Serverless II a

A Study of Database Techniques over Serverless Cloud Platforms

Milestone 2

Nikhil Gupta, Sahithi Kodali, Sachit Kothari

PURDUE UNIVERSITY | Department of Computer Science

Roadmap

- Achieved in milestone 1:
  - Introduction and overview of the three papers - Serverless in the wild, Lambada and Starling.
  - Discussed objectives and plan of the project.

- For Milestone 2:
  - Discussed the details of paper 1 - Serverless in the wild - completely.
  - Discussed the architecture details of paper 2 - Lambada - particularly the three strategies/operators. More details to be discussed in the next milestone.
  - Discussed the architecture details of paper 3 - Starling - including the details of internal working fo the query engine, technical approaches of query execution and data management. Aspects of paper that consists of more execution details will be covered in the next milestone.

Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud Provider

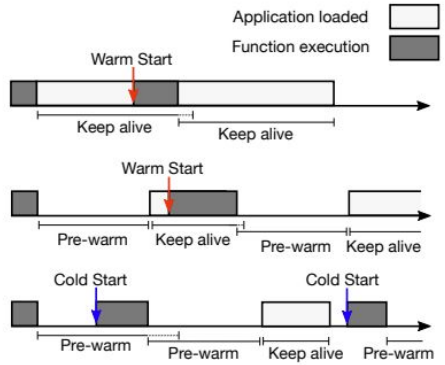
Introduction: Why?

- Cold starts in FaaS: Minimize by prewarming.
- Can’t always keep in memory, so need to predict.
- Currently keeps alive for fixed time for every function after trigger.
- So how can the keep alive time for each function and pre warm be predicted by looking at data?

Reference: Mohammad Shahrad, Rodrigo Fonseca, Íñigo Goiri, Gohar Chaudhry, Paul Batum, Jason Cooke, Eduardo Laureano, Colby Tresness, Mark Russinovich, and Ricardo Bianchini. Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud Provider. USENIX Annual Technical Conference, 2020.

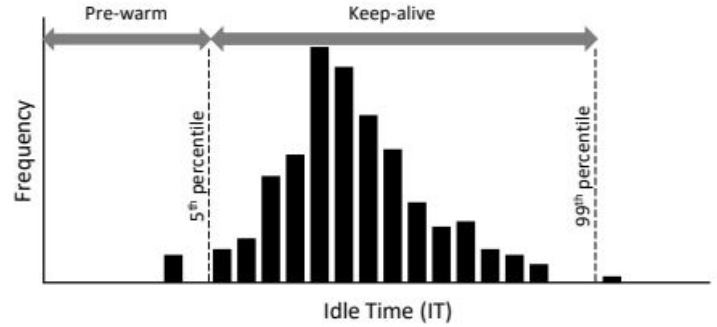
Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud Provider

How does it work?



- Hybrid histogram policy: adjusts for each app
- Pre warming window starts after function executions, keep alive starts after each pre warming. If pre warming window is 0, keep alive starts at execution.
- Policy has 3 parts: Histogram capturing each applications 'idle' times. Standard keep-alive approach when histogram is not representative. Time series forecast component for when histogram does not capture most idle times.
- Range limited histogram for IT's (1m bins)
- If not representative, pre warm = 0, keep alive = range of histogram (max 4 hours)

Coefficient of variation of its bin counts determines if histogram is representative.



Reference: Mohammad Shahrad, Rodrigo Fonseca, Íñigo Goiri, Gohar Chaudhry, Paul Batum, Jason Cooke, Eduardo Laureano, Colby Tresness, Mark Russinovich, and Ricardo Bianchini. Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud Provider. USENIX Annual Technical Conference, 2020.

Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud Provider

How it works continued

- If histogram has too many infrequent invocations there's a lot of entries outside its range. Then time-series analysis it used. ARIMA modelling is used to predict needed pre warming and keep alive time, and 15% margin is used to give prediction room for error.
- Implemented policy in Apache OpenWhisk, written in Scala.
- Evaluated using Simulator and real experiments. The result is overall a big improvement in resource efficiency as well as reducing the number of cold starts. Also explains how each part of the policy affects results.
- Policy implemented in Azure functions for HTTP triggered applications (when this paper was published). Likely has rolled out further since.

Reference: Mohammad Shahrad, Rodrigo Fonseca, Íñigo Goiri, Gohar Chaudhry, Paul Batum, Jason Cooke, Eduardo Laureano, Colby Tresness, Mark Russinovich, and Ricardo Bianchini. Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud Provider. USENIX Annual Technical Conference, 2020.

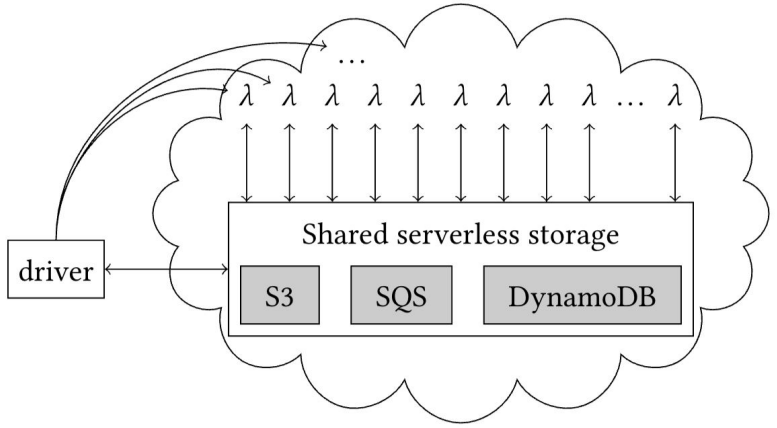
Lambada: Interactive Data Analytics on Cold Data using Serverless Cloud Infrastructure

- Lambada is a fully-serverless distributed data processing framework designed for data analytics.
- Serverless computing is most beneficial for interactive workloads on cold data, particularly in the initial exploratory phase of data analytics. Lambada offers a cost and performance advantage over existing solutions in scenarios where serverless computing is suitable.
- The authors address technical challenges in implementing data analytics on serverless computing and propose solutions for efficient batch-start of serverless workers, cloud-native scan operators, and inter-function communication.

Reference: Ingo Müller, Renato Marroquín, and Gustavo Alonso. 2020. Lambada: Interactive Data Analytics on Cold Data Using Serverless Cloud Infrastructure. *ACM SIGMOD International Conference on Management of Data (SIGMOD '20).*

Lambada Architecture Overview

- Design Goal: Utilize existing serverless components
- High-Level Components: Local driver, serverless workers, shared storage
- Serverless Workers (λ): Execute queries in a distributed data-parallel fashion
- Shared Storage: Amazon S3 and DynamoDB for data, SQS for messages
- Input/Output: Read and write data from/to shared storage
- Architectural Model: Resembles a shared storage database
- Communication Model: Workers communicate exclusively through shared storage
- No Direct Communication: No direct interaction between workers or with the driver



Reference: Ingo Müller, Renato Marroquín, and Gustavo Alonso. 2020. Lambada: Interactive Data Analytics on Cold Data Using Serverless Cloud Infrastructure. *ACM SIGMOD International Conference on Management of Data (SIGMOD '20).*

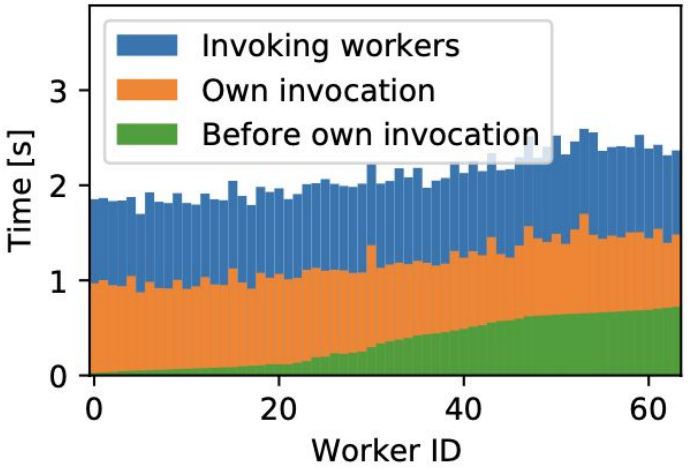
Key Components in Lambada System

- Data-parallel Query Plans:
  - Queries transformed into executable form.
  - Collection Virtual Machine (CVM) is the query compilation and execution framework. Most Lambada operators execute in serverless scopes. Driver-based scopes for pre/post-processing.
- Serverless Workers:
  - AWS Lambda functions with event handlers. Use dependency layer, metadata (memory, timeouts).
  - Extract worker ID, query plan, inputs. Results sent to driver via SQS queues.
- System Components for Serverless Analytics:
  - Trade-offs between SLAs, execution speed, and usage-based cost for each component.

Reference: Ingo Müller, Renato Marroquín, and Gustavo Alonso. 2020. Lambada: Interactive Data Analytics on Cold Data Using Serverless Cloud Infrastructure. *ACM SIGMOD International Conference on Management of Data (SIGMOD '20).*

Batch Invocation Strategy

- Limits of Sequential Invocation:
  - Challenging to invoke serverless workers quickly.
  - Invocation times vary based on location and data center.
  - Overlapping requests hides network round-trip latency.
- Lambada Two-level Invocation:
  - Parallelize the invocation process.
  - First-generation workers invoke second-generation workers.
  - Balance invocations between driver and first-level workers.
  - √P second-generation invocations each, where P is the total number of workers.
  - AWS Lambda invocation rate limit not a concern; focus on concurrent invocations (workers).



Example run of two-level invocation process.

Reference: Ingo Müller, Renato Marroquín, and Gustavo Alonso. 2020. Lambada: Interactive Data Analytics on Cold Data Using Serverless Cloud Infrastructure. *ACM SIGMOD International Conference on Management of Data (SIGMOD '20).*

Cloud Storage Scan Operator

- Network Characteristics:
  - Performance and cost analysis of accessing S3 from serverless workers.
  - Large files have a stable limit of around 90 MiB/s per worker.
  - Small files show variable network bandwidth, influenced by memory size.
  - Optimal performance achieved with multiple concurrent connections.
  - Size of individual requests directly impacts scan costs; smaller requests cost more.
- Lambada Cloud-native Scan:
  - Efficiently designed scan operator for Parquet files.
  - Supports selections and projections by working with row groups and column chunks.
  - Leverages concurrent connections at multiple levels to maximize bandwidth utilization.

Reference: Ingo Müller, Renato Marroquín, and Gustavo Alonso. 2020. Lambada: Interactive Data Analytics on Cold Data Using Serverless Cloud Infrastructure. *ACM SIGMOD International Conference on Management of Data (SIGMOD '20).*

Exchange Operator

- Exchange Operators for Serverless Workers:
  - Serverless workers in Lambada can't directly communicate with each other, necessitating the use of external storage for data exchange.
  - Workers partition data into *P* segments based on a partitioning routine, writing them to files named after their ID and the intended receiver.
  - Workers read files where they are listed as the receiver, potentially repeating the process until the file exists, as senders may be slower.
- Basic Exchange Algorithm:
  - We adopt the basic exchange algorithm, a method widely used by other researchers.
  - However, this algorithm exhibits a crucial issue: the number of files involved grows quadratically with the number of workers, leading to potential throttling due to cloud provider rate limits and increasing costs.

Reference: Ingo Müller, Renato Marroquín, and Gustavo Alonso. 2020. Lambada: Interactive Data Analytics on Cold Data Using Serverless Cloud Infrastructure. *ACM SIGMOD International Conference on Management of Data (SIGMOD '20).*

Exchange Operator

- Lambada Multi-Level Exchange:
  - To mitigate the issues with the basic algorithm, we introduce the concept of multi-level exchange.
  - This approach splits the exchange into multiple levels, each involving a subset of the workers.
  - The method uses projection functions for horizontal and vertical exchanges, dramatically reducing the number of requests and addressing both cost and rate limit concerns.
- Lambada Write Combining:
  - With write combining, data partitions generated by a single worker are consolidated into a single file, reducing the need to read multiple files per sender.
  - This technique efficiently minimizes the I/O operations required for data exchange.
- P√P complexity achieved

Reference: Ingo Müller, Renato Marroquín, and Gustavo Alonso. 2020. Lambada: Interactive Data Analytics on Cold Data Using Serverless Cloud Infrastructure. *ACM SIGMOD International Conference on Management of Data (SIGMOD '20).*

Starling: A Scalable Query Engine on Cloud Function Services

Introduction to Starling

- Starling is a query execution engine built to run on serverless platforms by leveraging the benefits of cloud function services (FaaS) like AWS Lambda and Azure.
- It can handle the challenges with FaaS cloud function services such as:
  - Managing hundreds of tiny stateless resource-constrained workers (computational units performing tasks with limited resources).
  - Handling stragglers i.e, workers that take longer to complete than other tasks within a parallel computation.
  - Data shuffling and aggregation (process of redistributing data across workers via stateful queries) through opaque cloud services.
- Provides interactive query latency at lower cost, lower query latency when reading from cloud stores and scales to large datasets.
- Supports low to moderate query volumes on data.

Reference: Perron, Matthew, et al. "Starling: A scalable query engine on cloud functions." Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. 2020.

Starling: A Scalable Query Engine on Cloud Function Services

Advancements of Starling

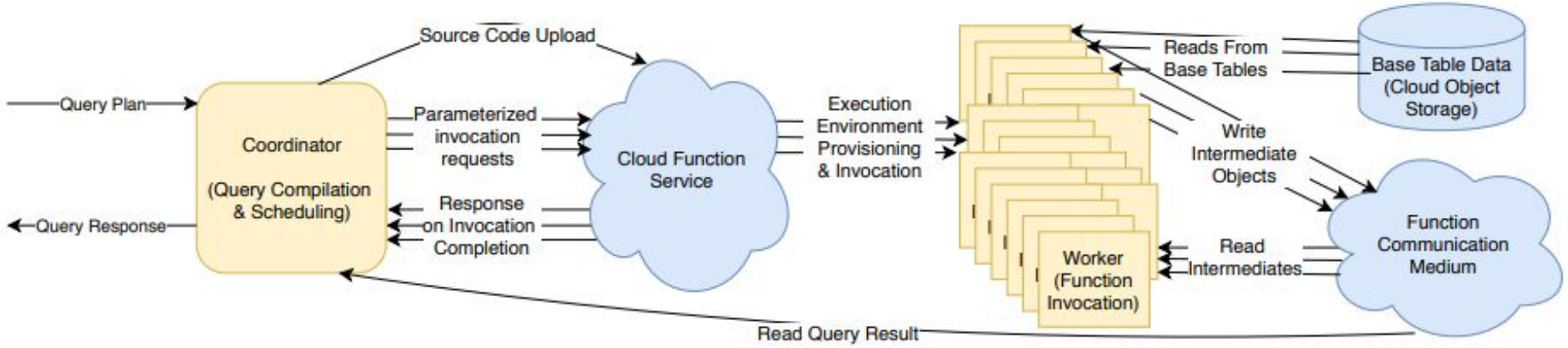
- High resource utilization and reduced cost
  - Maps tasks to function invocations to allow users pay-by-request.
  - Utilizes on-demand elasticity to shrink/grow the number of invocations as needed.
  - Materialize intermediate results into formats that optimize resource costs.
- Mitigation of stragglers and reduce query latency
  - Uses a tuned model to detect straggling requests and takes necessary steps to decrease query latency.
  - Data loading is not required.
  - Increase performance by tuning queries to desired performance/cost and adjust parallelism.

System	Does not require loading	Pay by query	Tunable performance
Amazon Athena	✓	✓	✗
Snowflake	✗	✓*	✓
Presto	✓	✗	✓
Amazon Redshift	✗	✗	✓
Redshift Spectrum	✓	✗	✓
Google BigQuery	✓	✓	✗
Azure SQL DW	✓	✗	✓
Starling	✓	✓	✓

Reference: Perron, Matthew, et al. "Starling: A scalable query engine on cloud functions." Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. 2020.

Starling: A Scalable Query Engine on Cloud Function Services

Architecture overview

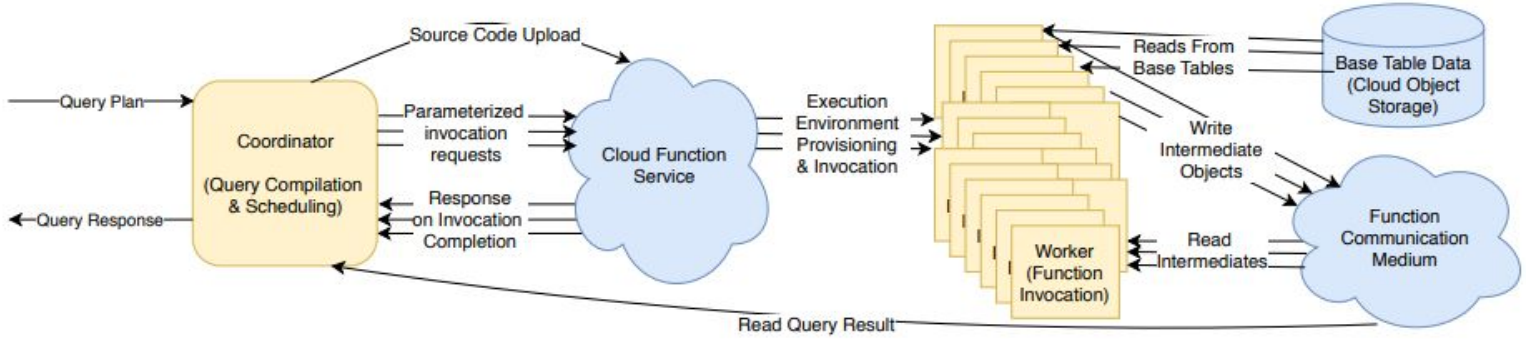


- The architecture aims at processing the query plan submitted by users and send back the response to the query. Includes three main parts: Coordinator, Cloud Function Service and the Worker.
- The coordinator compiles the query and uploads to cloud function service along with task schedule.
- The function service provides execution environments & invocation to the workers for their tasks.
- The workers execute query by accessing data and sends back the query response to the coordinator.

Reference: Perron, Matthew, et al. "Starling: A scalable query engine on cloud functions." Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. 2020.

Starling: A Scalable Query Engine on Cloud Function Services

Architecture details: Coordinator



- Compiles query and uploads executable (packages with necessary supporting files to execute query) to the cloud function services and manages query execution from end-to-end by task scheduling via scheduler.
- AWS Lambda is used as the cloud function service for its restriction free platform regarding language use and rate of function execution.
- Query execution starts by using the input from coordinator which is a single JSON file that refers to the physical query plan, stage dependencies and number of tasks in a stage.

Reference: Perron, Matthew, et al. "Starling: A scalable query engine on cloud functions." Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. 2020.

Starling: A Scalable Query Engine on Cloud Function Services

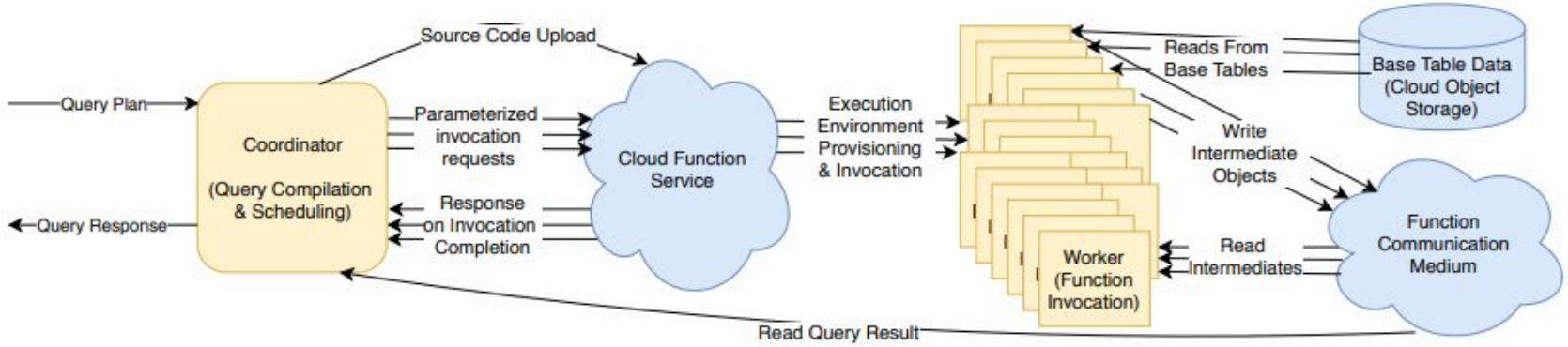
Query execution in Starling

- The query executable (C++ source code with dependencies) is uploaded to AWS Lambda. Each task input and output names are determined before worker execution.
- Relational operators
  - The base table data is scanned in parallel, reads necessary columns with projections, supports broadcast and partitioned hash joins, and performs aggregations in two steps.
  - Join tasks create hash tables and probe into partitions, while aggregation involves generating partial aggregates and reducing them into a final result.
- Handling data shuffles
  - Employs a partitioned intermediate format, allowing shuffling tasks to read only relevant partitions from input files, reducing data transfer overhead. For large-scale queries, Starling implements optimized shuffling strategies such as a multi-stage shuffle that combines tasks, reducing S3 reads significantly by reading contiguous subsets of partitions with negligible write costs.
- Task scheduling and pipelining
  - Finetunes the number of tasks per stage to find optimal balance between cost and query speed.
  - Pipelining is performed to allow consumers to start processing once significant portion of producer inputs are available. While pipelining accelerates query compilation, it can incur additional costs if producer experience delays. Can enable/disable pipeline based on the requirements.

Reference: Perron, Matthew, et al. "Starling: A scalable query engine on cloud functions." Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. 2020.

Starling: A Scalable Query Engine on Cloud Function Services

Architecture details: Worker



- Each worker runs single cloud function invocation that executes a part of query using the parameters sent by coordinator.
- Utilizes a communication medium (like a shared storage) to allow the stateless worker function to converse and exchange data as needed.
- The worker reads inputs from either the base table data or communication medium. It writes the output to the communication medium, which sends the result to the coordinator once all workers completes the task and exit.

Reference: Perron, Matthew, et al. "Starling: A scalable query engine on cloud functions." Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. 2020.

Starling: A Scalable Query Engine on Cloud Function Services

Data management in Starling

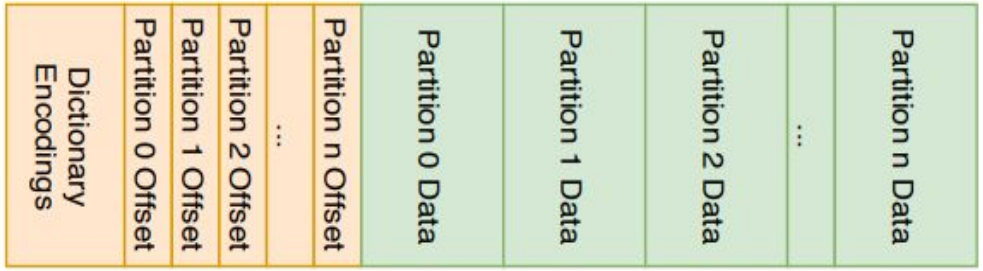
- AWS S3 is used for both base table data storage and the communication medium as it is the low-cost, high-throughput, low-latency, and scalable data exchange solution.
- Base table storage
  - Agnostic to base table formats but CSV, ORC and Parquet are the usual choices. Storing data as objects of a few hundred MBs enhance performance by allowing efficient parsing.
  - Open-source columnar formats like ORC enable selective column reading and skipping of unnecessary data using indexes improving query performance and saving time.

Reference: Perron, Matthew, et al. "Starling: A scalable query engine on cloud functions." Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. 2020.

Starling: A Scalable Query Engine on Cloud Function Services

Data management in Starling

- Managing intermediate stage
  - Allows passing of intermediate data between invocations. Producer workers write output as single S3 objects in the S3 buckets containing all partitions with predetermined keys. These partitioned files begin with metadata that indicates the end locations of partition in object.
  - Reader poll keys for new object and once found fetches the metadata and relevant partitions.
  - S3 is persistent i.e, workers can begin sending data before destination workers start execution.
  - Encodes low cardinality string columns using dictionary encoding. The metadata can be fetched with one read, followed by a read to fetch required partition. This supports multi-stage shuffles by fetching adjacent partition data with same number of GET requests as single partition data.



Reference: Perron, Matthew, et al. "Starling: A scalable query engine on cloud functions." Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. 2020.

Starling: A Scalable Query Engine on Cloud Function Services

Handling Stragglers

- Can occur while reading base table data or exchanging intermediate state, as stages must wait for their inputs to be available before query processing.
- Straggler optimization leverages the power of two choice technique, a well known theoretical framework using randomization and task duplication.
- Read Straggler Mitigation (RSM)
  - Uses a dynamic approach by detecting delays compared to expected completion times based on observed latency and throughput of S3 requests and AWS Lambda invocations. Workers open new connections and retry requests if responses exceed anticipated duration. Though the read expense is incurred, it significantly improves query latency.
- Write Straggler Mitigation (WSM)
  - Address delay in large S3 write requests by using a model similar to RSM to predict response times and initiate a second write request. Incurs additional write expenses but saves about 2100 seconds of compute time while incurring a cost of 314 seconds, thus giving low latency within optimized compute resources.

Reference: Perron, Matthew, et al. "Starling: A scalable query engine on cloud functions." Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. 2020.

Team members contribution

- All the team members contributed together to understand the goals and objectives for successful completion of this research
- Nikhil Gupta: Survey of Lambada and serverless architecture
- Sahithi Kodali: Survey of Starling paper and its methodologies
- Sachit Kothari: Survey of serverless optimization paper and its methodology

Next Steps

- Explain more on the technical details of the papers discussed.
- Find further related papers in the series of work to study and present where required.
- Understand how this research translates to real world application applications in cloud services. (e.g. AWS Athena)

THANK YOU

Q&A