Fivetran

Data Warehouse Benchmark

The source code for this benchmark is available at https://github.com/fivetran/benchmark

Data Warehouse Benchmark

Redshift, Snowflake, Azure, Presto, BigQuery

Fivetran is a data pipeline that syncs data from apps, databases and file stores into our customers' data warehouses. The question we get asked most often is "what data warehouse should I choose?" In order to better answer this question, we've performed a benchmark comparing the speed and cost of five of the most popular data warehouses:

- Amazon Redshift
- Snowflake
- Azure SQL Data Warehouse
- Presto
- Google BigQuery

Benchmarks are all about making choices: what kind of data will I use? How much? What kind of queries? How you make these choices matters a lot: **change the shape of your data or the structure of your queries and the fastest warehouse can become the slowest.** We've tried to make these choices in a way that represents a typical Fivetran user, so that the results will be useful the kind of company that uses Fivetran.

A typical Fivetran user might sync Salesforce, Zendesk, Marketo, Adwords and their production MySQL database into a data warehouse. These data sources aren't that large: a typical source will contain tens to hundreds of gigabytes. They are complex: they contain hundreds of tables in a normalized schema, and our customers write complex SQL queries to summarize this data.

The source code for this benchmark is available at https://github.com/fivetran/benchmark

What data did we query?

We generated the the TPC-DS[1] dataset at 100GB and 1TB scales. TPC-DS has 24 tables in a star schema; the tables represent web, catalog, and store sales of an imaginary retailer. The largest fact table had 400 million rows at 100GB scale and 4 billion rows at 1TB scale[2].

What queries did we run?

We ran 99 TPC-DS queries[3] in August-September of 2018. These queries are complex: they have lots of joins, aggregations, and subqueries. We ran each query only once, to prevent the warehouse from caching previous results. Because we ran each query only once, these times include both the time to compile the query and run the query. Modern data warehouses compile SQL queries into highly optimized programs, and this compilation takes time. For most data warehouses, the compile time is a small fraction of the overall time, but Redshift has an unusually slow compiler. You can observe this by running each query twice; the second run is much faster because Redshift re-uses the compiled query:

Geomean runtime for 1st and second run		
	1st Run	2nd Run
100 GB	18.1	3.8
1 TB	20.3	6.3

Redshift tries to compensate for its slow compiler by doing fine-grained caching of compiled query plans. Even if you make small changes to your query, it can usually re-use most of the compiled plan from the previous version of the query. The fact that Redshift is so dependent on the effectiveness of its compilation cache makes it difficult to summarize performance in a single number. If you give Redshift a query it's never seen before, it will be much slower---how big of a problem is this? How effective is Redshift's cache in practice? We'd like to study this issue in a future iteration of this benchmark; for now we chose the simple approach of "only present each query once", and acknowledge that this may be a bit unfair to Redshift.

---

[1] TPC-DS is an industry-standard benchmarking meant for data warehouses. Even though we used TPC-DS data and queries, this benchmark is not an official TPC-DS benchmark, because we only used one scale, we modified the queries slightly, and we didn't tune the data warehouses or generate alternative versions of the queries.

[2] This is a small scale by the standards of data warehouses, but most Fivetran users are interested data sources like Salesforce or MySQL, which have complex schemas but modest size.

[3] We had to modify the queries slightly to get them to run across all warehouses. The modifications we made were small, mostly changing type names. We used BigQuery standard-SQL, not legacy-SQL.

How did we configure the warehouses?

We set up each warehouse in a small and large configuration for the 100 GB and 1TB scales:

	100 GB		1 TB	
	Configuration	Cost / Hour	Configuration	Cost / Hour
Redshift	8x dc2.large	$2.00	4x dc2.8xlarge	$19.20
Snowflake	X-Small	$2.00	Large	$16.00
Azure[4]	DW200	$2.42	DW1500c	$18.12
Presto[5]	4x n1-standard-8	$1.23	32x n1-standard-8	$9.82
BigQuery[6]	On-demand			

We tried to match the configurations so that we were comparing similar configurations across warehouses.

How much does "tuning" improve Redshift?

These data warehouses each offer advanced features like sort keys, clustering keys, and date-partitioning. We chose not to use any of these features in this benchmark[7]. We've gotten some passionate feedback on this issue, both on github and at conferences where we've presented our work. We tested some alternative configurations of Redshift to explore the effect of tuning using SORTKEY, which improves the performance of WHERE clauses and joins, and DISTKEY, which improves the performance of joins.

	100 GB			1 TB		
	None	Dist Keys	Aggressive	None	Dist Keys	Aggressive
1st-run	18.1	16.2	16.1	20.3	18.5	17.0
Speedup		11%	11%		9%	16%
2nd-run	3.8	3.3	2.7	6.3	5.5	3.6
Speedup		12%	29%		12%	42%

[4] The DW1500c configuration for Azure uses Microsoft's Gen2 architecture; the Gen2 architecture is not yet available in a small warehouse size appropriate for the 100 GB scale, so we used the Gen1 DW200 configuration at that scale.

[5] Presto is an open-source query engine, so it isn't really comparable to the commercial data warehouses in this benchmark. But it has the potential to become an important open-source alternative in this space. We used v0.195e of the Starburst distribution of Presto. Cost is based on the on-demand cost of the instances on Google Cloud.

[6] BigQuery is a pure shared-resource query service, so there is no equivalent "configuration"; you simply send queries to BigQuery, and it sends you back results.

[7] If you know what kind of queries are going to run on your warehouse, you can use these features to tune your tables and make specific queries much faster. However, typical Fivetran users run all kinds of unpredictable queries on their warehouses, so there will always be a lot of queries that don't benefit from tuning.

We tested two levels of optimization, “Dist Keys” and “Aggressive”. “Dist Keys” represents a conservative level of optimization: we added DISTKEY hints on the columns we knew would be used in joins, and we added DISTSTYLE ALL to small tables. In “Aggressive” we added SORTKEY hints to columns that were used in WHERE clauses and joins; we consider it aggressive because these SORTKEY constraints sometimes contain multiple columns in an order that correspond to the exact way they are used in the TPC-DS queries; this level of optimization is probably not realistic for a real-world scenario where the set of queries isn’t fixed.

We report both 1st-run and 2nd-run times to show the impact of query compilation times. These tuning hints don’t make the Redshift compiler any faster, so they have a much more dramatic effect when you look at 2nd-run times. Azure SQL data warehouse has similar tuning features; we didn’t perform these experiments on Azure but it would probably show similar speed improvements. Snowflake, BigQuery and Presto each have a concept of “partitioned tables”, which can be used to improve the performance of WHERE clauses, similar to SORTKEY. Snowflake and BigQuery might infer a distribution-style in the background when you repeatedly run similar queries; we didn’t try to assess this. We did apply column compression encodings in Redshift; Snowflake, Azure and BigQuery apply compression automatically; Presto used ORC files in HDFS, which is a compressed format.

Presto is highly dependent on how you store the data

Presto recently got a cost-based query planner, developed over the last year by the Presto teams at Facebook and Starburst Data. This makes a huge difference in performance on the TPC-DS benchmark, but it depends on having accurate table statistics to estimate the cost of various query plans. Presto has “pluggable” storage backends: you can run Presto on HDFS, S3, GCS, and other “storage layers”. Since the cost-based query planner is relatively new, not all storage backends support table statistics yet. We ran on Orc files in HDFS using the HIve metastore, which has table statistics, but it will take some time for all the storage backends to “catch up”. In our experiments, running Presto without table statistics makes it about 2x slower in this benchmark.

Results

Redshift was about 2x slower than the fastest warehouses, due to its slower query compiler. However, if we had reported 2nd-run times rather than 1st-run times, Redshift might have been the fastest warehouse.

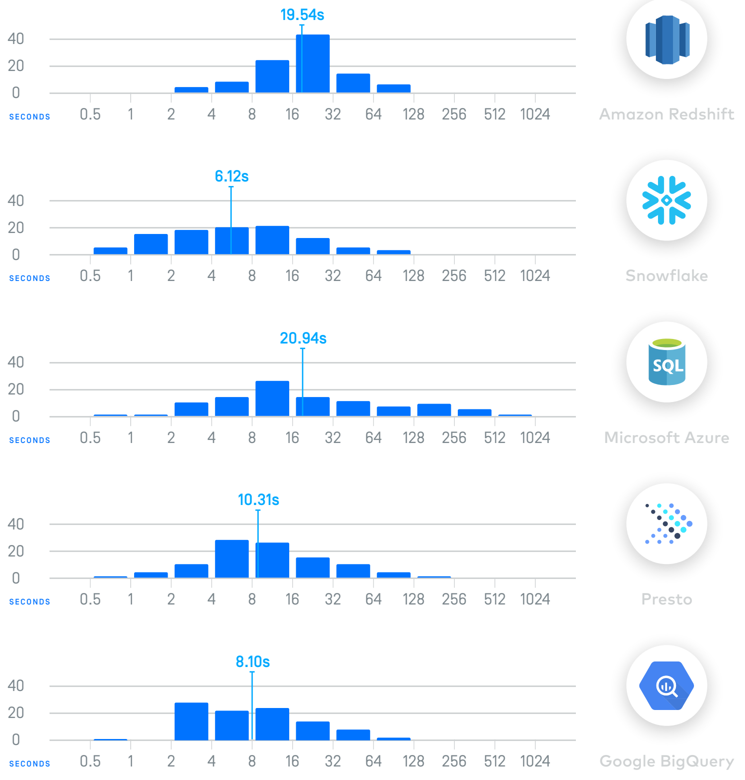
Azure was similar to the other warehouses at 1TB scale, but slower at 100GB scale. The reason for this is that we used the Gen2 architecture at 1TB scale and the Gen1 architecture at 100GB scale. The Gen2 architecture isn't yet available in a small size that would make sense for a 100GB benchmark.

The differences you see here are modest on the scale of data warehousing. Changes in workloads and how you structure your data can easily change performance by a factor of 2 in either direction. The qualitative differ-ences discussed later are much more important if you're making a decision between data warehouses.

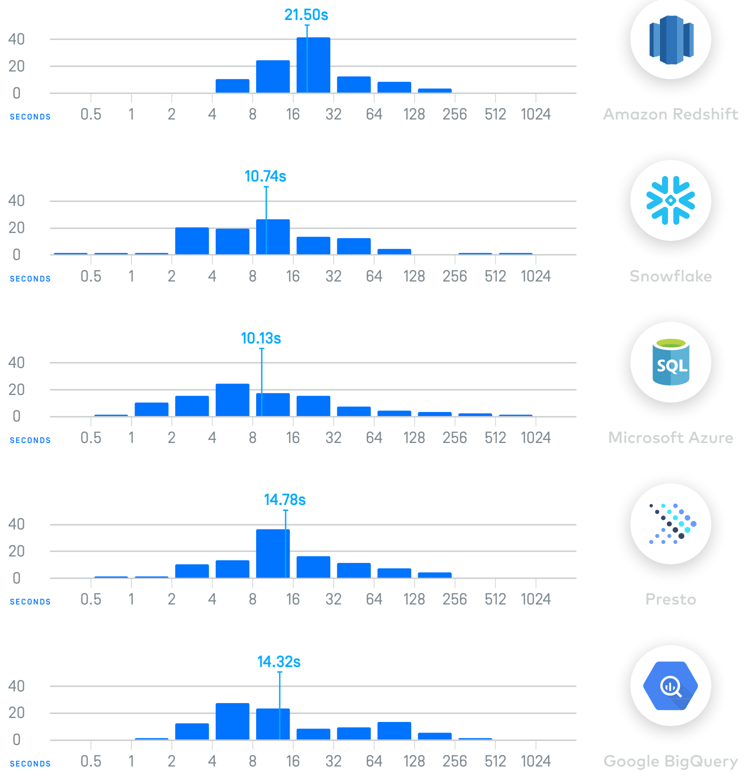
Which data warehouse is the fastest?

Histogram of performance for 99 TPC-DS queries (seconds, log scale) with geometric mean

100 GB



1 TB



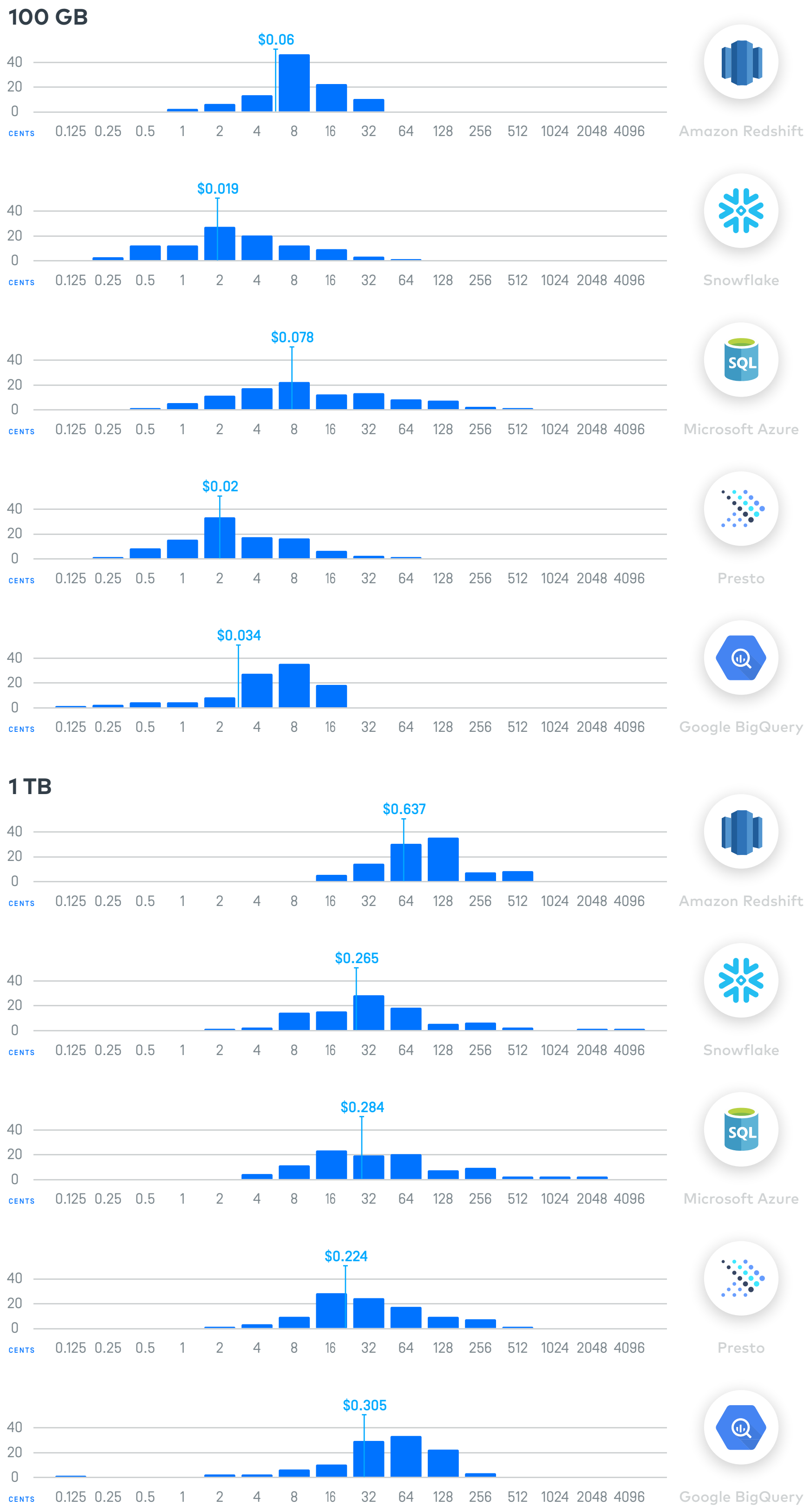
Results

BigQuery charges per-query, so we are showing the actual costs billed by Google Cloud. To calculate cost-per-query for other warehouses, we made an assumption about how much time a typical warehouse spends idle. For example, if you run a Snowflake X-Small warehouse for 1 hour at $2 / hour, and during that time you run 1 query that takes 30 minutes, that query cost you $2 and your warehouse was idle 50% of the time. On the other hand, if you run two 30-minute queries and the warehouse spends 0% of the time idle, each query only costs you $1. We looked at the actual usage data of a sample of Fivetran users with Redshift warehouses. The median Redshift cluster was idle 82% of the time[8].

This comparison depends heavily on our assumption about idleness[9]. If you have a very “spiky” workload, BigQuery would be much cheaper than the other warehouses. If you have a very “steady” workload, BigQuery would be much more expensive.

Which data warehouse is the cheapest?

Histogram of cost for 99 TPC-DS queries (cents, log scale) with geometric mean



[8] *Some readers may be surprised these data warehouses are idle most of the time. Fivetran users tend to use their warehouses for interactive queries, where a human is waiting for the result, so they need queries to return in a few seconds. To achieve this performance target, you need to provision a large warehouse relative to the size of your data, and that warehouse is going to be idle most of the time. Also, note that this doesn’t mean these warehouses are doing nothing for hours; it means that there are many small gaps of idleness interspersed between queries.*

[9] *The formula for calculating cost-per-query is [Query cost] = [Query execution time] * [Cluster cost] / [1 — [Cluster idle time]]*

Is BigQuery flat-rate pricing cheaper?

For customers with at least $40k in monthly spend, BigQuery offers flat-rate pricing. When you sign up for flat-rate pricing, you pay up-front for a fixed number of “compute slots” allocated to you; it’s equivalent to paying for a large Redshift cluster and leaving it on all the time. The BigQuery API reports the “compute-slot-milliseconds” used by each query, and we used this to estimate how flat-rate pricing compares to on-demand pricing at 1 TB scale:

1 TB			
	Geomean	Mean	Median
BigQuery On-demand	$0.31	$0.45	$0.38
BigQuery Flat-rate	$0.18	$0.56	$0.14

These comparisons are tricky to interpret because the distribution of query time and therefore cost is always skewed right: a small percentage of queries take a very long time and dominate the mean. The performance of these queries isn’t intrinsically more important than the other queries in the benchmark, and real-world users typically “fix” slow queries by hand-optimizing their SQL or setting up scheduled transformations that do the hard parts of the query in advance. The geomean is an alternative summary statistic that gives equal weight to each query; you can also think of it as the mean in log-scale. Because the query runtime distributions are skewed right, the mean is always higher than the geomean and the median.

BigQuery flat-rate cost is more or less proportional to runtime; we find that flat-rate is significantly cheaper than on-demand according to the geomean and median statistics, but not the mean. Intuitively, you can think of this as: flat-rate is cheaper if you plan to eliminate the very-long-running queries by optimizing your queries and setting up pre-transformations.

This cost analysis comes with one giant caveat, which is that flat-rate pricing is only available once you reach $40k / month in spend. This is much higher than the configurations we consider in this benchmark, which run ~$12k / month at 1TB scale on-demand. Also, you can get significant discounts by committing to a minimum monthly spend with the other warehouses. To do a fair comparison, we would need to run larger warehouses at a 10 TB scale, and build in the committed-use discounts available from Snowflake, Redshift and Azure.

Which warehouse is best?

Speed and cost are not the only considerations; these warehouses have important qualitative differences. Based on our experience with real customers, we believe there are 5 key features that differentiate data warehouses:

- Elasticity: how quickly can a data warehouse increase and decrease capacity in response to changing workloads?
- Availability: how does the warehouse provide high uptime?
- JSON support: can you store and query JSON data?
- Can you tune WHERE clauses by partitioning the data?
- Can you tune JOINs by specifying data distribution?

We've summarized how Redshift, Snowflake and BigQuery compare on these criteria; we don't yet have enough customers with Azure and Presto to include them in a qualitative comparison.

Feature Matrix Qualitative differences between Redshift, Snowflake, and BigQuery			
Elasticity	🔴 HOURS [10]	🟠 MINUTE [11]	🟢 QUERY [12]
Availability	🔴 BACKUP [13]	🟢 DISTRIBUTED SYSTEM	🟢 DISTRIBUTED SYSTEM
JSON support	🔴 NO ARRAYS [14]	🟢 NATIVE [15]	🟠 USER-DEFINED FUNCTIONS [16]
Can you tune WHERE clauses?	🟢 SORTKEY	🟢 PARTITION BY	🟠 DATE PARTITIONING [17]
Can you tune JOINs?	🟢 DISTKEY / DISTSTYLE [7]	🔴 NO	🔴 NO

Ease of use ↕ Tuning

Redshift is the closest-to-the metal, most "tuneable" system ← Snowflake is a compromise → Google BigQuery is a 100% shared distrbuted system

Elasticity

Data warehouse workloads tend to be highly variable. For example, if you're using your data warehouse to power internal dashboards, you may see little usage at night. Redshift is the least elastic warehouse, because it stores data on the same nodes that it uses to run queries[10]. It can take hours to change the size of a Redshift warehouse.

BigQuery is the most elastic warehouse: each query runs independently. If you aren't running any queries, you pay nothing.

Snowflake is not quite as elastic as BigQuery. It stores the data separately from the compute nodes, and you can spin up a new Snowflake warehouse in a few seconds, but the minimum billable time for a warehouse is 1 minute. So it's not realistic to scale your Snowflake capacity up and down with each query.

Availability

Snowflake and BigQuery are both distributed systems: at all times, your data is stored in multiple locations to provide resiliency against failure. We have observed occasional, short outages of both systems, but we've never observed a Snowflake or BigQuery data warehouse "die" and need to be restored from a backup.

Redshift protects against node failures by making periodic backups to S3. Fivetran has been supporting Redshift as a destination for over 3 years; in that time we have observed Redshift clusters "die" a few times. When this happens, you have to restore your cluster from the backup. This can take several hours, and you lose any data that was added between when the backup occurred and when the cluster failed. It's not a huge problem---if you use Fivetran, you can simply resync the data from the source---but it's definitely an inconvenience when it happens.

JSON support

A lot of real-world data is in JSON format and doesn't conform to a fixed schema. Snowflake has the best support for JSON, via it's native VARIANT type and its proprietary extension to SQL for querying nested fields and arrays.

BigQuery is second-best: you can store JSON in a text column, and create a user-defined-function that converts JSON into a nested data structure that BigQuery can natively understand.

Redshift can only support extremely simple JSON structures, because it has no FLATTEN or UNNEST operator. In practice, this means it is extremely awkward to work with JSON arrays in Redshift.

---

[10] Redshift Spectrum allows you to query data stored in S3, but it comes with significant operational complexity and performance trade-offs, and it's still not possible to automatically increase and decrease Redshift compute capacity according to load.

Can you tune WHERE clauses?

Redshift allows you to add a SORTKEY hint to one or more columns in each table. If you write a query that has a predicate on the SORTKEY column, Redshift can skip scanning most of the table, and the query will be dramatically faster.

Snowflake provides similar functionality via partitioned tables. BigQuery also supports partitioned tables, but only when the partition column is a date[11].

Can you tune JOINs?

Redshift allows you to add a DISTKEY hint to one or more columns in each table. If you perform an equi-join where both sides of the equi-join have the DISTKEY hint, the join will be performed in-place, with no shuffle step. This can make large joins dramatically faster. It's not a panacea---you can only specify one distribution style for each table, so if different queries perform different join patterns, you may not be able to optimize them all.

Snowflake and BigQuery don't have an equivalent feature, though they may be inferring some optimizations automatically in the background.

Tuning versus ease-of-use

The qualitative differences between data warehouses reflect their fundamental design choices. Snowflake and BigQuery are both distributed systems, and separate compute from storage. This makes them simpler to operate and more elastic. However, they aren't as tuneable as Redshift. There are some optimizations that are only possible in a "close-to-the-metal" system like Redshift.

Why are our results different than previous benchmarks?

Amazon's Redshift vs BigQuery benchmark

In October 2016, Amazon ran a version of the TPC-DS queries on both BigQuery and Redshift. Amazon reported that Redshift was 6x faster and that BigQuery execution times were typically greater than 1 minute. The key differences between their benchmark and ours are:

- They used a 10x larger dataset (10TB versus 1 TB) and a 2x larger Redshift cluster ($38.40 / hour versus $19.20 / hour).
- They tuned the warehouse using sort and dist keys, whereas we did not.
- BigQuery Standard-SQL was still in beta in October 2016, it may have gotten faster by late 2018 when we ran this benchmark.

---

[11] Shortly before this writing, BigQuery introduced clustered tables in beta. Clustered tables allow you to partition tables on a non-date column, if the table is already partitioned on a date column.

Benchmarks from vendors that claim their own product is the best should be taken with a grain of salt. There are many details not specified in Amazon's blog post. For example, they used a huge Redshift cluster — did they allocate all memory to a single user to make this benchmark complete super-fast, even though that's not a realistic configuration? We don't know. It would be great if AWS would publish the code necessary to reproduce their benchmark, so we could evaluate how realistic it is.

Periscope's Redshift vs Snowflake vs BigQuery benchmark

Also in October 2016, Periscope Data compared Redshift, Snowflake and BigQuery using three variations of an hourly-aggregation query that joined a 1-billion row fact table to a small dimension table. They found that Redshift was about the same speed as BigQuery, but Snowflake was 2x slower. The key differences between their benchmark and ours are:

- They ran the same queries multiple times, which eliminated Redshift's slow compilation times
- Their queries were much simpler than our TPC-DS queries

The problem with doing a benchmark with "easy" queries is that every warehouse is going to do pretty well on this test; it doesn't really matter if Snowflake does an easy query fast and Redshift does an easy query *really really* fast. What matters is whether you can do the hard queries fast enough.

Periscope also compared costs, but they used a somewhat different approach to calculate cost-per-query. Like us, they looked at their customer's actual usage data; but instead of using percentage-of-time idle, they looked at the number of queries-per-hour. They determined that most (but not all) Periscope customers would find Redshift cheaper, but it was not a huge difference.

Mark Litwintschik's 1.1 billion taxi-rides benchmark

Mark Litwintshik benchmarked BigQuery in April 2016 and Redshift in June 2016. He ran 4 simple queries against a single table with 1.1 billion rows. He found that BigQuery was about the same speed as a Redshift cluster about 2x bigger than ours ($41 / hour). Both warehouses completed his queries in 1–3 seconds, so this probably represents the "performance floor": there is a minimum execution time for even the simplest queries.

Conclusion

These warehouses all have excellent price and performance. We shouldn't be surprised that they are similar: the basic techniques for making a fast columnar data warehouse have been well-known since the C-Store paper was published in 2005. These data warehouses undoubtedly use the standard performance tricks: columnar storage, cost-based query planning, pipelined execution, and just-in-time compilation. We should be skeptical of any benchmark claiming one data warehouse is orders-of-magnitude faster than another.

The most important differences between warehouses are the qualitative differences caused by their design choices: some warehouses emphasize tunability, others ease-of-use. If you're evaluating data warehouses, you should demo multiple systems, and choose the one that strikes the right balance for you.

Fivetran

About the benchmark

Fivetran produces one of the only unbiased warehouse reviews available through a lens representing hundreds of real-world user. We load data into cloud warehouses for 400+ organizations, which means we are experienced with each warehouse vendor and can maintain a neutral opinion. Fivetran is committed to providing an annual, unbiased comparison of cloud warehouses to help users navigate a hard IT decision for their organization.

About Fivetran:

After a five-minute setup, Fivetran replicates all your applications, databases, events and files into a high-performance data warehouse. Our standardized cloud pipelines are fully managed and zero-maintenance. Shaped by the real-world needs of data analysts, Fivetran technology supports agile analytics, enabling data-backed decisions across organizations. We're focused, transparent and trusted by businesses that run on data.

To learn more, visit fivetran.com