries.

3 Problem Statement and Method

The questions I had were as follows:

- *Which platform has the best performance in the JOB queries given similar setups?*
- *Why would certain queries have performance differences amongst the systems?*
- *How does adding additional nodes affect the speedup of the queries?*

Running all 113 queries is out of the scope of this project as I would be running each multiple times to get a more accurate measurement. Instead I ran a sample of the queries to get a better sense of the runtimes and chose 10 I found that had interesting joins and reasonable runtimes. The process I took is detailed below.

1. Acquire the data from IMDB and upload to S3

2. Ingest the data into a RedShift cluster with 2 dc2.large nodes

3. Run each query with a hot cache. Record the runtimes

4. Change the cluster to 4 nodes and repeat runs

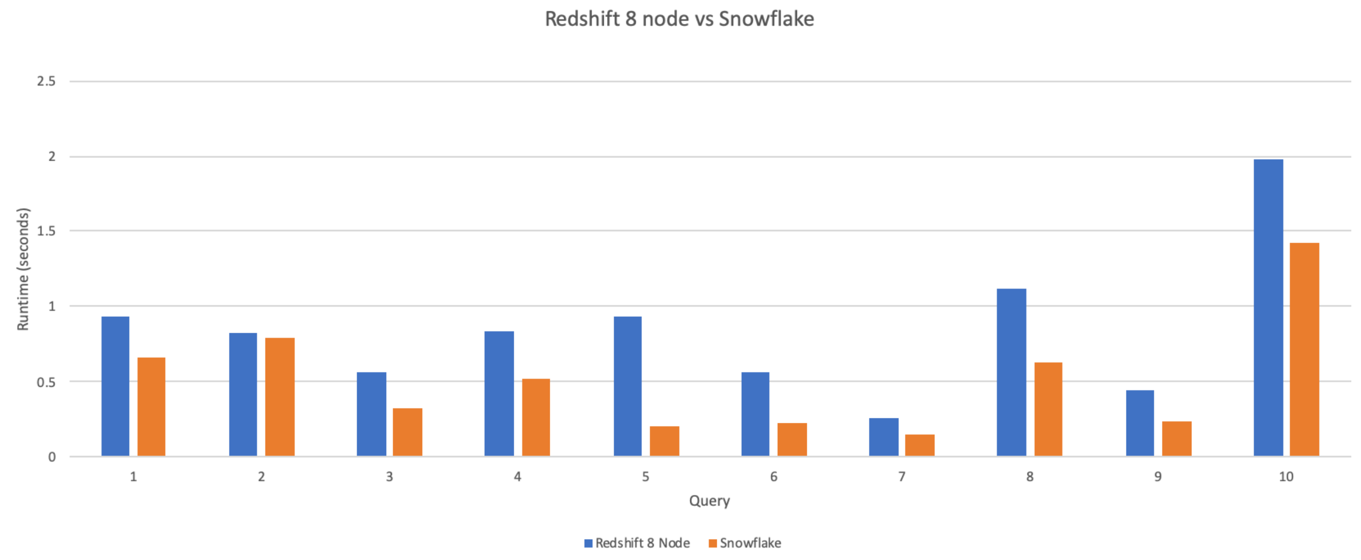
5. Change the cluster to 8 nodes and repeat runs.

5. Ingest the data into Snowflake warehouse

6. Run each query with a hot cache and record the runtimes

7. Plot the query runtimes. Re-run any queries with interesting differences using EXPLAIN to compare the query plans and cost estimates

I have access to a Snowflake XL virtual warehouse. According to the documentation, there is 16 server/cluster in the XLarge size[5]. This is the reason I have chosen a 8 node Redshift cluster to compare, as it is as close as I can get given my available resources. I understand that there are also differences in the node speeds and sizes.



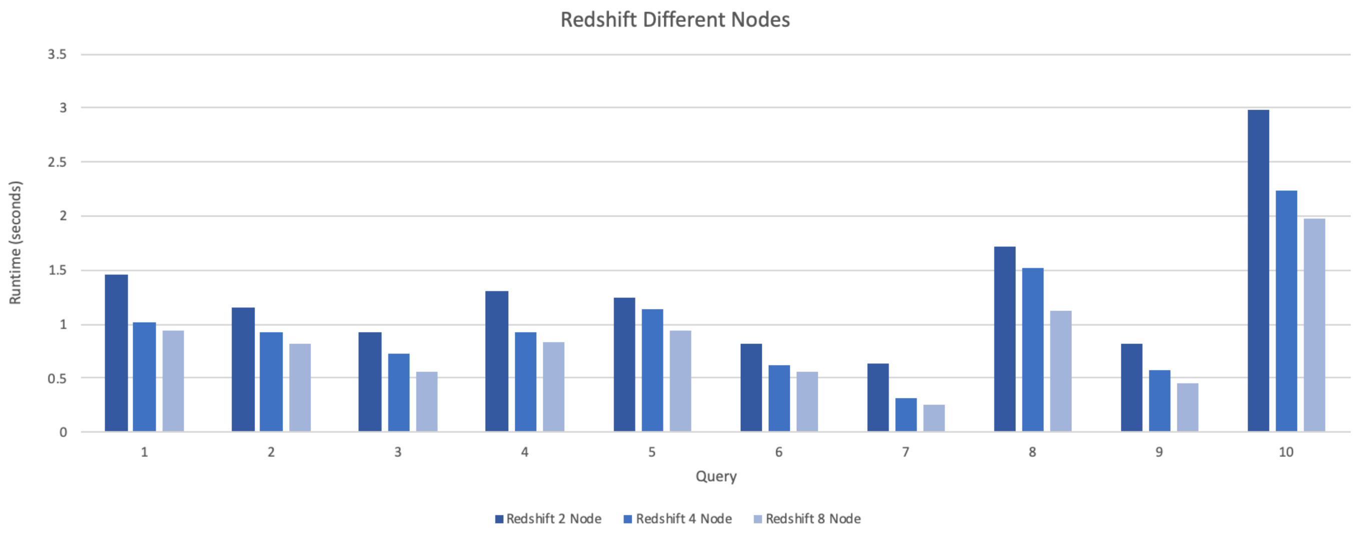


Figure 3: A selection of some interesting query runtime differences.

4 Results

When comparing runtime across node configurations, I found that Redshift does not achieve anything close to linear speedup. The distribution in query 1 was what I saw the most of. A sizable drop when going from 2 to 4 nodes, then a small drop when going to 8. Many queries only had a fractional runtime decrease. This isn't all surprising as the dataset is designed to be heavily skewed so the bulk of the runtime might have been waiting for the few nodes who were assigned more work. For reference, query 10 was JOB query 27a and involved 25 join conditions with 3 correlated 'IN's and 2 'LIKE's. It's difficult to discern the reason for the query runtimes. Each query has dozens of join conditions and correlations.

Snowflake was generally much faster than the Redshift 8 node cluster. It was sometimes more than twice as fast. This could be due to differences in the node speeds as I cannot control those to be the exact same. There was one case where Snowflake and RedShift were very close, that being JOB query 7b, including 10 join conditions and many other filtering conditions. Viewing the EXPLAIN plan (Query profile for Snowflake) shed a bit of light on this. Both were filtering the same tables in the start but used completely different join orders for the other tables. The greater patterns in join orders are difficult to parse by just looking at a few, very different queries. I would need much more time and compute resources to investigate this further.

5 Conclusion

In this project I ran the Join Order Benchmark on different configurations of RedShift and Snowflake. I found that the highly skewed nature of the data made for poor speedup as nodes are added. I also found that Snowflake computes join orders quite differently from RedShift leading to runtimes that are usually a few seconds faster, but not always.

REFERENCES

[1] Leis, Viktor, et al. "How good are query optimizers, really?." Proceedings of the VLDB Endowment 9.3 (2015): 204-215.

[2] (IMDB May 2013) http://homepages.cwi.nl/~boncz/job/imdb.tgz

[3] “Redshift Architecture and Its Components” https://hevodata.com/blog/redshift-architecture/

[4] Snowflake Key Concepts & Architecture https://docs.snowflake.net/manuals/user-guide/intro-key-concepts.html

[5] Snowflake Overview of Warehouses https://docs.snowflake.net/manuals/user-guide/warehouses-overview.html#warehouse-size