2024년 09월 06일

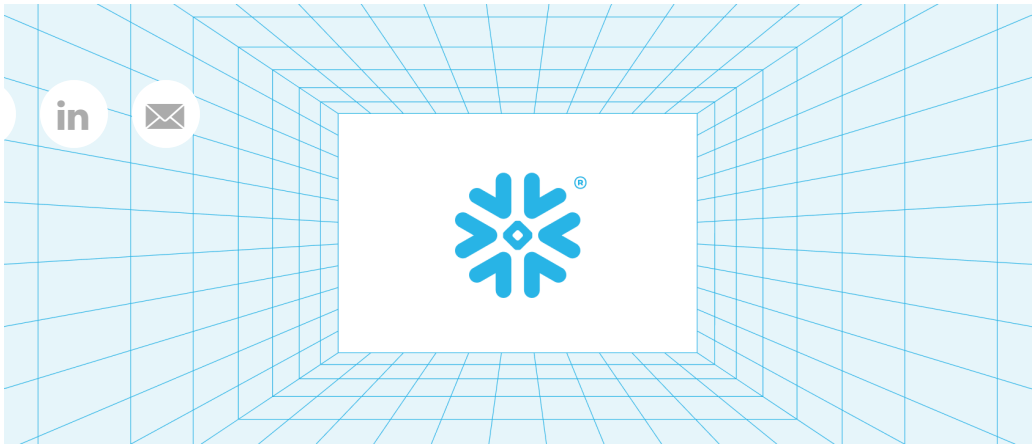
AUTHOR

Louis Magarshack

Benchmarking Real World Customer-Experienced Performance Using the Snowflake Performance Index (SPI)

Core Platform

SHARE



I'm excited to share some details about one of the projects that I've been working on as part of the Snowflake Product Data Science team in collaboration with the Snowflake Engineering team and the Snowflake Product Management team. Today, we announced the public launch of the Snowflake Performance Index (SPI), an aggregate index for measuring improvements in Snowflake performance experienced by customers over time.

One of the key elements of any cloud data service is the price/performance of the platform. In the past, vendors have turned to synthetic benchmarks as a proxy to showcase and claim performance advantages over competitors. However, this approach doesn't account for the characteristics of real-world customer workloads such as demand elasticity, complex security access policies, and schema design. Synthetic benchmarks tell you one thing — how fast a system performs on that specific benchmark.

To deliver the best possible price/performance profile for our customers, the SPI helps to proactively **discover and measure performance**

improvement opportunities using customer workloads instead of synthetic benchmarks. The Snowflake Performance Index identifies stable customer workloads across hundreds of thousands of Snowflake Virtual Warehouses and measures key performance indicators for the workloads running on these warehouses.

The metrics tracked by the SPI include metrics such as total query duration, query compilation time, query execution time, and bytes written per query over time, and we plan to track and analyze additional metrics to help us quantify how the improvements we make to Snowflake impacts customers. By tracking a set of key metrics for stable workloads over time, the SPI allows us to quantitatively measure performance improvements experienced by customers over time for their production workloads.

AUTHOR

Louis Magarshack

The Challenge

The limitations of synthetic benchmarks are well known. For example, this VLDB paper clearly breaks down what one can hope to measure from running TPC-DS. We derive value from TPC-DS as one of the steps in our testing and release pipeline. However, we find that it does not properly represent customer workloads and potential pain points. So, it cannot be used to prioritize performance work, and it cannot be used to measure real impact of new changes.

An important missing performance indicator in TPC-DS queries are Data Manipulation queries (DMLs) which is the focus of a significant amount of engineering effort at Snowflake. In modern data applications, there are always continuous DML statements executing, so there is a much more complex interaction between these DML statements and queries that TPC-DS does not adequately represent.

Measuring performance in the most actionable way possible is hard. The difficulty of using customer workloads comes from the amount of noise they introduce: people run different queries, on different tables from one week to another, warehouse configurations & load both vary. To measure real-world performance, we needed to find a way to decouple workload changes with Snowflake system performance improvements so that we can track them separately.

We already have capabilities that can help gauge the impact of Snowflake improvements on customer workloads. Our Snowtrail testing framework lets us test out new changes on real customer queries and data without impacting customer production workloads. We use Snowtrail in every release to detect regressions, and we also use it to measure performance

improvements for specific changes. However, Snowtrail provides point of time measurements only, and Snowtrail runs that take place a few weeks apart are not comparable due to changes in underlying data. We also have an extensive A/B testing framework, but the same problem arises: we cannot keep long term holdouts to look at the impact of our work across many weeks. Once a feature is tested and is confirmed to improve customer performance, it has to roll out everywhere in a safe but prompt manner.

AUTHOR

Louis Magarshack

Solution

Above, we described how the noisiness of customer workloads makes measuring the performance of real workloads a challenging endeavor. Therefore, our first priority was to find a solution to manage the amount of noise we would have in our measurements. The most important concept was to automatically identify the stable workloads that our customers have. Whether it is pipelines or dashboards, some queries

tedly and should have predictable performance.

We define stable workloads in two different ways: **stable warehouses and stable recurrent queries.**

Building blocks

Stable warehouses

Those should be warehouses doing the same amount of work for each period, and we select them by identifying a set of metrics that should be stable as well as some fixed characteristics.

Some example stability indicator metrics include: warehouse credit consumption, number of queries, total bytes written, total bytes read, total execution time, total compilation time.

Stability indicators that should remain fixed include: warehouse size, warehouse scaling policy, maximum number of clusters, minimum number of clusters.

In the case of stable metrics, we define stability as a low enough coefficient of variation (standard deviation / mean), , computed across a fixed period of time. The threshold is set such that **the resulting set of warehouses is representative**: the type of customer queries running is comparable with the overall population, and **the number of warehouses is stable from one period to the next.**

This group of warehouses gives us a **great way to measure the performance impacts of broadly applicable changes**: representative set of

queries without as much noise. However, it is not as helpful to measure the performance impacts of very targeted changes. This is why we also use stable recurrent queries.

This is how the distribution of customer query statement type breaks down between stable and unstable warehouses:

AUTHOR

Louis Magarshack

The overwhelming majority of our customers have at least one stable warehouse.

We find that from one week to the next the set of stable warehouses ch s l wer than 5% including both churned and newly stable warehouses. Churn occurs once a warehouse stops meeting the definition above. We observed that over the course of a year, fewer than 30% of stable warehouses are different. This is relatively large and driven by the addition of new stable warehouses and workloads by customers as well as the churn of existing stable warehouses. The bulk of stable warehouses remain stable for a long period of time.

For example, here is an illustration of the churn from March 2023 to May 2023, using March as the basis for 100%:

We see that the number of stable warehouses changed by a couple of percentage points with churn & growth mostly compensating each other.

This leaves us with a set of warehouses that are both representative of and fixed.

Stable recurrent queries

This concept is very similar to stable warehouses. Here, we look for queries running every day and doing the same thing every day. We identify comparable queries via parameterized query text: removing literals, whitespace, comments and hashing it. We then use the same

framework as warehouses: identify a set of metrics that should be stable as well as some fixed characteristics.

Metrics that should be stable remain stable in terms of variance to mean ratio: number of queries, total bytes written, total bytes read, total execution time, total compilation time.

AUTHOR

Louis Magarshack

Characteristics that should remain fixed: warehouse used. Again, we define *stability* as a low enough coefficient of variation.

Stable recurrent queries are less representative compared to stable warehouses, for example DML operations are more repetitive than BI-oriented queries, however **they are very useful to measure the impact of targeted changes**. For example, a performance optimization impacting a small fraction of INSERT queries by a few percentage points might not be detectable at the warehouse level if that warehouse performs other operations. However, it will be detectable by looking at individual stable queries after the change has been rolled out.

This use-case is much closer to the one for **Snowtrail** we described earlier. It is essentially an extension with 2 key benefits. First we get a more economical solution, we do not need to re-run thousands of customer queries on our own virtual warehouses. Second, we get a safety net for changes that were not specifically targeted by Snowtrail: we should still be able to measure their impact using stable recurrent queries.

Applications

Indexing performance

A key question we want to answer is, how much faster has Snowflake become over time. To solve this we use **stable warehouses**. We present the information as a performance index, which is simply the **change in average performance for specific metrics in stable warehouses from one point in time until today.** We rely on the ratio of change rather than actual runtime because it gives us the flexibility to later improve our definition without the need to run potentially expensive backfilling operations. For example, at some point, we might want to change the definition to change the weight of individual customers such that no individual account contributes too much to the overall value.

Finally, we re-center the index on 0 to make it easier to read improvement numbers.

We call this resulting index the **Snowflake Performance Index, or SPI for short:**

AUTHOR

Louis Magarshack

SHARE

Visit **this** page to learn more about the most recent SPI result.

It is easy to read that **since we started tracking the index in August 2022, the relative query performance of Snowflake has improved by 15%**. This is caused by a number of factors: core performance optimizations (for example eliminating unnecessary joins), cloud infrastructure improvements (for example changing the CPU architecture we rely on), and concurrency improvements (for example improved query scheduling) across categories of changes in that time window. Those are very broadly applicable changes that we're able to deliver to all of our customers.

Those two views of the index are available for anyone to view on our **website** and they will be refreshed regularly as we keep improving performance.

Insight into the impact of specific improvements

When rolling out new performance improvements we always want to know: how much faster are queries now? To solve this we use both **stable warehouses and stable recurrent queries based on the type of change**.

For changes that impact all queries we would use stable warehouses, and for changes that impact specific queries we would use stable recurrent queries. In both cases we can **read off the impact from a table** such as this one:

Here we see a mix of performance metrics: Average query execution time, Average scheduling time, and cost metrics: Average bytes written, Average read requests.

In this case we determine significance by performing a paired hypothesis test where each pair is a repeat query before and after the change was applied.

Understanding component-level drivers of performance changes

AUTHOR

Louis Magarshack

Making sure that our performance is moving in the right direction is a great first step. The Snowflake Performance Index also helps answer the follow-up questions: why is the performance getting better? Is it caused by a specific dimension?

To solve this problem we use **stable warehouses** and compute the contribution of different dimensions to the change in index value. The dimensions we consider are: cloud region, customer, warehouse, statement type, time spent location and workload type. This will help quickly narrow down changes in performance driven by a combination of the factors. Time spent location includes things such as execution or compilation, but also more granular steps such as time spent on scan, aggregation or pruning.

We can then analyze the contribution from a bar chart:

For example, this is showing that MERGE statements did not improve as quickly as the rest for our selected time window: the index value would have been 0.01 points higher without their contribution.

Future plans

We will be sharing changes to our performance index regularly to hold ourselves accountable for continuously improving customer experience.

We will keep improving our definition of stable warehouses and stable recurrent queries: assigning different weights to different customers based on size to limit the impact of any single account, assigning more importance to warehouses that have been stable for a longer time to limit the impact of new warehouses over the course of a year.

We will introduce a definition of stable objects: tables, materialized views with fixed rates of churn, including tables that never change. This will

give us a good set of objects to analyze for changes focused on background services such as auto-clustering, search optimization or materialized views.

Credits

Special thanks to Josh Klahr, Rudi Leibbrandt (Rudi Leibbrandt), Shiyu Qu, Zerui Wei, Jiaqi Yan for all the contributions to this project.

AUTHOR

Louis Magarshack

SHARE