2024년 09월 06일

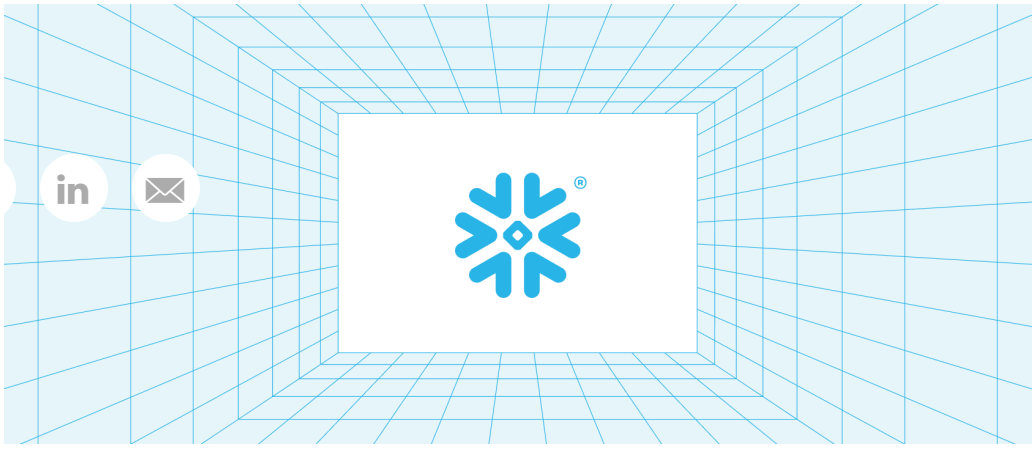
AUTHOR

Bowei Chen

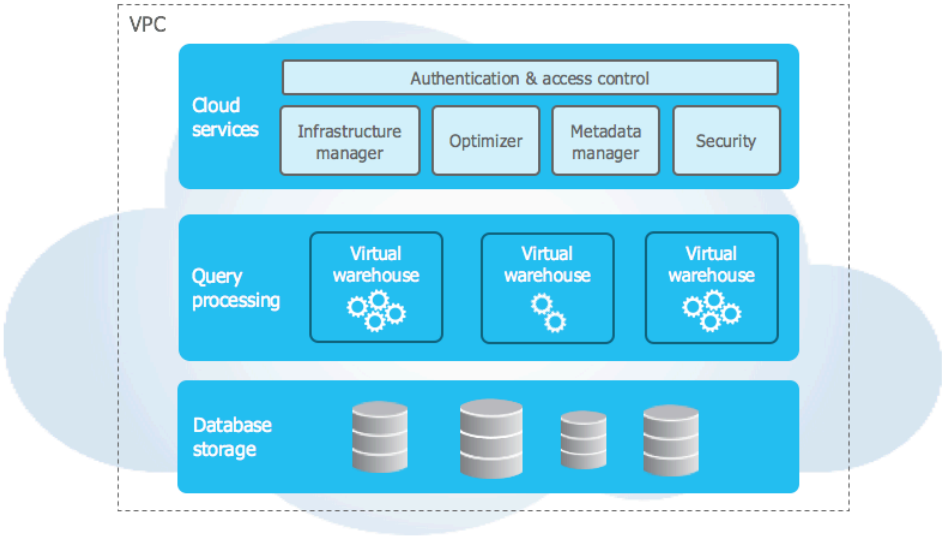
Aggregation Placement — An Adaptive Query Optimization for Snowflake

Core Platform

SHARE



Snowflake's Data Cloud is backed by a data platform designed from the ground up to leverage cloud computing technology. The platform is delivered as a fully managed service, providing a user-friendly experience to run complex analytical workloads easily and efficiently without the burden of managing on-premise infrastructure. Snowflake's architecture separates the compute layer from the storage layer. Compute workloads on the same dataset can scale independently and run in isolation without interfering with each other, and compute resources could be allocated and scaled on demand within seconds. The cloud-native architecture makes Snowflake a powerful platform for data warehousing, data engineering, data science, and many other types of applications. More about Snowflake architecture can be found in Key Concepts & Architecture documentation and the Snowflake Elastic Data Warehouse research paper.



AUTHOR

Bowei Chen

Figure 1. Snowflake Technical Architecture

SHARE

Snowflake's core query processing ability is backed by a query engine that serves billions of queries daily on average. The query engine consists of a query compiler and a query execution engine. The query compiler transforms SQL queries into executable query plans and applies various query optimization techniques to the plan. The optimized query plan is sent to be executed on the highly parallel, distributed execution engine to generate the query results. The ability to efficiently run online analytical processing (OLAP) queries is essential for Snowflake. Our team keeps working on developing novel query optimization and query execution techniques to deliver industry-leading performance.

In this series of blog posts, we focus on the query processing layer. We introduce aggregation placement, a query optimization technique where aggregations are placed at different positions in a join tree in addition to places where they are specified in the query text. This query optimization technique has been implemented in many other database systems, however, they have some limitations. With decades of experience in query optimization and query execution, aggregation placement is designed and implemented in a unique way in Snowflake to overcome the shortcomings of the existing implementations.

In traditional approaches to aggregation placement, the query optimizer considers alternative query plans during cost-based optimization and picks the plan with the lowest cost calculated by the optimizer's cost model. The problem with this approach is that the optimizer's cost model is based on compiler statistics. Compiler statistics could be missing, stale, too coarse, expensive to maintain, and often deviate from the actual statistics in non-trivial cases. Another limitation of the existing implementations is that they are not generic enough.

A core philosophy behind Snowflake's query engine is to make query execution runtime-adaptive whenever possible and hands-free for the users. This is a key differentiator between Snowflake and other database systems. To align with the design philosophy of Snowflake's query engine, aggregation placement is implemented differently in Snowflake with the following goals:

AUTHOR

Bowei Chen

Adaptiveness: Query execution should be able to take advantage of aggregation placement even when the optimizer has imperfect knowledge.

No regression: To avoid performance degradation, the additional aggregations need to be disabled during execution if they are not beneficial enough.

Generic: The optimization should be generally applicable to all aggregation functions and all join types.

In this blog post, we introduce aggregation placement briefly and show the performance improvements it brings to real-world workloads. In the next blog post of this series, we will deep dive into how the optimization is implemented in Snowflake from query processing's perspective and demonstrate how a complex feature like this is rolled out to customers.

Aggregation Placement Overview

When the query engine receives a SQL query, the compiler parses the query and transforms it into a query plan which represents how data is retrieved and processed. The query optimizer applies various transformations to the original query plan aiming to produce the most efficient way to execute the query. As a result of query optimization, the query plan that is deemed optimal is generated and sent to be executed by the query execution platform.

Each step of retrieving and manipulating data in the query plan is represented as an operator. For example, the scan operator is for retrieving data from a data source and the sort operator is for ordering the input data. Joins and aggregations are two common operators in analytical workloads. Joins are used for combining multiple inputs based on common fields, a.k.a join keys. Aggregations are used for grouping together multiple input rows into a single value based on common group-by keys. The following diagram shows a query plan containing a join and an aggregation that is generated for a SQL query.

AUTHOR

Bowei Chen

SHARE

In the example, a query to get the number of students enrolled in every class is transformed into a query plan. First, we scan the two tables that store the student and enrollment information. Then we join them using the **student_id** field. Finally, we perform an aggregation to compute the number of students associated with each **class_id**.

A common access pattern of analytical queries is that multiple data sources are first combined through joins, forming a join tree, and the results are grouped together through an aggregation node. Aggregation placement is a query optimization where aggregations are placed at different positions in the join tree as opposed to their initial position as specified in the SQL query. If a join is exploding, producing way more results than its input, and/or the aggregation has a low group count, then it might be more efficient to first evaluate the aggregation on the join input. In some cases, the optimization can drastically improve query performance.

The following diagram illustrates how aggregation placement improves query performance. We apply the transformation rule known as eager transformation in [1] and [2] by adding an aggregation to the right input of the exploding join. In the original query where aggregation placement is not applied, the join consumes 10k rows from its right input and produces 100k to the downstream aggregation. The additional aggregation added through aggregation placement reduces the join's right input and output row count to 1k and 10k rows respectively. Let's use a simplified model to compute the cost of join and aggregations in the query plan fragment where the cost of operators is linear to the input row count. The cost of the original and the transformed query plan fragment are 110k and 21k respectively. Query performance can be improved by 5x in this case.

Improvements for Real-world Workloads

Aggregation placement is now enabled by default in all of Snowflake's production deployments. We built some dashboards to continuously monitor its impact.

The following chart shows the total number of queries where aggregation placement kicks in to transform the query plan. The query optimization is applied to millions of queries on all production deployments daily. These queries are from more than 8,000 customers (as of April 30, 2023). Aggregation placement is an optimization that benefits a large number of queries for workloads different Snowflake customers run on a daily basis.

AUTHOR

Bowei Chen

We prepared some queries before and after aggregation placement is rolled out to study the impact of the optimization, which demonstrates that the optimization is very beneficial. Let's look at one production query in detail. The following diagram generated from the Query Profile page that is shown in the Snowflake UI is the query plan fragment for a query without applying aggregation placement. The table names and expressions in the query plan are blurred out to anonymize the query. The operators and their execution time in percentage are shown in rectangles. The numbers on the arrows represent the number of rows passed from one operator to another. Processing the plan fragment takes up more than 50% of the total query processing time. The main reason is that Join#25 is exploding. It has 4.735 and 17.34 million input rows from its left and right children each and explodes into 8.588 billion output rows. This is making Join#25 and its downstream operator Aggregate#11 expensive.

The same query plan fragment with aggregation placement being applied is shown in the diagram below. The main difference to the original fragment is that **Aggregate#24** and **Aggregate#22** are placed below **Join#25** and **Join#23**. In the new query plan, **Aggregate#22** improves query performance drastically by reducing

the data that are eventually processed by **Join#25**. This results in less processing time in **Join#25** and **Aggregate#26**. The overall query performance is improved by more than 2x.

AUTHOR

Bowei Chen

Another interesting aspect is that we found aggregations being pushed below multiple joins can be very effective. With compiler statistics, it can be difficult to make an accurate estimation of whether an aggregation node is beneficial at each level. And as a result, we might miss the opportunity to optimize at some places or apply the optimization where it is not needed. Our runtime adaptive approach, which we will discuss more in the next post, naturally makes sure that we can apply the optimization wherever it is beneficial and will not introduce overhead at other places.

Conclusion

In this blog post, we introduced aggregation placement, a query optimization technique to accelerate query execution for complex analytical workloads. A unique aspect of the implementation of aggregation placement in Snowflake is the optimization can adapt to different characteristics in customer workloads. We showed how aggregation placement broadly improves query performance for real-world customer queries. In the following blog post, we are going to dive into some technical details of aggregation placement and demonstrate the effort we put into finally enabling the optimization for all production customers. At Snowflake, we continue to deliver new features in query optimization and query execution to bring industry-leading query performance to our customers.

References

1. Including Group-By in Query Optimization

2. Eager Aggregation and Lazy Aggregation

Snowflake Inc.

AUTHOR

Bowei Chen