

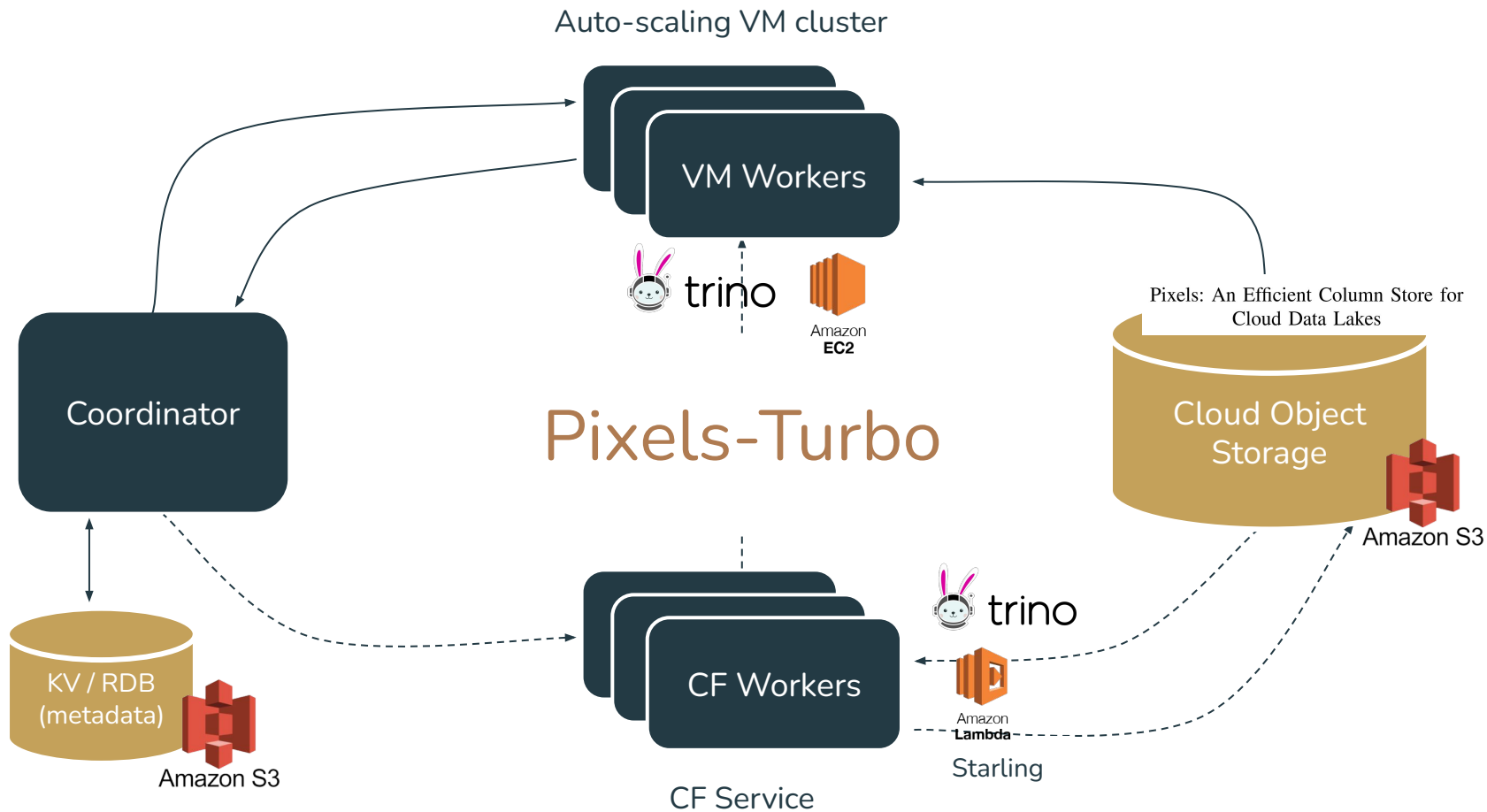


Pixels-Turbo

A Cloud-Based Query Engine for Elastic Data Analytics

Jingzhi Yan, Noah Picarelli-Kombert, Jinpeng Huang,
Kathlyn Sinaga, Zhiyuan Chen, David Lee



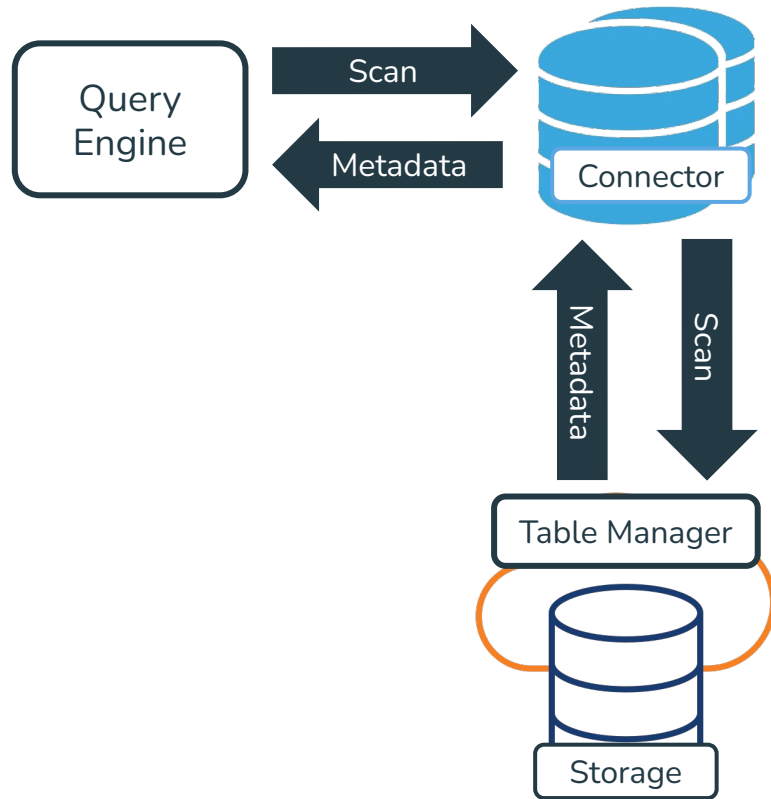




Background

Cloud Data Lakes

- Decoupled storage and query execution



Query Services of a Data Lake

- Previously, cloud users use virtual machines (VMs) to run queries
 - Maintenance cost for these VMs took a large portion of the total cost
- Solution: cloud providers developed **serverless query services**
 - Users only pay for executed queries instead of a whole VM
 - Cloud providers maintain the processing cluster for the user





Query Services of a Data Lake



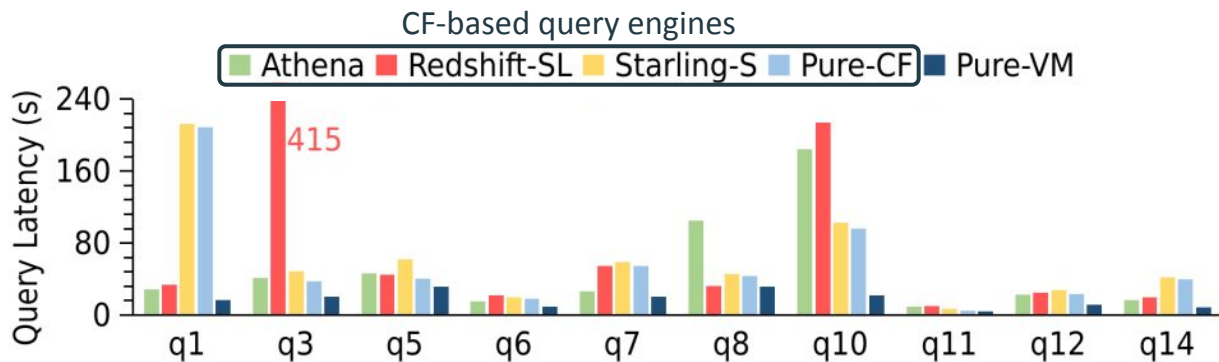
Cloud Functions (CF)

- A category of serverless computing service
- Individual functions are uploaded to a shared cloud server and executed independently
 - Invoked, executed, and billed individually
 - No direct communication between functions



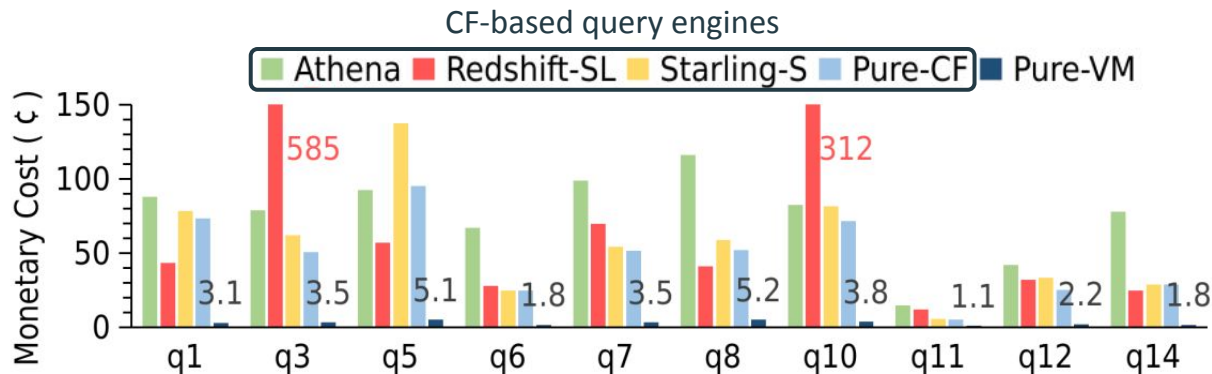
Cloud Functions vs. Virtual Machines

Query Latency on TPC-H



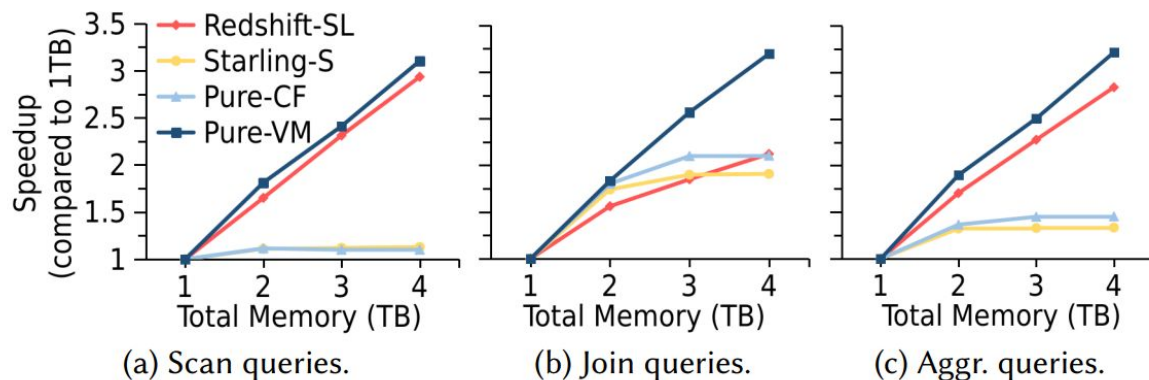
Pure-VM has the best query latency compared to CF-based query engines because of higher I/O bandwidth and an efficient massively parallel processing (MPP) cluster.

Monetary Cost on TPC-H



Pure-VM is 1-2 orders of magnitude **more cost-efficient** compared to CF solutions. CFs accumulate an additional cost on S3 requests, but optimizations can lower this cost.

Scalability: Speedup vs. Memory capacities



Systems designed for **clusters with inter-node communication** (Pure-VM and Redshift-SL) **have better scalability** when not affected by the sub-optimal query plan.

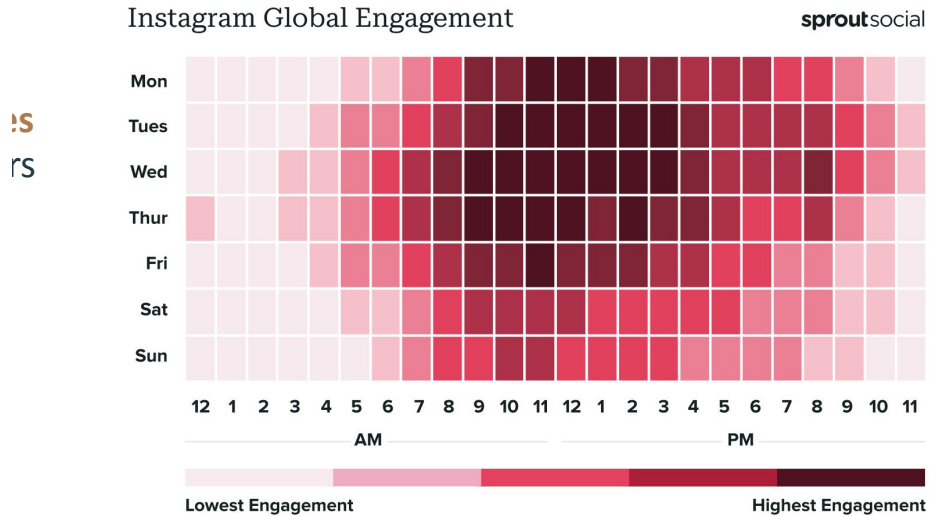
Cloud Function vs. Virtual Machine

	Cloud Function	Virtual Machine
Resources		
Storage		
Startup Time		
Elasticity		
Execution Time		
Billing		



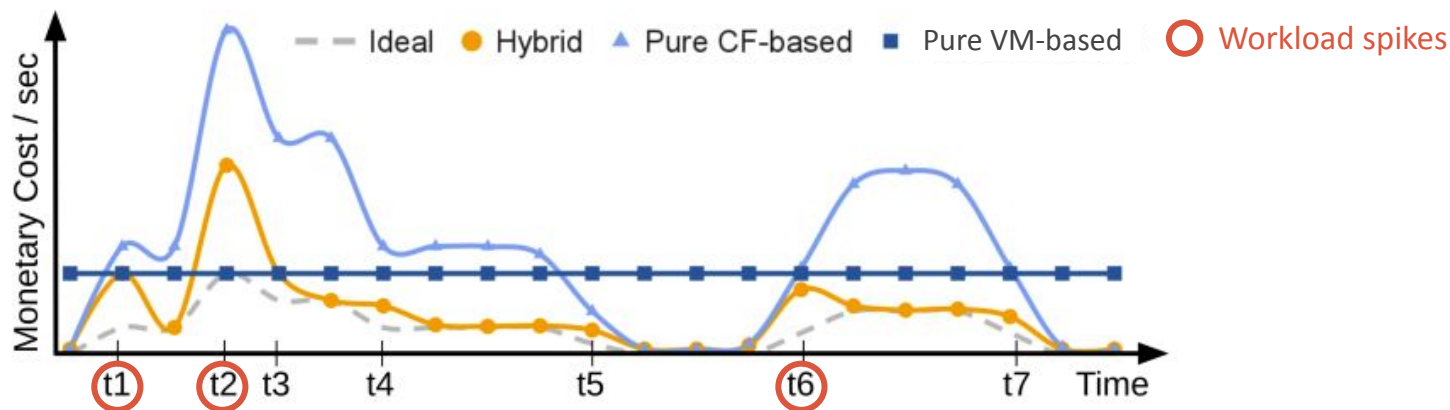
Motivation

The Problem



*All time frames are recorded globally, meaning you should be able to publish with the times provided in any timezone and see positive engagement results.

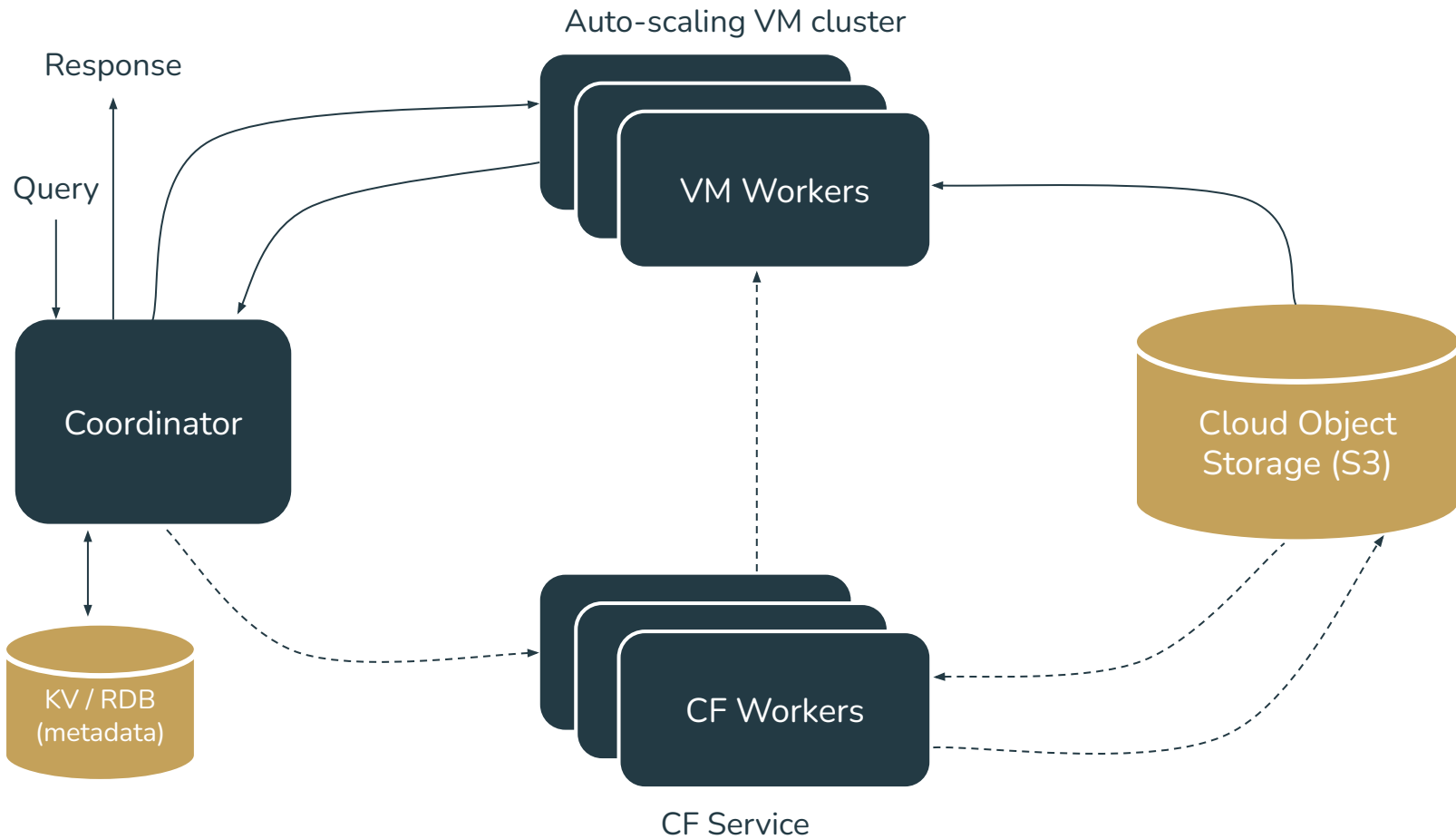
Cost Comparisons on a Dynamic Workload



- Pure CF-based system still costs more than a pure VM-based system for a sustained, dynamic workload
- Hybrid achieves a lower cumulative cost

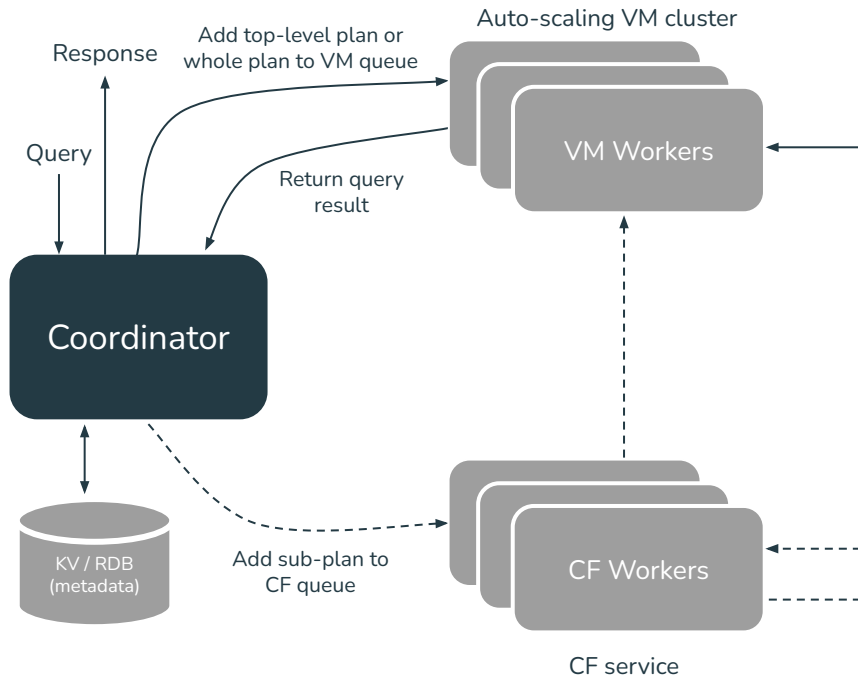


Architecture



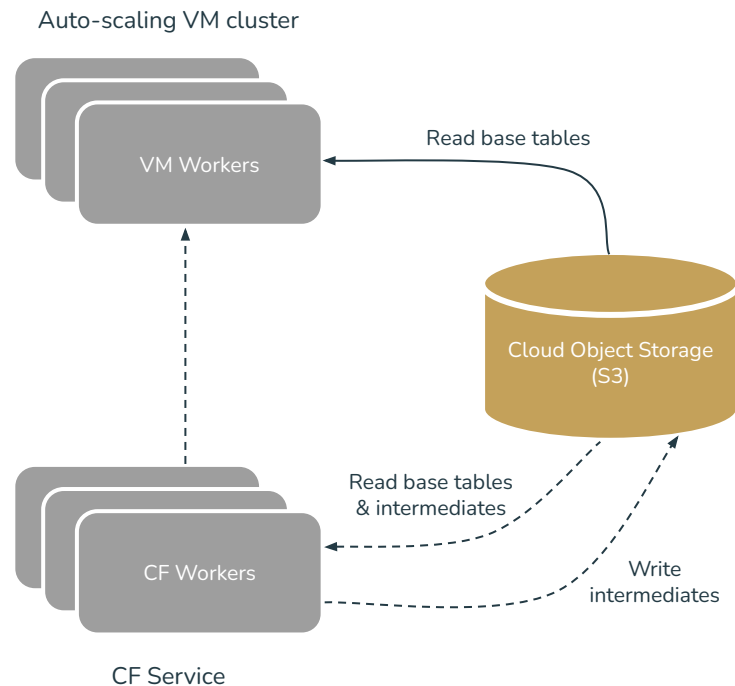
The Coordinator

- The only long-running component
 - It can run in a small VM



Cloud Storage

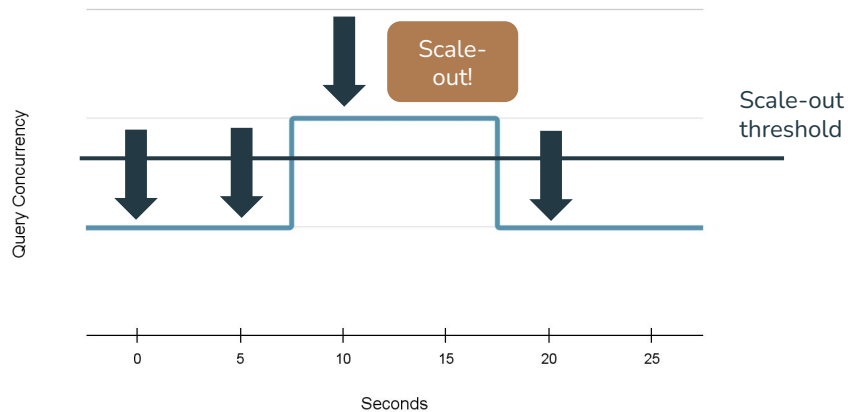
- Cloud storage contains the tables to be queried



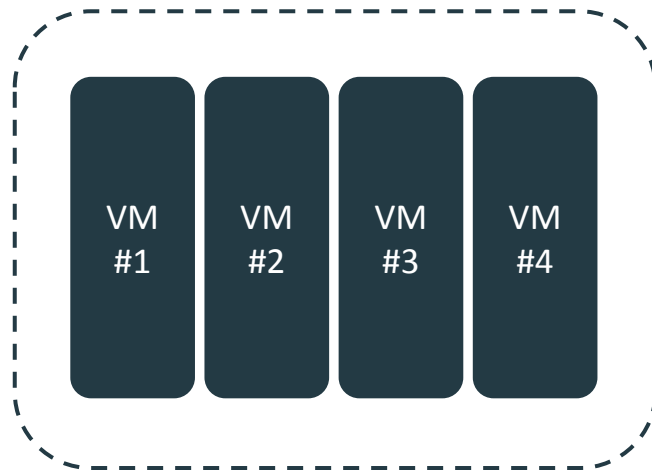
Auto-scaling VM Cluster

- Auto-framework to add (scale-out) or remove (scale-in) VMs consists of three parts

Active Scale-out

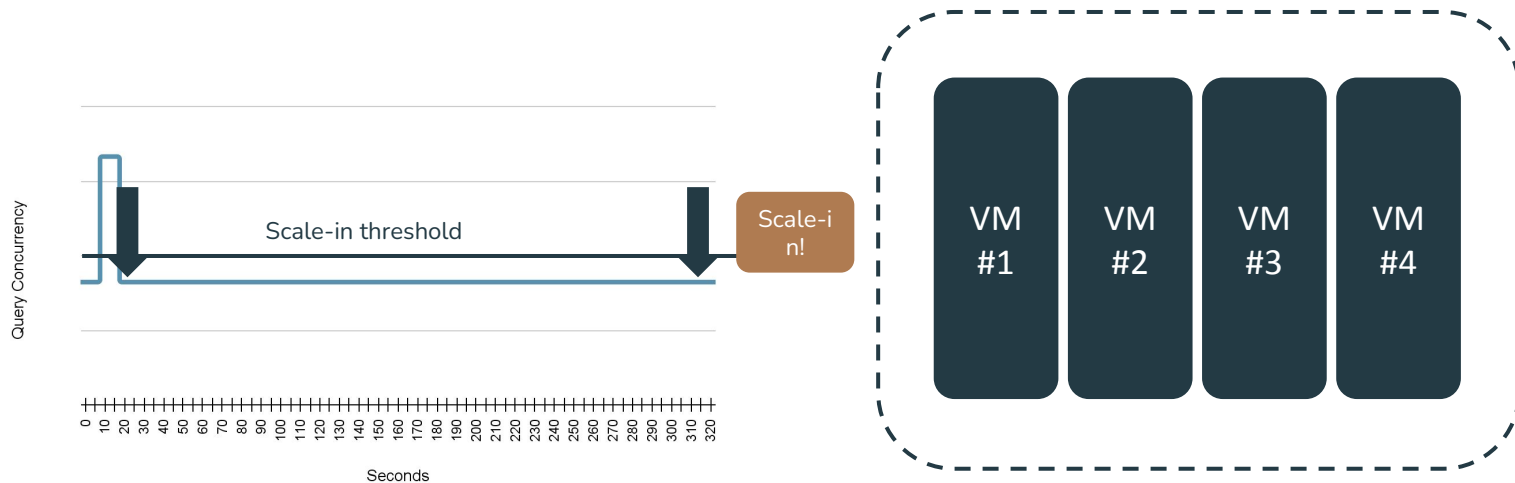


Scale-out actions can occur every five seconds, but scale-in actions only occur every five minutes. These periods are configurable.



VM Cluster

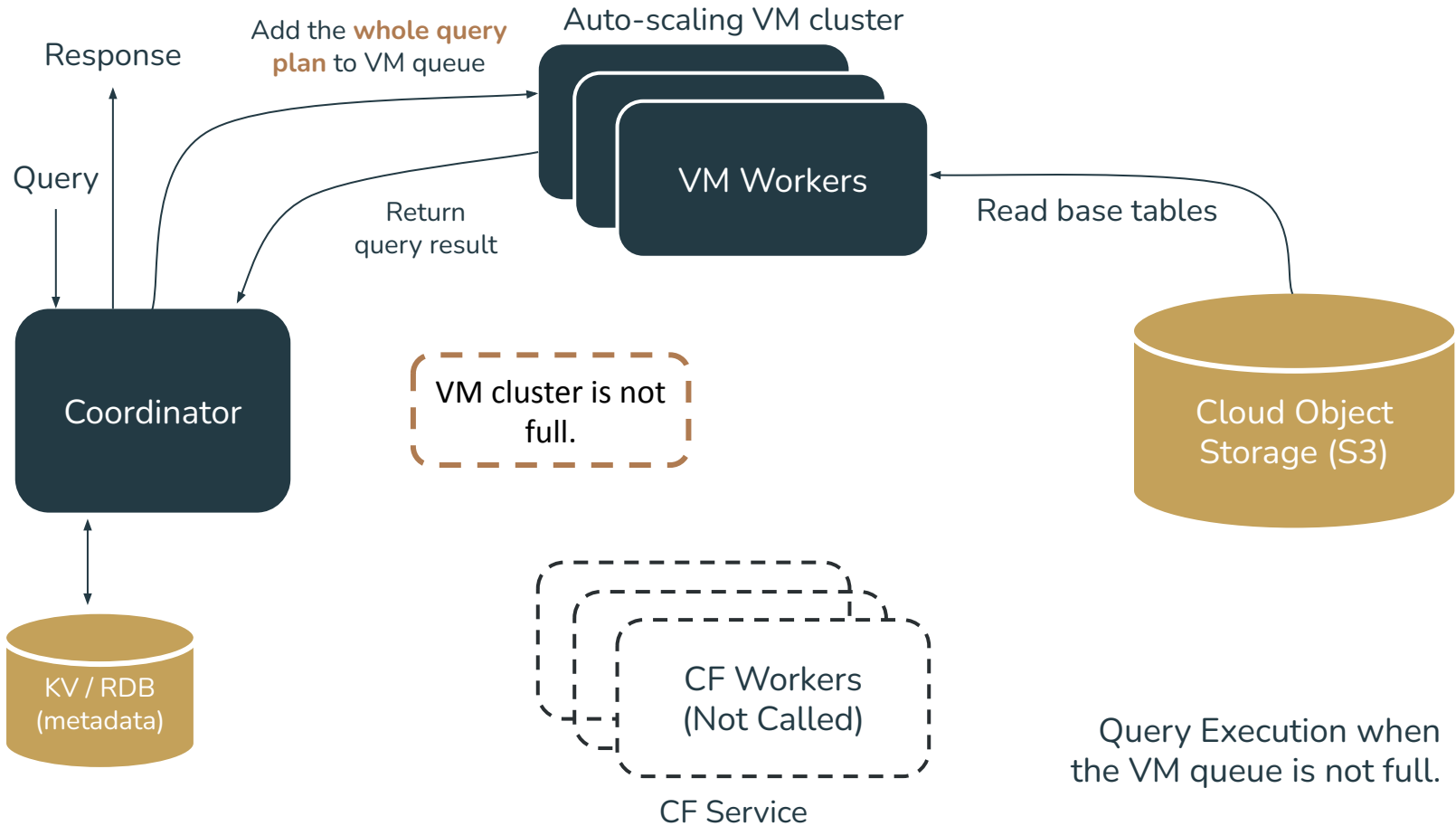
Lazy Scale-in

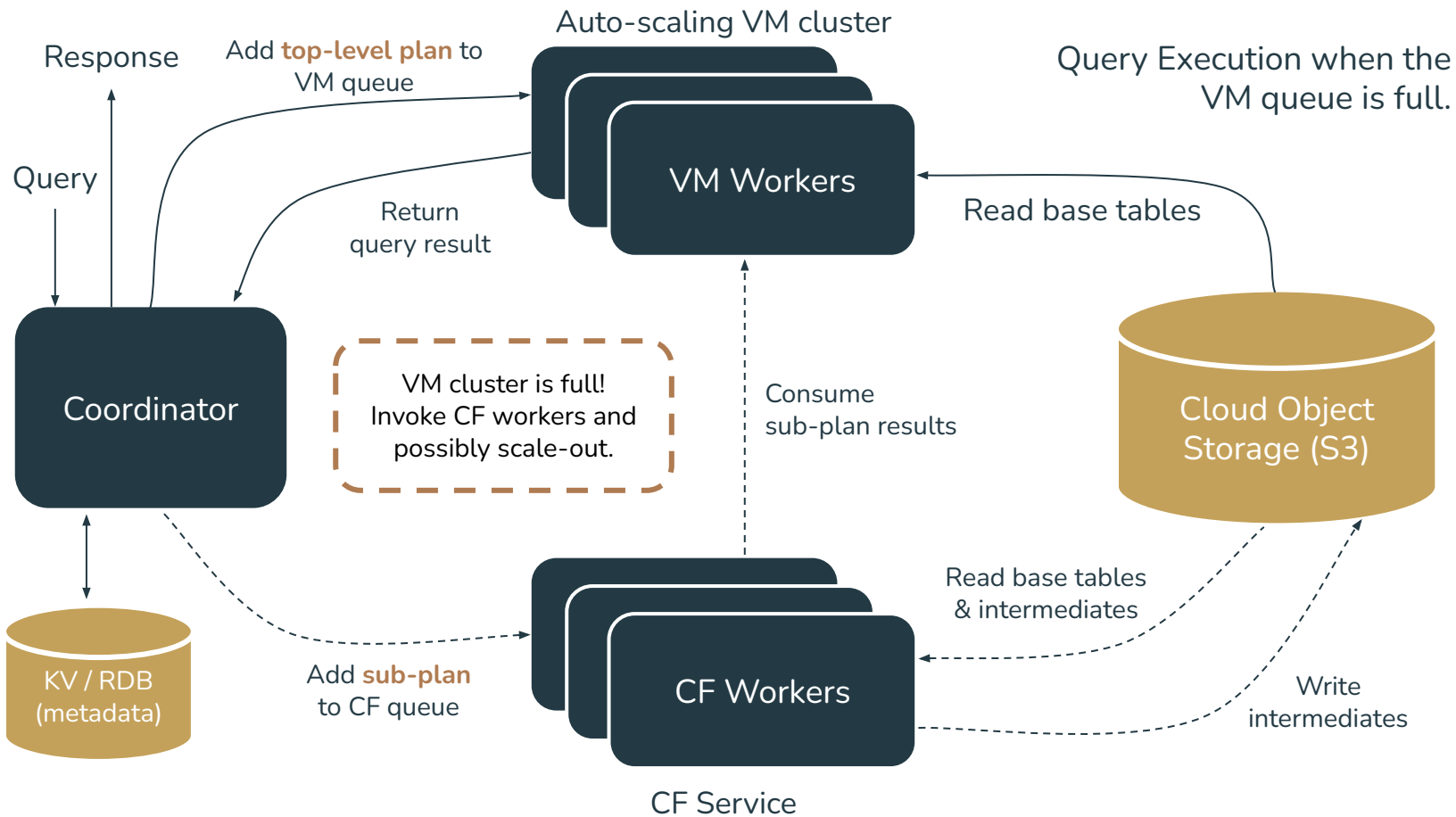


Scale-out actions can occur every five seconds, but scale-in actions only occur every five minutes. These periods are configurable.



Query Execution







Sub-plan Operations in Cloud Functions

Table Scans

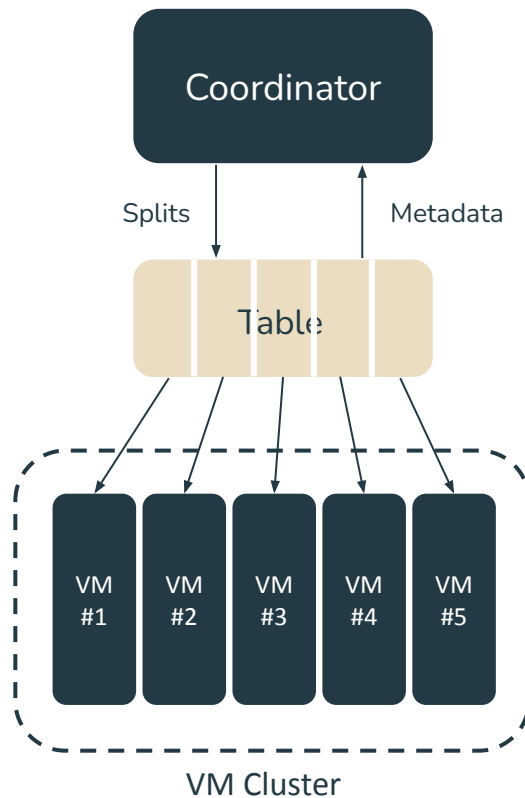
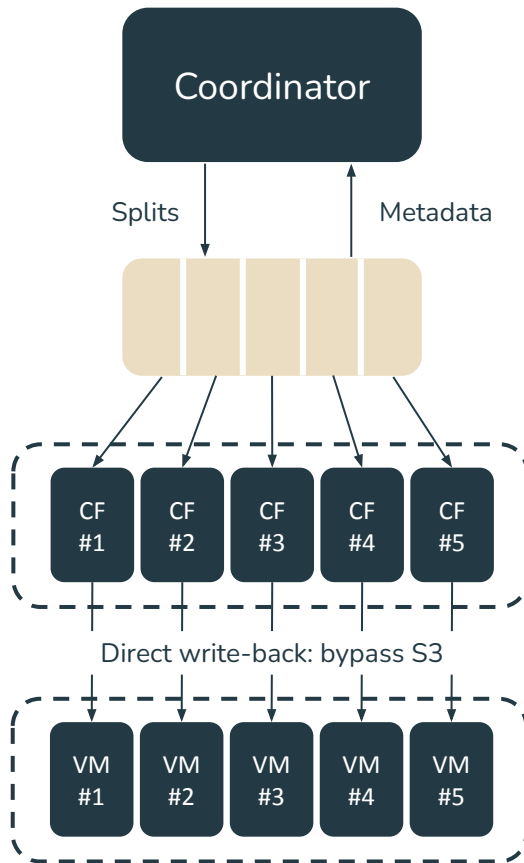
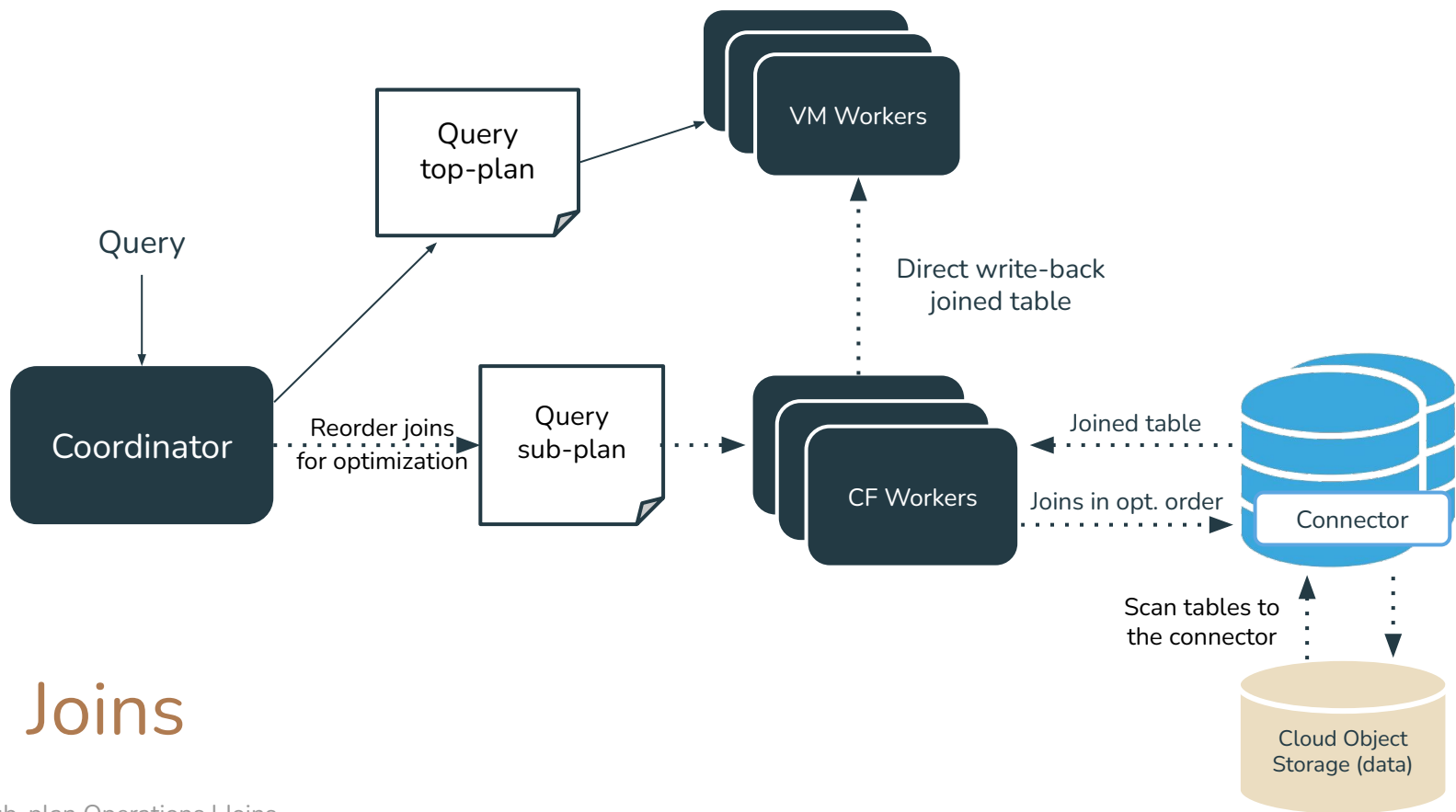


Table Scans

- Coordinator creates splits containing metadata and row partitions
- If **VMs are used**, splits are sent to VMs, which each process the query on given splits over multiple threads
 - VMs can use a file format library to directly scan the partitions in the splits

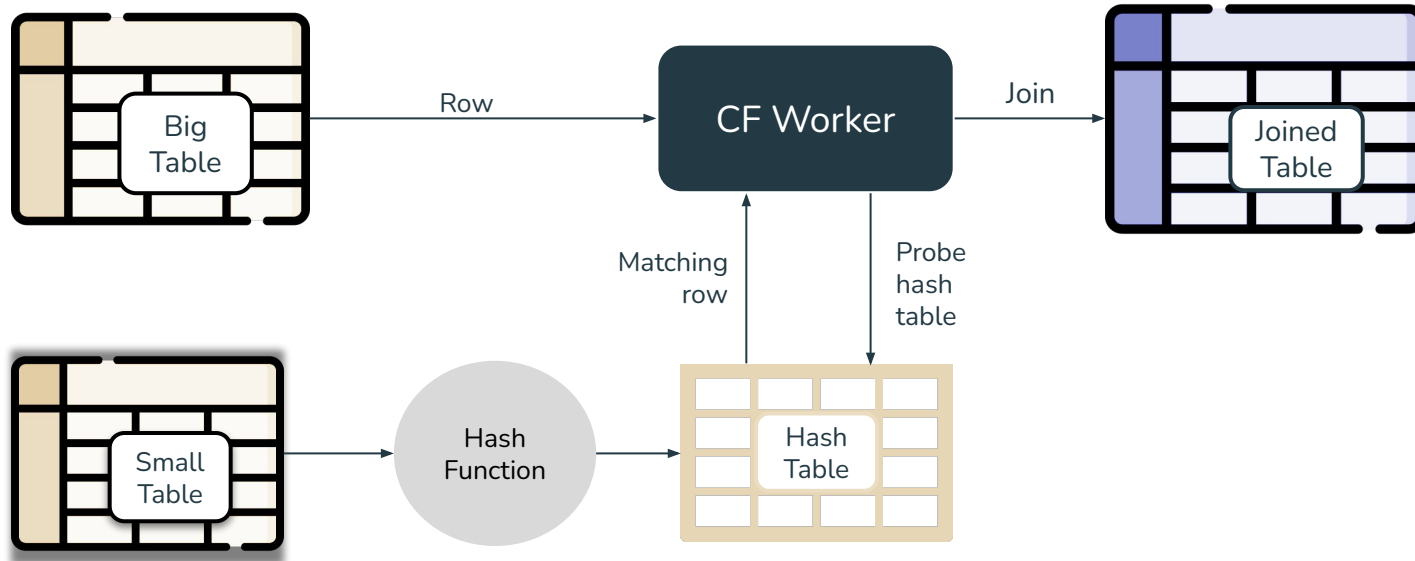




Joins

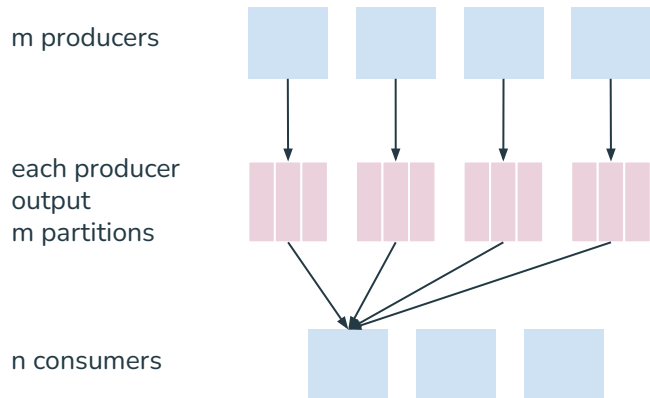
Broadcast Joins

Tables to join:



Shuffling

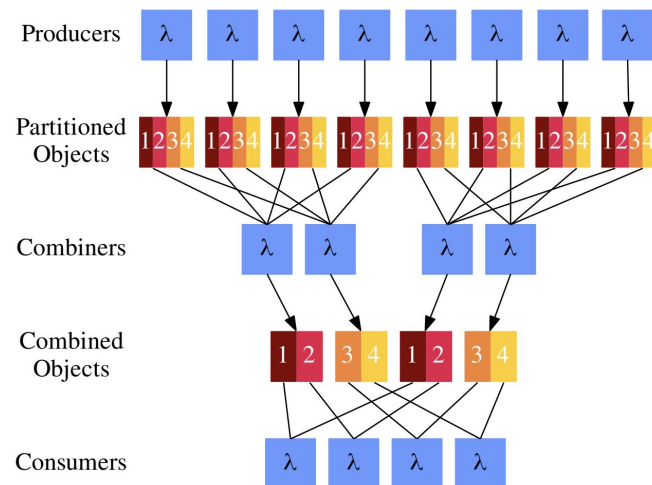
Standard Shuffle



Problem: S3 Throttling

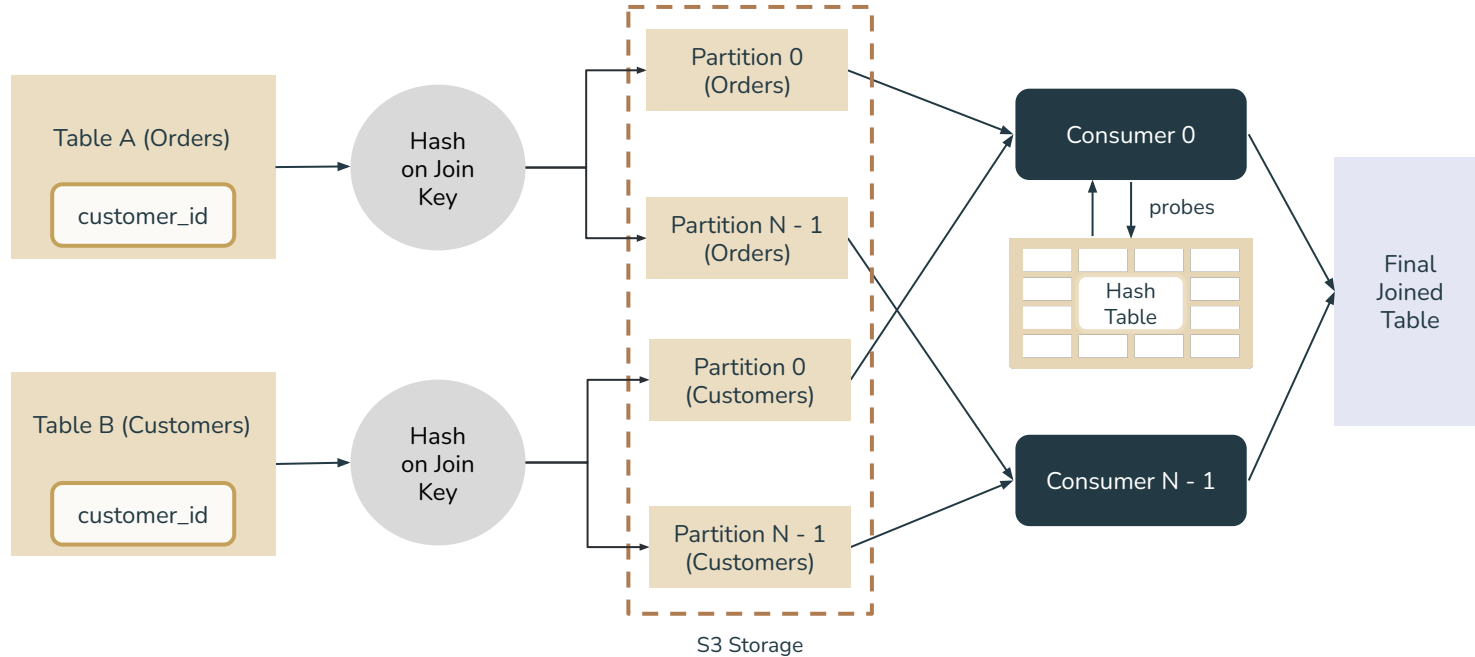
Sub-plan Operations | Joins

Introducing Combiners



Trades higher I/O for better load balancing and lower per-consumer stress on S3

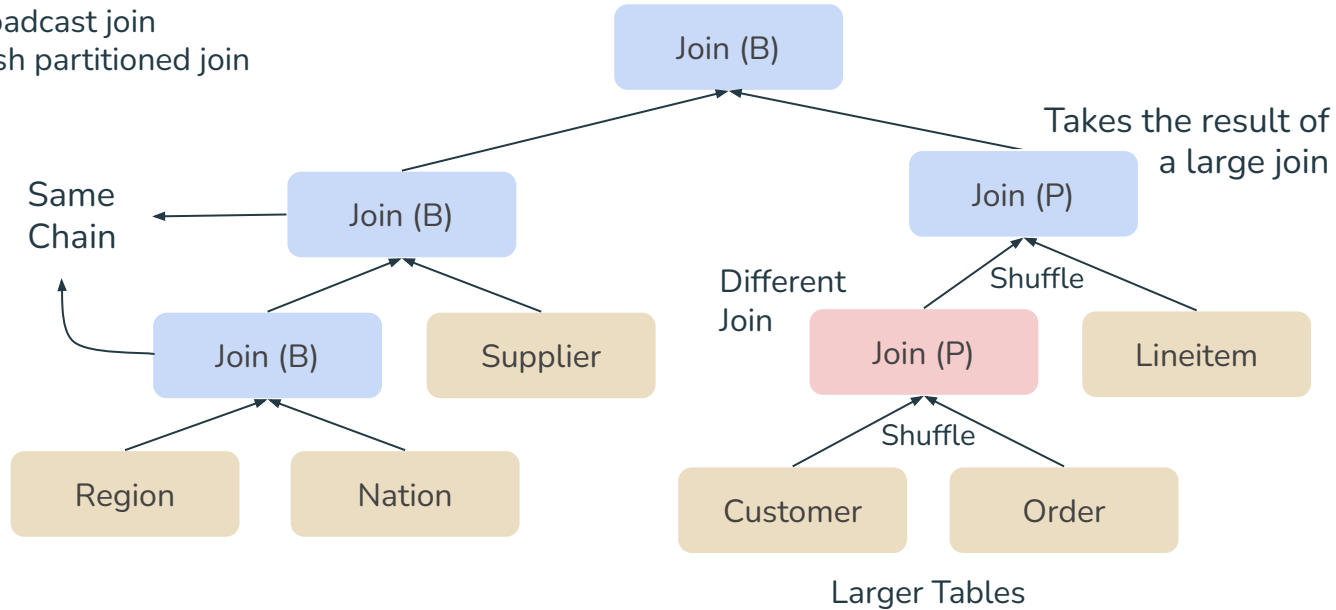
Hash Partitioned Joins



Chain Join: Bypass S3

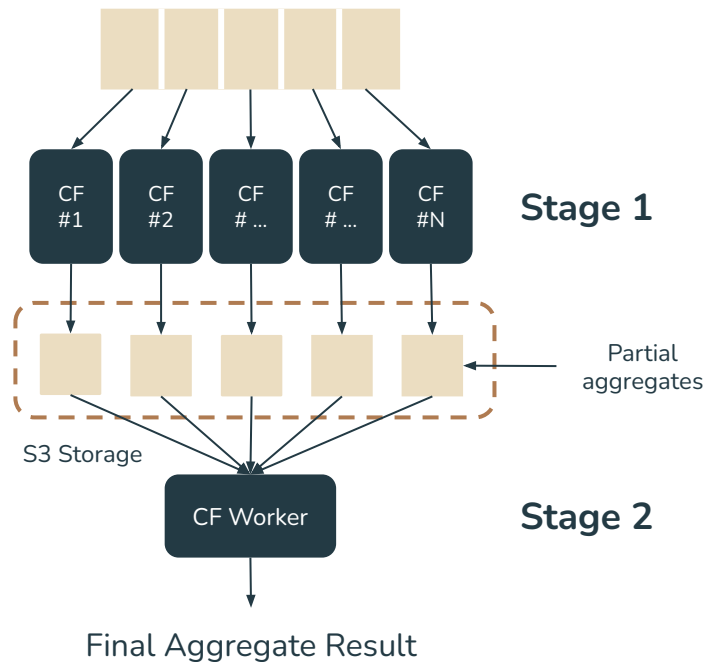
B: Broadcast join

P: Hash partitioned join



Direct Aggregation

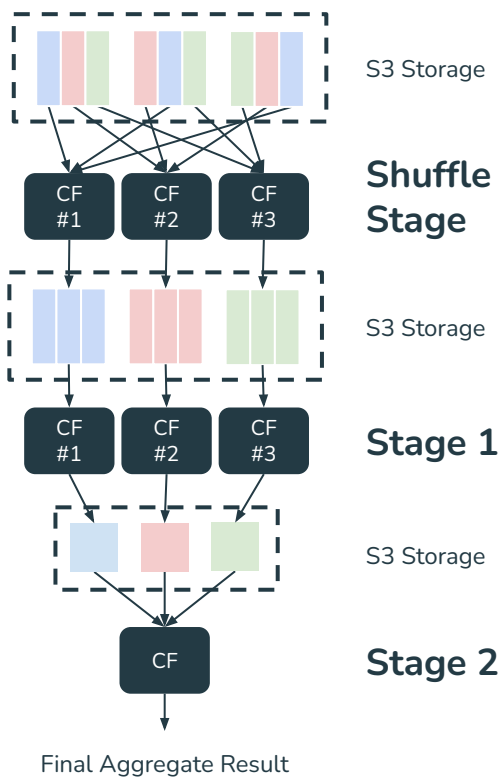
- The coordinator pushes aggregations after table scans and joins



Shuffled Aggregation: Starling

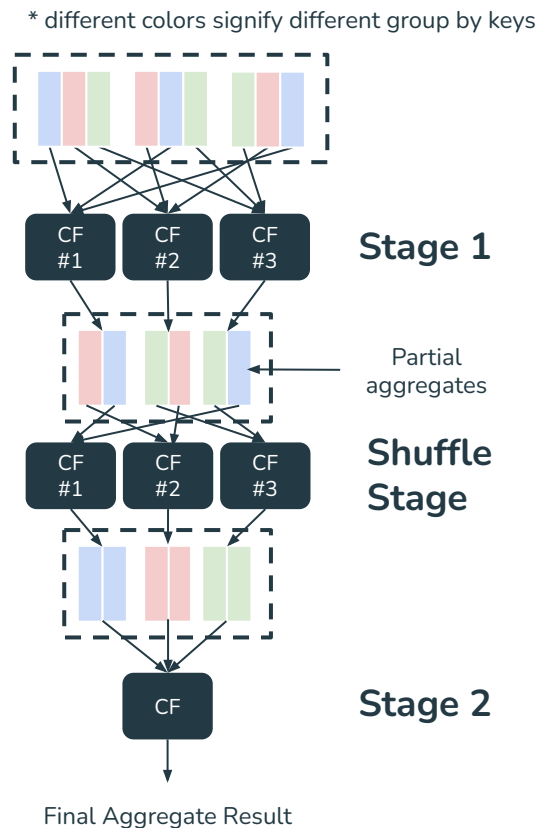
- Used for larger aggregates

* different colors signify different group by keys



Shuffled Aggregation: Pixels-Turbo

- Late Shuffling: postpone the shuffling stage



Implementation Optimizations

Implementation Optimizations



Finding Optimal Settings for Tuning Knobs in CFs

Size of CF Instances

- Use the **largest permitted size** for the CF instance
 - Larger CF instances result in fewer CFs
 - Executor runtime has a constant memory overhead, so larger instances tend to be more memory efficient
 - Testing suggests that the largest instances perform the best and result in the lowest cost

Split Size on Base Tables

$$S_s = \min \left(\frac{cap_{sb}}{\sum_{c_i \in C} \text{chunk_size}(c_i)}, \frac{g \cdot cap_{sr}}{s_r} \right)$$

S_s = split size

cap_{sb} = cap on number of bytes to read from a split

cap_{sr} = cap on number of filtered rows per split

g = number of row groups in the table

C = set of columns to read

chunk_size = average size of column chunks

S_r = filtered rows on the tables

- The **split size determines parallelism** for operators that read base tables
- cap_{sb} and cap_{sr} ensures CF workers will not block on I/O or CPU or fail on out-of-memory (OOM) errors

Join Algorithm Selection

$$S_b = \sum_{c_i \in C} size(c_i)$$

$$S_r = n \cdot \min_{c_i \in C_p} selectivity(c_i)$$

S_b = hash table size (bytes)

S_r = hash table size (rows)

C = columns to read

n = table row count

C_p = predicate columns

- Broadcast tables are limited by bytes (cap_{bb}) and number of rows (cap_{br})
 - cap_{bb} controls reading and decoding cost of the broadcast table
 - cap_{br} controls building cost and collision rate
- Joint selectivity is estimated since we don't have histograms
 - Selectivity may be overestimated, but it avoids broadcasting oversized tables

Number of partitions for shuffle-based operators

- Any shuffle-based operation requires calculating the number of partitions
- Hash partitioned join
 - Partition size is limited by bytes (cap_{pb}) and number of rows (cap_{pr})

$$P_j = \max \left(\frac{S_{b_1} + S_{b_2}}{cap_{pb}}, \frac{S_{r_1} + \alpha \cdot S_{r_2}}{cap_{pr}} \right)$$

P_j = number of partitions for joins

S_{b_1}, S_{b_2} = size of two tables (bytes)

S_{r_1}, S_{r_2} = size of two tables (rows)

α = speed ratio of hashing building and hash probing

cap_{pb} = partition size cap (bytes)

cap_{pr} = partition size cap (rows)

Number of partitions for shuffle-based operators

- Aggregation
 - Number of aggregation partitions relies on the size of the aggregates, which rely on the group-by key cardinality

$$D_g = \max_{c_i \in C_g} \text{cardinality}(c_i)$$

$$P_a = \frac{D_g}{\text{cap}_{ar}}$$

D_g = joint cardinality of group-by keys

C_g = set of group-by key columns

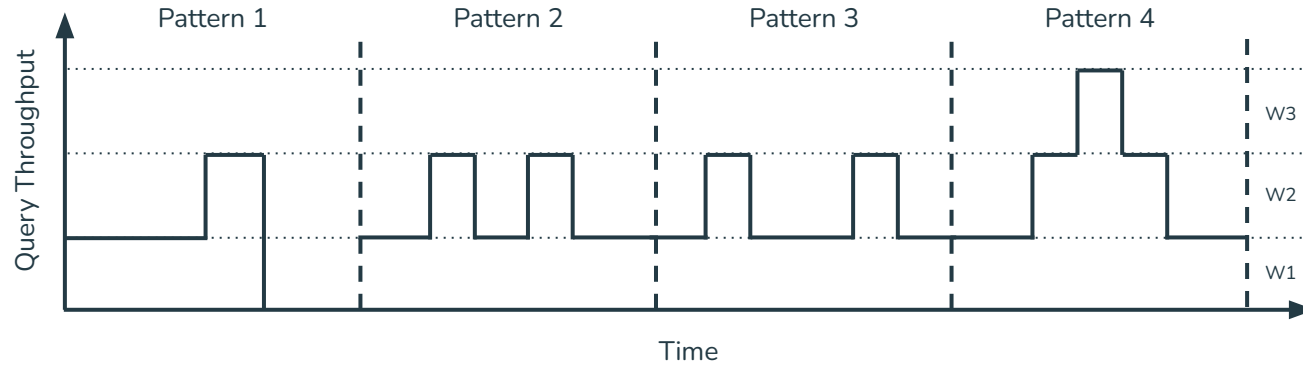
P_a = number of partitions for aggregation

cap_{ar} = max rows per partition in aggregation

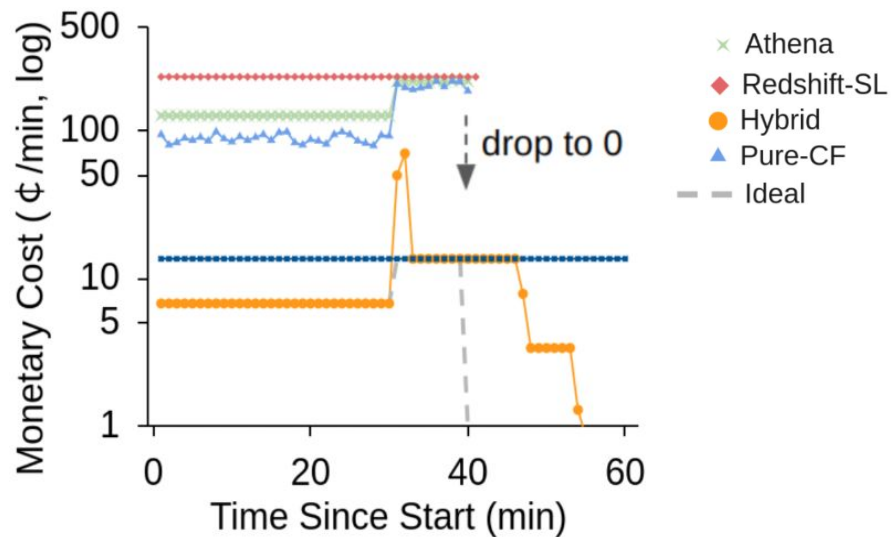
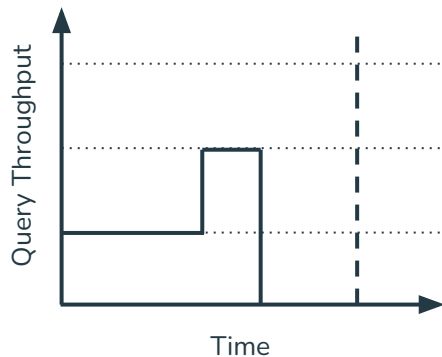


Evaluation

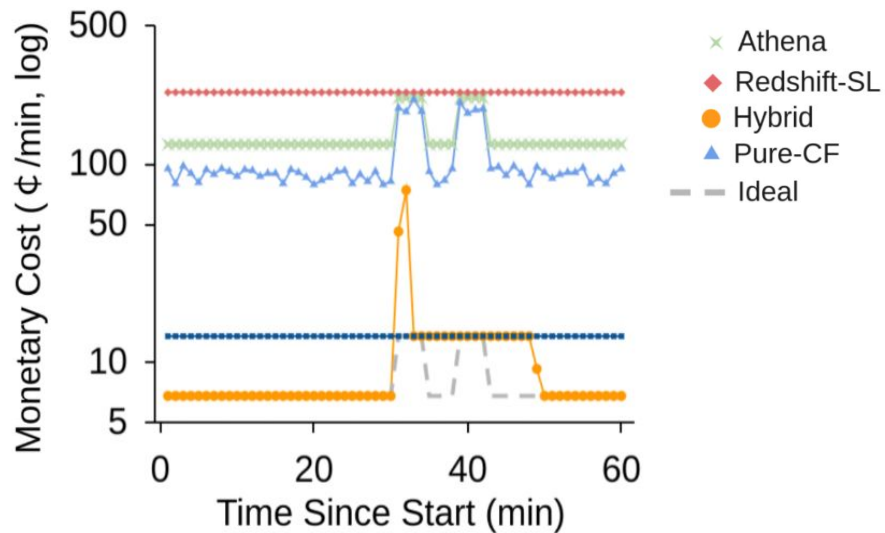
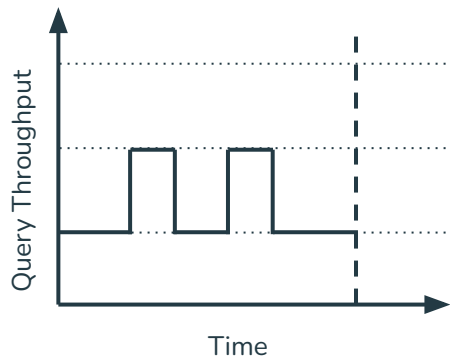
Experiment Workload Patterns



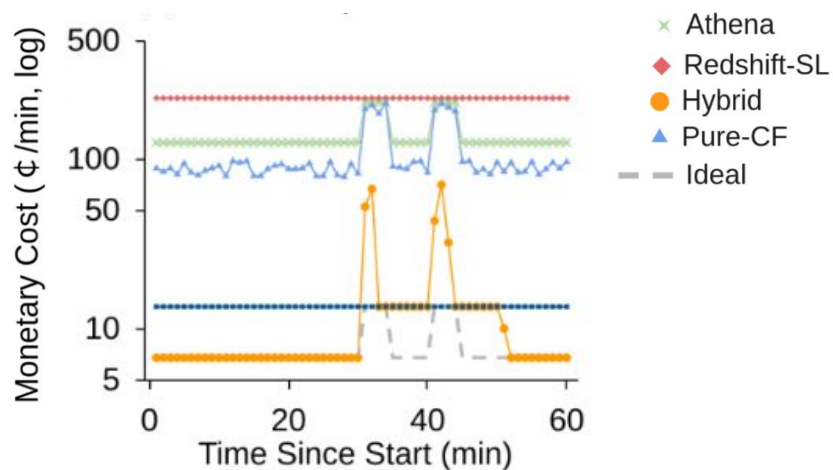
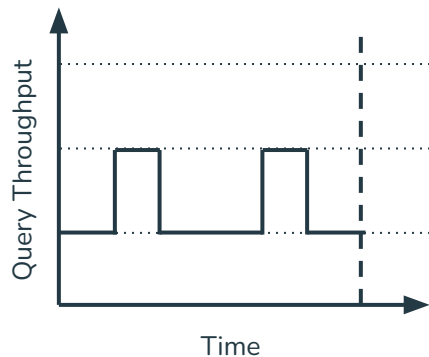
Experimental Results: Pattern 1



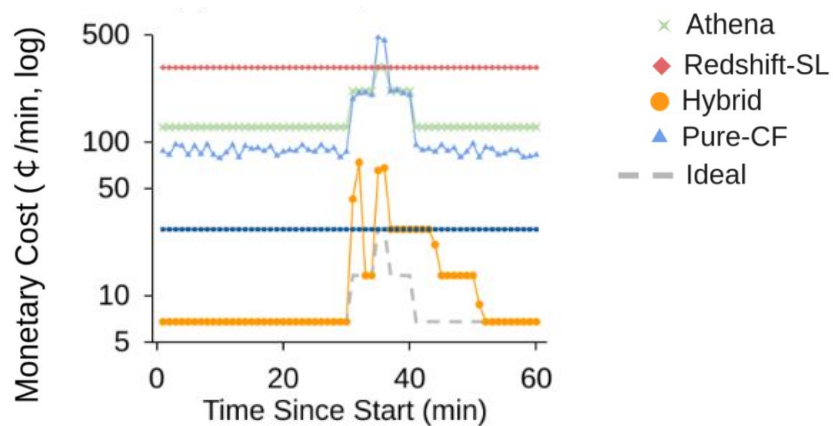
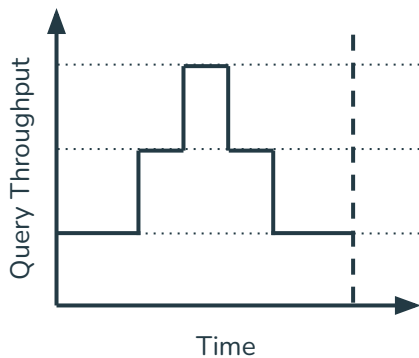
Experimental Results: Pattern 2



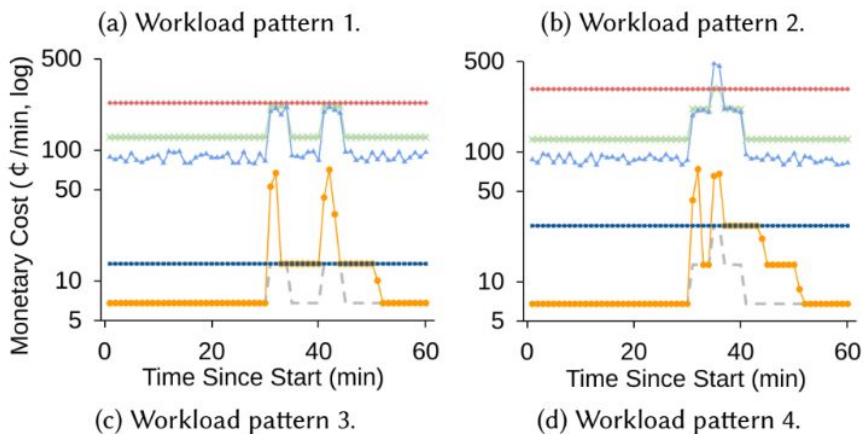
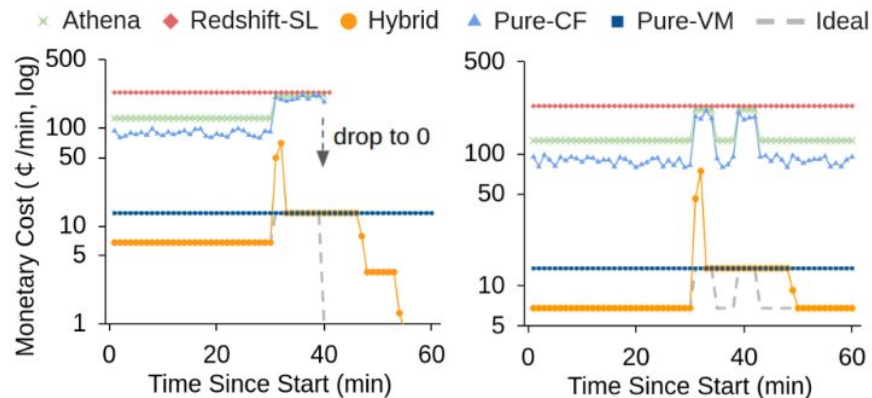
Experimental Results: Pattern 3



Experimental Results: Pattern 4



Experimental Results: Monetary Cost



Monetary Cost

- The hybrid system is capable of scaling to zero
- When consecutive spikes are close enough in time, the CFs of the first spike are reused for the second
- When the workload repeatedly experiences building spikes, the hybrid cost repeatedly spikes as well

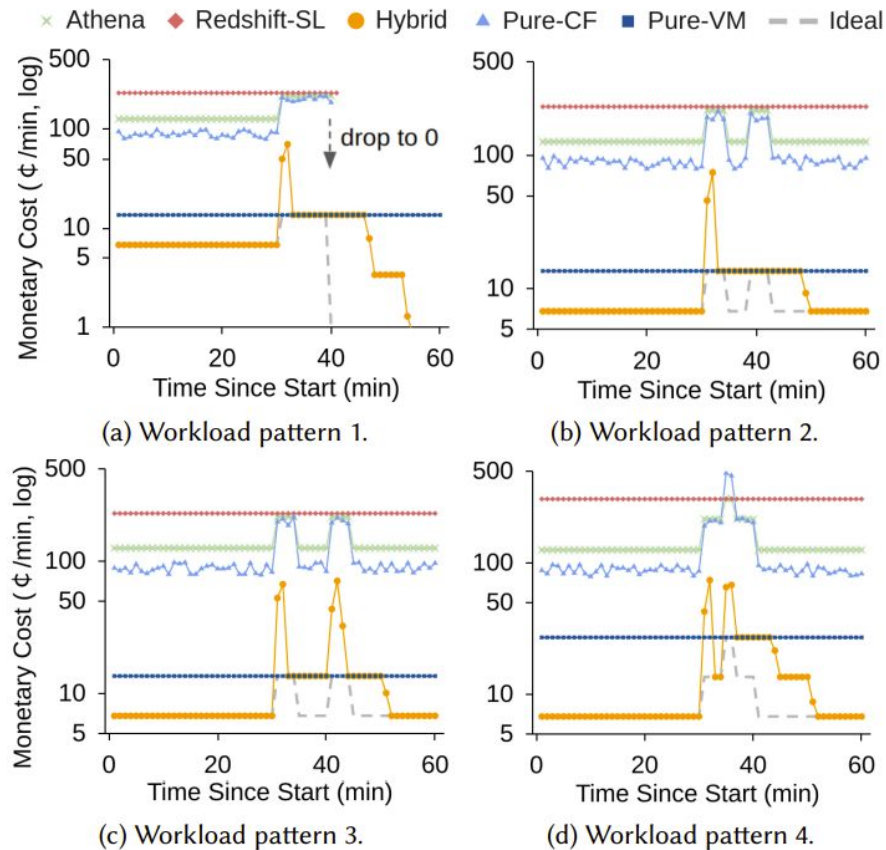


Fig. 9. Monetary cost per minute of the workload patterns.

Query Latency

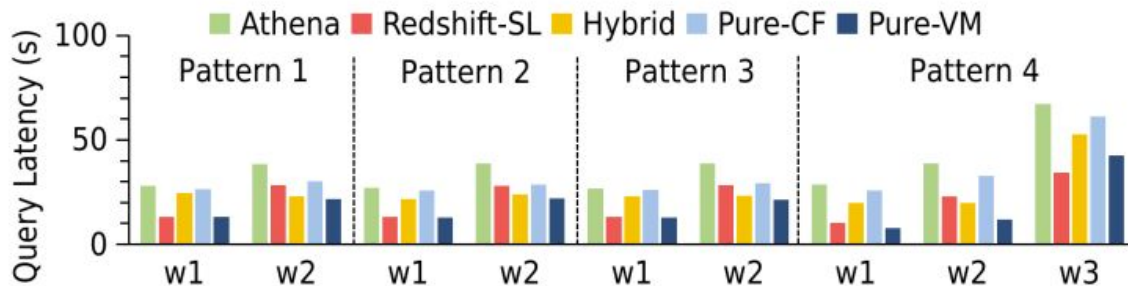


Fig. 10. Average query latency of the workload patterns.

- Hybrid provides lower query latency than auto-scalable baselines (Athena, Pure-CF) in all cases because most queries are processed in the VM cluster.
- Pure-VM is faster than Hybrid in all cases.
 - Redshift and Pure-VM were provisioned for spikes at an increased monetary and operational cost

Discussion

Is monetary cost the only benefit to Pixels-Turbo?

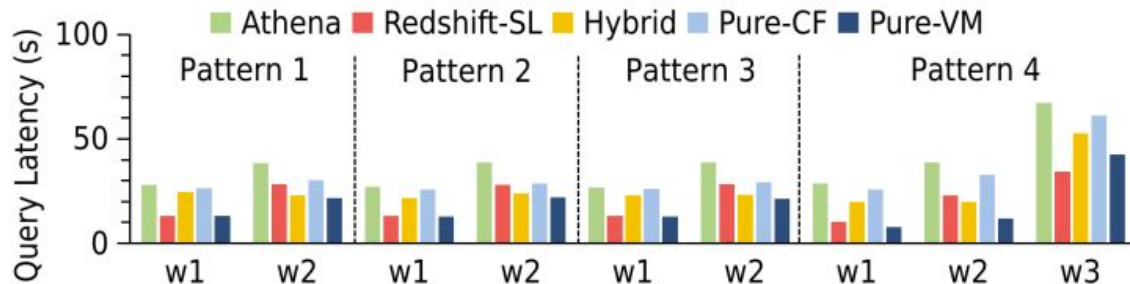
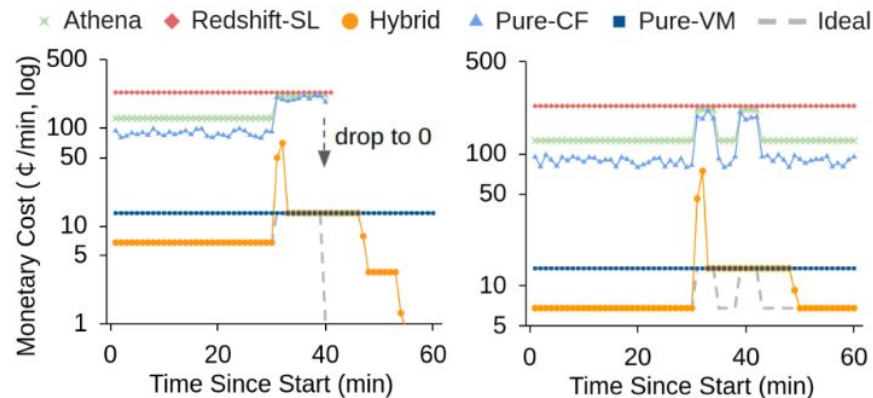


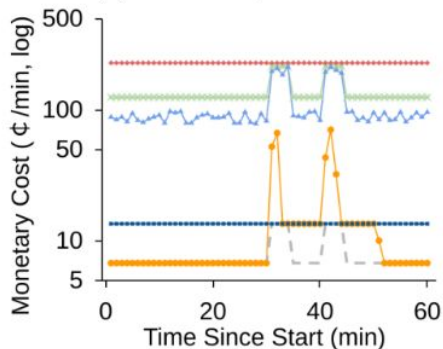
Fig. 10. Average query latency of the workload patterns.

Pure-VM was
overprovisioned.

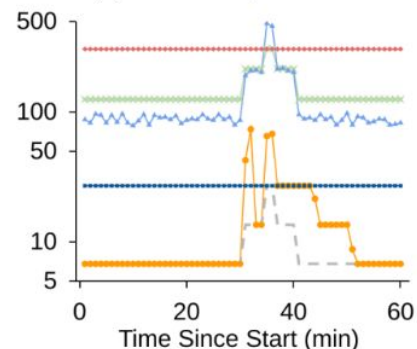


(a) Workload pattern 1.

(b) Workload pattern 2.

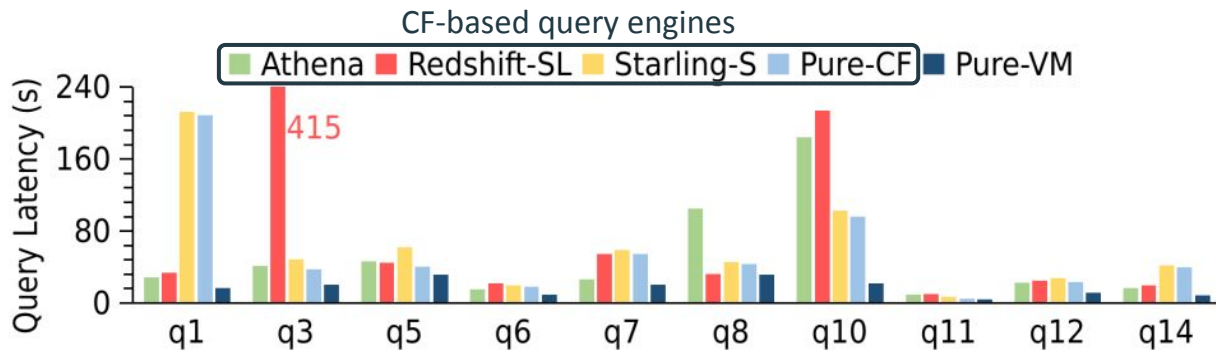


(c) Workload pattern 3.

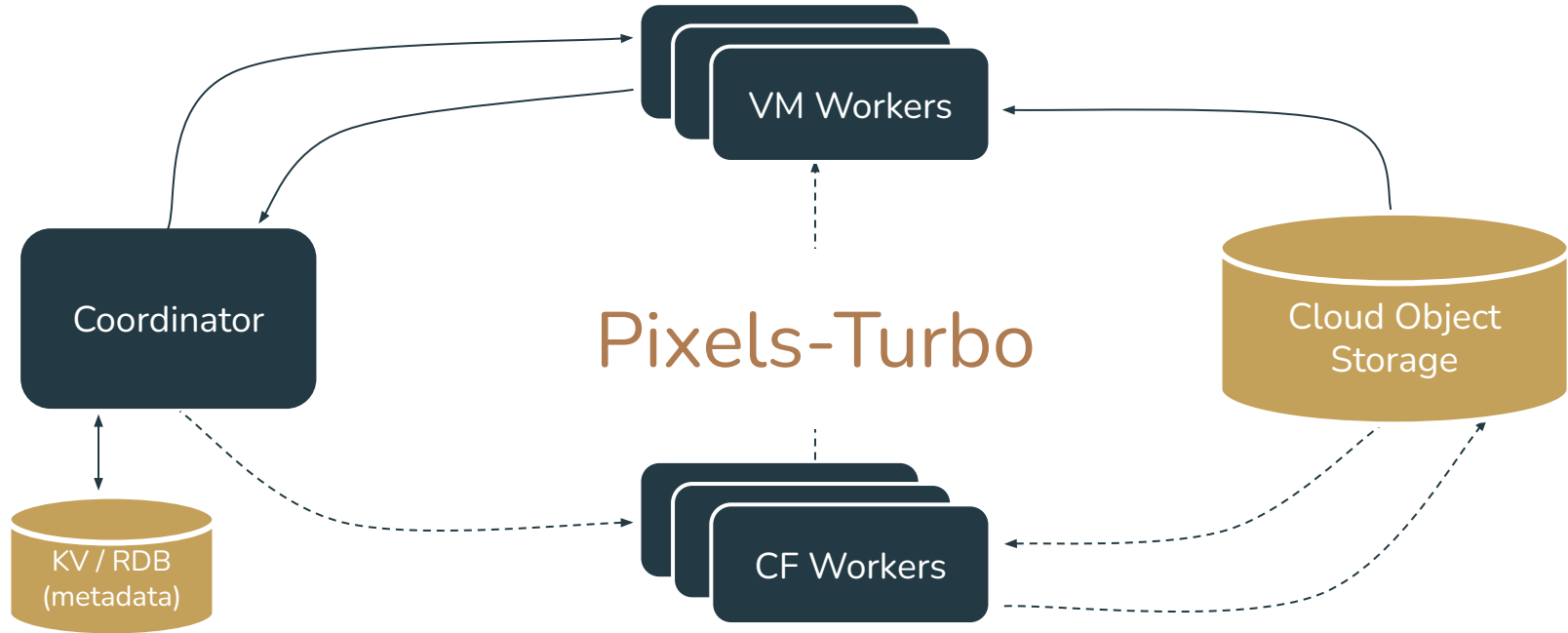


(d) Workload pattern 4.

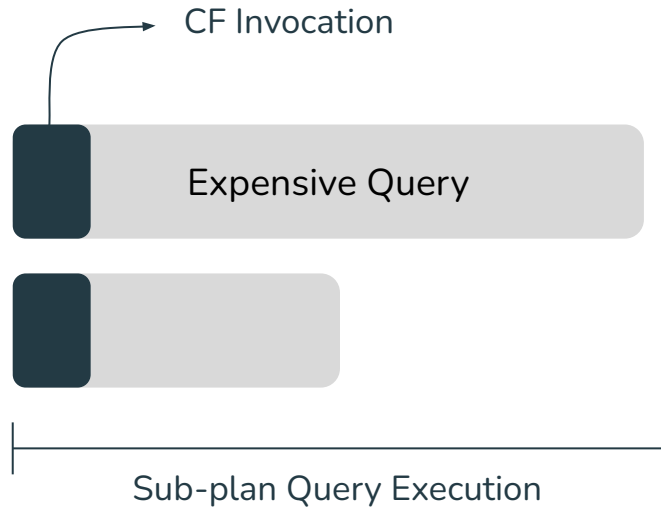
Why are operations pushed down to sub-plans more expensive if VMs have higher query latency?



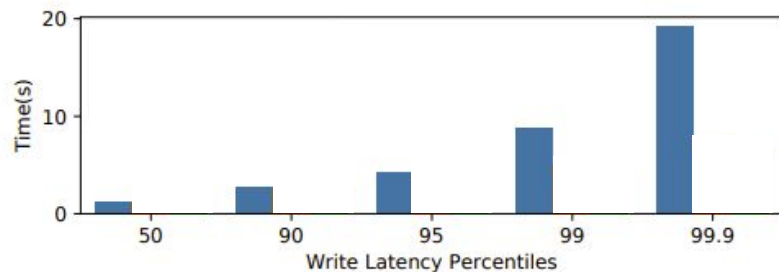
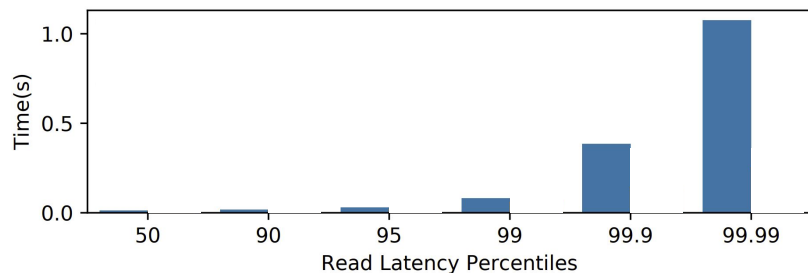
CFs should support VMs



CFs are charged per invocation.



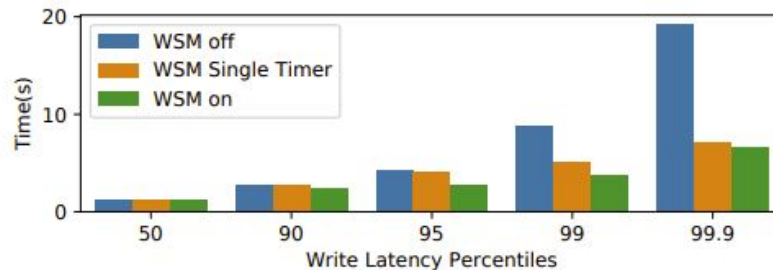
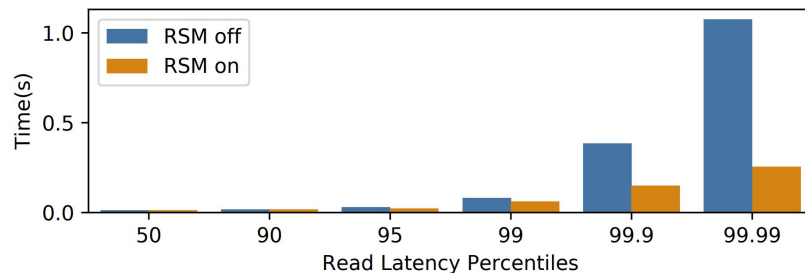
How does the system handle cases where network instability or limited I/O performance could affect sub-plan execution?



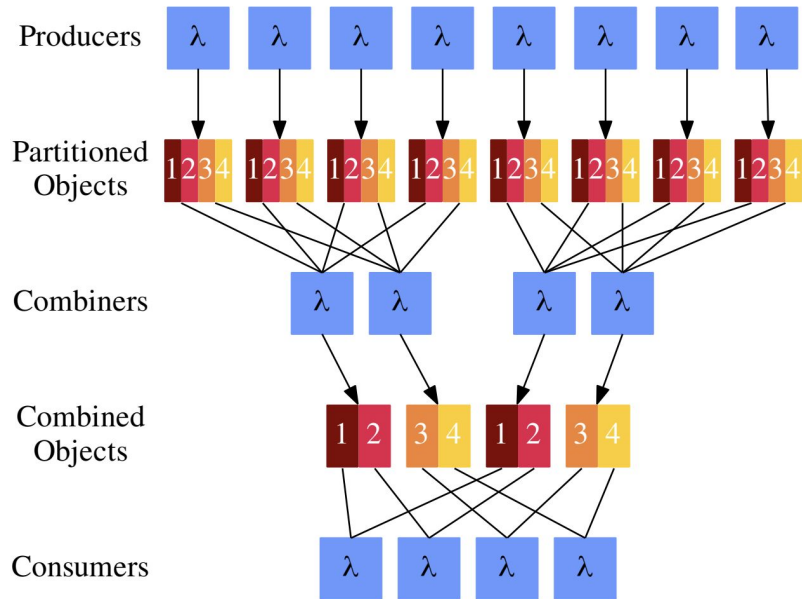
Solution proposed by Starling: Read Straggler Mitigation (RSM) and Write Straggler Mitigation (WSM)

$$r = \left(l + \frac{b}{t \cdot c} \right)$$

r: expected response time
b: number of bytes requested
c: number of concurrent readers
l: latency of CF service
t: throughput of CF service



What is the benefit of adding combiners?



It reduces the cost, even with extra writes.

$$\text{S3 Reads without Combiners} = 2mn$$

$$\text{S3 Reads with Combiners} = 2 \left(\frac{m}{p} + \frac{n}{f} \right)$$

m: number of producers

n: number of consumers

p: fraction of partitions each combiner reads

f: fraction of files each combiner reads

If accessing S3 (cloud storage) costs so much, what about adding network connections to CFs?

- Shuffling: add combiners to mitigate S3 throttling
- Direct write-back: avoid an extra read/write to S3 by sending results directly to VMs
- Chain Join: avoid S3 by sequentially processing joins in the same CF group
- Direct Aggregation: combine the first stage with a previous operation if possible to avoid S3

Pixels-Turbo seems to be very complicated.
What are the benefits of all the optimizations it is adding?

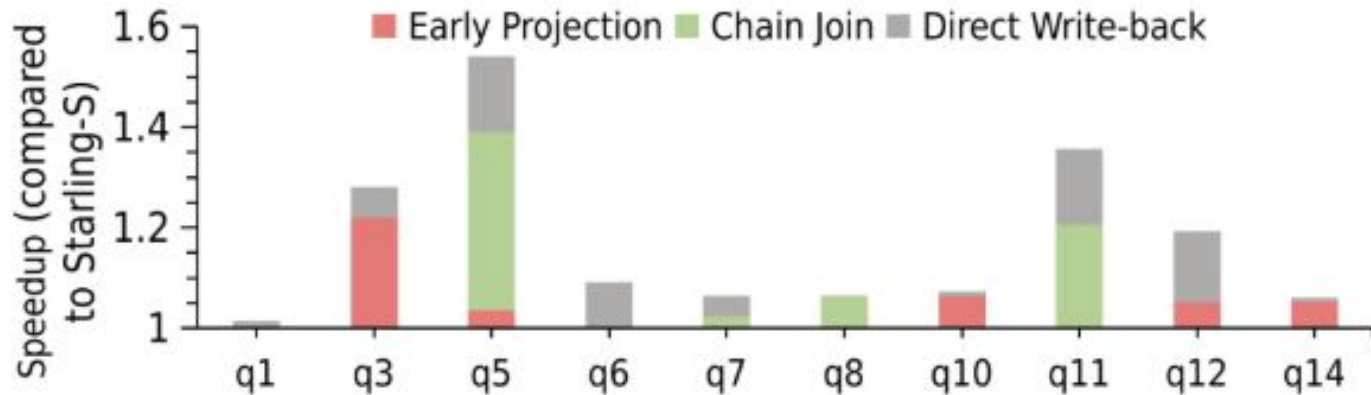


Fig. 11. Speedup of the optimizations on TPC-H 1TB.

Pixels-Turbo seems to be stuck to specific services, is it worth adopting?



trino



AWS Lambda



Amazon
EC2



Amazon S3

Pixels: An Efficient Column Store for
Cloud Data Lakes

Treat Pixels-Turbo as a framework.

