Cost-Aware SQL Query Planning on Serverless Functions

Research Project Proposal | CS848 - Modern Database Systems (Fall 2026)

Chanaka Lakmal
chanaka.lakmal@uwaterloo.ca

1 Problem and Motivation

Serverless computing is a emerging paradigm for analytical SQL queries: it scales to thousands of parallel functions within seconds and charges only for the time they run. However, whether a query actually benefits depends heavily on its physical plan: how tables join (shuffled, broadcast, or already co-partitioned), how many function instances each stage gets, which operators share a function, and how much memory each function has.

Existing serverless query systems either reuse plans from a data-warehouse optimizer [8] or choose them with fixed rules [5]. Neither approach reflects the cost structure of serverless execution. This project builds a *query planner* whose cost model reflects serverless execution, and tests whether its plans achieve lower latency and lower cost than state-of-the-art serverless query planning.

2 Related Work

Skyrise [5] is a serverless SQL query processor. It compiles SQL into pipelines of physical operators, cuts pipelines at exchanges, and runs each pipeline fragment as a function, with all intermediate data exchanged through object storage. Its optimizer is rule-based: plans are shaped by fixed rewrite rules and heuristics rather than by comparing the estimated cost of alternative plans, and the number of workers per pipeline is set from the input size.

Starling [8] runs analytical queries on cloud functions, with base tables and intermediate data in cloud object storage, and introduces techniques to handle stragglers and shuffle data through storage. It executes plans obtained from Amazon Redshift [2].

3 Approach

We build our planner into a query layer on top of an existing serverless workflow system, which deploys functions, connects them into a directed acyclic graph (DAG), and invokes them. Given an SQL query, the layer obtains a physical plan from a planner, turns it into a DAG of functions, and runs it on the workflow system. Users can plug in their own planner or configure a physical plan directly, so plans from different sources run on the same platform.

Pluggable Planners: We use this feature to compare our planner with Skyrise's planning rules [5], which we re-implement because its released source code [4] cannot compile SQL and provides plans for only three TPC-H queries. Both planners start from the same logical plan, produced by Apache Calcite [6], so the comparison isolates their physical decisions. Supporting additional planners, such as plans from Amazon Redshift [2] (Starling's approach), is left for future work. Skyrise's plans do not specify function memory, so we run them at every memory size our planner may choose and report the best result. Additionally, each plan uses the same executor and data-passing mechanism. Therefore, the comparison evaluates planning quality rather than engine performance.

Cost Model: Before query execution, the system collects data statistics once from the stored tables and measures a platform profile once per machine using micro-benchmarks. These benchmarks assess operator processing speed across different memory sizes, function-invocation overhead, and data transfer rates. From these, we estimate two quantities for each candidate plan:

(1) *latency*, as the critical path through the stages (computation + data transfer)
(2) *dollar cost*, in the style of AWS Lambda [3], that is, active execution time × configured memory (GB-seconds), where, as in AWS Lambda, a function's CPU share grows with its memory

Plan Selection: For each query, our planner considers four kinds of physical decisions together: (1) how each join is executed (shuffling both inputs, broadcasting the smaller input to every instance, or joining data that is already co-partitioned); (2) how many function instances each stage uses; (3) which consecutive operators are combined into one function; and (4) how much memory each function receives. These decisions are made jointly because they interact. Since faster plans are typically more expensive, the planner returns a set of plans that are not outperformed by any other plan in both latency and cost, representing the latency–cost trade-off. When a single plan is required, the planner selects the least expensive plan from this set that satisfies a specified latency constraint.

4 Evaluation

Workload: We use a set of TPC-H queries, selected to cover different join and aggregation patterns, at a single scale factor; extending the evaluation to all 22 queries and to multiple scale factors is left for future work. Base tables are stored as Parquet files [7] in AWS S3-compatible [1] object storage.

Setup: We compare our planner with Skyrise's rules on a single cloud instance by query latency and cost. We plan to extend the experiments to multiple machines in future work.

References

[1] Amazon. 2026. Amazon S3 - Cloud Object Storage. https://aws.amazon.com/s3/
[2] Amazon. 2026. Cloud Data Warehouse – Amazon Redshift – AWS. https://aws.amazon.com/redshift/
[3] Amazon. 2026. Serverless Function, FaaS Serverless - AWS Lambda. https://aws.amazon.com/lambda/
[4] Thomas Bodner. 2026. Skyrise - HPI Data Engineering Systems. https://github.com/hpides/skyrise/tree/dasp2025
[5] Thomas Bodner, Daniel Ritter, Martin Boissier, and Tilmann Rabl. 2025. Skyrise: Exploiting Serverless Cloud Infrastructure for Elastic Data Processing. *Datenbank-Spektrum* 25, 1 (01 Mar 2025), 29–38. doi:10.1007/s13222-025-00496-7
[6] Apache Software Foundation. 2026. Apache Calcite. https://calcite.apache.org/
[7] Apache Software Foundation. 2026. Apache Parquet. https://parquet.apache.org/
[8] Matthew Perron, Raul Castro Fernandez, David DeWitt, and Samuel Madden. 2020. Starling: A Scalable Query Engine on Cloud Functions. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (Portland, OR, USA) *(SIGMOD '20)*. Association for Computing Machinery, New York, NY, USA, 131–141. doi:10.1145/3318464.3380609