Comparison of AWS Redshift and Snowflake on Real-World Data

Anushna Prakash and Preston Stringham

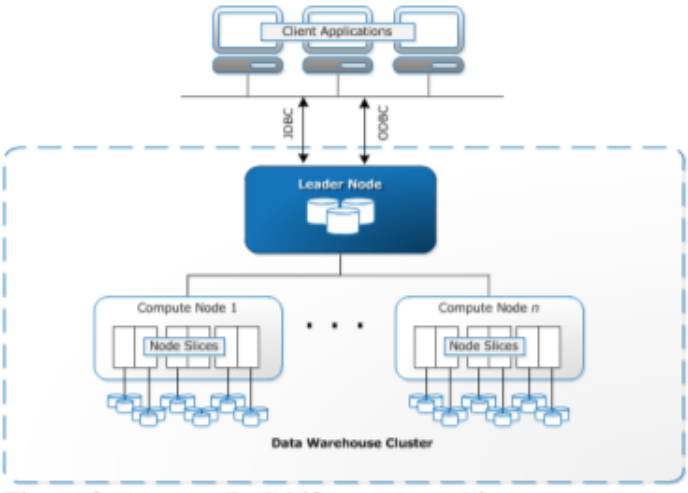
I. Introduction

AWS Redshift is a well-established database system that has an ability to scale and is widely used. Snowflake is a newer database system that can read data from Amazon S3 buckets and also boasts an ability to scale quickly and easily. However, how do these systems compare to one another on real-world, messy data? We will use the data available from TPC-H and IMDb under their personal and non-commercial license to write queries and compare the execution times on each system, with varying levels of complexity.

TPC-H is a commonly-used data set that was created to be of industry-wide relevance. It is used for benchmarking across different systems. The paper "How Good are Query Optimizers, Really?" by Viktor Leis et al. [1] performed analyses that challenged using TPC-H as a benchmarking tool by comparing some unnamed database systems as well as PostGres and HyPer on 2013 IMDb data. While the join-order benchmark (JOB) queries written for that paper had increasing numbers of joins, they did not include any aggregations. We will compare performance in terms of query runtime on queries with and without aggregations, with varying numbers of joins to increase complexity and better replicate common use cases for data analysts. We also revisit the IMDb data from the same source, with the data having grown from 3.9GB to 5.5GB since 2013. In this way, we can compare how Redshift and Snowflake perform on the standard benchmarking dataset, but also on real-world data to see if the performance is noticeably different between platforms.

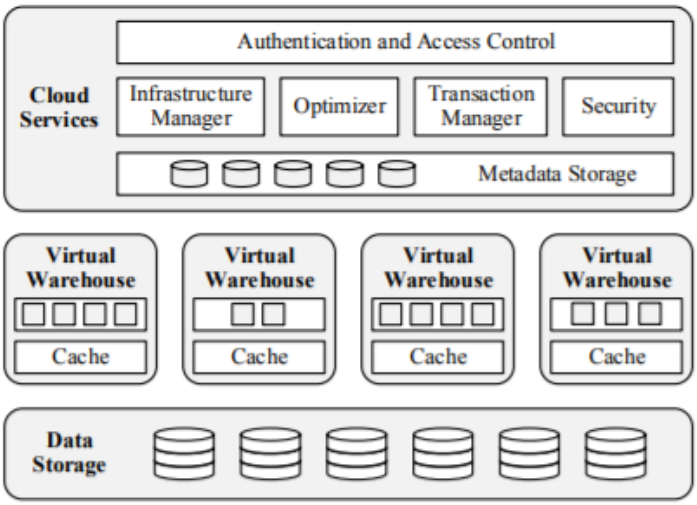
II. Evaluated Systems

One of the two systems that we are comparing is Amazon Redshift. Redshift is a popular cloud-based database system that boasts the ability for its users to change the number of nodes in the database cluster for more processing power. It has at least two nodes, a minimum of one compute node and one leader node [2]. The leader node accepts connections from clients, parses the request, and performs aggregations when required. The worker node(s) does the heavy computational lifting, by doing the query processing and data manipulations.



We will compare Redshift against Snowflake. Snowflake allows users to access their data through S3 buckets and shares that data with all virtual warehouses in a data lake. Warehouses can elastically scale up the number of nodes within t-shirt sizes like X-Small, Small, Medium, etc. having nodes that scale up by a factor of 2 for each size. There is a cloud services layer as well which is the so-called brain of the system, handling the "virtual warehouses, queries, transactions, and all the metadata that goes around that: database schemas,

access control information, encryption keys, usage statistics and so forth" [3].



III. Problem Statement and Methods

The question that we are trying to answer is how popular database systems scale with the number of joins and aggregations. This is based on the work done by Leis et al.'s, "How Good Are Query Optimizers, Really?" In this paper, a similar process of comparing popular database systems was done, but the queries used had no aggregations. This is the motivation for our work as aggregations are used very frequently among data scientists and analysts. To do this, we follow the steps of Leis et al., we use the IMDb and TPC-H databases to run our queries against. We will use Amazon's Redshift and Snowflake as our two benchmarking platforms. Both of these database systems are used at large scales by companies in industry. For this reason we feel these database systems are perfect candidates for benchmarking our queries.

While datasets like the TPC-H are commonly used for benchmarking performance, the data can be simplistic and unskewed, which causes it to resemble actual use cases less. In this paper, we use the data available from IMDb to measure performance on increasing the number of joins and with aggregations. We expect to see a drop in performance as the number of both joins and aggregations increase. The IMDb dataset consists of seven tables that combined are about 5.5GB of data.

Our queries put an emphasis on aggregation which differs from the work of Leis et al. by addressing a key concept used by data scientists and analysts on a daily basis. We compile a suite of queries to test the database systems. This suite contains queries varying in number of joins and aggregations to see how other database systems compare.

We utilize a dc.2 large cluster in Redshift with 1, 2, and 4 nodes and compare against Snowflake's x-small (1 node), small (2 node), and medium (4 node) warehouse. The differences in compute power of each node is obscured, particularly in Snowflake, so we gathered data on these three configurations to get a better understanding of the performance on each configuration. Query runtimes are averaged over 5 runs and disabling results-caching on both platforms.

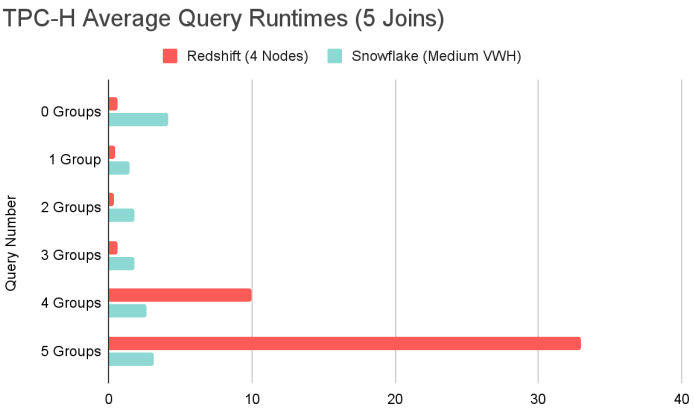
IV. Results

Across almost all configurations and data sets, Snowflake had the lowest average runtimes. Even as the joins and group-bys increased, Snowflake had little variance in its runtime and stayed well below 6 seconds, whereas Redshift had varying performance across configurations, increasing in runtime as the complexity increased. While downgrading the warehouse size in Snowflake did not increase the query runtimes by more than a second at most, there was a significant difference between 1 node, 2 nodes, and 4 nodes in Redshift, with 4 nodes having the best performance of these configurations.

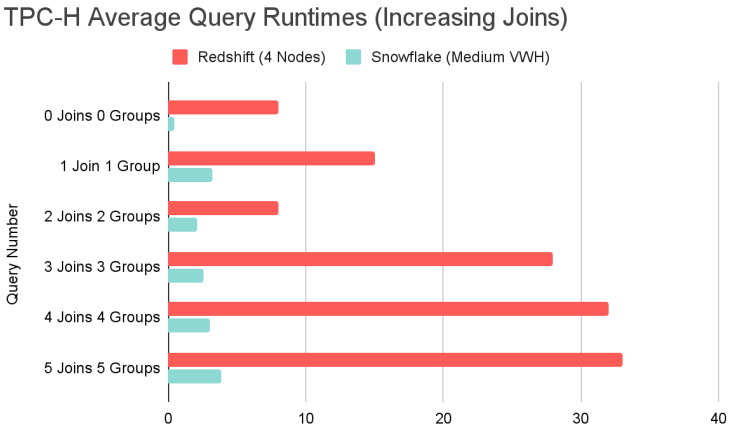
A. TPC-H

Redshift was quicker to join 5 tables with 0, 1, 2, and 3 group-by statements, but

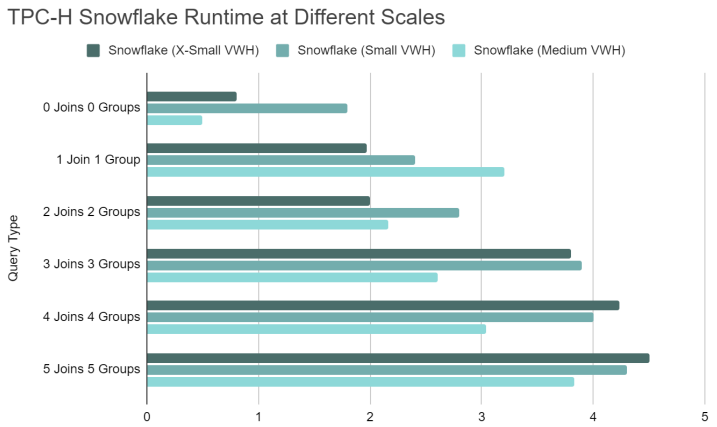
increased nearly exponentially in runtime as the groups increased. Redshift had higher average runtimes on TPC-H, the larger dataset, by up to 25 seconds.

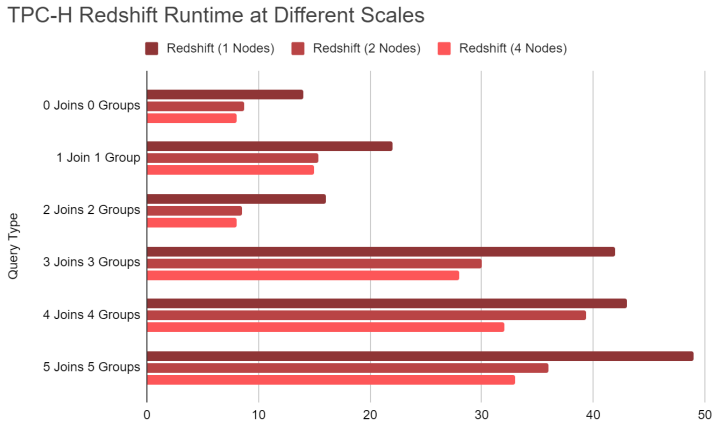


In general, the query with the longest runtime was the one with the most number of joins and group-by statements. For both Redshift, runtimes were lowest for 2 joins and group-clauses, and for Snowflake was lowest for 0 joins and group-bys.



Whereas Snowflake ran all queries in all configurations in under 5 seconds, Redshift took anywhere from 8 to 50 seconds, with the 1 node configuration being the most suboptimal and 4 nodes being the most optimal.





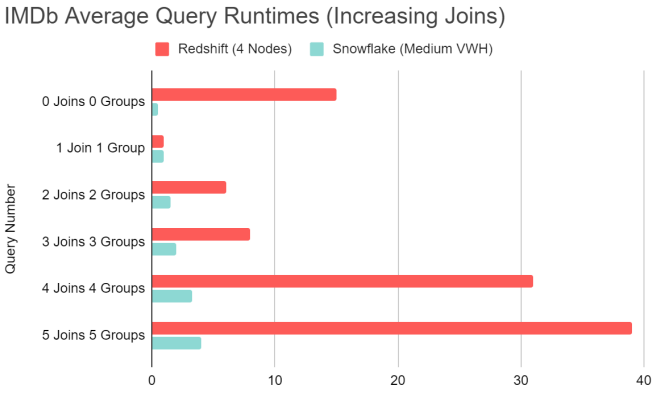
Interestingly, when we hold the number of joins constant at 5 and increase the number of groups, Redshift outperforms Snowflake from 0 to 3 groups. However, after 3 groups, Snowflake's runtimes do not vary significantly while Redshift takes up to 30 seconds. Overall, Redshift was unable to handle higher aggregation complexity as well as Snowflake.

B. IMDb

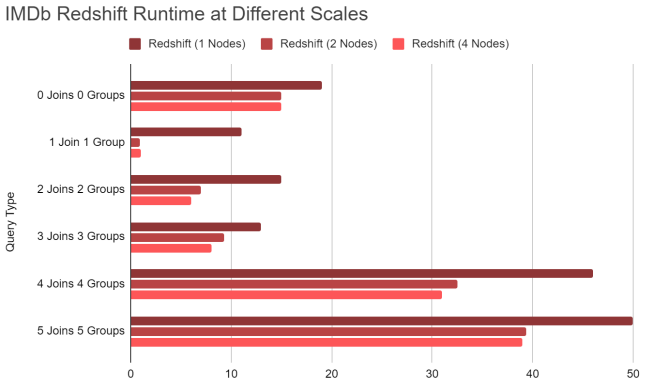
Despite the IMDb dataset being almost half the size of TPC-H, query runtimes were longer on Redshift. Snowflake saw some variance in average runtimes, but all queries under all configurations still completed in under 6 seconds. Here as well we saw a positive correlation between query complexity and runtimes. Again, the overall winner was

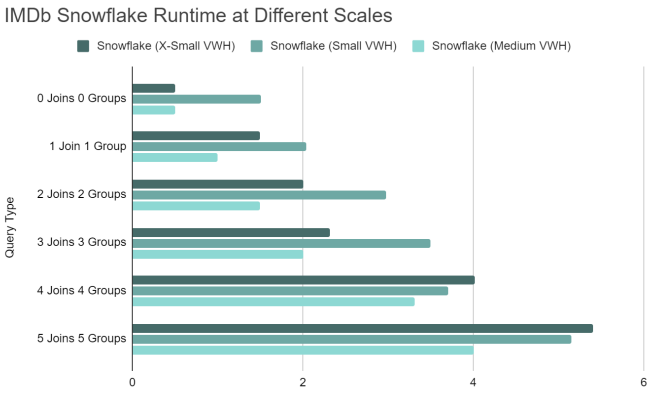
Snowflake on the medium warehouse, and Redshift had its lowest runtimes on a four node configuration.

Redshift had the fastest runtimes on the query with 1 join and group-by statement, taking about 1 second. Interestingly, the query with no joins nor group-bys took over 15 seconds to run. The range across all configurations for Snowflake's query runtimes was between 0.5s to 5.6s, and for Redshift was between 1s to 49s.

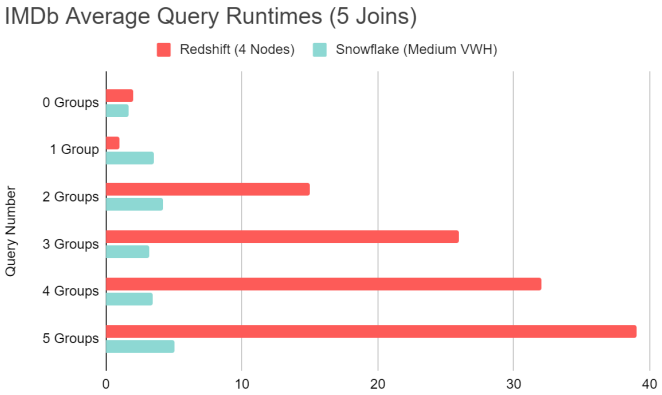


We found that Redshift with 1 compute node performed poorly when compared to Snowflake on a X-Small virtual warehouse. This makes sense considering the amount of overhead that comes with using a large SaaS platform such as AWS's Redshift. In general, we found that the runtime increased as the number of joins and groups increased, which does correspond to what we learned in the JOB paper.





Next, we ran queries that focused just on the group by clause in particular. For this test, we held the number of joins constant at 5 joins and varied the number of groups. We see that it clearly takes more time for both Redshift and Snowflake to form groups, but Redshift in particular behaved much more inefficiently especially with a larger number of groups. Snowflake, on the other hand, performed much more to our expectations. Redshift's timing clearly increased linearly as the number of groups increased.



V. Conclusion

The overwhelming winner of this competition is Snowflake. It is possible that the overhead in Redshift is causing much variation in the runtime of queries with an increasing number of groups. Queries run in Redshift do not behave as we would expect. In particular, we saw that in the low number of joins and groups, Redshift outperformed

Snowflake. However, this did not scale as we increased the number of joins and groups where we saw Snowflake outperform Redshift substantially.

To understand why this may be occurring, we tried to look into the query plans for each system. However, this was inconsistent between platforms. Redshift has a query plan which shows each step and its estimated cost, whereas Snowflake obscures its cost estimation and instead shows the steps and the number of partitions involved. This made it difficult to compare why Redshift made the decisions that it made in its query plan that could explain why it was behaving so inefficiently, as was done in the Join Order Benchmark paper.

Another challenge was creating identical configurations between each system. Through the free trial for AWS, we are only able to use dc.2 large nodes. It is unclear if the computational power provided by a single node in Snowflake is identical. Unfortunately, we were unable to determine if Snowflake had equivalent types of nodes and were unable to control for this difference.

It is also possible that Redshift's need to compile queries in C first is contributing to the longer runtimes. This would be an obvious disadvantage to Redshift as it would increase query times, but we did see cases in which Redshift outperformed Snowflake. Ultimately, further research is warranted to tease out the effects of individual node computational ability, query plan development, and level of overhead on average query runtimes.

VI. References

[1] Leis, V., Gubichev, A., Mirchev, A., Boncz, P., Kemper, A. & Neumann, T. (2015). How Good Are Query Optimizers, Really?. *Proc. VLDB Endow.*, 9, 204--215. doi: 10.14778/2850583.2850594

[2] Anurag Gupta, Deepak Agarwal, Derek Tan, Jakub Kulesza, Rahul Pathak, Stefano Stefani, and Vidhya Srinivasan. 2015. Amazon Redshift and the Case for Simpler Data Warehouses. In Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD '15). Association for Computing Machinery, New York, NY, USA, 1917–1923. DOI: 10.1145/2723372.2742795

[3] Benoit Dageville, Thierry Cruanes, Marcin Zukowski, Vadim Antonov, Artin Avanes, Jon Bock, Jonathan Claybaugh, Daniel Engovatov, Martin Hentschel, Jiansheng Huang, Allison W. Lee, Ashish Motivala, Abdul Q. Munir, Steven Pelley, Peter Povinec, Greg Rahn, Spyridon Triantafyllis, and Philipp Unterbrunner. 2016. The Snowflake Elastic Data Warehouse. In Proceedings of the 2016 International Conference on Management of Data (SIGMOD '16). Association for Computing Machinery, New York, NY, USA, 215–226. DOI: 10.1145/2882903.2903741