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

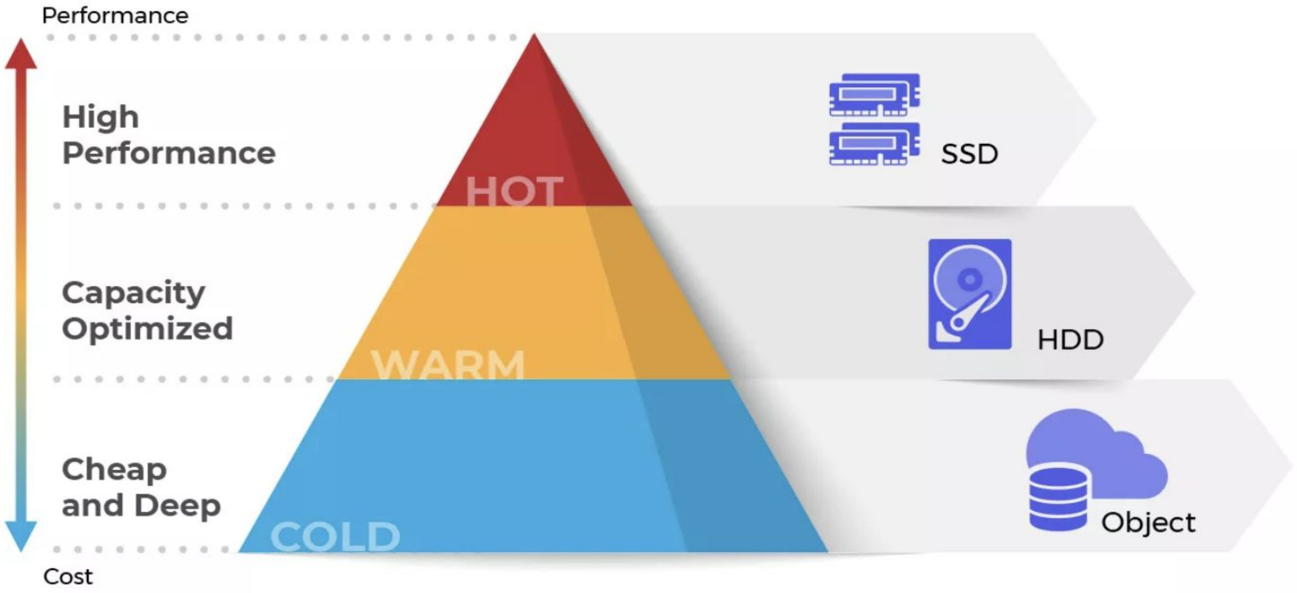
Ingo Müller , Renato Marroquín, Gustavo Alonso

Presenters : Harshitha, Greeshma, Vishwas, Shivangi

Hot Data & Cold Data

What is Hot Data?

What is Cold Data?



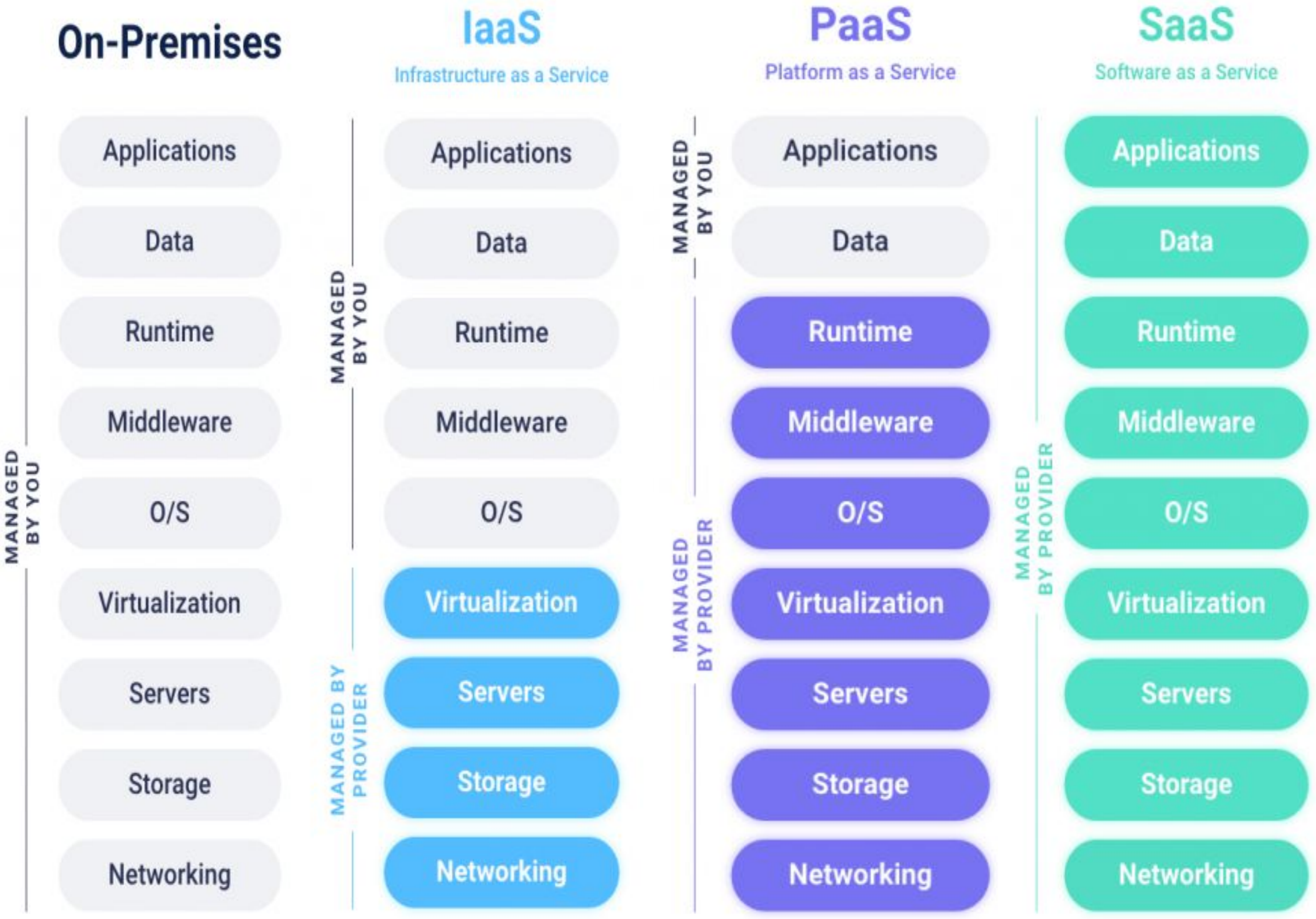
Cloud Computing

Google Cloud

AWS

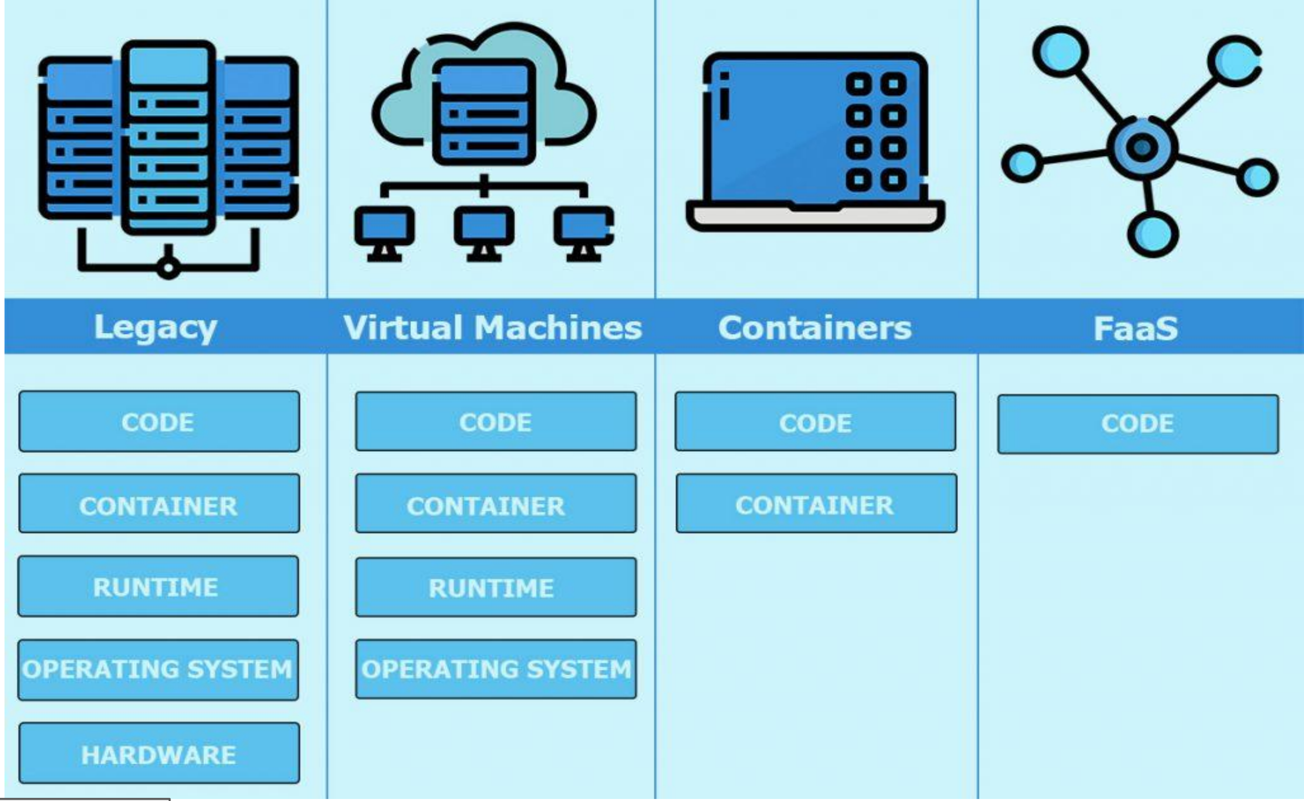
Azure

OPENSHIFT



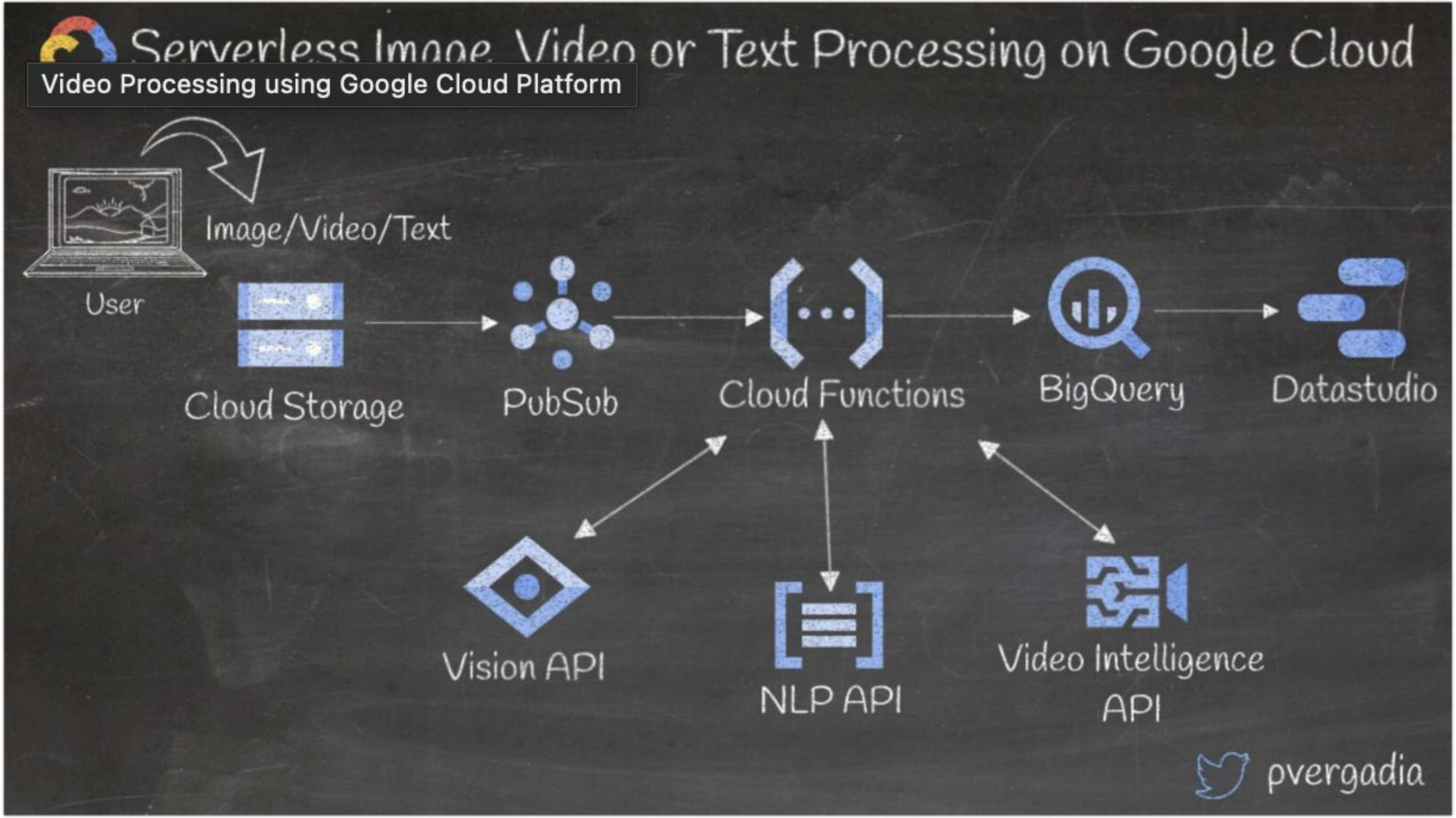
FaaS

FaaS



write and update a piece of code on the fly

executed in response to certain events like just a click

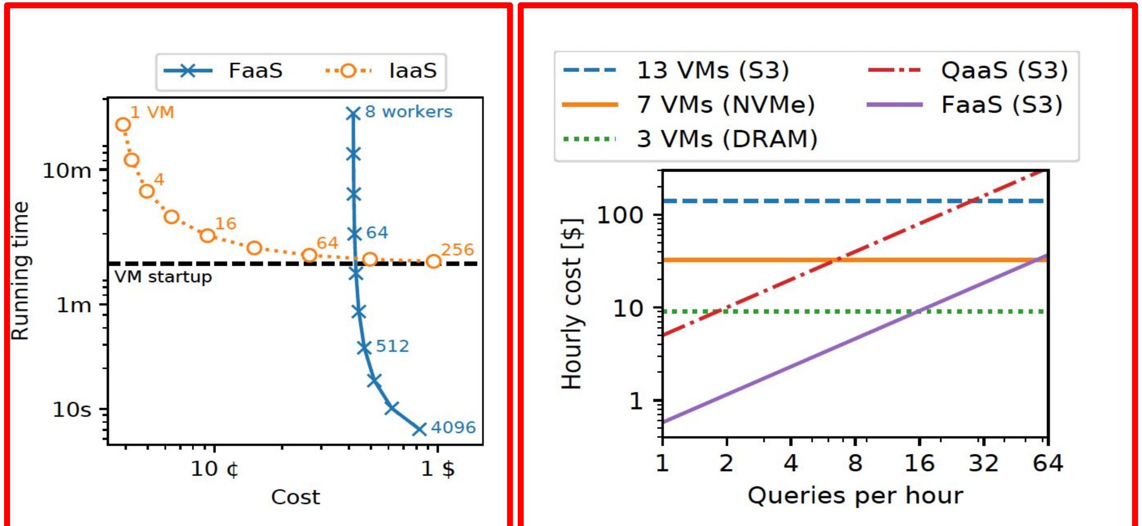

Serverless Image, Video or Text Processing on Google Cloud
Video Processing using Google Cloud Platform
Image/Video/Text
User
Cloud Storage
PubSub
Cloud Functions
BigQuery
Datastudio
Vision API
NLP API
Video Intelligence API
pvergadia

Why was FaaS chosen over other Cloud Services

Researchers ran few experiments for this using **VM** and **Workers**

IaaS vs FaaS

- Lack of control over scheduling of functions.
- Communication between function invocation.



(a) Job-scoped resources.

(b) Always-on resources.

Figure 1: Comparison of cloud architectures.

What are the limitations of FaaS?

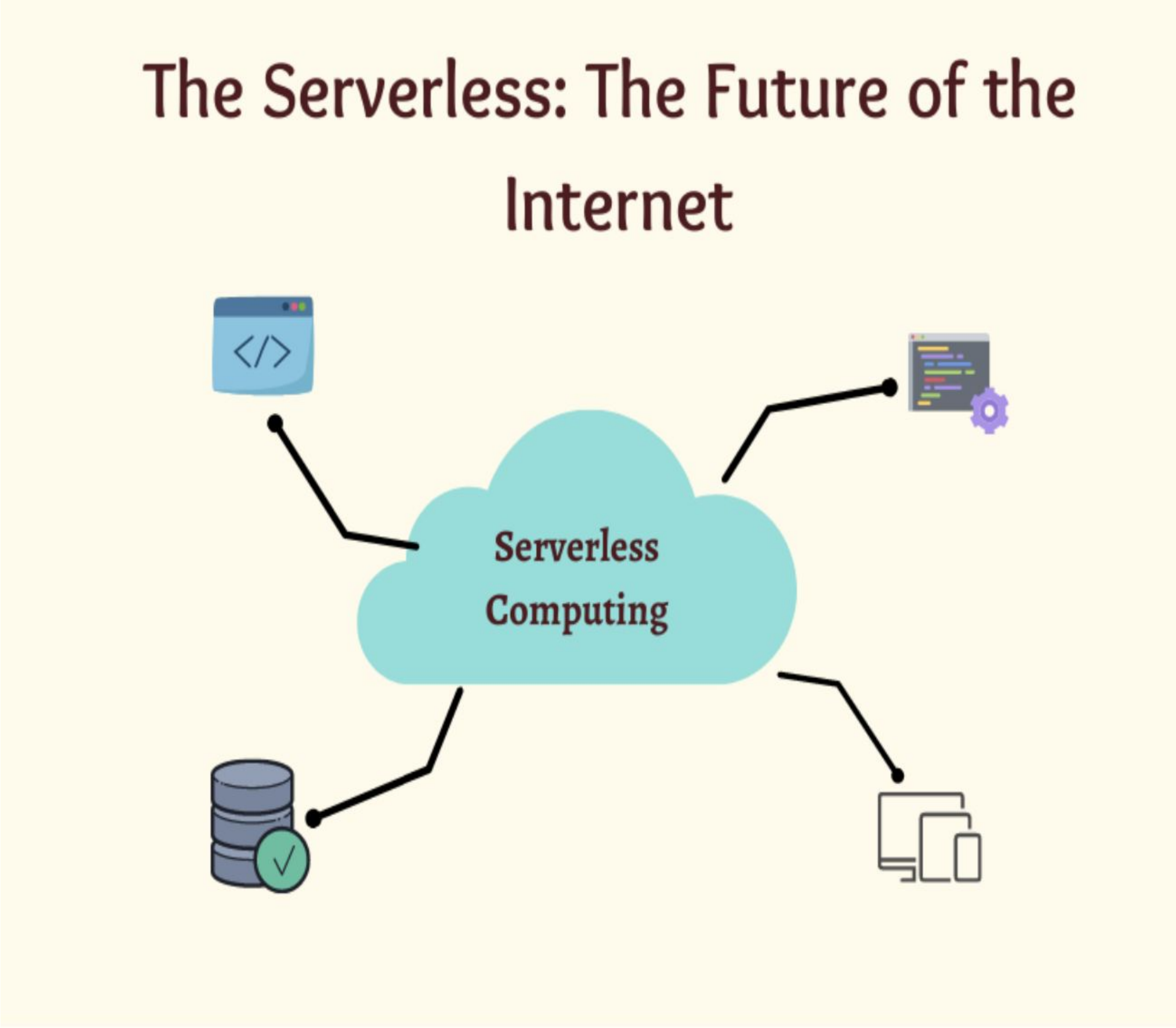
What is serverless?

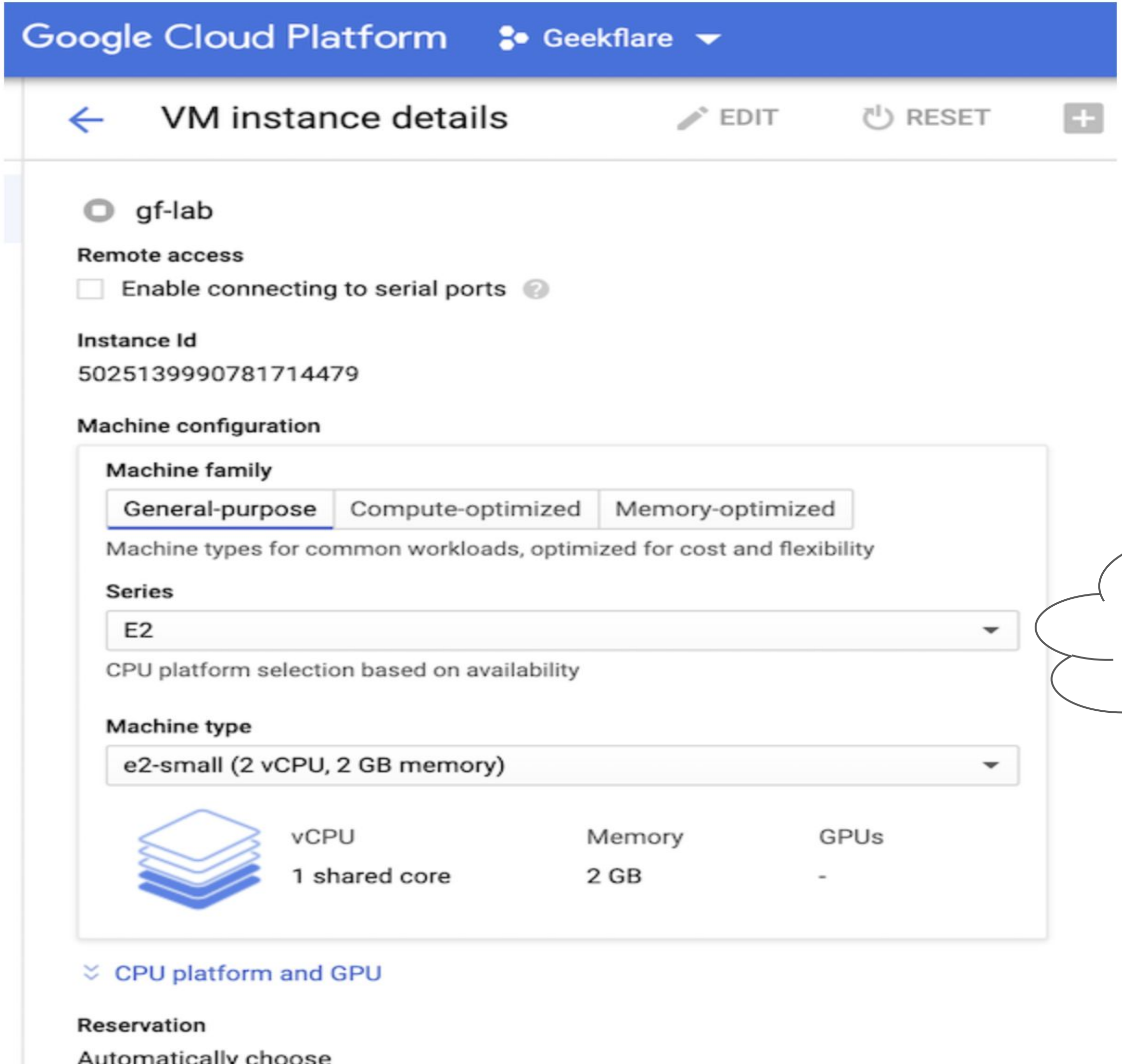
Ex: GCP, Amazon services, Azure, IBMwhisk

Is serverless really server less?

distribution of data among different nodes

Distributed data processing?

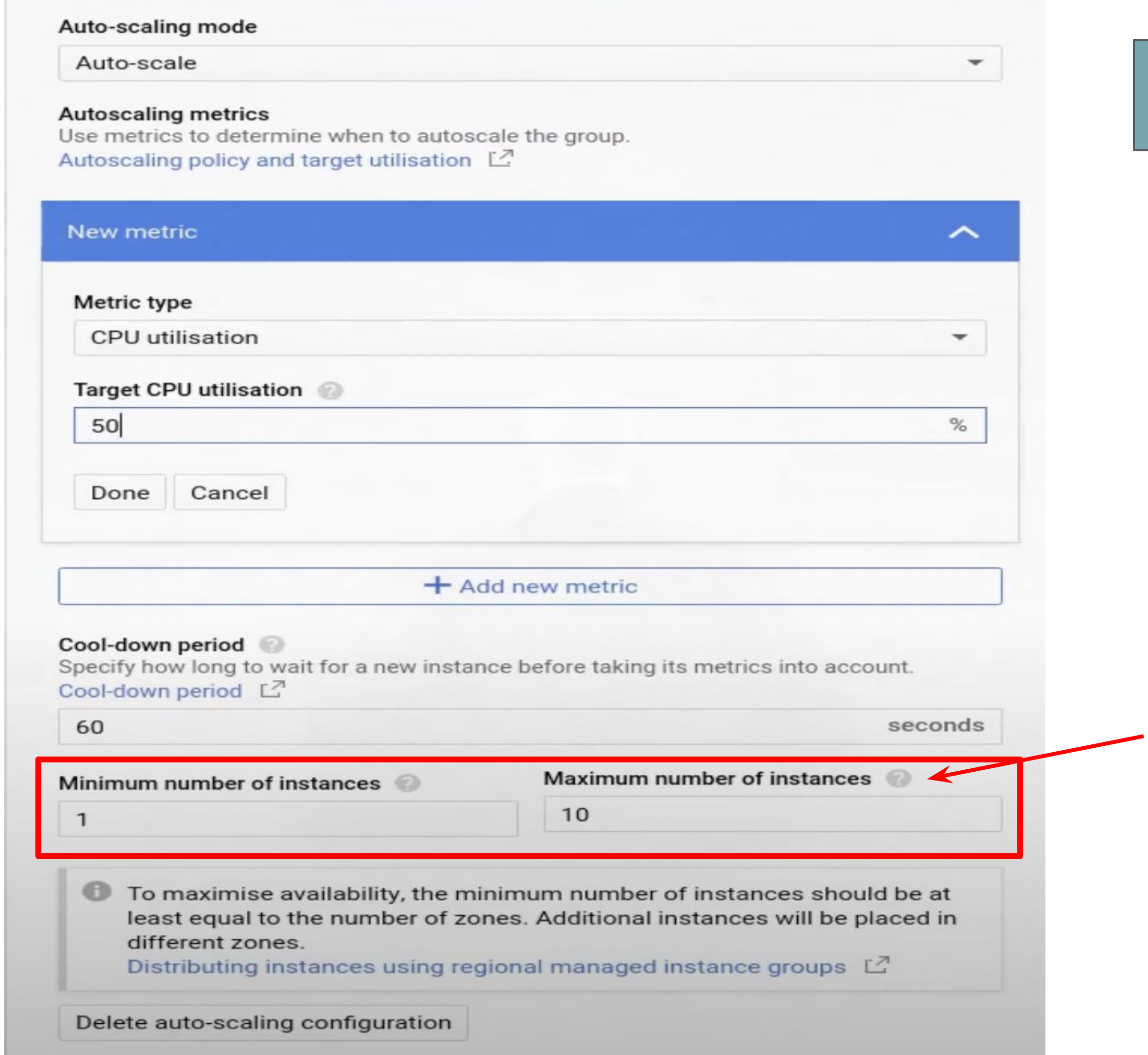




Operational simplicity?

Keeping the infrastructure as simple as possible

Ultimate Elasticity?



LAMBADA

It is a **data analytics system** on top of FaaS.

What is LAMBADA?

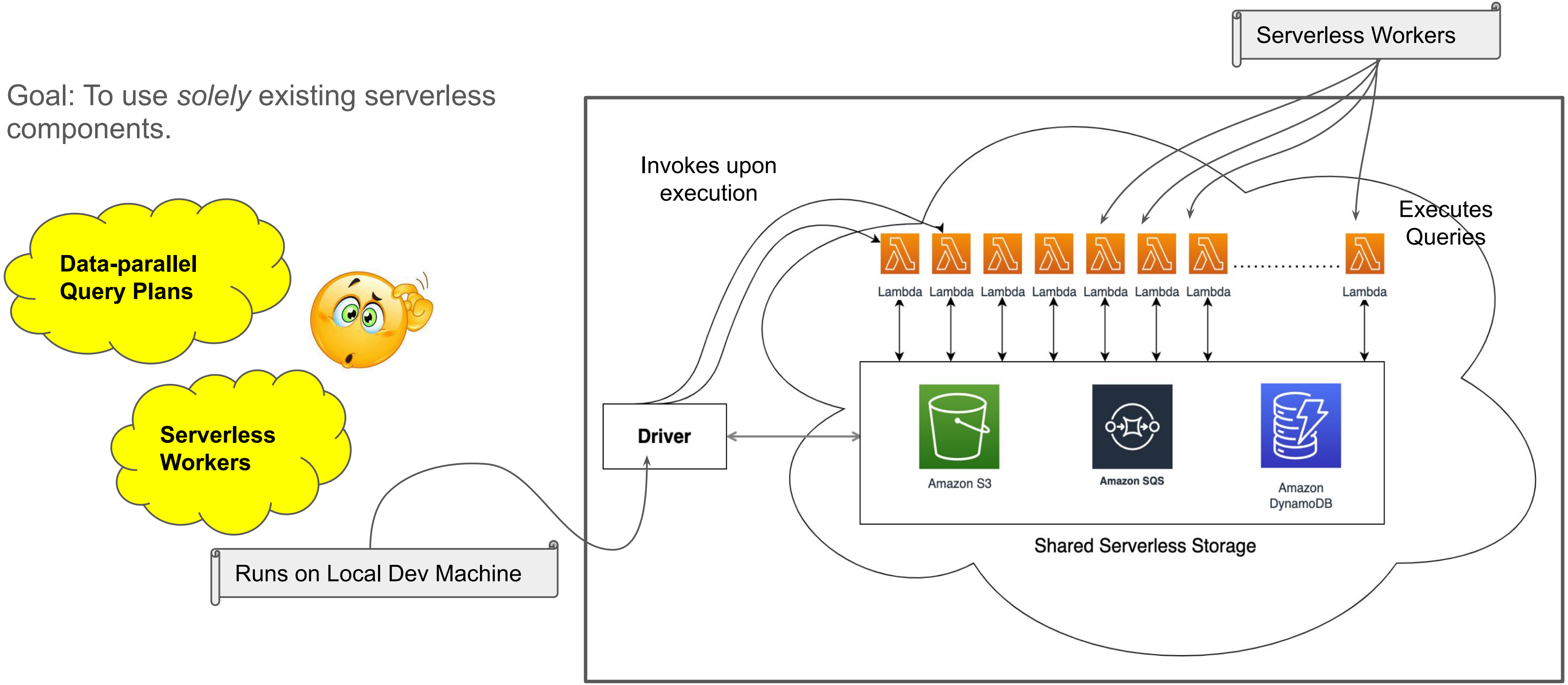
Uses **only** serverless components to overcome limitations of FaaS.

Answers **ad-hoc queries** on **Cold Data** in interactive query latency!

2x Cheaper and **1x Faster** than normal QaaS

Architecture

Goal: To use *solely* existing serverless components.



Challenges in Existing Serverless Paradigm

1. Balancing cost of reading time and performance — Cloud Storage Scan Operator

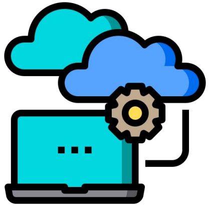
2. Delays in start up time of worker functions — Tree Based Invocation Strategy

3. Efficient data transfer between workers and driver — Exchange Operator

What is the challenge with reading the data?

How do we address this challenge?

What is Cloud Storage Scan Operator?



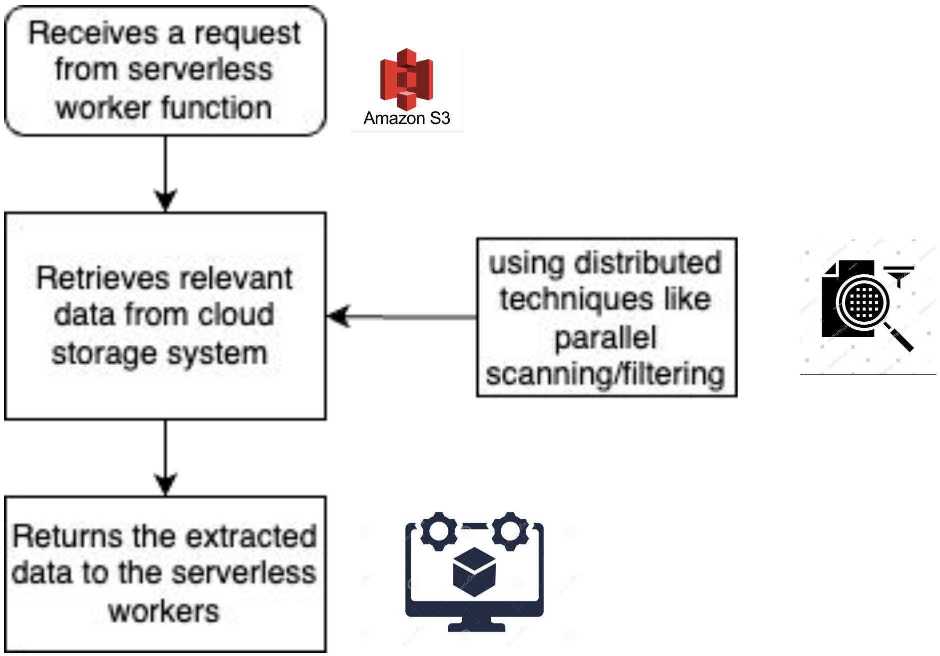
Cloud Storage Scan Operator

The cloud storage scan operator is a mechanism used to detect and prevent malicious content from being uploaded or stored in cloud storage systems. This operator is designed to scan files that are being uploaded to cloud storage and identify any potential security threats.

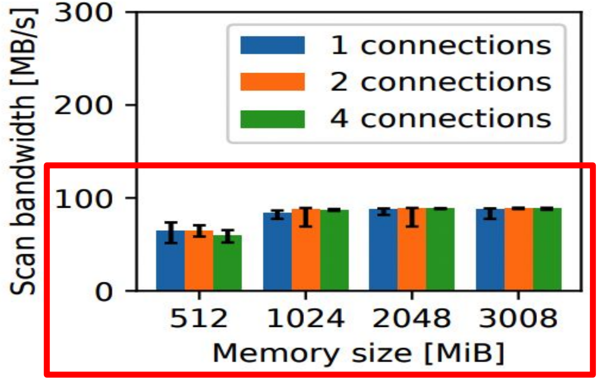
The paper examines the performance and cost of accessing S3 from serverless workers, and uses microbenchmarks to evaluate the download speeds of large and small files from S3 into serverless workers

However,the memory size of the workers has an influence on the network bandwidth because the cloud provider allocates CPU resources to each function that is proportional to its memory size.The Author suggests that using multiple concurrent connections can maximize performance for short-running scans and hide latencies with concurrent requests

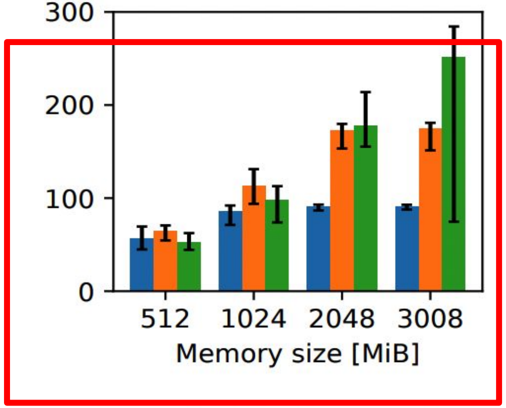
Cloud Storage Scan Operator



(i) Workflow of scan operator



(a) Large files (1 GB).



(b) Small files (100 MB).

Overall Analysis: Size of each request is directly proportional to cost of scan and inversely proportional to the number of requests made

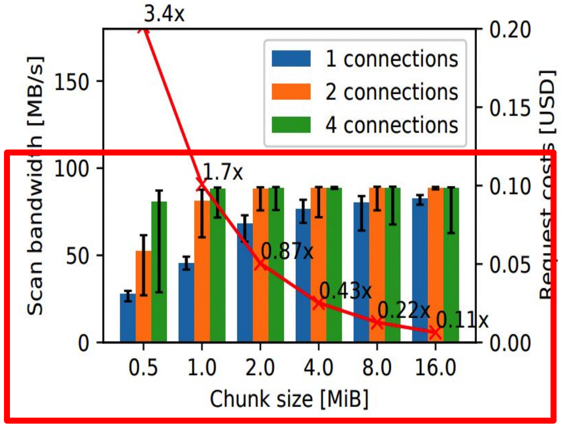


Figure 5: Impact of the chunk size on scan characteristics.

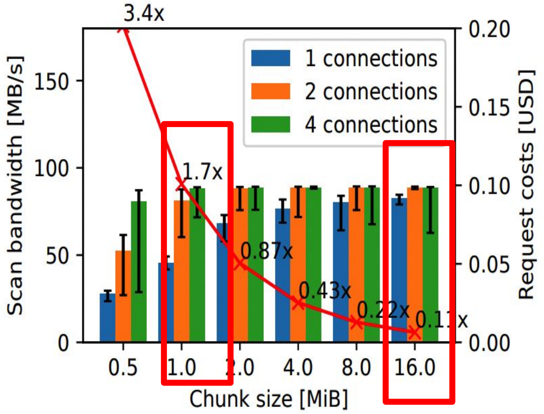


Figure 5: Impact of the chunk size on scan characteristics.

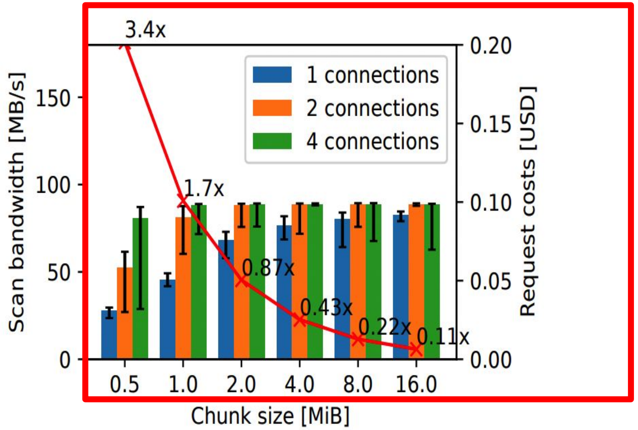


Figure 5: Impact of the chunk size on scan characteristics.

Overall Analysis:

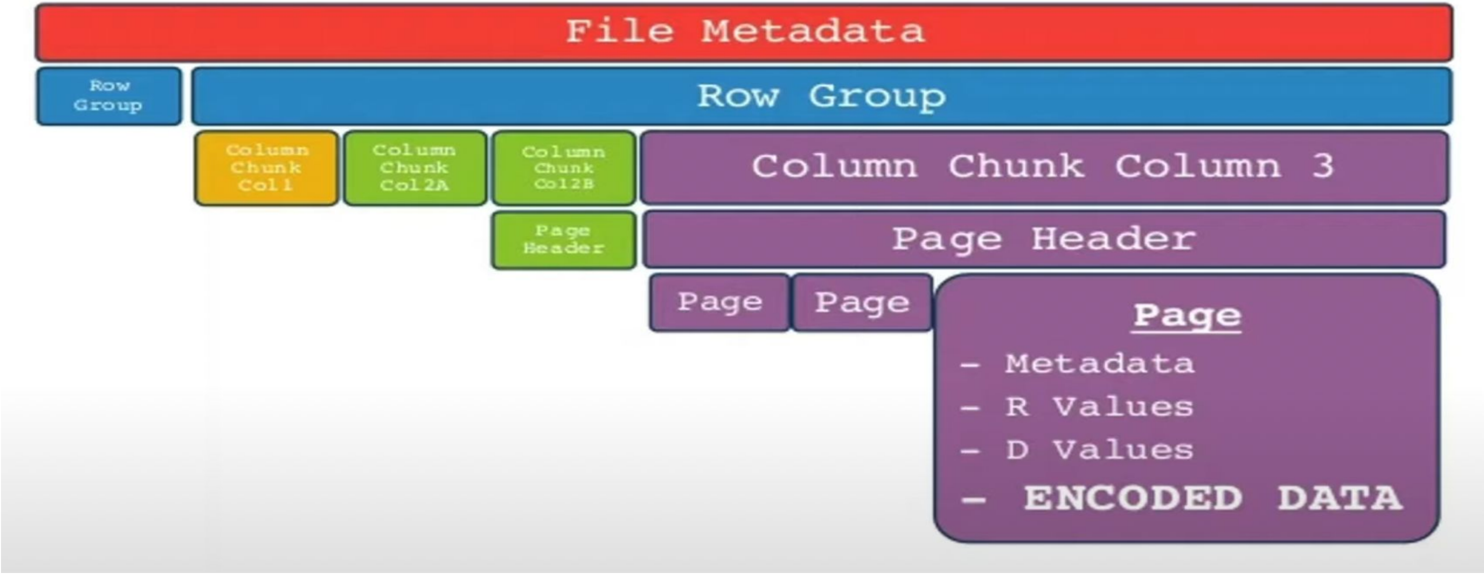
- Smaller reads increase cost but can be supported using several inflight methods.
- Larger chunk size can result in higher throughput but might result in higher cost if few requests are made to S3.

What are different kinds of file formats?

- CSV
- Parquet

Will the Cloud scan operator work on these?

Parquet File format



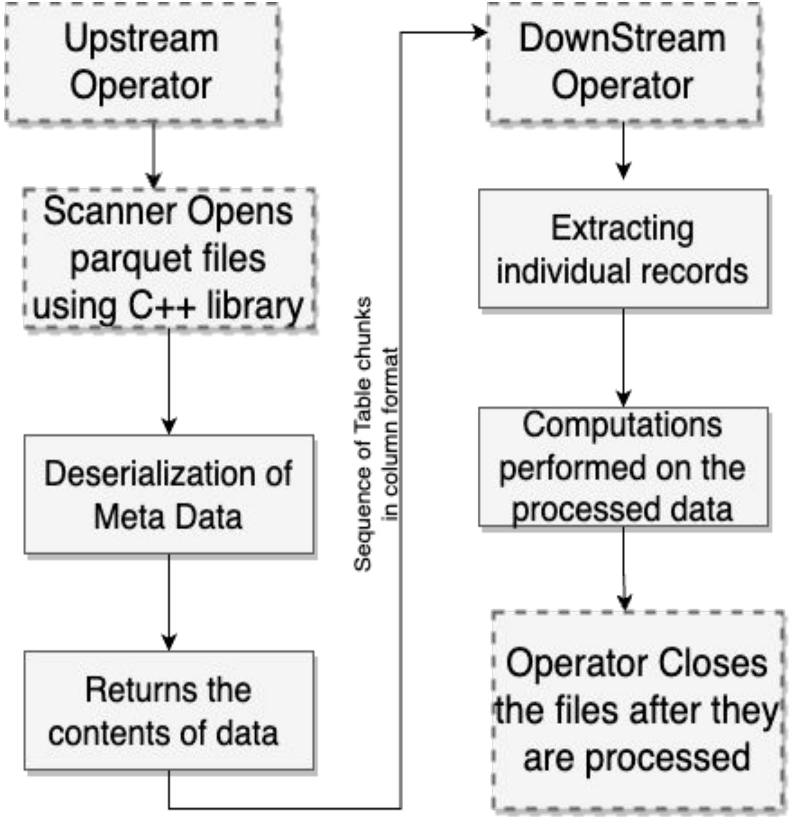
Compression methods

Category	LZO	GZIP	Snappy
Compression size	Medium	Smallest	Medium
Compression speed	Fast	Slow	Fastest (> LZO)
Decompression speed	Fastest (> Snappy)	Slow	Fast
Frequent of data usage	More frequent (hot)	Less frequent (cold)	More frequent (hot)
Splitable	Yes	No	Yes
Best For	General usage	Long term storage	General usage (> LZO)

CSV vs Parquet

File Format	Query Time (sec)	Size (GB)
CSV	2892.3	437.46
Parquet: LZO	50.6	55.6
Parquet: Uncompressed	43.4	138.54
Parquet: GZIP	40.3	36.78
Parquet: Snappy	28.9	54.83

Parquet Scan Operator



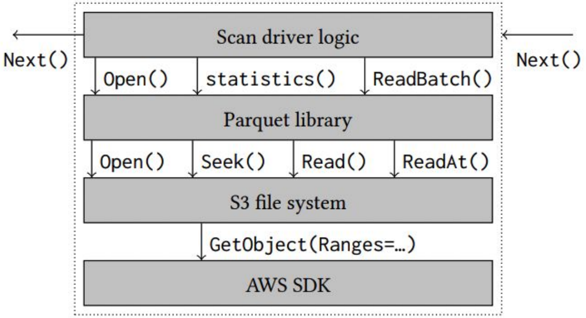
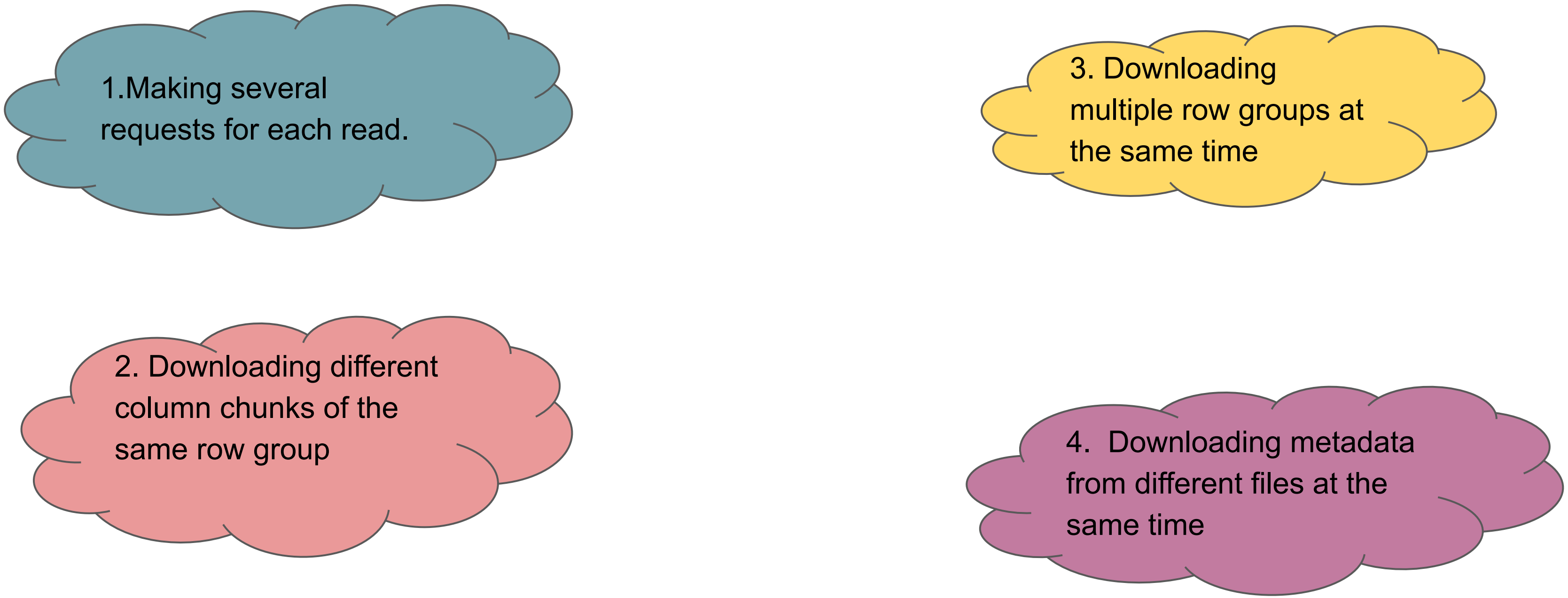


Figure 6: Components of the Parquet scan operator.

(ii) workflow of Parquet Scanner

Identify how we could achieve maximum bandwidth utilization on small files during Scan operation?


1.Making several requests for each read.
2. Downloading different column chunks of the same row group
3. Downloading multiple row groups at the same time
4. Downloading metadata from different files at the same time

Evaluations: Scan Operators

Dataset and Methodology

Lambada vs Google BigQuery vs Amazon Athena

Performance **Cost**

TPC-H Benchmark

Scale Factor(SF) of 1k is equal to **502 GiB dataset**

dbgen is **modified** to generate **only integers** as Lambada does not support strings

Scan Heavy Queries

Two Scan Heavy TPC-H Queries

Q1: Selects 98% of the Relation and uses seven attributes

Q6: Selects 2% of the Relation and uses four attributes

Query 1

```
SELECT
    l_returnflag,
    l_linestatus,
    sum(l_quantity) as sum_qty,
    sum(l_extendedprice) as sum_base_price,
    sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
    sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
    avg(l_quantity) as avg_qty,
    avg(l_extendedprice) as avg_price,
    avg(l_discount) as avg_disc,
    count(*) as count_order
FROM
    lineitem
WHERE
    l_shipdate <= date '1998-12-01' - interval '90' day
GROUP BY
    l_returnflag,
    l_linestatus
ORDER BY
    l_returnflag,
    l_linestatus;
```

Query 6

```
SELECT
    sum(l_extendedprice * l_discount) as revenue
FROM
    lineitem
WHERE
    l_shipdate >= date '1994-01-01'
    AND l_shipdate < date '1994-01-01' + interval '1' year
    AND l_discount between 0.06 - 0.01 AND 0.06 + 0.01
    AND l_quantity < 24;
```

TPC-H Queries

Query 1

```
SELECT
    l_returnflag,
    l_linestatus,
    sum(l_quantity) as sum_qty,
    sum(l_extendedprice) as sum_base_price,
    sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
    sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
    avg(l_quantity) as avg_qty,
    avg(l_extendedprice) as avg_price,
    avg(l_discount) as avg_disc,
    count(*) as count_order
FROM
    lineitem
WHERE
    l_shipdate <= date '1998-12-01' - interval '90' day
GROUP BY
    l_returnflag,
    l_linestatus
ORDER BY
    l_returnflag,
    l_linestatus;
```

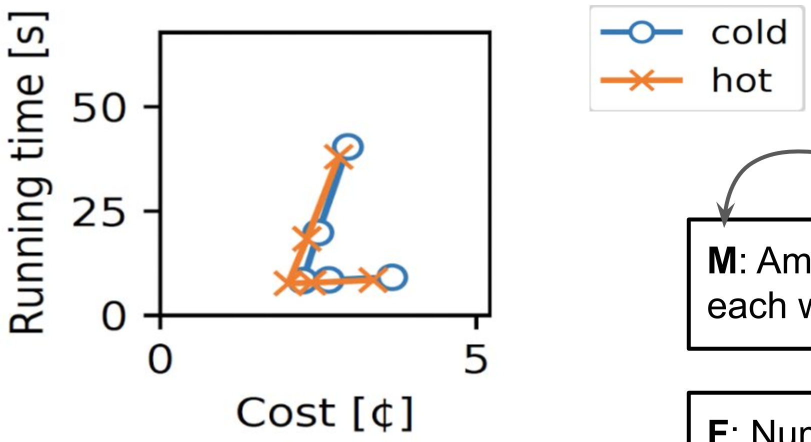
Query 6

```
SELECT
    sum(l_extendedprice * l_discount) as revenue
FROM
    lineitem
WHERE
    l_shipdate >= date '1994-01-01'
    AND l_shipdate < date '1994-01-01' + interval '1' year
    AND l_discount between 0.06 - 0.01 AND 0.06 + 0.01
    AND l_quantity < 24;
```

Q1: Selects 98% of the Relation and uses seven attributes

Q6: Selects 2% of the Relation and uses four attributes

Effect of Worker Configuration



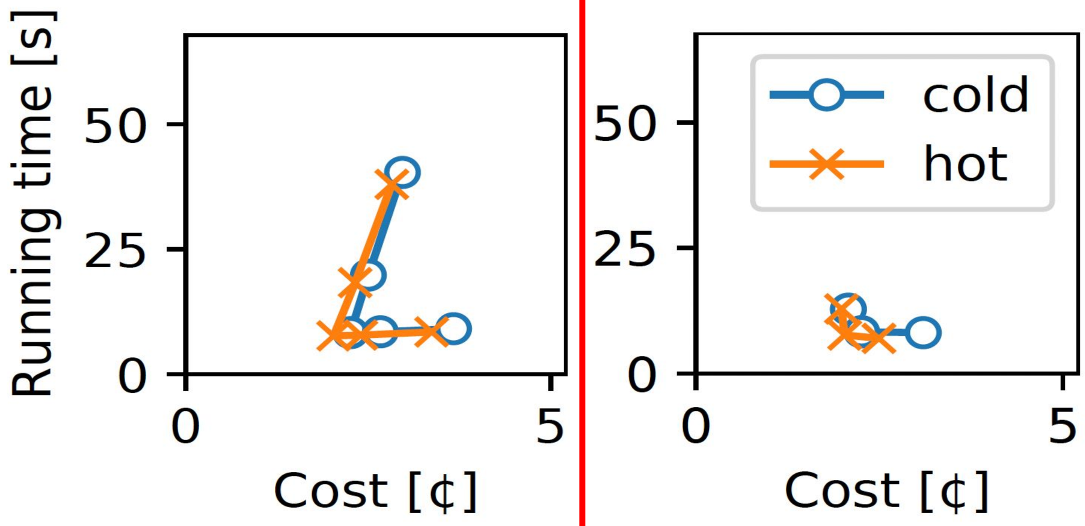
(a) $F = 1$, **varying** M.

This influences the number of CPU cycles the functions can use

M: Amount of main memory for each worker

F: Number of files that each worker can process

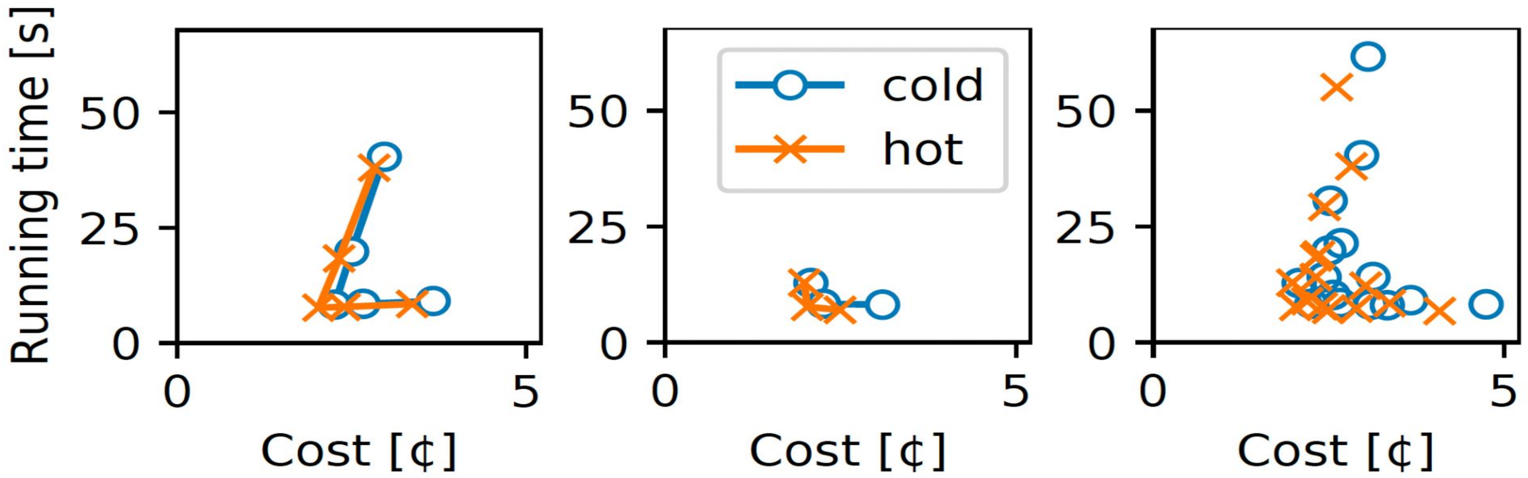
Effect of Worker Configuration



(a) $F = 1$, **varying** M.

(b) $M = 1792\,\mathrm{MiB}$, **varying** F.

Effect of Worker Configuration

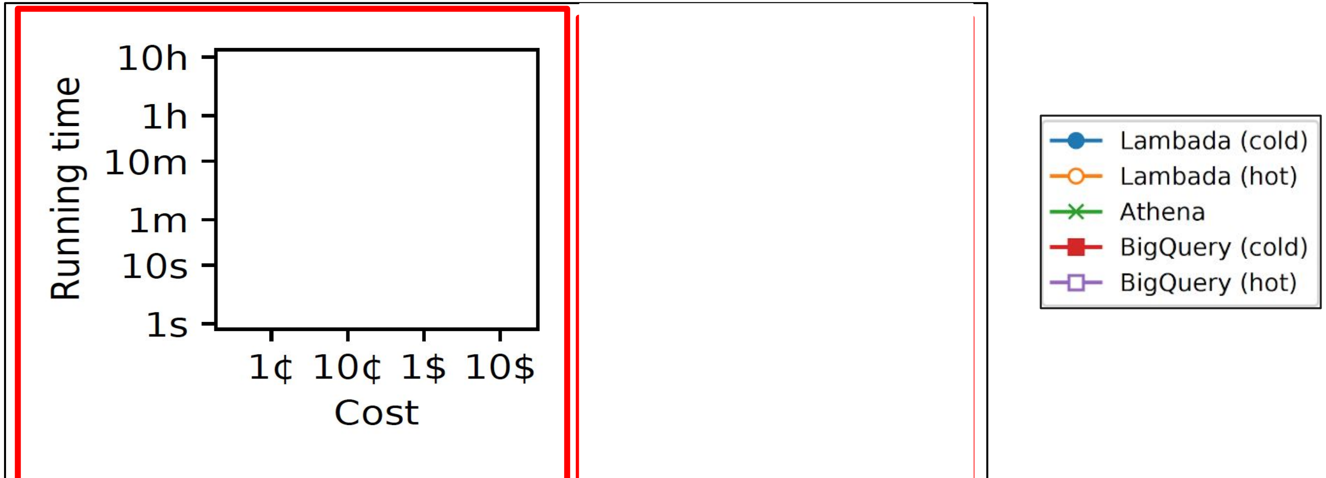


(a) $F = 1$, **varying** M. **(b)** $M = 1792$ MiB, **varying** F. **(c) Varying** M **and** F.

Comparison with Other QaaS Systems

VS Google BigQuery VS Amazon Athena

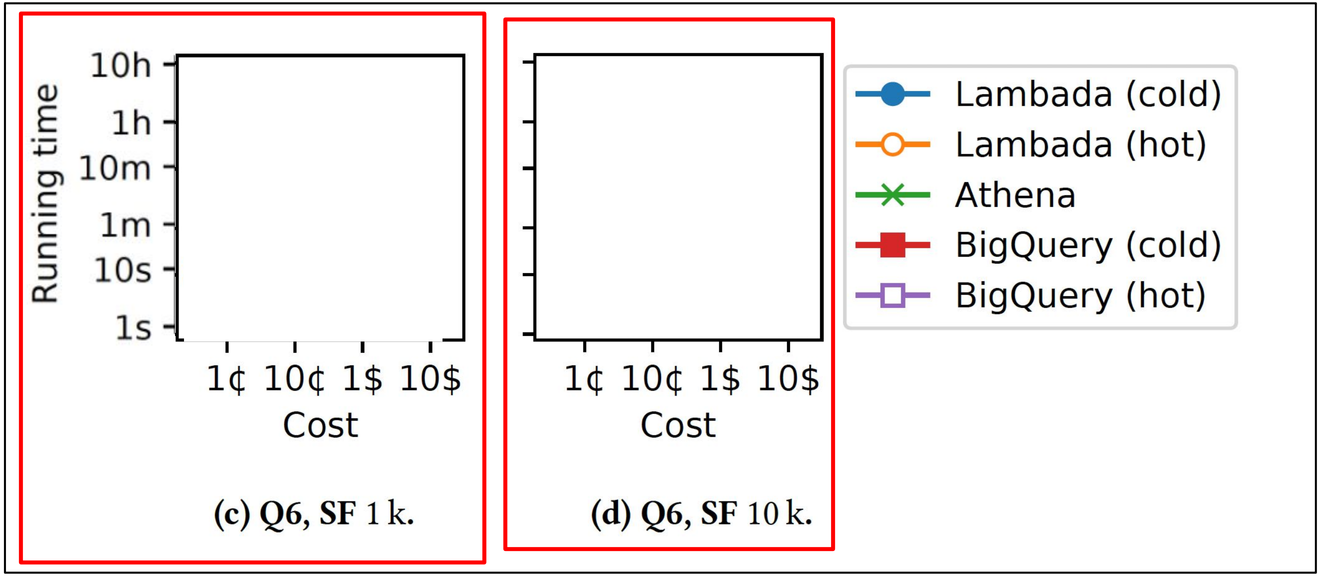
Comparison with Other QaaS Systems - Query 1



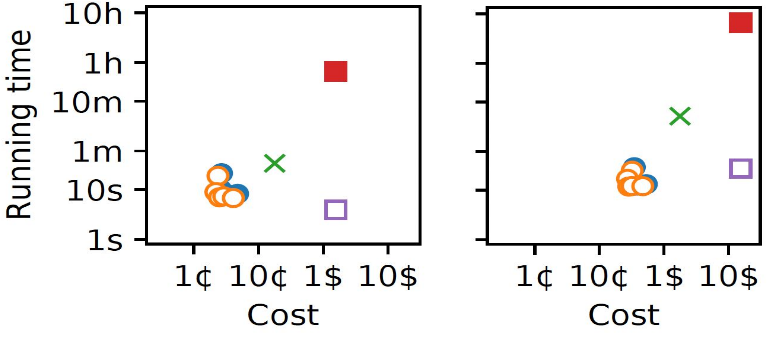
(a) Q1, SF 1 k.

Where would be the ideal place for the points to lie in this graph?

Comparison with Other QaaS Systems - Query 6



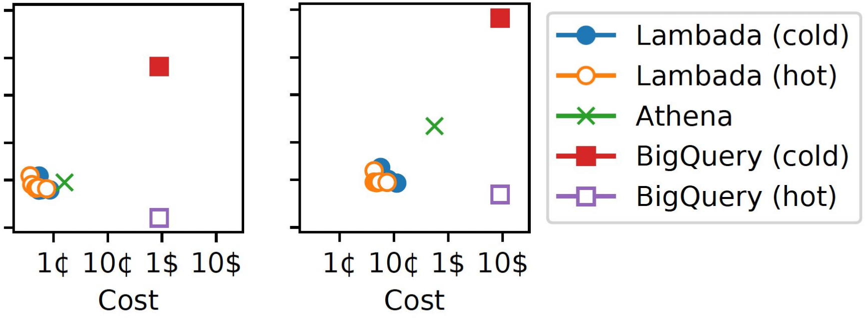
Comparison with Other QaaS Systems



(a) Q1, SF 1 k. **(b) Q1, SF 10 k.**

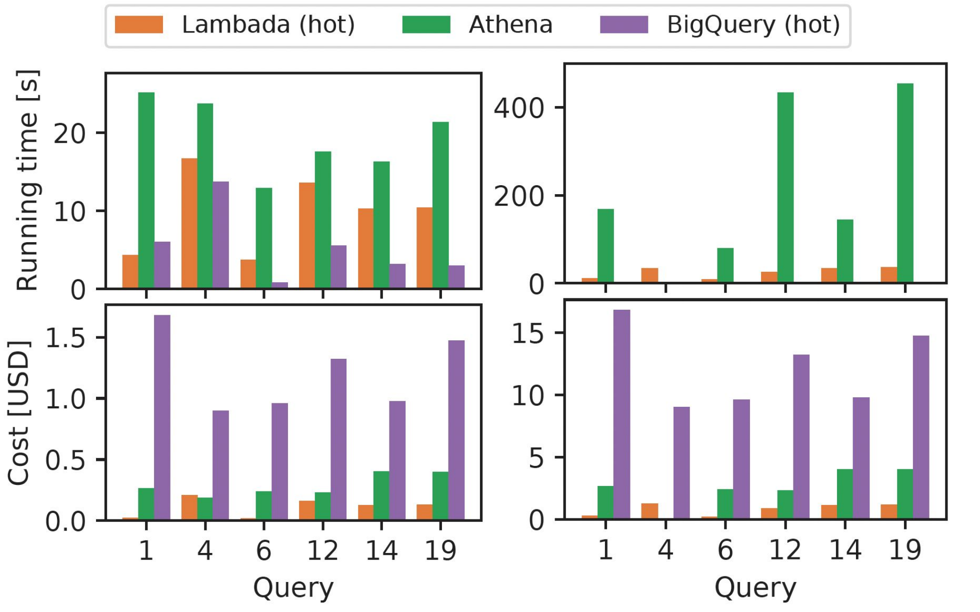
Compared to Amazon Athena, Lambada
4x Faster for Q1 (SF 1k)
On par for Q6 (SF 1k)
26x and **15x** respectively (SF 10k)

Lambada is cheaper than both systems



(c) Q6, SF 1 k. **(d) Q6, SF 10 k.**

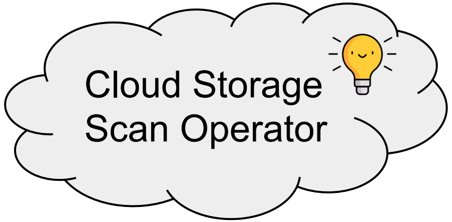
End to End Workloads



(a) SF 1 k, $W = 512$, $F = 2$. **(b) SF** 10 k, $W = 1280$, $F = 8$.

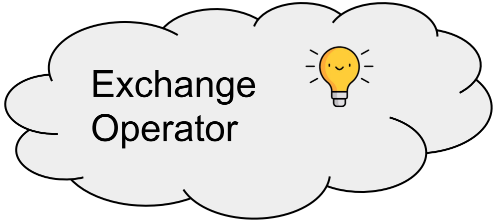
Challenges in Existing Serverless Paradigm

1. Balancing cost of reading time and performance

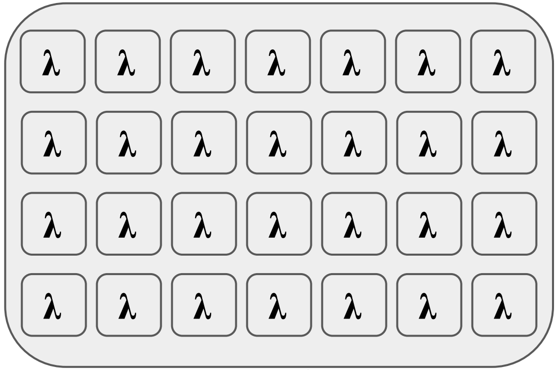


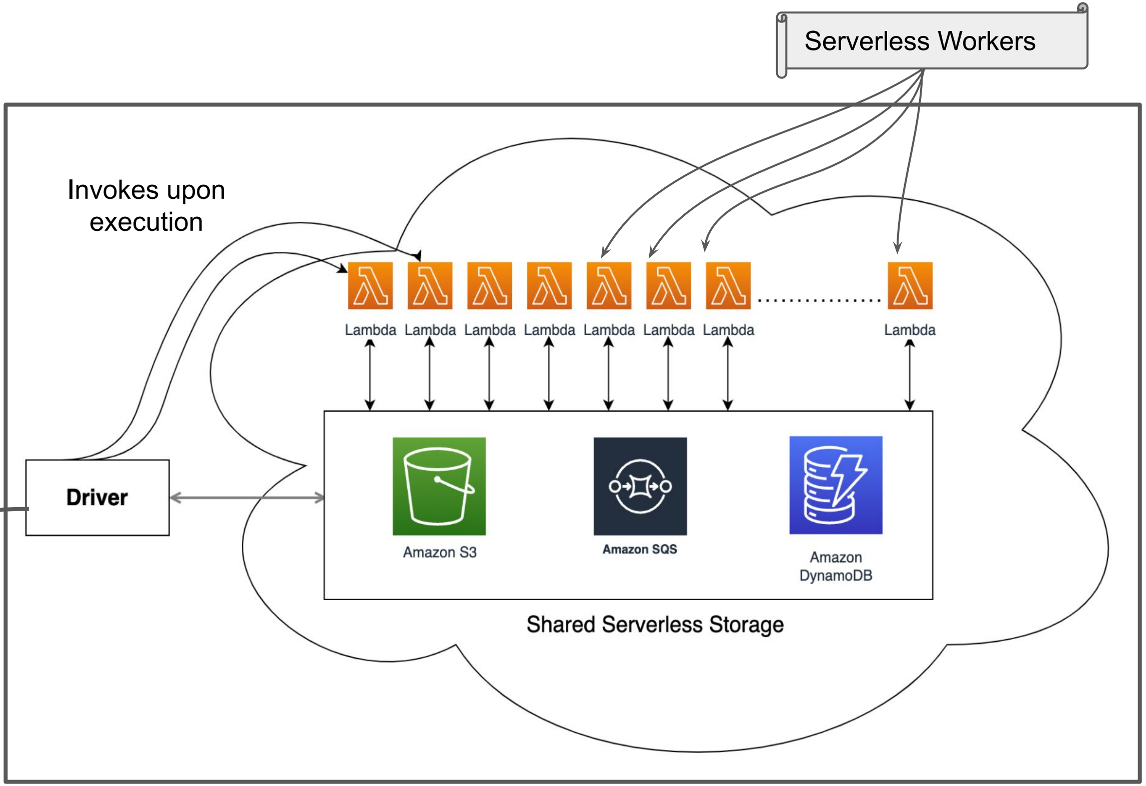
2. Delays in start up time of worker functions

3. Efficient data transfer between workers and driver



Sequential Invocation

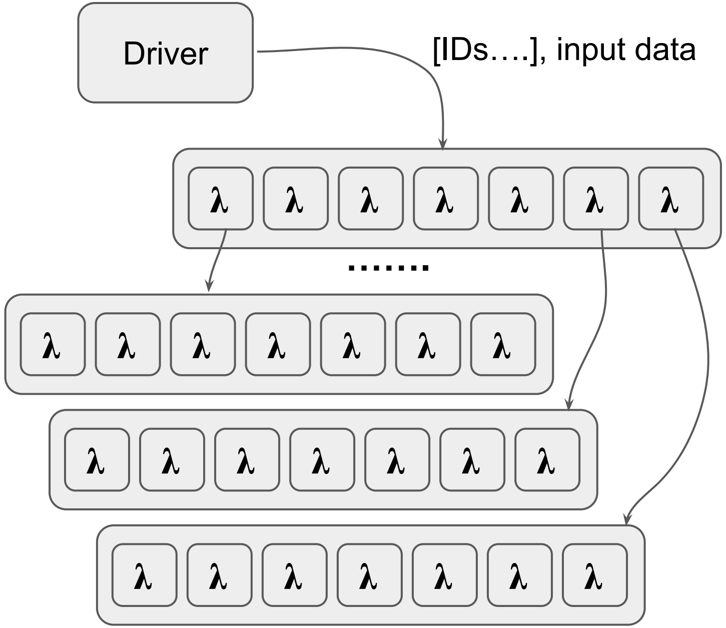




What are the issues that arises with this kind of invocation of the workers?

Lambada Two-level Invocation

How does this work?



Challenges in Existing Serverless Paradigm

1. Balancing cost of reading time and performance — Cloud Storage Scan Operator

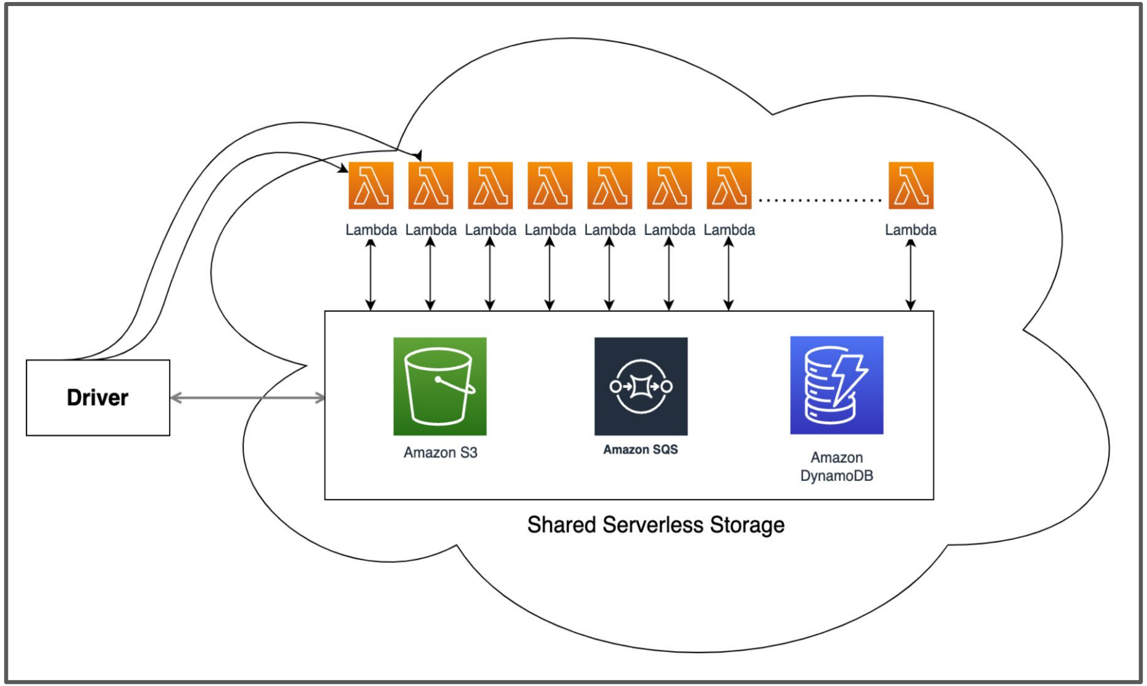
2. Delays in start up time of worker functions — Tree Based Invocation Strategy

3. Efficient data transfer between workers and driver — Exchange Operator

Exchange Operators

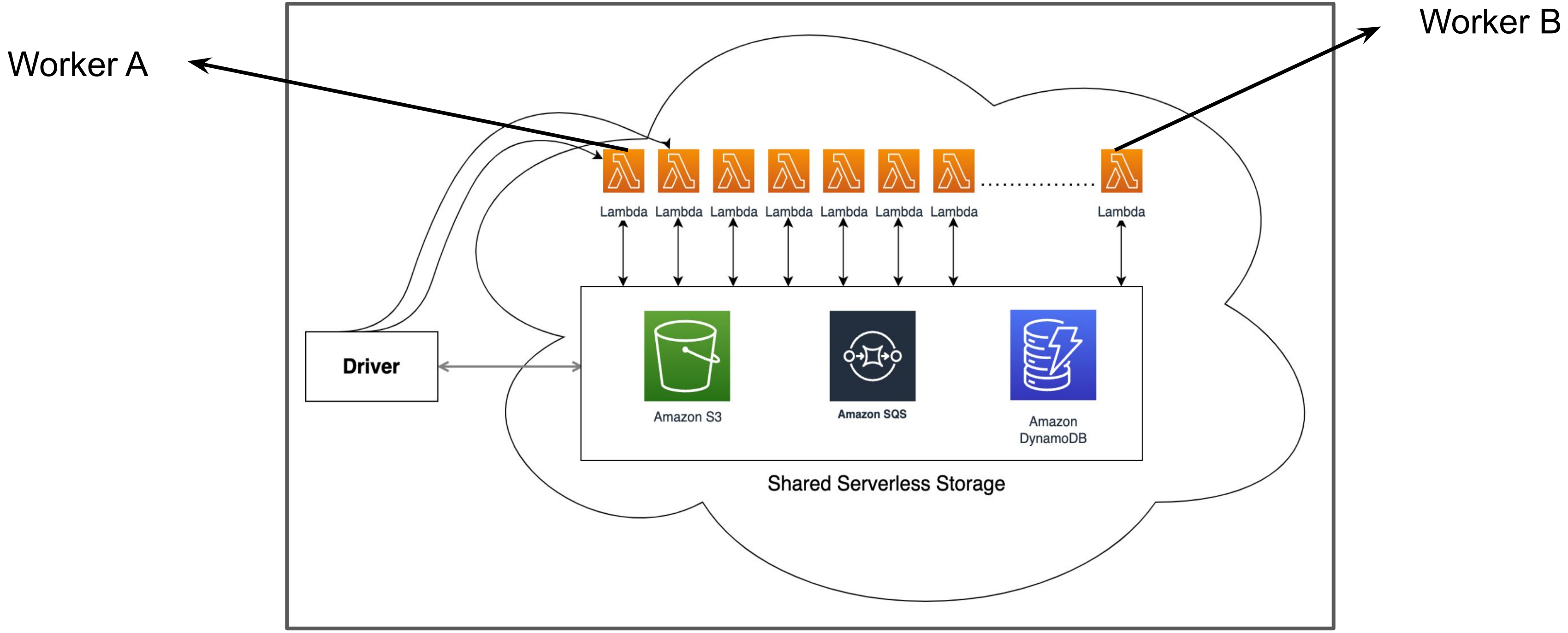
Used in exchanging data among workers and workers and drivers

Exchange Operator



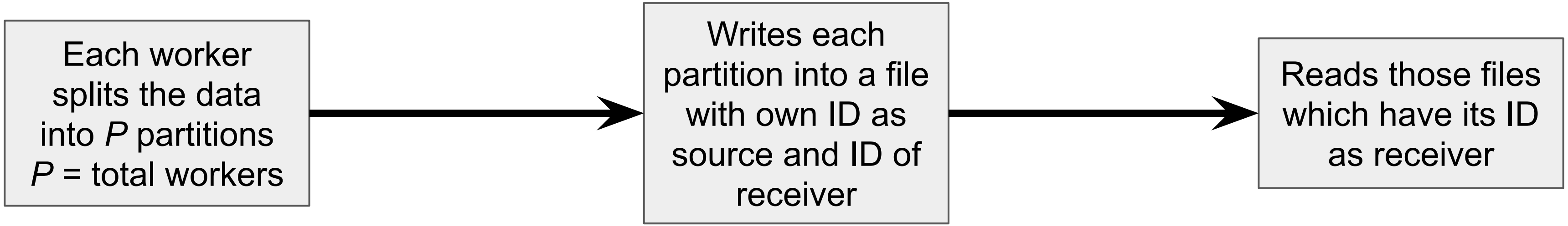
Function in FaaS that is used for parallel processing

Exchange Operator



What if worker A requires to exchange data with worker B for computation of some query?

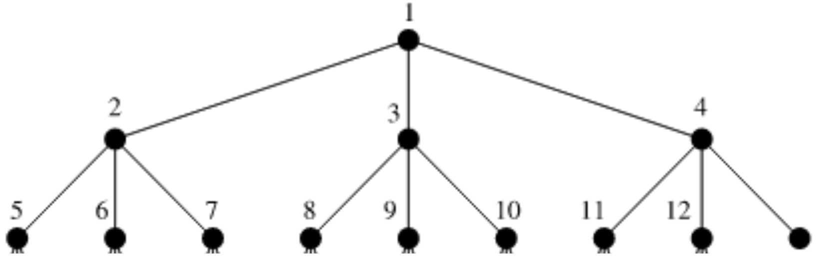
Basic Exchange Algorithm (Single level)



What could be the limitations of this approach?

Challenges with Basic Exchange Algorithm

What happens to the number of files if we increase the number of workers?



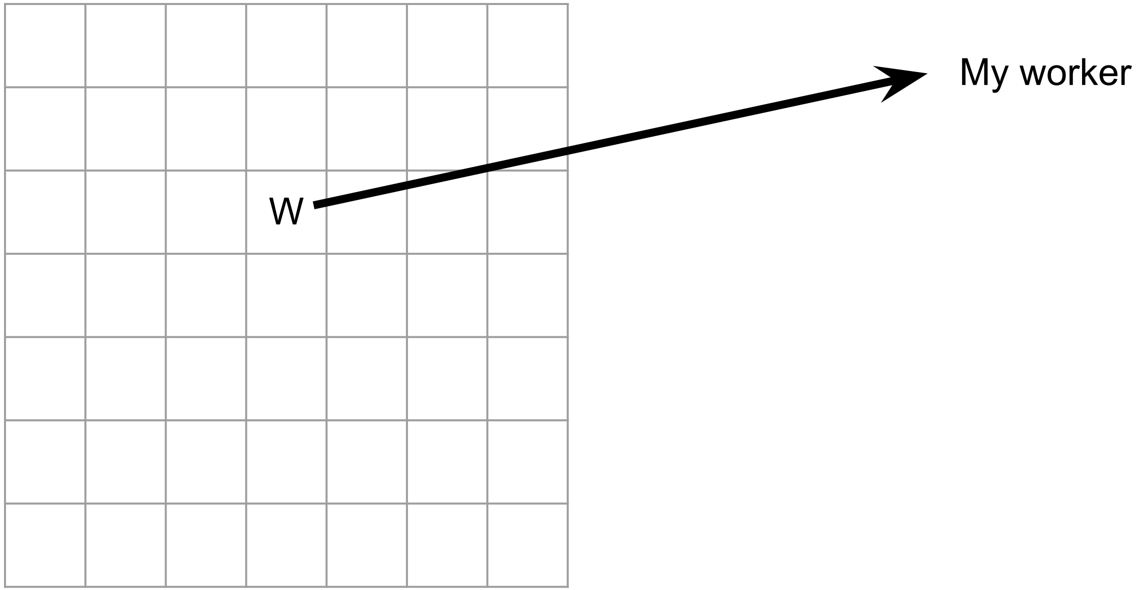
Optimizations that reduces number of requests

Exchange through multiple levels

Writing all partitions into a single file

Lambada Multi Level Exchange

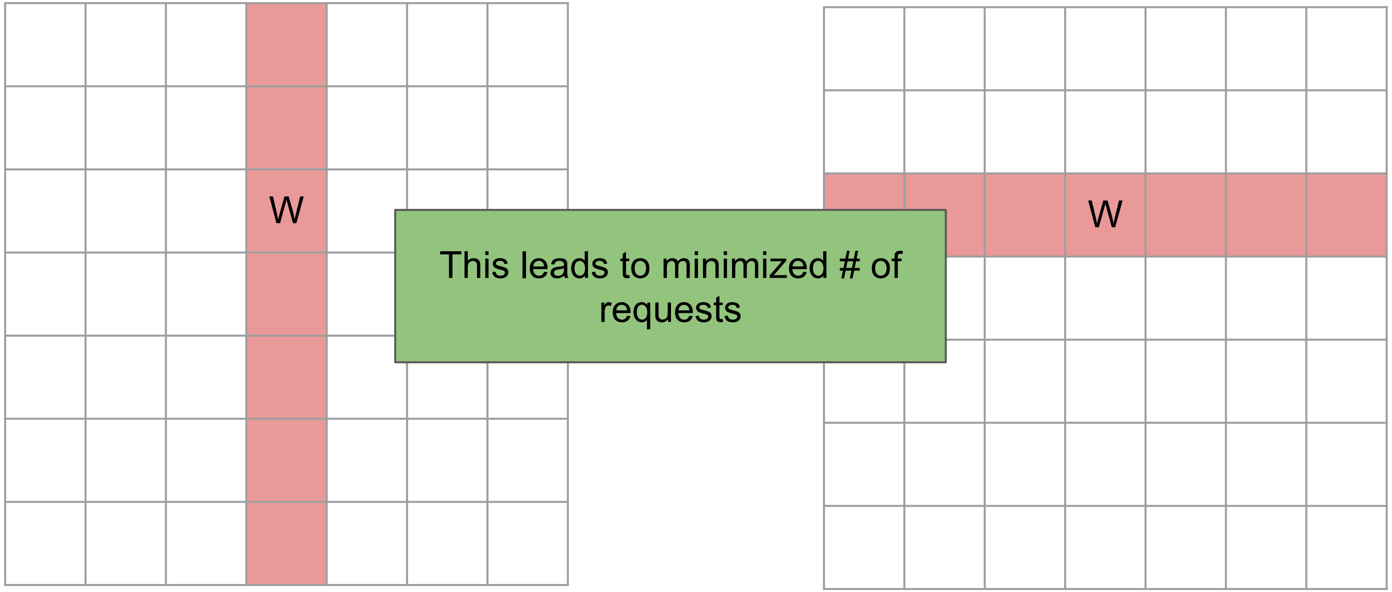
Workers are divided into subsets



Lambada Multi Level Exchange

Each worker performs two exchange

Vertical Exchange **Horizontal Exchange**



Optimizations that reduces number of requests

Exchange through multiple levels

Writing all partitions into a single file

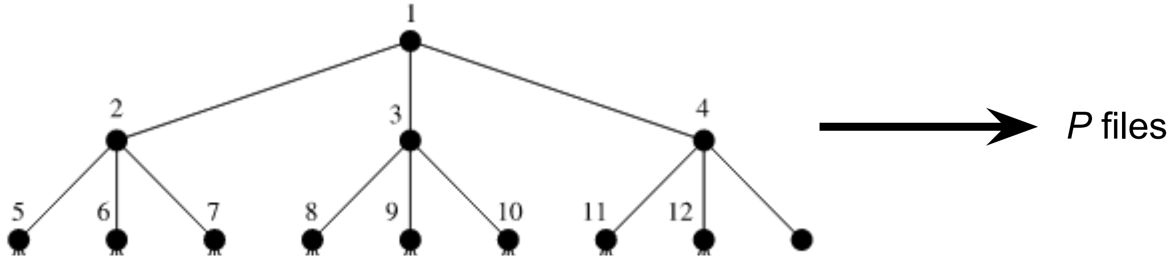
Lambada Write Combining

Writing all partitions into one file

Receiver only has to read a part of file

Row groups in parquet files

Lambada Write Combining



Amount of data that needs to be read is reduced → Significant Performance Improvement

Evaluations: Exchange Operators

Performance of Exchange Operators in Lambada

Experiment 1: 100GB Dataset

Implementation	Main Memory of Workers
Locus	1536 MB
Qubole	1536 MB
Pocket	3008 MB
Lambada	2048 MB

Results

Implementation	# of Workers	Running Time	Always On?
Pocket	250	98s	True
Locus	Dynamic	80s - 140s	False
Qubole	400	580s	True
Lambada	250	22s	False
	500	15s	
	1000	13s	

Results

Implementation	# of Workers	Running Time	Always On?
Pocket	250	98s	True
Locus	Dynamic	80s - 140s	False
Qubole	400	580s	True
Lambada	250	22s	False
	500	15s	
	1000	13s	

Lambada runs **5x faster** than Pocket

Results

Implementation	# of Workers	Running Time	Always On?
Pocket	250	98s	True
Locus	Dynamic	80s - 140s	False
Qubole	400	580s	True
Lambada	250	22s	False
	500	15s	
	1000	13s	

multiple buckets to partition the input data → sublinear amount of requests

Results

Implementation	# of Workers	Running Time	Always On?
Pocket	250	98s	True
Locus	Dynamic	80s - 140s	False
Qubole	400	580s	True
Lambada	250	22s	False
	500	15s	
	1000	13s	

Locus uses dynamic number of workers, but even with 250 workers Lambada is faster

Results

Implementation	# of Workers	Running Time	Always On?
Pocket	250	98s	True
Locus	Dynamic	80s - 140s	False
Qubole	400	580s	True
Lambada	250	22s	False
	500	15s	
	1000	13s	

Leads to wastage of resources

Evaluation: Performance of Exchange Operators in Lambada

Experiment 2: 1TB Dataset

Baseline:

- Compared with Locus (1TB)

	Locus	**Lambada**
Running Time	39s	56s
Cost	High	Low

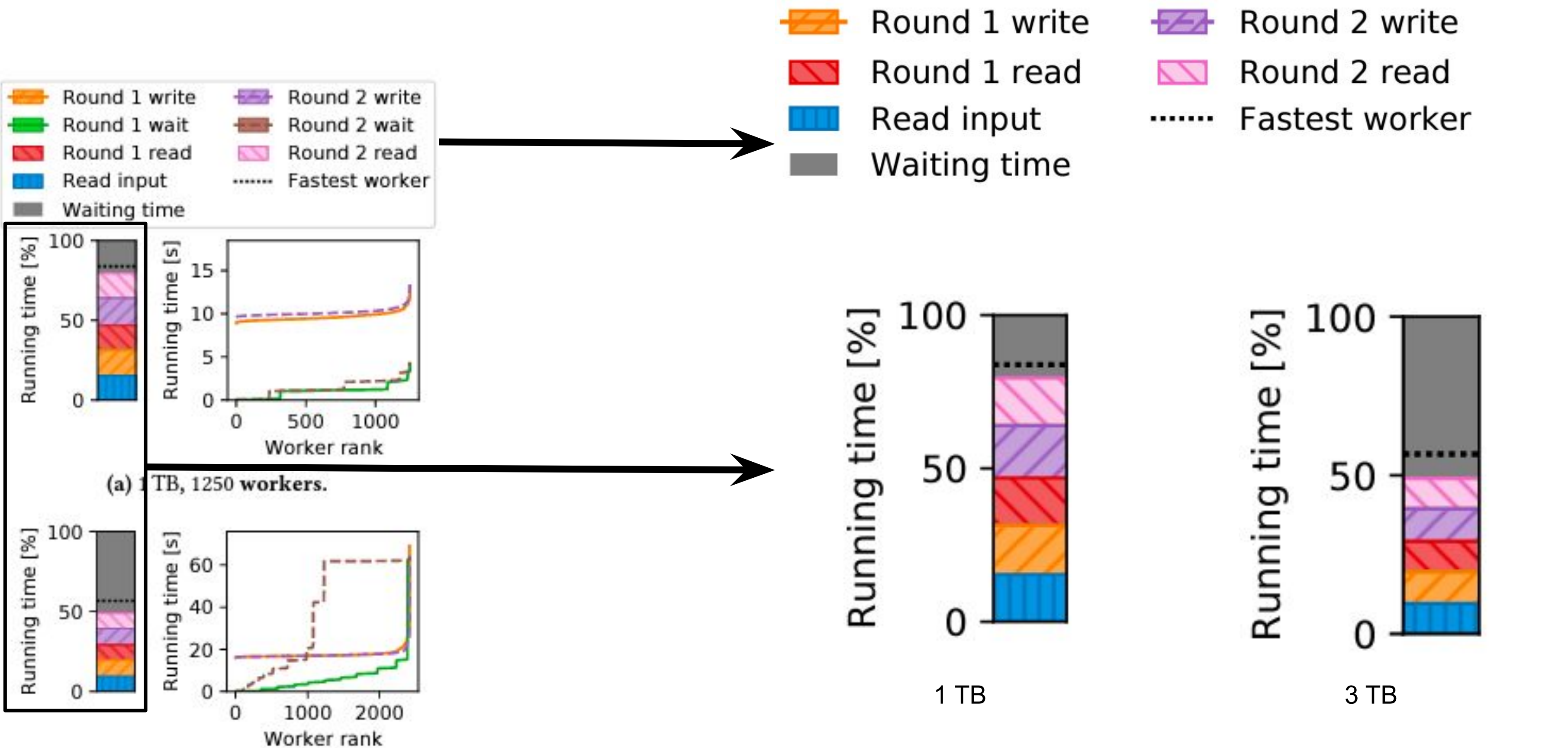
Locus uses VM-based fast storage for intermediate results which is expensive but fast

How do stragglers impact the performance of Lambada?

Wait a minute!
Who is a straggler?

Straggler: a worker that takes significantly longer than other workers to complete its tasks.

Fastest running time of each phase for any worker as a fraction of end to end latency



(b) 3 TB, 2500 workers.

Sum of Running time of all phases = Lower bound on Running Time

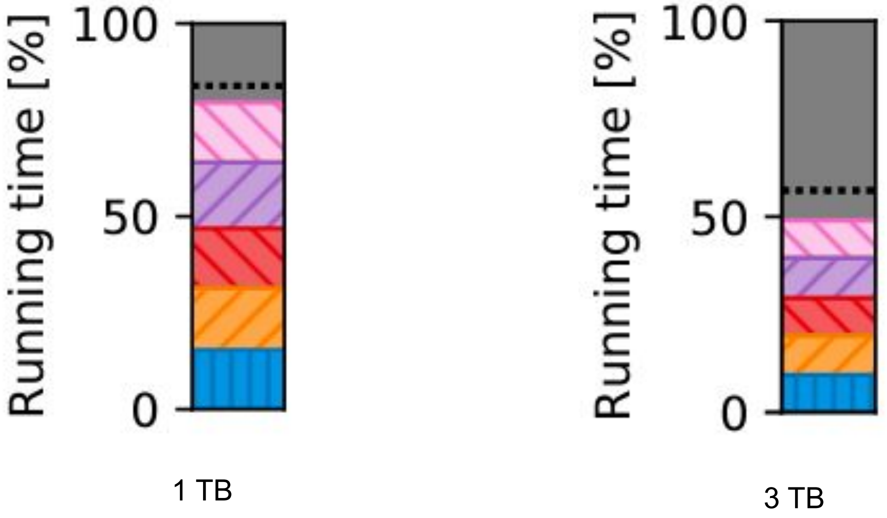
1 TB Workload

Fastest worker takes around 85% of the total time

3 TB Workload

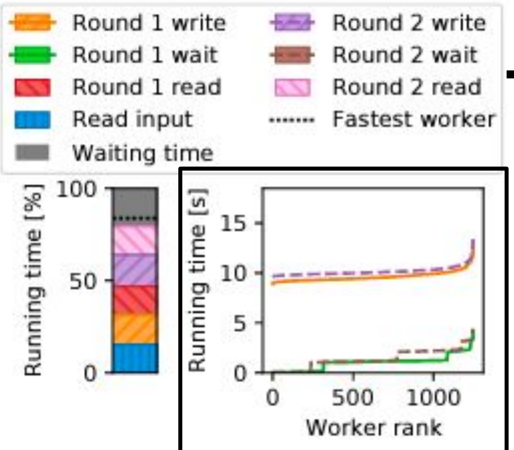
More than 50% of the total run time is due to stragglers

Fastest running time of each phase for any worker as a fraction of end to end latency

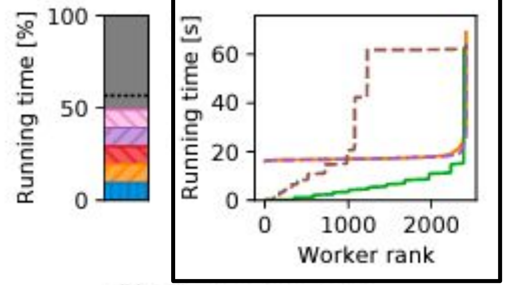


Stragglers impact 3TB workload more than 1TB

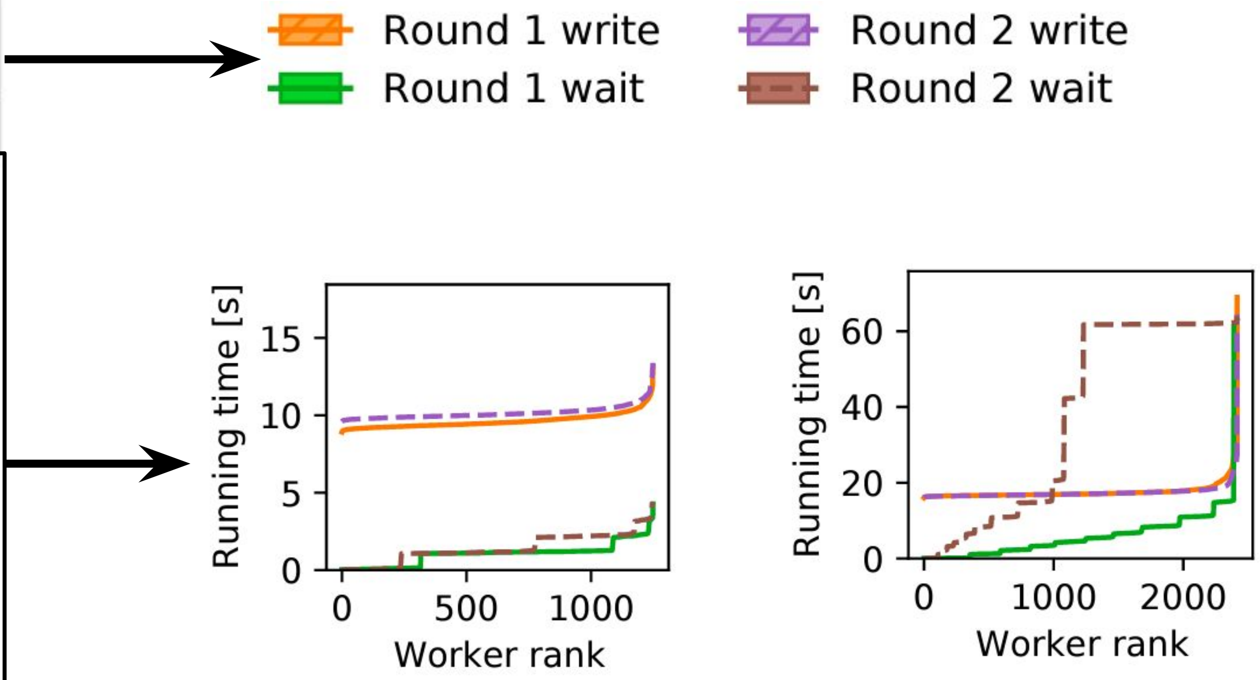
Distribution of the running time of each worker ordered by increasing running time



(a) 1 TB, 1250 workers.



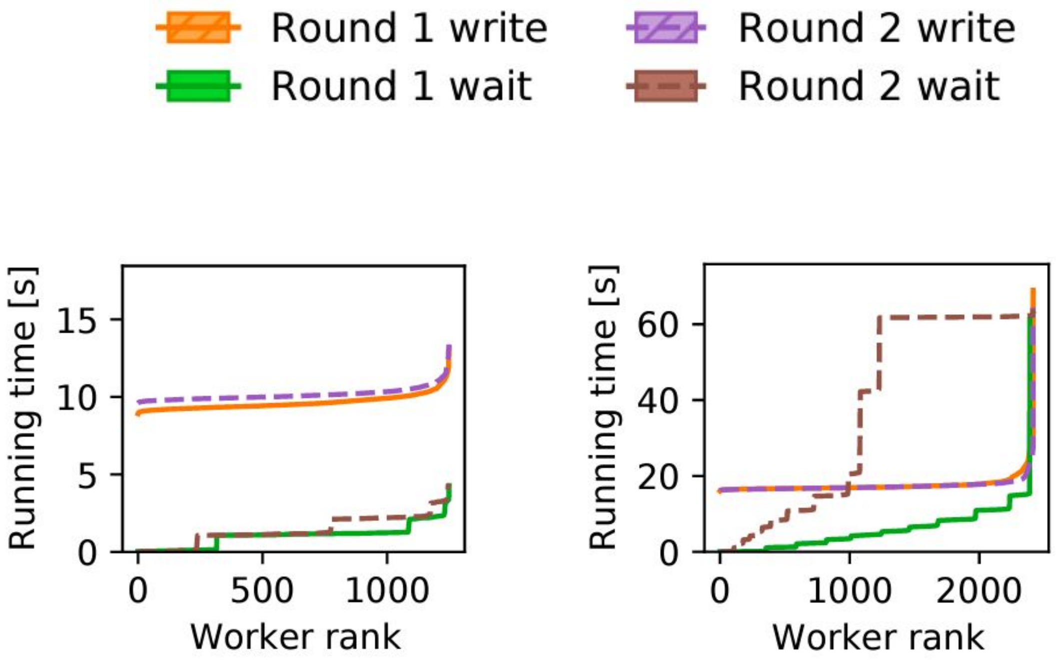
(b) 3 TB, 2500 workers.



1 TB

3 TB

Distribution of the running time of each worker ordered by increasing running time



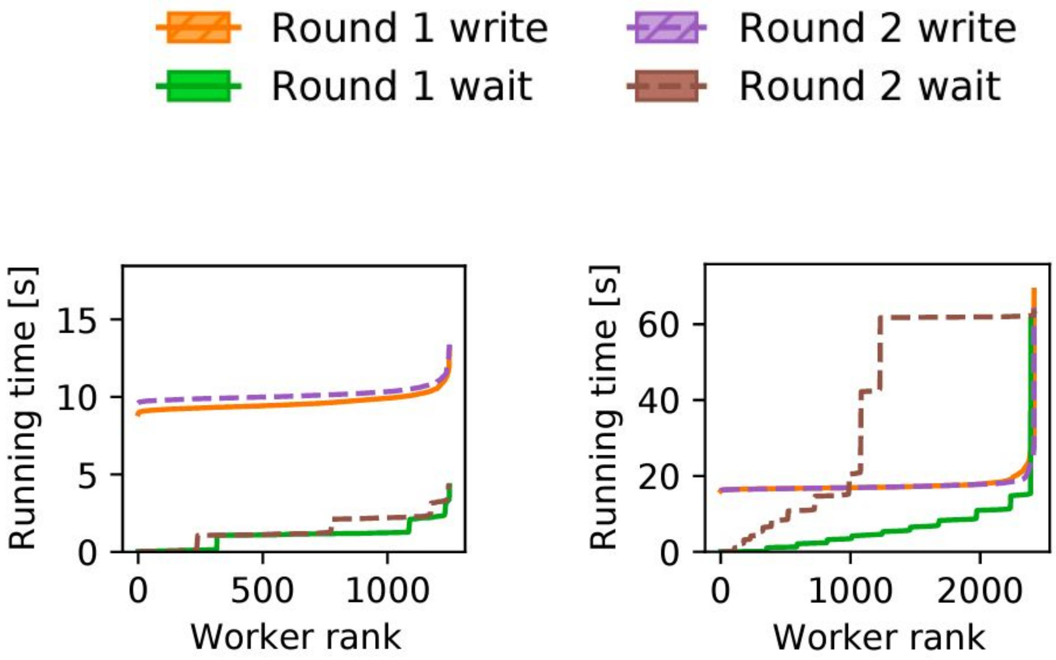
1 TB

3 TB

Write phases have a stable run time until the 95-percentile

Stragglers

Distribution of the running time of each worker ordered by increasing running time



1 TB

3 TB

Wait time dominates the execution time for the larger dataset → **tail latencies**

One slow worker slows down the others too

Conclusion

Interesting Parts

Exchange Operators

Scan Operators

Batch Invocation

Evaluations

Comparison with other implementations

Faster and Cheaper

Missing Parts

Integrate ML

Compatibility with more cloud providers

Conclusions

- Data analytics on serverless computing is technically possible and economically viable
- Tree-based invocation of workers for fast startup
- Design for scan operators that balances cost and performance of cloud storage
- Purely serverless exchange operator
- Lambada can answer queries on more than 1TB of data in about 15 s,

thank you

ANY Questions?

Adobe Stock | #255392447