Serverless II a

A Study of Database Techniques over Serverless Cloud Platforms

Milestone 3

Nikhil Gupta, Sahithi Kodali, Sachit Kothari

PURDUE UNIVERSITY | Department of Computer Science

Roadmap

- **Achieved in milestone 1 and 2:**
  - Discussed the paper Serverless in the wild in detail.
  - Discussed the architecture details, particularly the three strategies/operators in Lambda.
  - Discussed the architecture, query execution approaches, data management and handling of stragglers in Starling.

- **For Milestone 3:**
  - Discussed the details of a new paper - Agile cold starts for scalable services.
  - Discussed the data exchange operators, algorithms used and optimization approaches used in Lambda.
  - Discussed data management approaches in base table and intermediate table in detail, explained how relational operators, data shuffles, task scheduling and pipelining are performed in query execution, with more straggler management approaches extended to final presentation.

Agile cold starts for scalable services

Quick recap and breakdown of cold starts

- Load everything from memory in FaaS.
- Cant keep everything in memory because of resource efficiency.
- The graph shows breakdown of function invocation for concurrent functions.
- Large part of setup is startup time.
- The startup time is further broken down into Container Create, Network Create, and Network Connect times.
- Network Create and Network Connect steps account for 90% of the startup time.

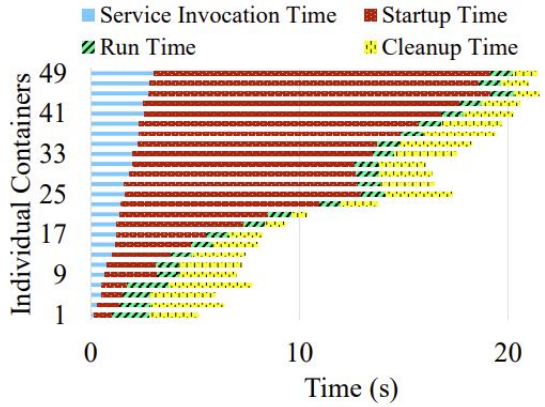


Figure 3: Timeline of 50 concurrent functions with alternate containers. Startup time accounts for 90% of total time.

Mohan, A., Sane, H., Doshi, K., Edupuganti, S., Nayak, N., & Sukhomlinov, V. (2019). Agile cold starts for scalable serverless. In 11th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 19).

Agile cold starts for scalable services

Methodology to reduce network setup time

- Pause Container Pool Management (PCPM)

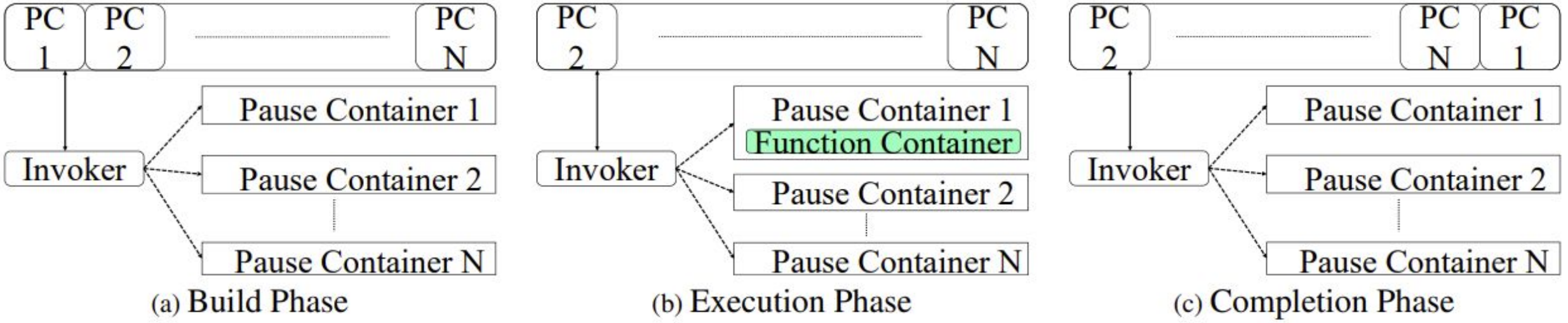


Figure 5: Different phases of Pause Container Pool Manager (PCPM)

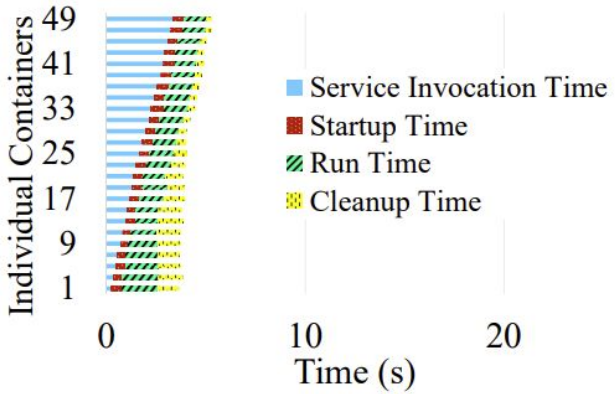
Mohan, A., Sane, H., Doshi, K., Edupuganti, S., Nayak, N., & Sukhomlinov, V. (2019). Agile cold starts for scalable serverless. In 11th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 19).

Agile cold starts for scalable services

Why it works and results

- PC's have no data retention and take negligible memory. Keep a large number ready.
- Evaluated with unmodified Open Whisk. 80% improvement in execution time, several order in magnitude of memory saved compared to pre warming containers.

Container Type	Environment Dependency	Function Dependency	Mem Usage (MB) (50 Containers)	Total Time (s) (50 Functions)
Cold	NO	NO	0	20.01
Warm	YES	YES	1600	0.78
Pre-Warm	YES	NO	1500	1.05
PCPM	**NO**	**NO**	**2**	**4.84**



Mohan, A., Sane, H., Doshi, K., Edupuganti, S., Nayak, N., & Sukhomlinov, V. (2019). Agile cold starts for scalable serverless. In 11th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 19).

Lambada: Exchange Operators for Serverless Data Processing

- Purpose: Facilitate data exchange for serverless workers without direct connections, using external storage (e.g., S3).
- Use Cases: Critical for operations like joins, sorting, and grouping in data processing.
- Functionality: Transfers data among workers based on partitioning criteria.
- Parallel Execution:
  - Enables parallel operations by distributing tasks across workers.
  - For instance, a parallel equi-join uses exchange operators to route join partners to the same worker for a local join operation.
- Significance: Provides a necessary and efficient method for data-parallel processing in serverless architectures.

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: Basic Exchange Algorithm

- Algorithm Overview:
  - Workers partition data into P^2 files for exchange.
  - High file operations due to quadratic scaling with worker count (P).
- Key Challenges:
  - Rate Limits: Potential throttling by cloud providers.
  - High Costs: Quadratic cost increase with more workers.
- Cost Example:
  - 256 workers: S3 costs > worker run costs.
  - 4k workers: S3 costs ~$100 vs. worker costs ~$3.3 for 4 TiB data.
- Solution Direction:
  - Introduce optimizations to reduce request count and costs.

Algorithm 1 Basic S3-based exchange operator.

```
1: func BasicExchange(p: Int, 𝒫: Int[1..P], R: Record[1..N],
                      FormatFileName: Int × Int → String)
2:     partitions ← DramPartitioning(R, 𝒫)
3:     for ⟨receiver, data⟩ in partitions do
4:         WriteFile(FormatFileName(receiver, p), data)
5:     for source in 𝒫 do
6:         data ← data ∪ ReadFile(FormatFileName(p, source))
7:     return data
```

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: Optimizing Data Exchange with Lambda Multi-Level Approach

- Concept:
  - Reduce operations by exchanging data in multiple levels.
  - Each level involves a smaller subset of workers.
- Technique:
  - Two-level exchange: Horizontal followed by vertical.
  - Utilizes a projection function to map worker IDs onto a grid.
- Implementation:
  - BasicGroupExchange with a projection function for partitioning.
- Efficiency:
  - Significantly lowers the total number of requests.
  - Optimal when the group size is the square root of the total number of workers.

Algorithm 2 Two-level S3-based exchange operator.

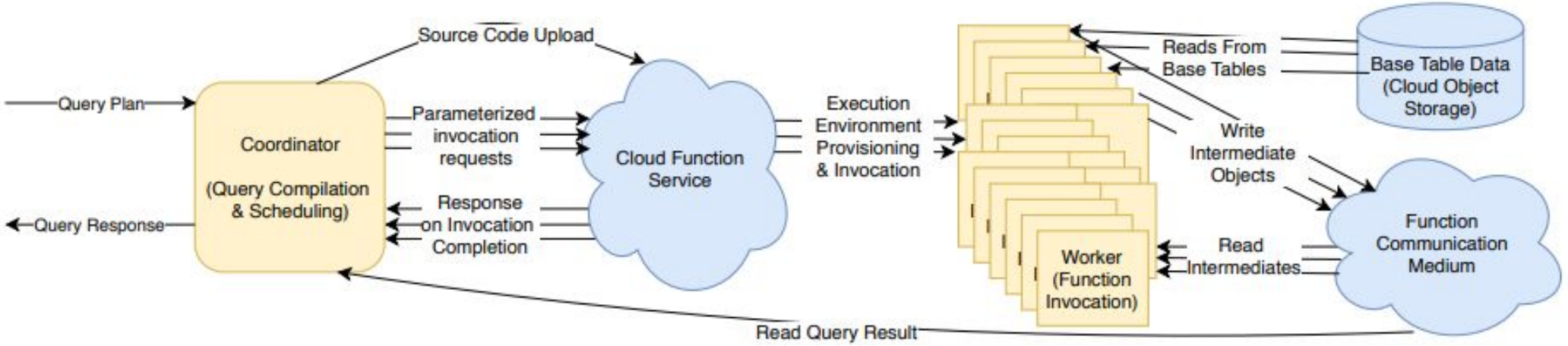
1: **func** TwoLevelExchange(p: **int**, P: **int**, R: **Record** $[1..N]$)
2: $\langle p_1, p_2 \rangle \leftarrow H_s(p)$
3: $\mathcal{P}_i \leftarrow \{q | q \in \{1..P\} : q_i = p_i\}$ **for** $i = 1, 2$
4: $f_i \leftarrow \langle s, t \rangle \mapsto$ "s3://b{i}/snd{s}/rcv{r}" **for** $i = 1, 2$
5: tmp $\leftarrow$ BasicGroupExchange($p, \mathcal{P}_1, f_1, R, H_s^2$)
6: **return** BasicGroupExchange($p, \mathcal{P}_2, f_2$, tmp, H_s^1)

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

Previously Discussed

- Introduction to Starling
- Architecture overview and details
- Overview of query execution, data management including handling of stragglers in Starling

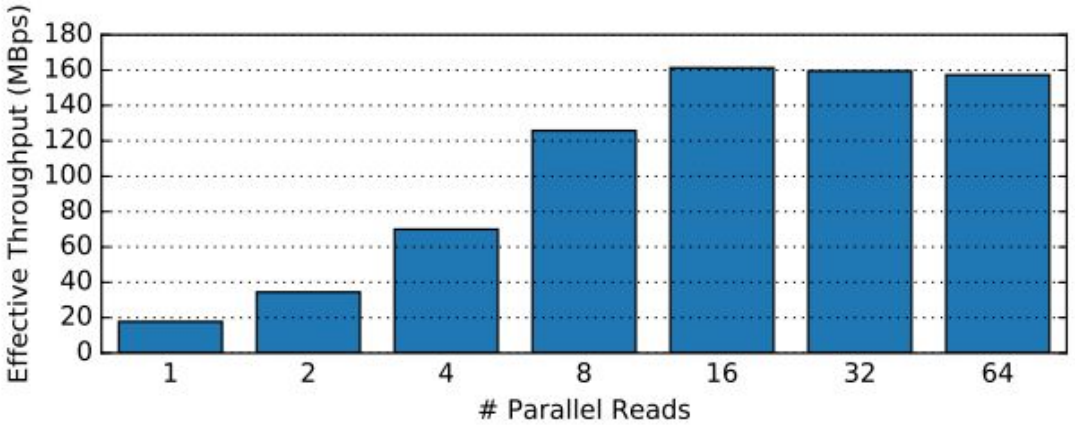


Reference: Perron, Matthew, et al. "Starling: A scalable query engine on cloud functions." Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. 2020.

Starling: A Scalable Query Engine on Cloud Function Services

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.
- S3 suffers from high storage latency, but an approach of several parallel reads by a task reduces this latency.
- Parallelizing reads keeps a task busy, making the time of idleness less by investing more time on the query processing.
- Figure shows total throughput of a function invocation reading many 256KB reads. As reads increase, effective throughput is higher until 16 parallel reads.



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.

Data management in Starling

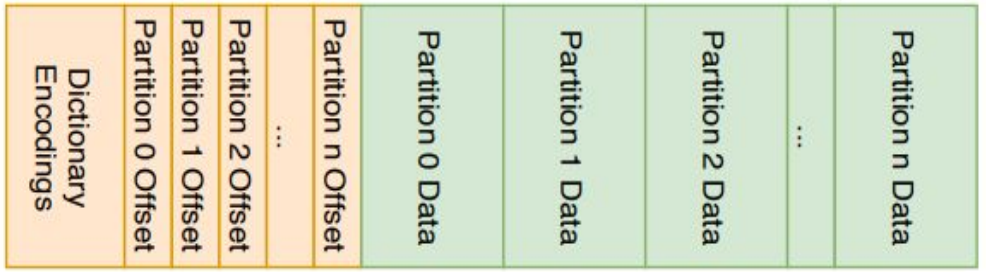
- **Base table storage**
  - Executes queries over data in S3. Starling design is agnostic to base table formats but CSV, ORC and Parquet are the usual choices.
  - Rows of a specified schema are parsed from source objects in S3. Data is stored as objects of few hundred MB for best performance.
  - Open-source columnar formats like ORC enable selective column reading and skipping of unnecessary data using indexes improving query performance and saving time.
  - S3 is a write-once system, hence objects can only be replaced. S3 provides atomic reads and writes, ensuring that readers never see data from two separate writes in the same read.

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 state**
  - Relies on AWS for data shuffling and passing intermediate data between function invocations.
  - Uses one to many communication where producer tasks write objects that are visible to all readers. All to all communication (shuffle operations) is difficult at low cost & writing one object per partition incurs high costs.
  - Starling optimizes the challenge with shuffle process by writing **single partitioned file to S3**, allowing consumer tasks to read only relevant portions.
  - Workers write output as a single S3 object in the S3 buckets with **predetermined keys**. Reader poll these keys for new object and once found fetches the metadata and relevant partitions.
  - The partitioned files begin with metadata that indicates the end locations of partition in object. Encodes low cardinality string columns using dictionary encoding.
  - The **metadata can be fetched with one read, followed by a read to fetch required partition**. Allows 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

Query execution in Starling

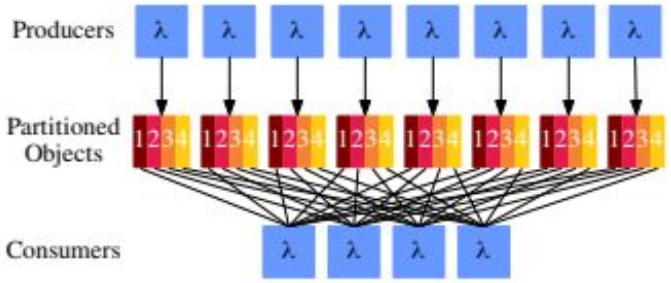
- Coordinator uses single JSON file of physical query plan as input. The query executable (C++ source code with dependencies) is uploaded to AWS Lambda. Each task input and output object names are determined before each stage tasks execution to minimize workers communication.
- **Relational operator implementation**
  - Reads input files data from base table in parallel, reads necessary columns with projections and relational operators are implemented as nested loops allowing pipelining that can reduce query latency.
  - Supports **broadcast joins**: each input task for inner relation writes single S3 object, and outer relation tasks read all data from inner relation and their own subset. Join is performed by comparing data in outer relation and inner relation stored in S3 objects.
  - Supports **partitioned hash joins**: The relations are scanned and data is partitioned with a join key. Join tasks create hash tables for its partition of relation and scans other relation, probing into partitions to perform join.
  - Performs **aggregations** in two steps. Each task performs preliminary operations to give sets of partial aggregates as S3 objects. A final task reduces these aggregated into a single final aggregate.
  - If necessary, data is shuffled for partitioning the data based on a group key for grouping.

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.

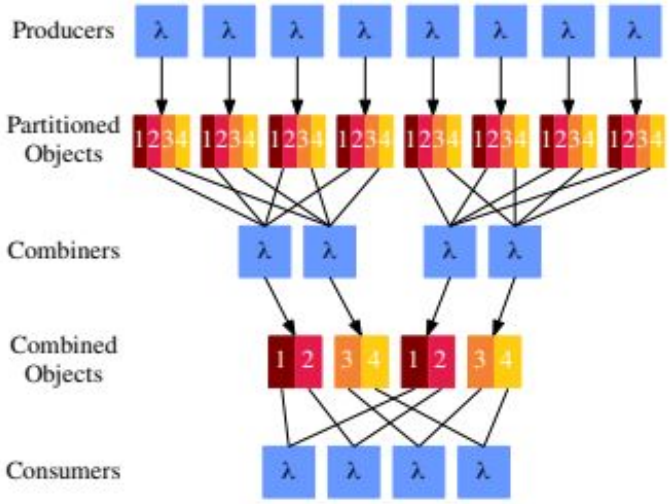
Starling: A Scalable Query Engine on Cloud Function Services

Query execution in Starling

- **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.



(a) Standard Shuffle



(b) Multistage Shuffle

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.

Next Steps

- Complete missing implementation details of discussed papers. Include related works in connection with the research papers where required.
- Understand how this research translates to real world applications in cloud services.

THANK YOU

Q&A