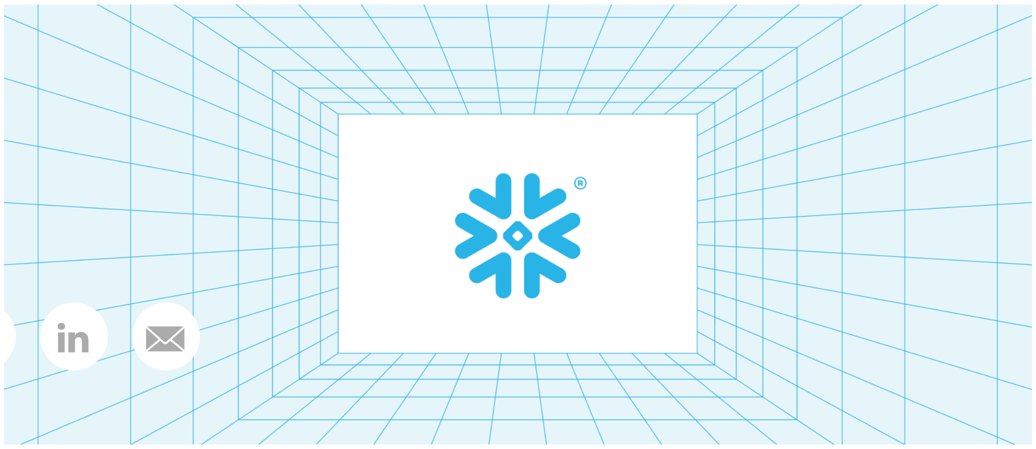
2024년 09월 06일

AUTHOR

Ryan Shelly

Automatic Clustering at Snowflake

Core Platform



SHARE

Snowflake's Data Cloud is built on top of major cloud computing vendors (AWS, Azure, GCP) to provide a cloud-agnostic platform for its users to store, query and share data. At its core, it is powered by an elastic and performant query execution engine to provide users with reliable, low-cost storage as well as easy, fast, and intelligent infrastructure to power data workloads. As a result, it becomes very easy and common for customers to create extremely large tables. This can lead to a number of very interesting technical challenges ensuring our query engine is highly performant when processing such large tables.

One way Snowflake improves query efficiency on such large tables is by utilizing automatic clustering, which maintains a pre-defined approximate ordering of the tables' underlying data. Maintaining this ordering on large tables in the face of customer DML (Data Manipulation Language) statements without negatively impacting customer workloads presents its own challenges. Snowflake's approach to maintaining clustering automatically as a service drastically reduces management overhead for end-users and is non-blocking and resource-efficient.

Snowflake Table Structure

Unlike traditional data warehousing offerings, which rely on static partitioning schemes for handling large tables, Snowflake relies on automatic micro-partitioning to physically store data in tables to provide a highly scalable and performant database solution. Micro-partitions do not need to be explicitly defined or maintained by users and are written automatically by customer DMLs. Micro-partitions are columnarized sets of rows and are inherently small in size and structure, allowing for extremely granular pruning on large tables, which could have hundreds of millions of micro-partitions. The columnar structure implies that columns are stored independently within the micro-partitions, allowing for efficient column scanning and filtering as well as automatic per-column compression. If none of the rows in a given micro-partition need to be modified by an incoming DML, then that micro-partition will not be touched at all by that DML. This enables extremely efficient DMLs as they limit the amount of data that needs to be written during a given operation. Similarly, if we are able to filter out all rows in a micro-partition given their range of values, then we can skip scanning that micro-partition entirely.

AUTHOR

Ryan Shelly

SHARE

Scanning entire tables is often not an option for acceptable query performance, which makes pruning efficiency critical. Metadata about the rows contained by each micro-partition such as the range and number of distinct values for each column, as well as other properties allows for various query optimizations and highly efficient query processing. As micro-partitions are automatically created by customer DMLs, the underlying data in customer tables is stored in its natural ingestion order, but oftentimes this is not the optimal physical ordering of data for customer workload patterns.

Clustered Tables

A clustered table is one where the underlying data is physically stored in the order of a user-defined set of key(s), rather than by natural ingestion order (see Figure 1 as an example). Selecting proper clustering keys is critical and requires an in-depth understanding of the common workloads and access patterns against the table in question.

Figure 1: Natural Ordering vs Clustered Ordering

The main goal in clustering a table is to achieve better pruning efficiency for the queries in a user's relevant workloads. In combination with the metadata maintained by Snowflake for all of its tables, sorting the data in a table can allow the query optimizer to prune away large portions of data in clustered tables based on filters, predicates, and order-preserving operators in incoming queries. For example, looking at Figure 1, the query is:

```
select name, country from t1 where type = 2 and
date = '11/2';
```

AUTHOR

Ryan Shelly

This query must scan all 4 micro-partitions in the naturally ordered table, whereas on the clustered table it only needs to scan 1 micro-partition (micro-partition 5) because we are able to prune out micro-partitions 6–8 with the filters on “type” and “date”.

Snowflake’s Approach to Clustering Tables

SHARE

Large tables in the system are rarely static, and both the volume of data as well as the numerous possible DML patterns present challenges for efficient clustering maintenance. Doing a full-table sort can be extremely expensive and may also block ongoing customer workloads while the sort is doing the work. To consistently perform full-table sorts, there would need to be periods of downtime in the customer workloads which in modern database systems is often impossible. Simply sorting new data as it is added does not solve the problem because it could not only slow down customer DML performance but also due to each DML’s data being locally sorted, the more DMLs that occur on the target table, the less effectively sorted it would become (as each distinct DML’s new data is not sorted relative to the data added by other DMLs). To deal with these challenges, Snowflake performs incremental clustering to keep up with DMLs on large tables with low-overhead. A key observation is that a table does not need to be fully sorted for query performance to benefit from clustering, so Snowflake strives to find the right balance between clustering well enough to achieve high pruning efficiency while maintaining low overhead cost.

The idea of bringing the table to a good enough state begs the question of how to determine how well-clustered a table actually is. In essence, Snowflake determines how well-clustered a table is by looking at how many partitions in the table overlap with each other. Two of the main metrics are:

width — the number of partitions that overlap with a given partition

depth — the number of partitions that overlap at the same point

AUTHOR

Ryan Shelly

Figure 2: Measuring the State of Clustering

Figure 2 shows four examples of sets of 5 micro-partitions for a given table and how they overlap with each other. In this scenario, the four tables get progressively more well-clustered as we go down the figure. So the first example with 5 overlapping micro-partitions is the least well-clustered as all of its micro-partitions overlap with each other – this table is totally unclustered. The fourth (bottom) example is perfectly clustered as none of the partitions overlap with each other.

Automatic clustering is broken down into two types of jobs:

SHARE

1. partition-selection jobs

2. clustering-execution jobs.

Clustered tables' micro-partitions are split into clustering levels that are modeled after an LSM-tree where each level is a proxy for the number of times the data in that micro-partition has been clustered. This representation is used to reduce write amplification, a metric for how much data we write performing clustering vs the amount of data written during customer DMLs, across clustered tables and to minimize the work required to bring a table to a well-clustered state. It also allows us to easily think about how much work we have already put into clustering each portion of the table and helps us know which portions of the table to prioritize for reclustering going forward. We maintain the aggregated depths and widths of partitions at each level as well as some other level-specific metrics which give us an idea of the current clustering state of that level. Partition-selection jobs select partitions from a particular level that will benefit the overall state of clustering as much as possible and the resulting sorted partitions are inserted in the level above. We use heuristics based on level ordering and the average depth of partitions in a level to choose the optimal level to work on for a given round of partition selection. By inherently preferring lower levels over the higher levels in the table, we focus on newer data which is often the most important data to be queryable for a customer and it also further helps reduce write amplification because we protect ourselves against selecting micro-partitions at higher levels until there is a sufficient amount of work to be done at that level.

Overlapping partitions are included within a batch, and the result of a partition-selection job is a sorted set of batches of fixed size (a batch-set), each of which are to be sorted individually. Partitions are grouped into batches such that there is a large amount of overlap among the partitions in a single batch.

AUTHOR

Ryan Shelly

Figure 3: Partition Selection Ideology

Figure 3 demonstrates how we think about constructing batches out of the widest overlapping micro-partitions with respect to the clustering key expression. After selecting the batches of overlapping micro-partitions (in this case we select two batches), clustering execution jobs are each assigned a set of batches from the batch-set, and each thread in the execution job will scan the partitions from a particular batch, sort them, and re-insert them into the table as new, sorted, and potentially consolidated partitions in the level above where they were selected. The idea is to achieve gradual clustering of the table efficiently and as we bring the micro-partitions to higher and higher levels, each micro-partition should overlap with fewer and fewer other micro-partitions in its level as demonstrated by Figure 4. As more and more partitions are moved to higher levels, the global clustering state of the table gets better and better and once a partition reaches the maximum level in the table, we will not select that partition for reclustering until the table grows large enough to increase its maximum level. This is done to keep write amplification and therefore cost low.

SHARE

Figure 4: Clustering Levels

Decoupling clustering into two types of jobs as well as using fixed-size batches allows us to control the amount of work each task is responsible for as partition selection jobs are usually computationally intensive jobs, regardless of the amount of resulting work for its successor execution jobs. Execution jobs each have a fixed amount of work assigned to them so will take a fixed amount of time and this allows us to fine-tune utilization of each execution node and to achieve linear scalability.

Optimistic Locking

AUTHOR

Ryan Shelly

To avoid disruption of customer queries, execution jobs utilize optimistic locking so that no queries are blocked on these background maintenance operations, which alleviates a huge pain especially with how compute-intensive these operations usually are. Optimistic locking implies that clustering jobs do not preemptively grab any locks for their target tables, but are able to proceed *optimistically* assuming that none of the micro-partitions that are to be sorted and rewritten by this particular recluster are modified by any concurrent jobs, then after all of the major work has been accomplished, immediately prior to the final commit, the job gets the necessary locks and ensures that none of the micro-partitions that this job has rewritten were modified by customer DMLs. If it turns out ha y e micro-partitions that were rewritten by this job were in fact modified by some customer query, then this clustering execution job must roll back to avoid any inconsistencies. We are working on further optimizations to make clustering jobs smarter about avoiding these data conflict errors as much as they can while continuing to make progress on the table as a whole. This non-blocking attribute means there is no effect on customer queries and allows Snowflake to maintain high availability for all tables.

SHARE

Automatic Clustering as a Service

Performing automatic clustering incrementally in the background in a non-blocking fashion provides our customers with a nearly effortless way of maintaining clustering properties across their tables without hindering any of their foreground workloads. Ease of use is a core product principle at Snowflake and automatic clustering is inline with this principle. While it would have been easier to just ask the customer to cluster their tables manually when they notice performance isn't meeting expectations, we strive to always give our customers the best possible experience when using our product — which includes hiding the complexity of a feature whenever possible. As such, customers do not need to concern themselves with the details of problems such as when they should recluster their table and how they find a maintenance window that won't impact their other DML workloads, how they determine how well-clustered the table is, or what size warehouse to use to efficiently recluster their table. Choosing the wrong size warehouse could lead to wasted resources or slow clustering operations and knowing when to recluster a table based on the current state of the data can be tricky.

Automatic clustering makes it as simple as selecting your clustering keys, and letting the service do the rest to ensure that you are able to achieve expected performance across large tables in customers' workloads.

Snowflake's automatic clustering service is one of many services built on top of an internal compute service framework which manages scheduling and execution of compute-intensive tasks and handles asynchronous execution of internally-generated queries. Units of work are represented as tasks (each with a corresponding SQL statement) and these tasks are then executed on internally-managed compute resources. This gives us fine-grained control over the workload to generate and scheduling requirements to allow us to achieve higher resource utilization. The compute service also takes advantage of the elasticity of the cloud and removes any resource contention with foreground customer queries.

AUTHOR

Ryan Shelly

SHARE

Figure 5: Clustering Workflow

Figure 5 depicts the standard workflow on a clustered table. The rate at which automatic clustering tasks are enqueued and dequeued to the compute service framework depends both on the size of the table in question as well as the rate of DMLs against that table. Once DMLs trigger clustering partition-selection tasks, these tasks decide whether or not there is work to be done based on the state of the table and if there is work, will kick off one or more successor execution tasks, each given a batch-set from their parent partition selection task. If there is no work to be done, the partition selection job will not produce any batch-sets or kick off any successors and no more tasks will be scheduled until new DMLs come in, potentially creating new work for the service to handle.

Conclusion

In this blog post, we introduced some of the benefits and challenges of maintaining clustered tables automatically and provided a high-level overview of Snowflake's approach to clustering tables. Automatic clustering is one of the most widely used features at Snowflake and there is an exciting roadmap ahead to continue improving and evolving Snowflake's clustering service. If you found this post interesting, please contact us and share your thoughts! If we piqued your interest with the problems we're working on and you want to work on these problems, check out our open job listings! We have engineering offices all around the world.

SHARE

AUTHOR

Snowflake Inc.

Ryan Shelly

SHARE