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

Manind Gera, Aditya Pal, Harsh Mutha, Joseph Mitchell

Evolution of Data Processing

Infrastructure as a Service

- Bare computing resources.
- Starts a set of resources for the duration of a single job.
- Scheduling jobs onto resources that are kept running.

Platform as a Service

- Cloud is used as a rented computing infrastructure.
- Elasticity can lead to a lower total cost than owned infrastructure, limited by how fast the infrastructure can be started and stopped, and services migrated

Software as a Service

- Customers do not rent infrastructure per se but the use of a given software functionality.

Query as a Service

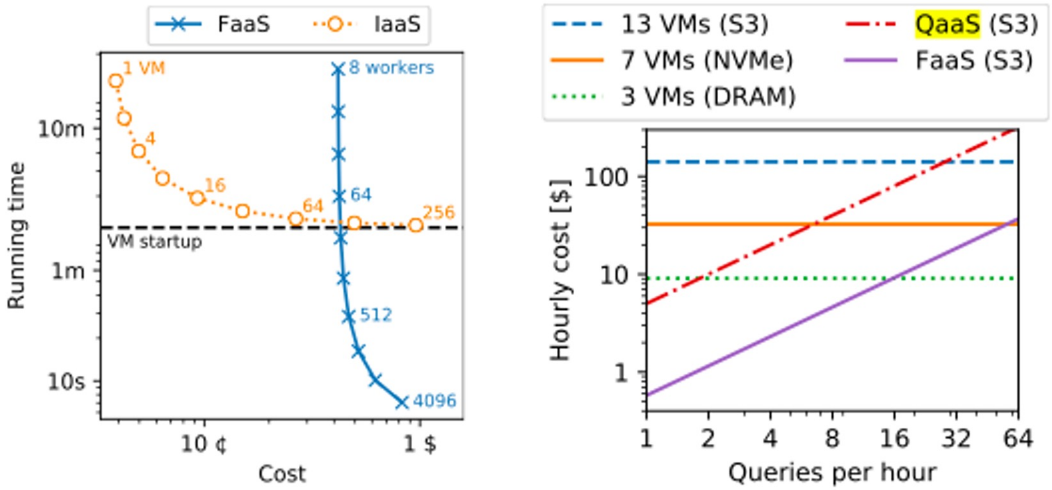
- Usage based pricing models
- Amazon Athena, BigQuery.

Function as a Service

- Way to implement serverless computing
- No need to install, operate, and manage a server (infrastructure) to get computations done.

Comparison of cloud Architectures

Workload: Scan 1 TB from cloud storage



(a) Job-scoped resources. **(b) Always-on resources.**

Goals of the paper

- Identify the **technical limitations** of a concrete implementation of **FaaS, AWS Lambda;**
- Propose **suitable solutions** to the limitations that do not fundamentally reduce their economic advantage, i.e., solutions that **require only serverless components;**
- **Clarify the use cases** in which the cost model behind Lambdas makes sense.
- Design a number of data processing components that **accommodate the existing limitations** of serverless cloud infrastructure to build Lambada.

Contributions

- Characterize **interactive analytics on cold data** as the sweet spot for using FaaS.
- Show that AWS Lambda currently **exposes a small amount of intra-function parallelism**
- Identify the process of **invoking a large number of functions naively as incompatible** with the interactivity requirement
- **effect of the input block size** on the performance and monetary cost of reading data from cloud storage
- **characterizing the competitiveness of FaaS in this domain**.
- Design a purely serverless **exchange operator that overcomes the rate limit of cloud storage**

Suitable Cloud Infrastructure

- **Use other cloud services** to complement them.
- These services should **not incur any cost for idle infrastructure**
- Amazon offers **AWS Lambda, AWS Fargate, and Amazon EC2** to run code in a function, a container, and a virtual machine
- Only **AWS Lambda** has low enough start-up times for interactive analytics.
- For storage, Amazon offers **DynamoDB and S3**, which both scale to zero if used for temporary data during query execution.
- **Amazon SQS** and **AWS Step Functions**

Architecture Overview

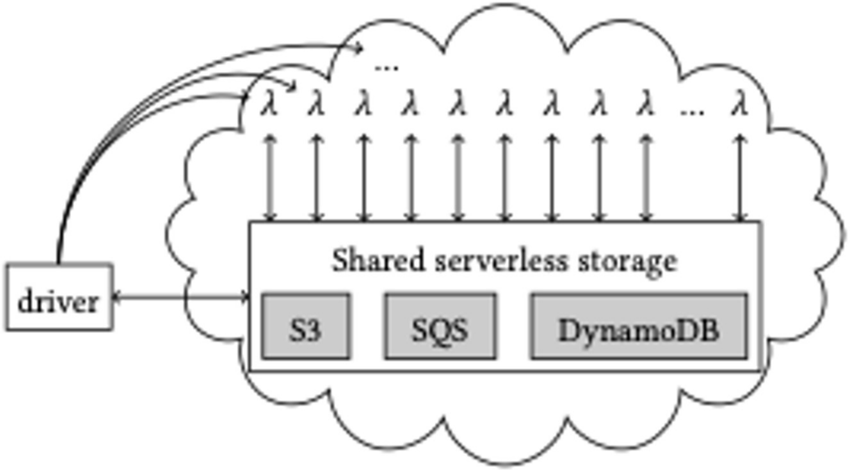
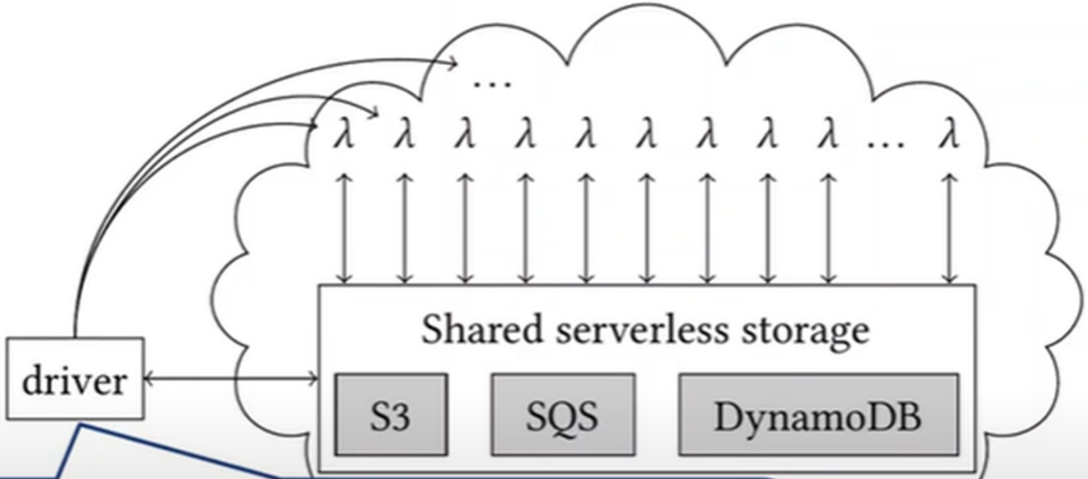


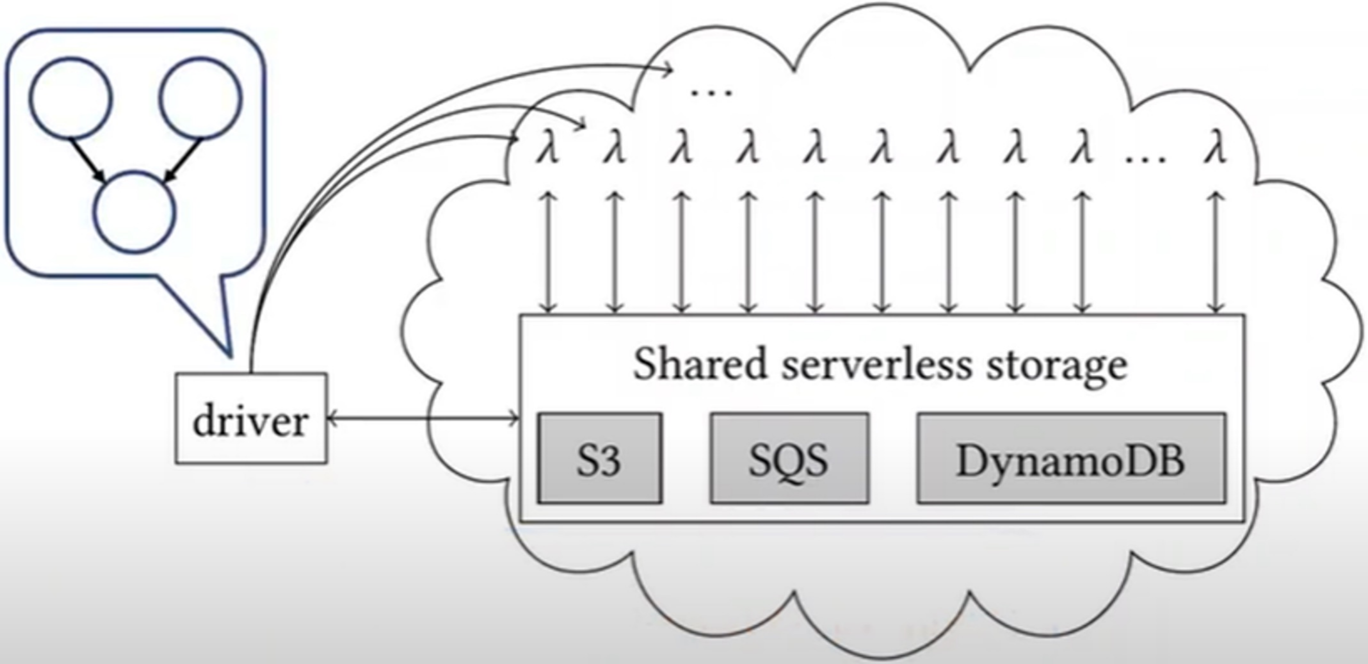
Figure 2: Architecture overview of Lambada.

Lambada: Architecture

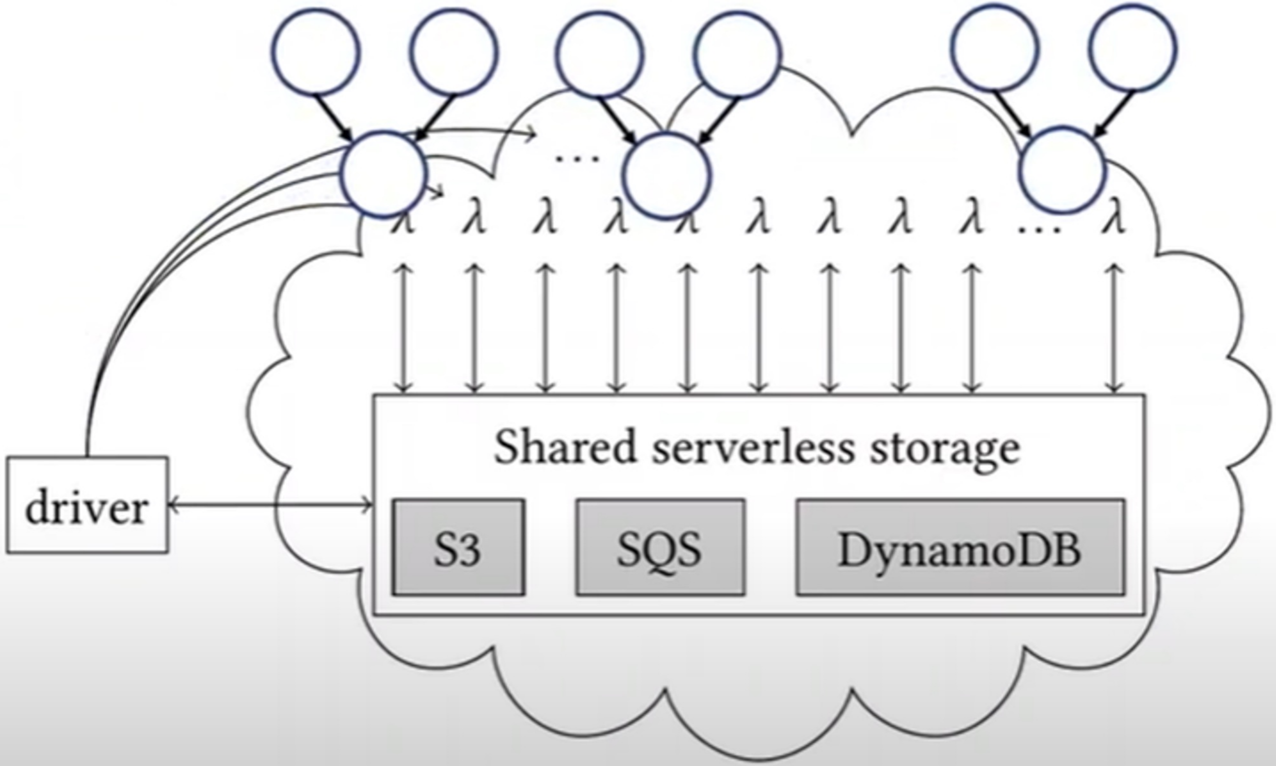



```
data = lambada \
    .from_parquet('s3://bucket/*.parquet')
    .filter(lambda x: x[1] >= 0.05)
    .map(lambda x: x[1] * x[2])
    .reduce(lambda x, y: x + y)
```

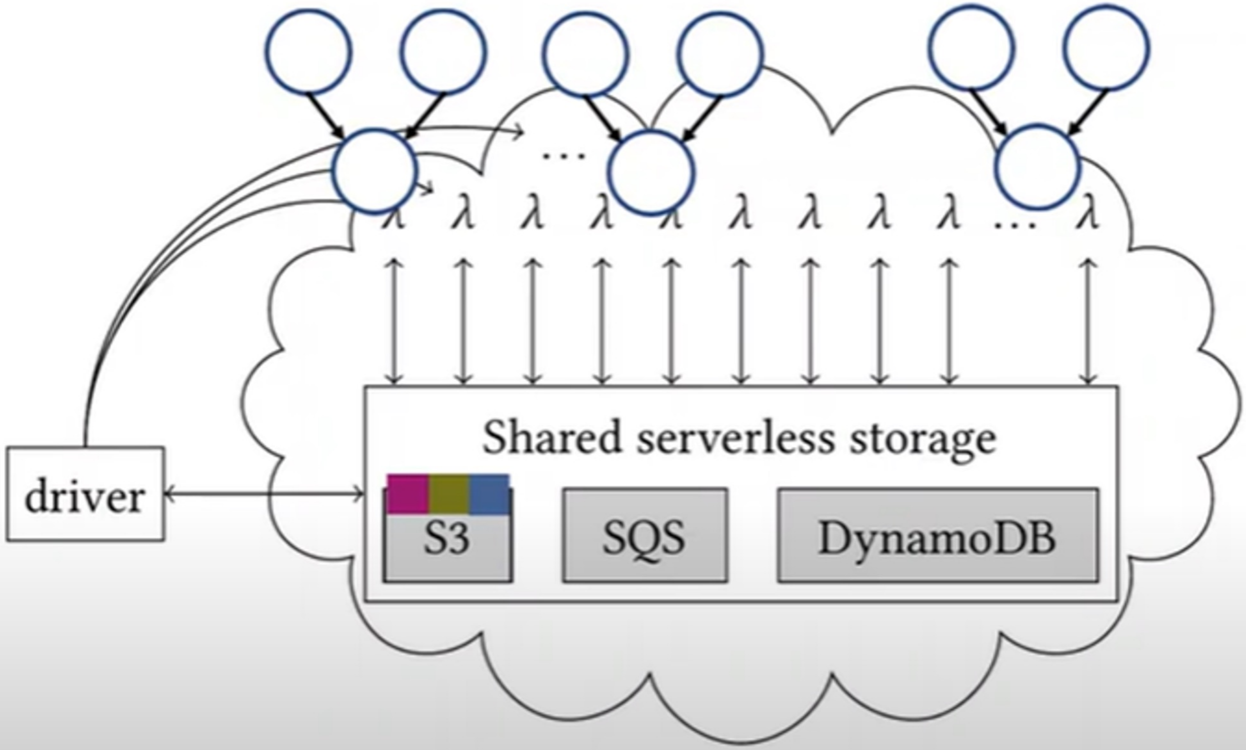
Lambada: Architecture

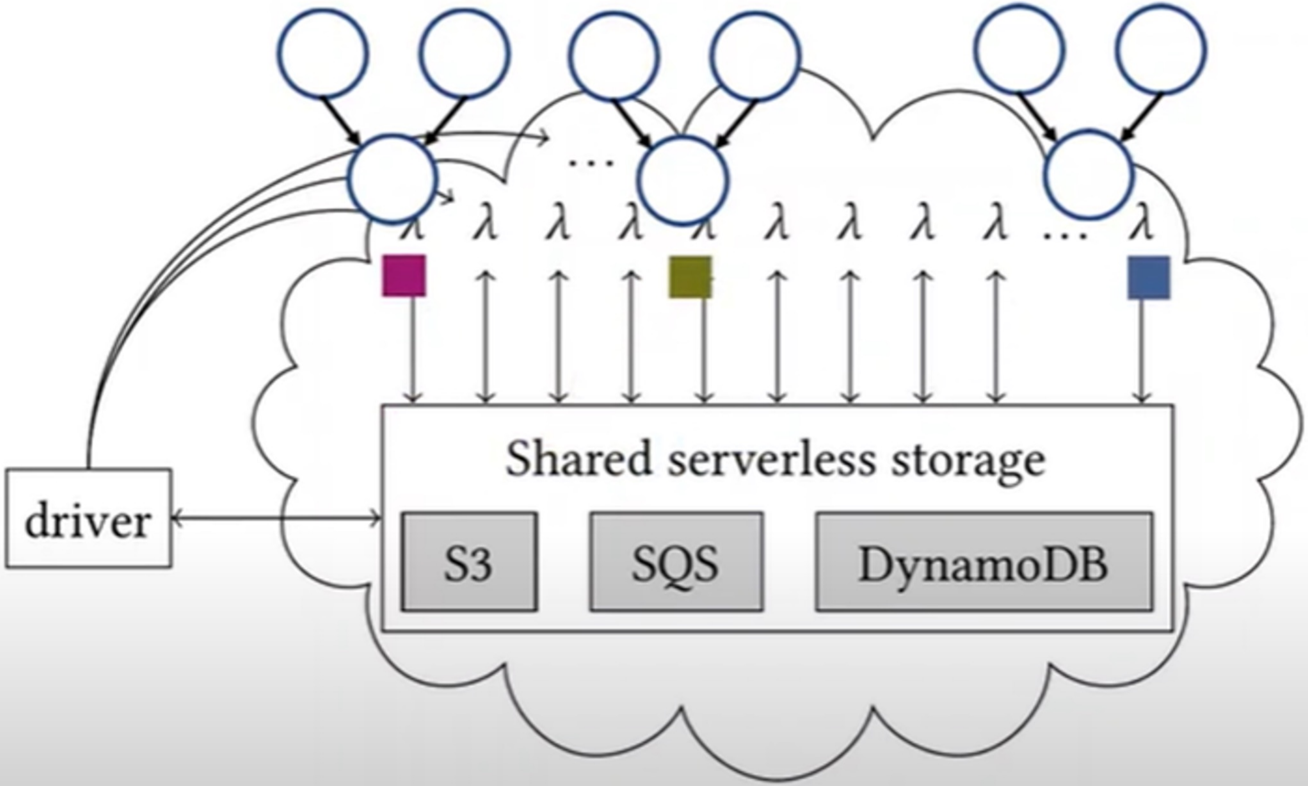


Lambada: Architecture



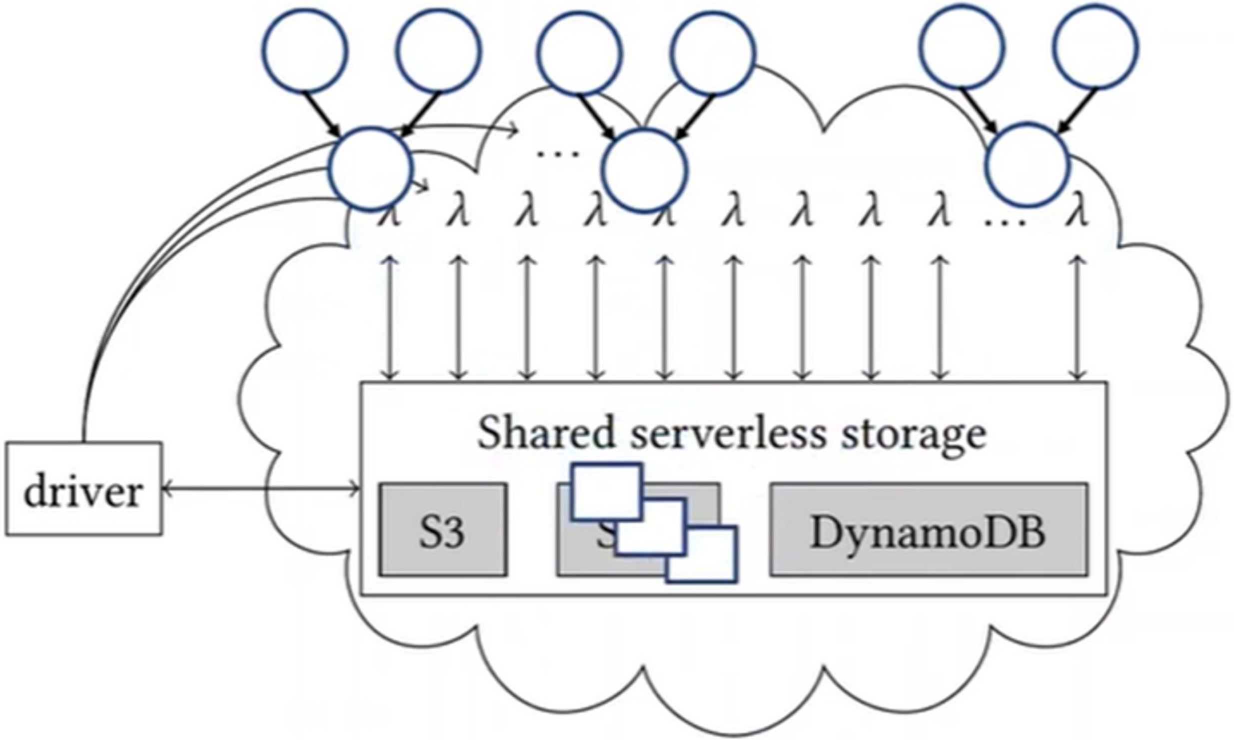
Lambada: Architecture



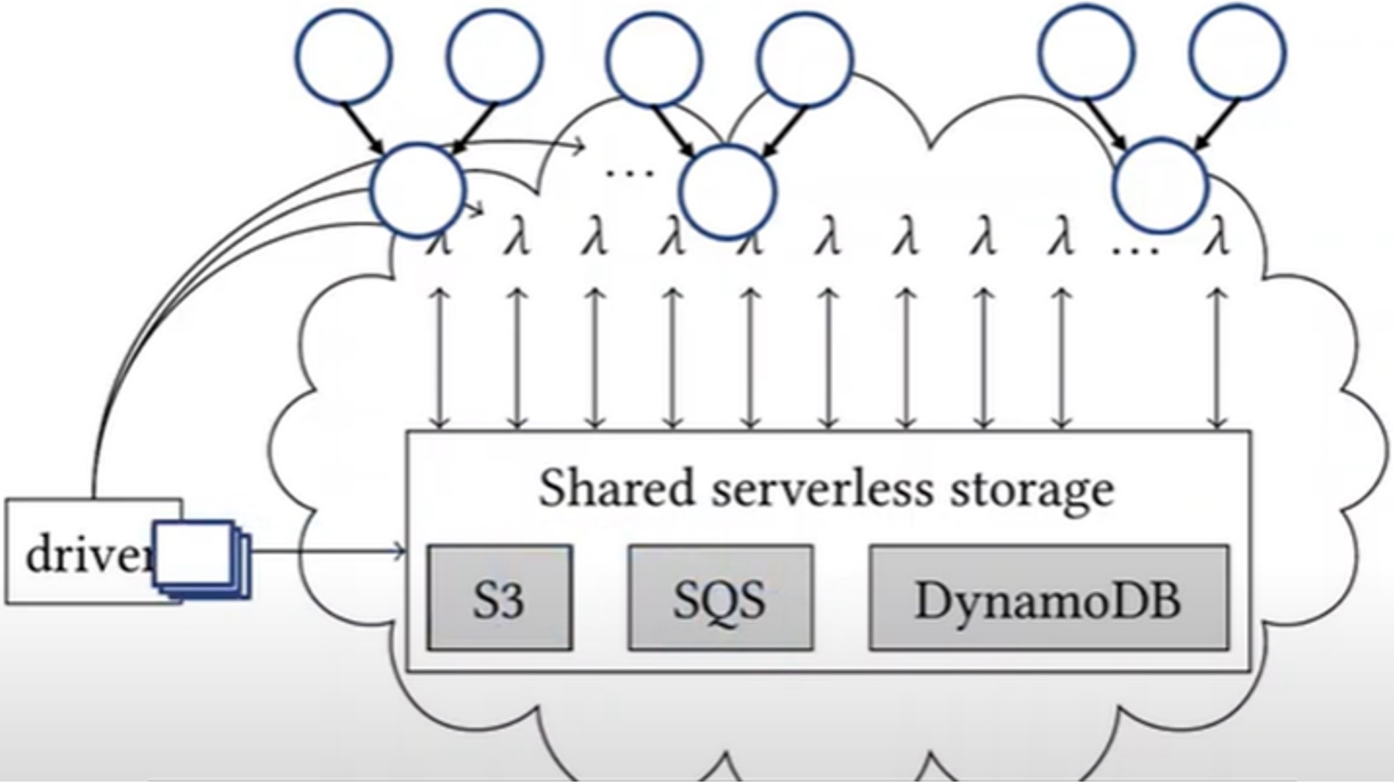


λ λ λ λ λ λ λ λ λ … λ
Shared serverless storage
S3
SQS
DynamoDB
driver

Lambada: Architecture



Lambada: Architecture



Data parallel query plans

- Queries are written in a thin **Python front-end** and go through a **series of translations** that transform it into an executable form.
- A query plan in CVM is **divided into scopes**, each of which may run in a different target platform.
- Most operators in a typical plan of Lambada **run in a serverless scope**
- However, queries may also contain **small scopes** running on the driver

Serverless workers

- The serverless workers run as a function in **AWS Lambda**, which is set up at installation time.
- Consists of an **event handler, a "dependency layer" and some metadata**
- Dependency layer contains the **same execution framewor**k that also runs on the driver and an event handler as a wrapper around it **implemented in Python.**
- Event handler **extracts the ID of the worker, the query plan fragment, and its input** from the invocation parameters of the function and forwards them to the execution framework.
- When the execution engine finishes its computation, the **handler forwards its results to the driver.**
- If an error occurs or the computation finished successfully, the **handler posts a corresponding message** into a result queue in SQS, from which the driver polls until it has heard back from all workers.

System components for Serverless Analytics

- **Hard quotas and limits** from the service-level agreements (SLAs) of the cloud provider such as a limit on the request rate to S3,
- **Execution speed** under the given constraints (from service limits or from de-facto performance of a resource)
- **Usage-based cost** of the various serverless services, such those from the running time of the serverless workers but also from the number of requests to the various systems.

Limits of Sequential Invocation

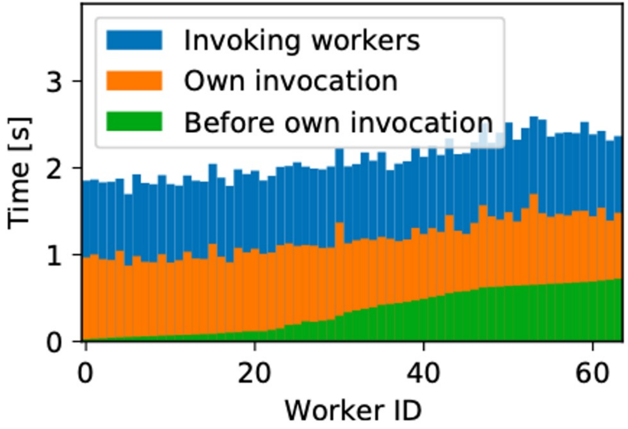
Metric	**Region**			
	eu	**us**	**sa**	**ap**
Single invocation time [ms]	36	363	474	536
Concurrent inv. rate [inv./s]	294	276	243	222
Intra-region rate [inv./s]	81	79	84	81

Table 1: Characteristics of function invocations.

Lambada Two-level Invocation

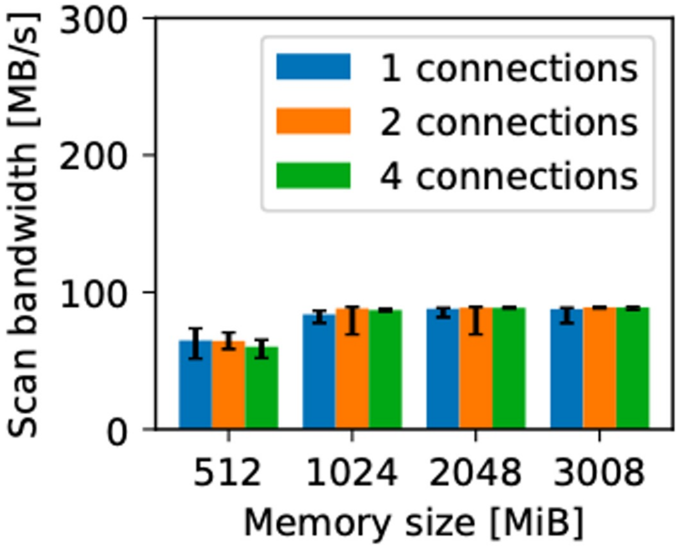
Every first generation worker works in a three phase timeline:

- **The time the driver took** before it initiated their invocation (namely, to launch all previous workers),
- **The time their invocation took**, i.e., the time between their invocation was initiated and they were actually running, and
- The time they took to do the **second- generation invocations.**



Network Characteristics: Large Files

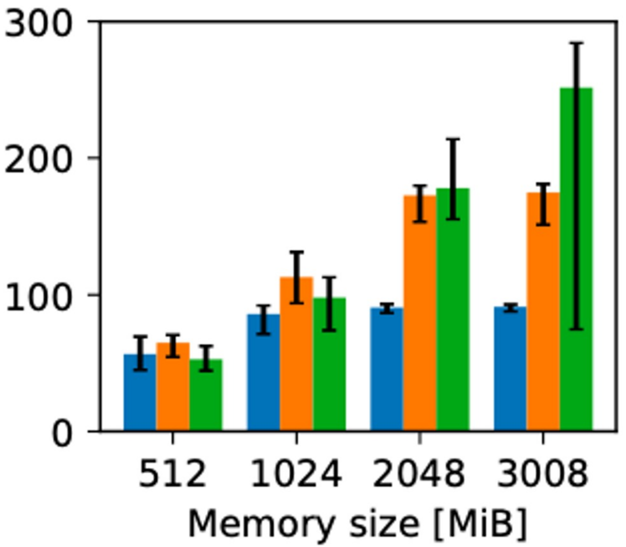
- There is a very stable limit of about **90 MiB/s per worker**.
- Workers of virtually any size have **fast enough network** to achieve this limit.
- Only workers with less than **1 GB of main memory see a slightly lower ingress** bandwidth.
- **Using more network connections does not significantly change the overall bandwidth.**



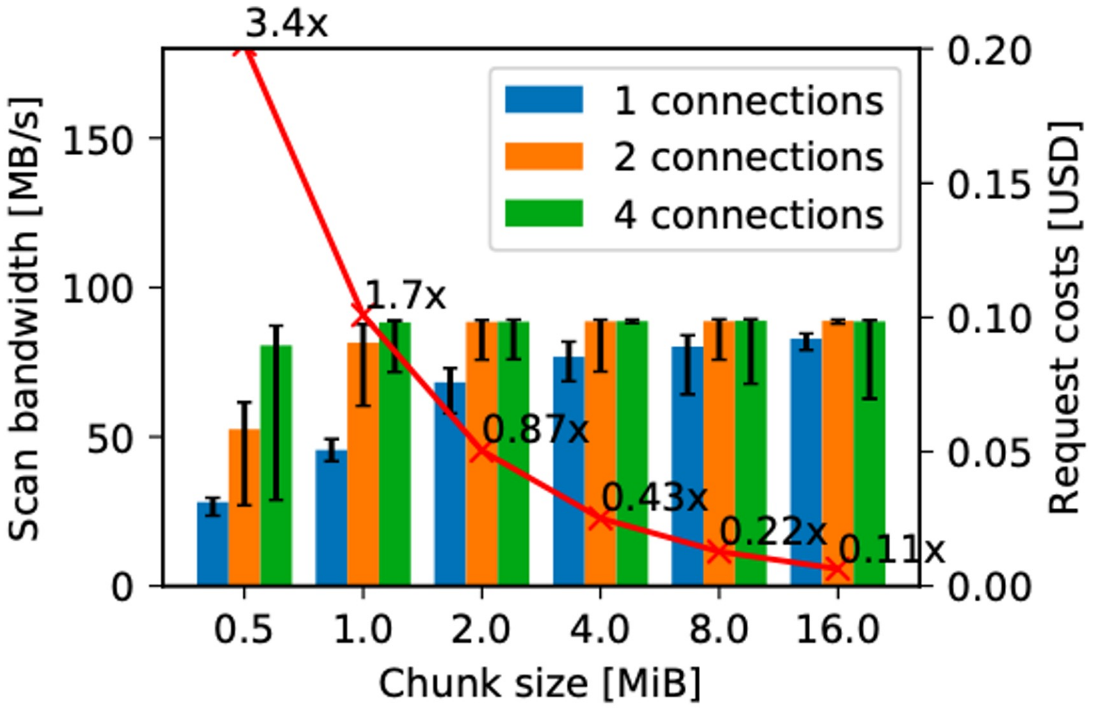
(a) Large files (1 GB).

Network Characteristics: Small Files

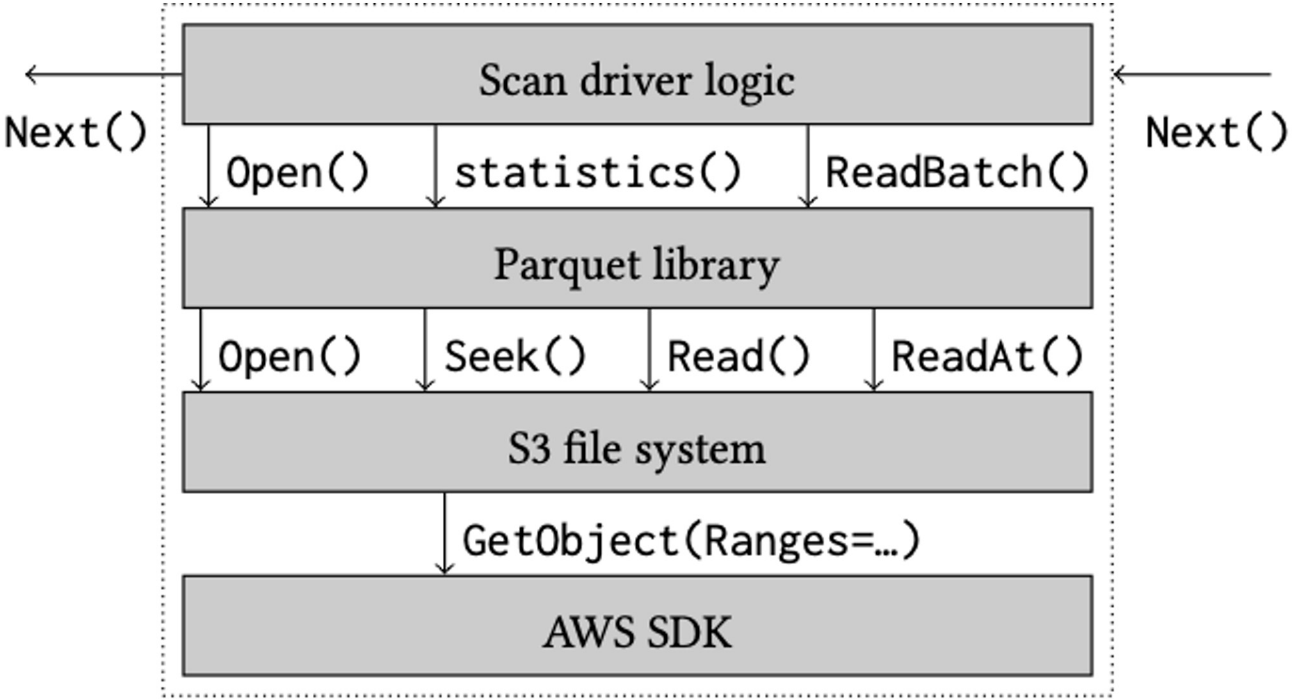
- Workers with large amounts of memory observe a much **higher network bandwidth**, occasionally reaching almost 300 MiB/s.
- This is only the case if they use **several network connections** at the same time.
- It is observed that the time span during which the burst may exceed the target is a **small number of seconds.**
- In order to maximize performance for short-running scans, we thus need to use **multiple concurrent connections.**



Impact of Memory size



Lambada Cloud Native Scan



Exchange in Joins, Sorting and Grouping

- The exchange operator transfers its input among the workers such that all t**uples belonging to the same partition** (according to some partitioning criteria) **end up at the same worker.**
- Joins, sorting, and grouping **can be executed in parallel** with the help of one or more exchange operators; no further operator with communication logic is required
- The proposed operator at the same time **necessary and sufficient** for data-parallel processing on serverless workers.

Basic Exchange Operator

Algorithm 1 Basic S3-based exchange operator.

func BasicExchange(p: **Int**, $\mathcal{P}$: **Int**[1..P], R: **Record**[1..N],
    FormatFileName: **Int** × **Int** → **String**)
  partitions ← DramPartitioning(R, $\mathcal{P}$)
  for ⟨*receiver*, data⟩ **in** partitions **do**
    WriteFile(FormatFileName(*receiver*, p), data)
  for *source* **in** $\mathcal{P}$ **do**
    data ← data ∪ ReadFile(FormatFileName(p, *source*))
  return data

Lambada Multi Level Exchange

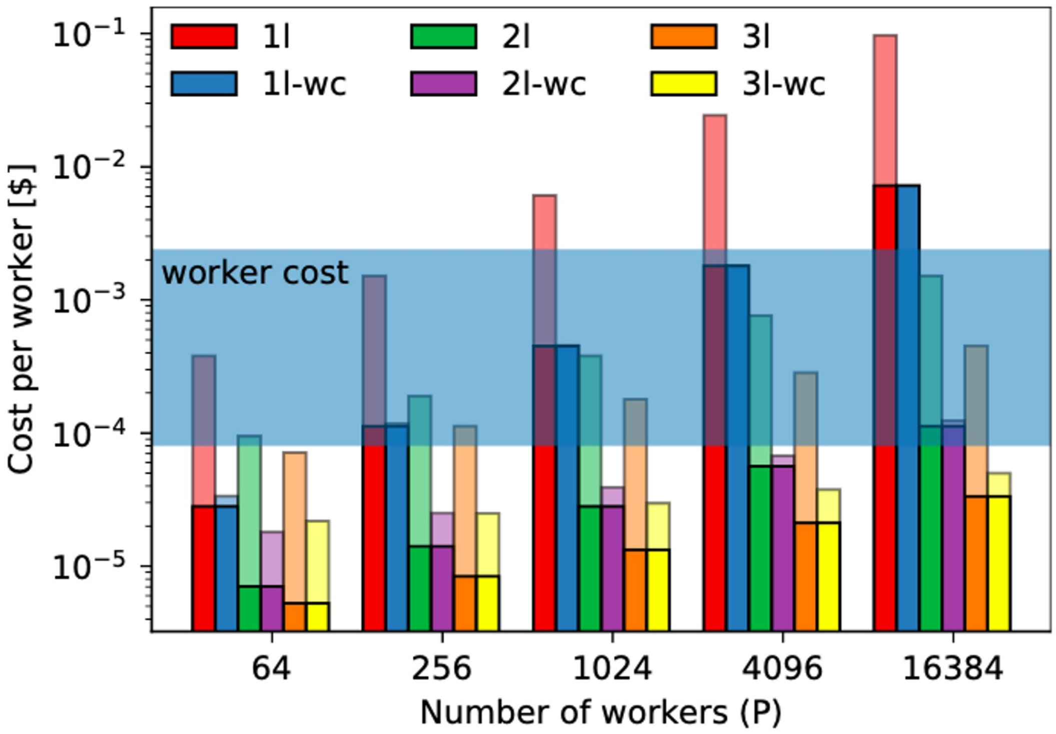
Algorithm 2 Two-level S3-based exchange operator.

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

Table 2: Cost models of S3-based exchange algorithms.

Algorithm	#reads	#writes	#lists	#scans
1l	P^2	P^2	$O(P)$	1
1l-wc	P^2	P	$O(P)$	1
2l	$2P\sqrt{P}$	$2P\sqrt{P}$	$O(P)$	2
2l-wc	$2P\sqrt{P}$	$2P$	$O(P)$	2
3l	$3P\sqrt[3]{P}$	$3P\sqrt[3]{P}$	$O(P)$	3
3l-wc	$3P\sqrt[3]{P}$	$3P$	$O(P)$	3

Complexity and Cost Analysis



Dataset and Methodology

- Most experiments use the TPC-H benchmark
- Lambada does not support strings
  - Dbgen modified to generate numbers instead
- Scale factor is 1K, size of data set is 502 GiB
  - In Parquet - standard encoding, GZIP compression, size 273 GiB

End to End Query Latency

- Accounts for:
  - Serverless workers' invocation time
  - Useful work carried out
  - Fetching results from result queue in Amazon SQS
- Median of three runs are reported, same data center:
  - Using a different data center showed negligible variation

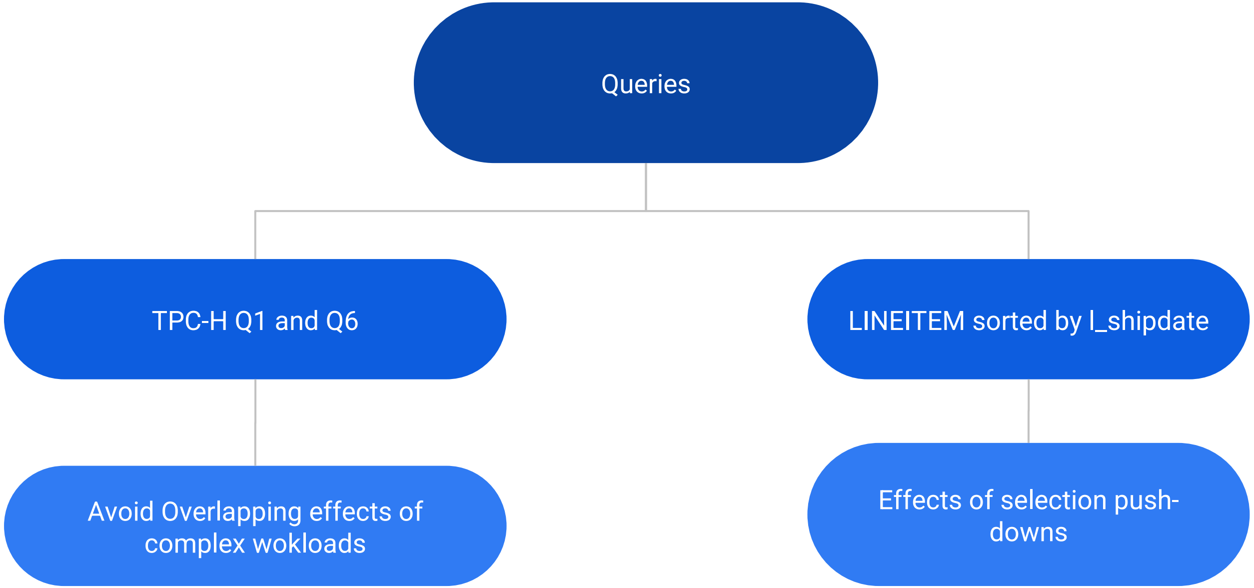
Comparison with QaaS

- Lambada is compared with **Google BigQuery** and **Amazon Athena**
- QaaS - similar operational simplicity as Lambada
- Queries without need for startup or maintenance
- Usage based pricing model
- Therefore, well suited for cold data interactive analytics
- PaaS solutions are not considered due to **running on VMs and hourly pricing model**

Google BigQuery

Amazon Athena

Scan Heavy Queries



Effect of worker configuration

◦Parameter space of worker configurations are explored

◦Amount of main memory of each worker, **M**, is varied

- ‣ Influences number of CPU cycles the function can use
- ‣ Influences number of files, **F**, that each worker may process

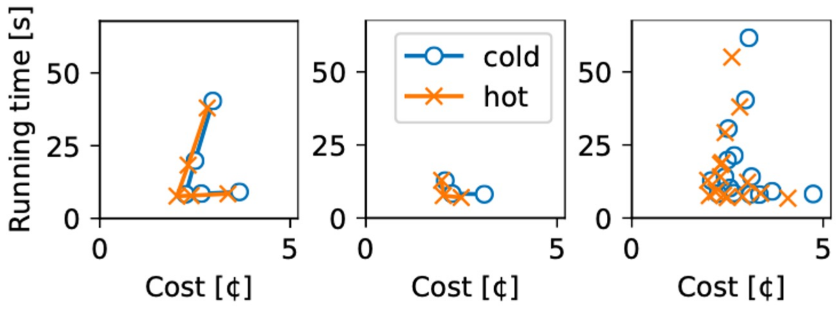
• F indirectly defines number of workers*

◦Table is stored in 320 files

‣ **W** = 320/F workers

TPC-H Query 1

- Selects 98% of LINEITEM
- Aggregates selection to very small amount of groups
- This eliminates effects of more complex plans
- Query is ran twice: first is cold run, second is hot run
- Fresh function is created for each configuration and repetition



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

Effect of push-downs

◦Effect of pushing down selections and projects into the scan operator are studied

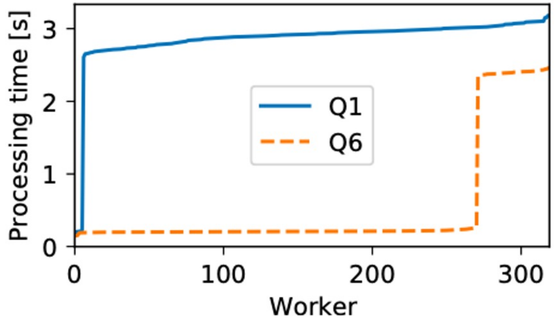
◦TPC-H Query 1 and Query 6 are used

‣ The two most scan-bound queries of TPC-H

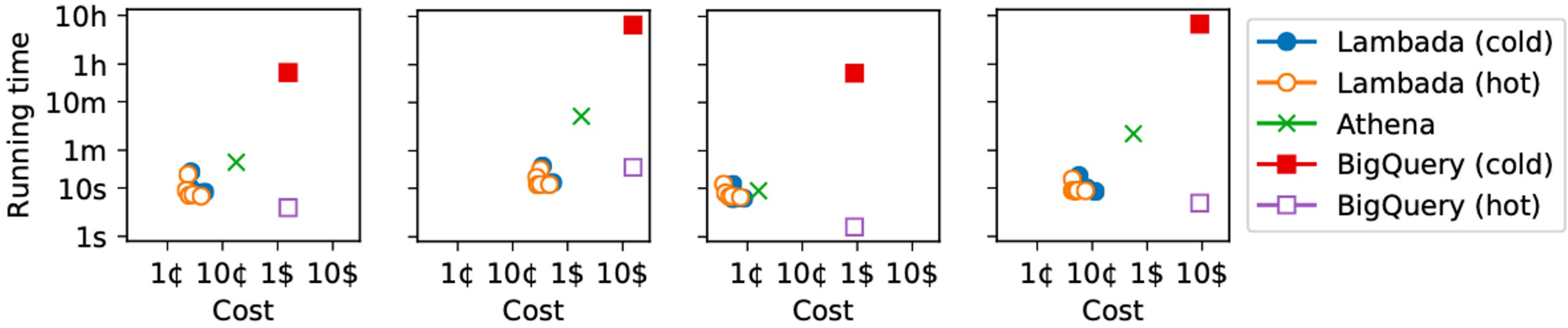
- Query 1 - selects 98% of relation, uses 7 attributes
- Query 6 - selects 2% of relation, uses 4 attributes

◦Only processing time is measured

‣ Eliminates unrelated effects such as invocation time



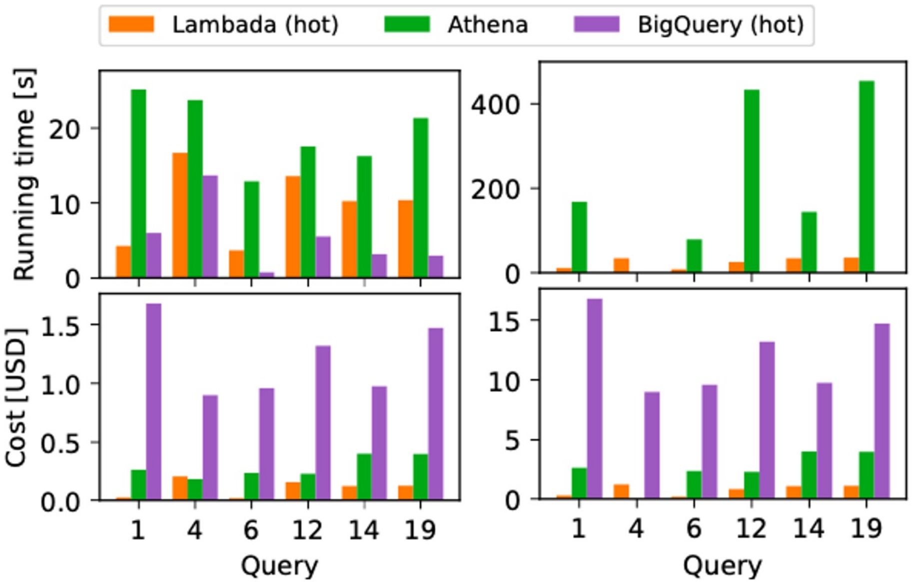
Comparison with QaaS systems



(a) Q1, SF 1 k. **(b) Q1, SF** 10 k. **(c) Q6, SF** 1 k. **(d) Q6, SF** 10 k.

Figure 10: Comparison of Lambada (using $F = 1$ and varying M) with commercial QaaS systems.

End to End Workloads



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

Figure 11: TPC-H queries on Lambada ($M = 2$ GiB).

Exchange operator

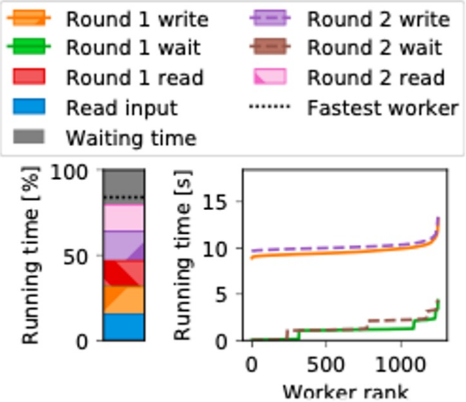
- Dataset of 100GB is used
- Locus and Qubole: use workers with 1536 MiB of main memory
- Pocket: uses 3008 MiB workers
- Lambada: uses 2048 MiB of allocated memory

Table 3: Running time of S3-based exchange operators.

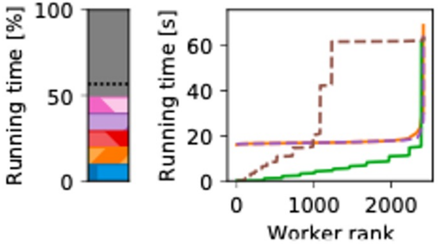
	#Workers	Storage Layer	
		VMs	S3
Pocket [27]	250	58 s	98 s
	500	28 s	
	1000	18 s	
Locus [38]	dynamic		80 s to 140 s
Qubole [41]	400		580 s
Lambada	250		22 s
	500		15 s
	1000		13 s

Two Level Exchange

It is shown that “exchange operators can be implemented under a purely serverless paradigm and even outperform approaches with always-on infrastructure”



(a) 1 TB, 1250 **workers.**



(b) 3 TB, 2500 **workers.**

Does the paper support its claims?

- Yes!
- Data analytics on serverless computing is possible and economically viable
- Lambada can answer on 1Tb data in 15s
- Competitive with conventional QaaS and faster than job-scoped VMs

Possible next steps

- Explore the concept of serverless clusters
- Improve PyWren, Flint using the Lambda optimizations