Lambada

Serverless analytics on cold data

Goal

To run **data analytics** on **serverless computing framework**

in a **cost effective manner**.

Goal

To run **data analytics** on **serverless computing framework**

in a **cost effective manner**.

Q. What is serverless and why?

Q. What kind of data analytics are cost effective on serverless?

Cloud Computing - Trend

Infrastructure-as-a-service

- Virtualize computing resources
- AWS(2006), GCP (2008), Azure (2010)

Platform-as-a-service

- Build and Deploy applications on cloud
- Amazon Elastic bean, Google App services

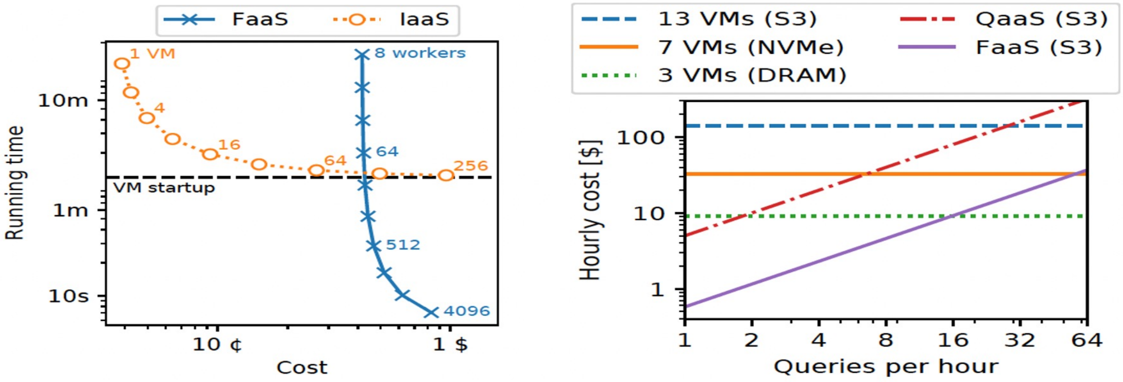
Software-as-a-service

- DBaaS, Data Security as a service
- Snowflake, Spanner

Function-as-a-service

- Ultimate granularity, resource utilization
- AWS Lambda, Google Cloud Functions.

Serverless - SWOT



(a) Job-scoped resources.

(b) Always-on resources.

Good for	Bad for
Low-latency queries (interactive)	Long running analytics
Sporadic query load (ie, likely cold data)	High sustained query load

Goal

To run **data analytics** on **serverless computing framework**

in a **cost effective manner**.

Q. Why serverless?

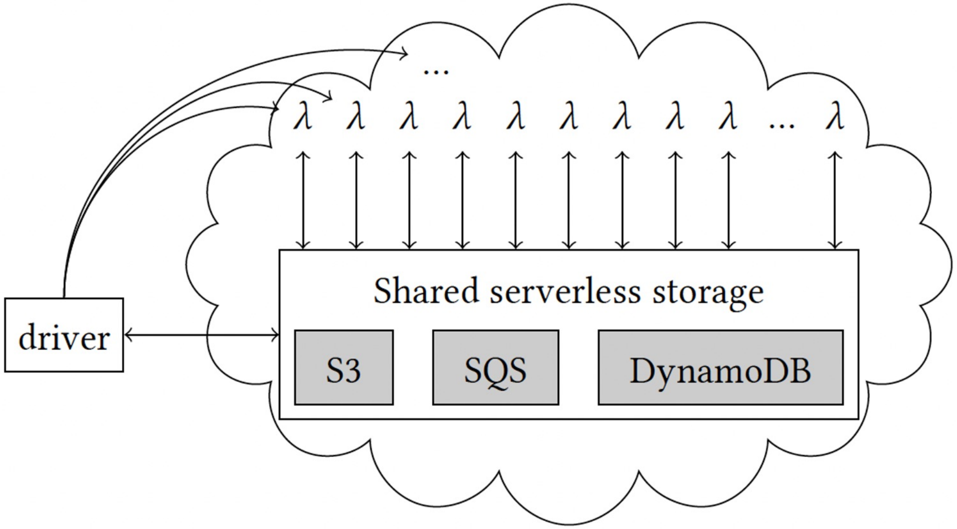
- ❖ Ultimate elasticity, granularity, pay-as-you-go model.

Q. What kind of data analytics are cost effective on serverless?

- ❖ Interactive, infrequent queries – `lone-wolf data scientist use case`

~ Into the serverless

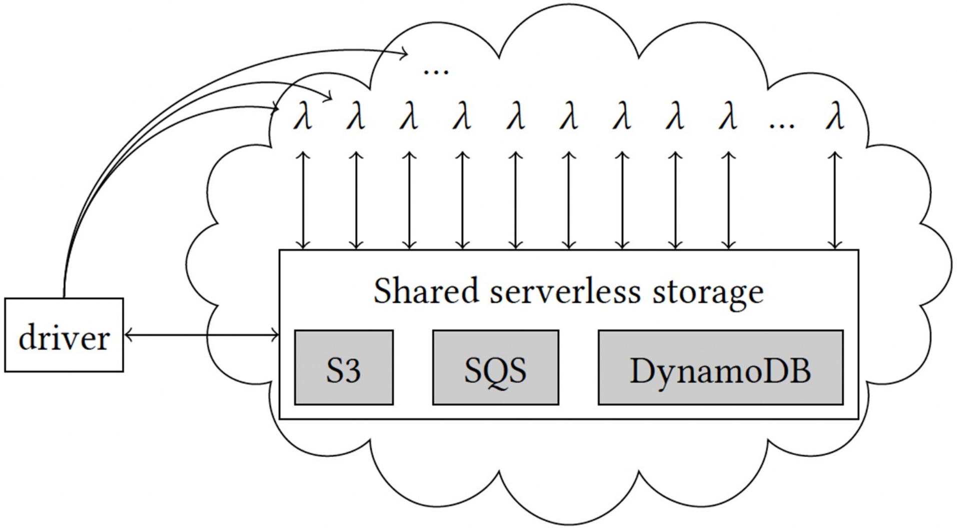
Architecture - for data analytics



- A local coordinating driver
- Data parallel query plans (

  difference with starling?)
- Serverless workers
- Shared serverless storage for intermediate and output data

Architecture - for data analytics



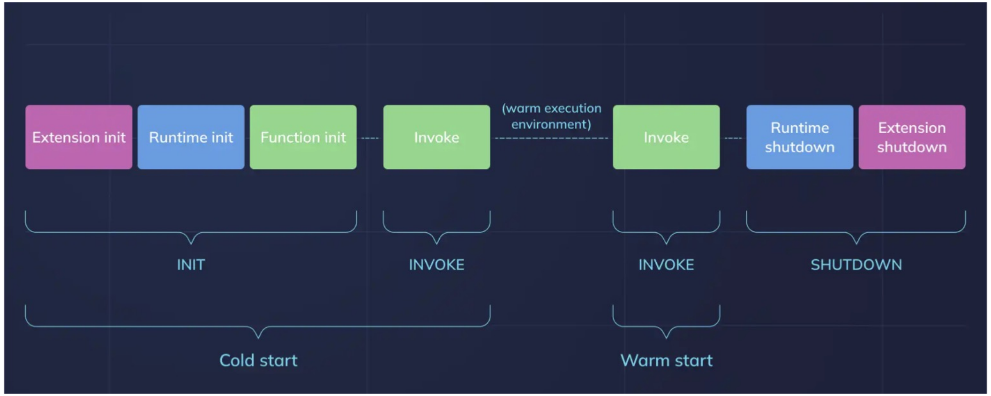
Challenges ?

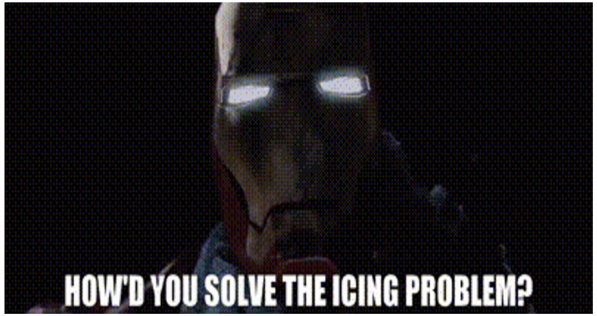
- Cold start problem
- Network-Efficient Scan
- Efficient Exchange Operator

Prob #1: The cold start

Q. How to invoke thousands of AWS lambda workers fast?

- Invoke sequentially?
- Use concurrent invocation in driver?

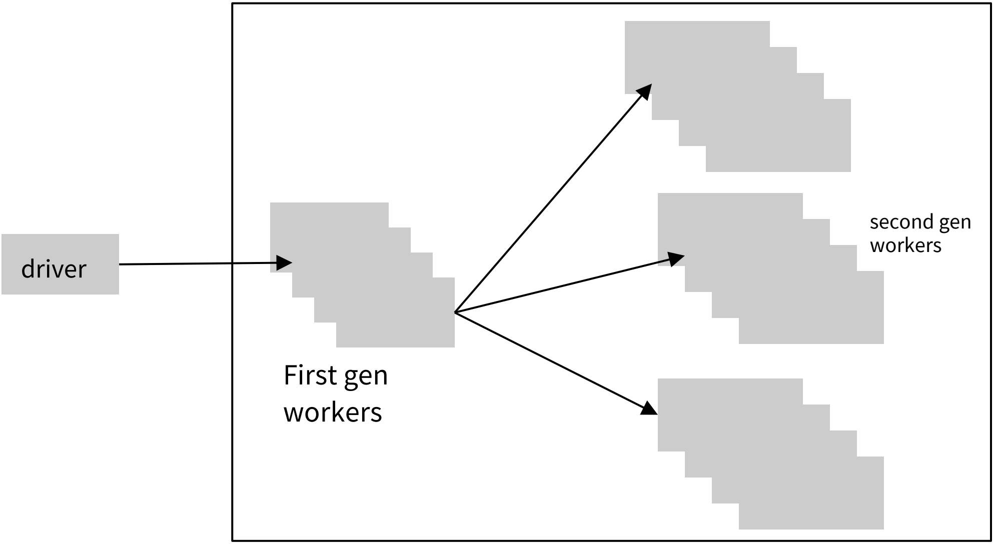


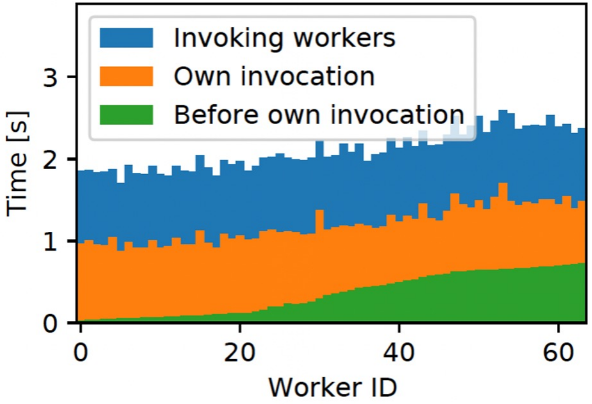


Prob #1: The cold start

Q. How to invoke thousands of AWS lambda workers fast?

Idea: Multi-level invocation - 3-4x speedup over concurrent driver invocations

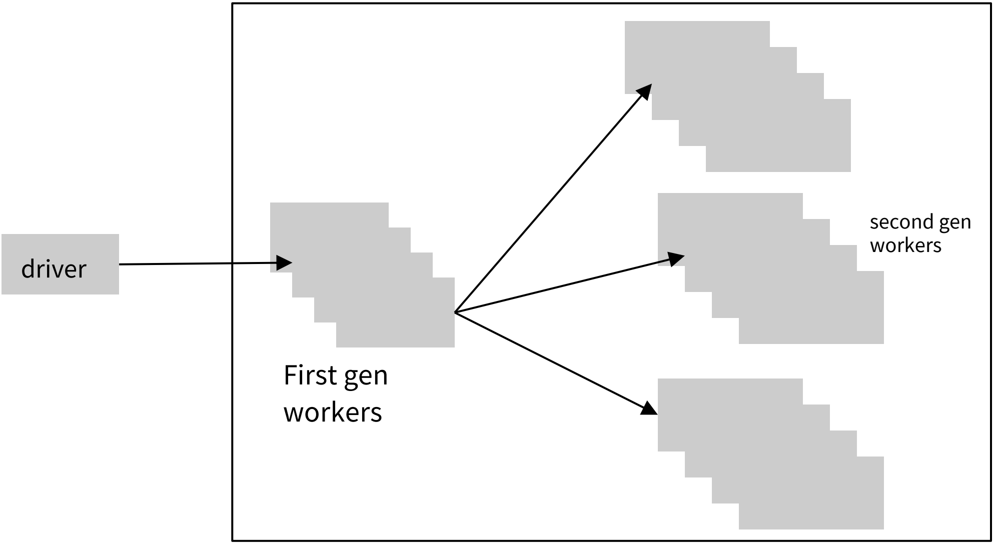


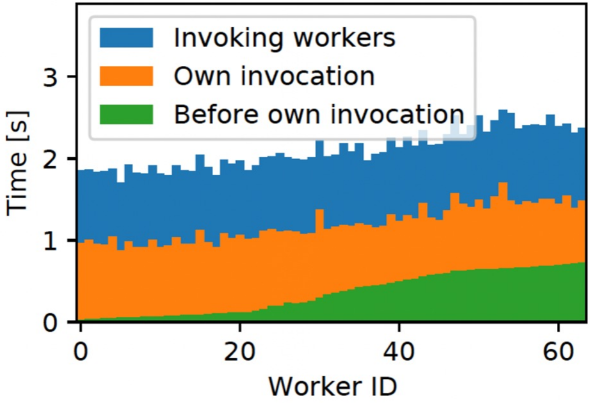


Prob #1: The cold start

Q. How to invoke thousands of AWS lambda workers fast?

Idea: Multi-level invocation - 3-4x speedup over concurrent driver invocations





Q. Why better though?

Q. Is this method resilient to failures?

Prob #2: Efficient Network Scan

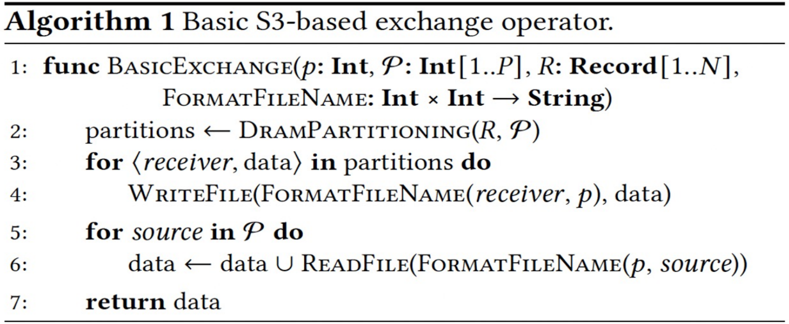
Q. How to do perform a network efficient scan?

- Use parquet format
  - Columnar storage of row groups
  - Pruning, efficient projection
  - ~ idea as in snowflake
- Use concurrency (~ **Starling**)
  - Download data from multiple files
  - Download multiple rows and different columns from same row if possible
  - Chunk-size vs latency trade off
- Can use computational push-down as well

Prob #3: Efficient Exchange Op

Q. How to exchange data amongst workers efficiently?

First cut:

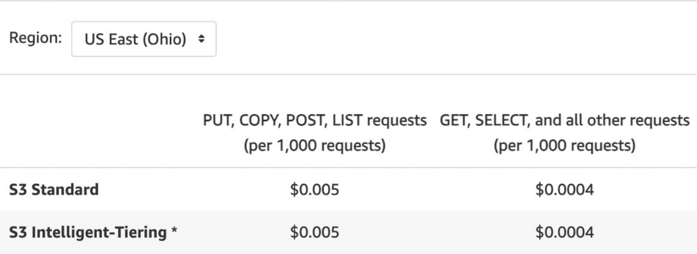


Algorithm 1 Basic S3-based exchange operator.

```
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
```

Issues:

- ❖ Quadratic number of requests!!
- ❖ Billing on number of requests



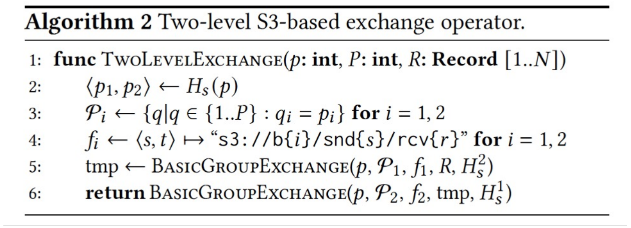
Region: US East (Ohio)

	PUT, COPY, POST, LIST requests (per 1,000 requests)	GET, SELECT, and all other requests (per 1,000 requests)
S3 Standard	$0.005	$0.0004
S3 Intelligent-Tiering *	$0.005	$0.0004

Prob #3: Efficient Exchange Op

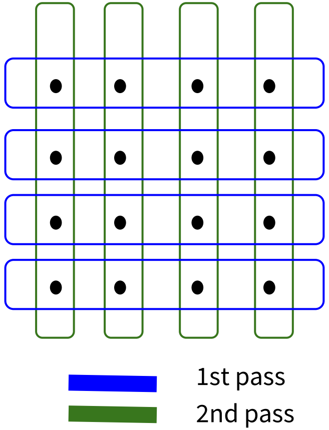
Q. How to exchange data amongst workers efficiently?

- Multi-level exchange



Algorithm 2 Two-level S3-based exchange operator.

```
func TwoLevelExchange($p$: int, $P$: int, $R$: 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 ← BasicGroupExchange($p$, $\mathcal{P}_1$, $f_1$, $R$, $H_s^2$)
    return BasicGroupExchange($p$, $\mathcal{P}_2$, $f_2$, tmp, $H_s^1$)
```

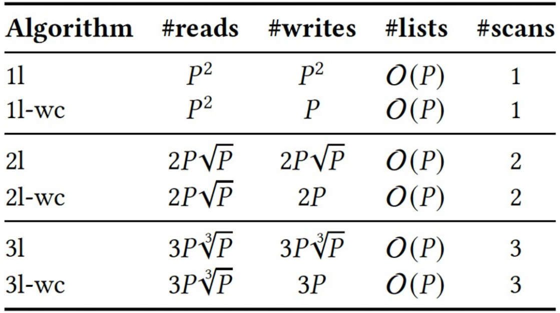


🟦 1st pass

🟩 2nd pass

- Write combining
  - Each worker writes all data to be shared in a single file.

Q. How does this compare with starling?



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

Evaluation

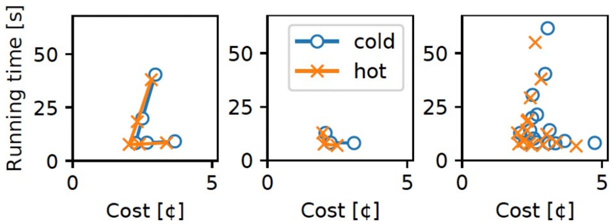
Q1. How does varying AWS Lambda memory and num of workers affect query run-time and cost?

Q2. How does Lambada compare against commercial QaaS services?

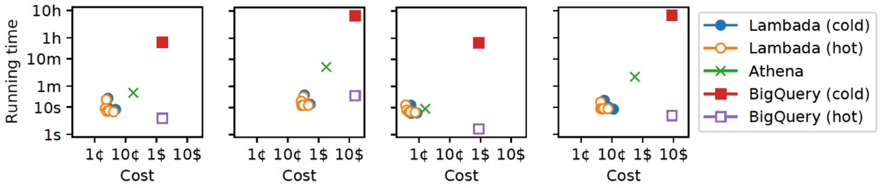
Q. How does Lambada perform over realistic workloads?

Q. How good is Lambada’s exchange operator?

Evaluation



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



(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.

A1. Increasing workers and memory speeds up execution but at diminishing rates and increasing cost

A2. a) Lambada, on most workloads, has competitive performance with commercial QaaS.

b) The pricing model of Lambada reflects the resources utilized more accurately than QaaS systems.

Evaluation

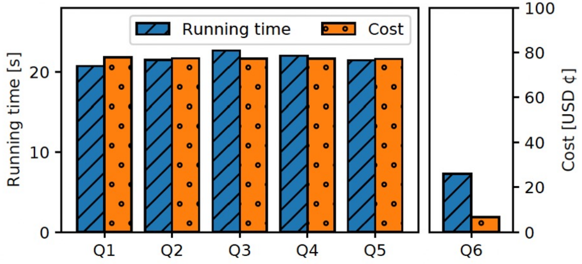
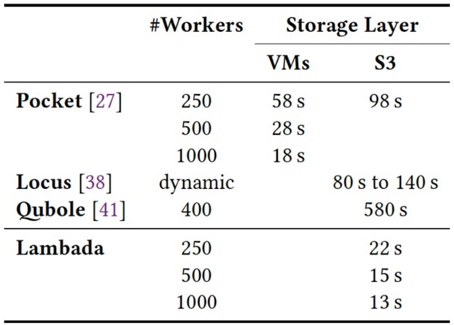


Figure 12: Hydrologist (Q1-Q5) and HEP queries (Q6).

A3. Can process TBs of scientific data within seconds at sub-dollar cost.

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



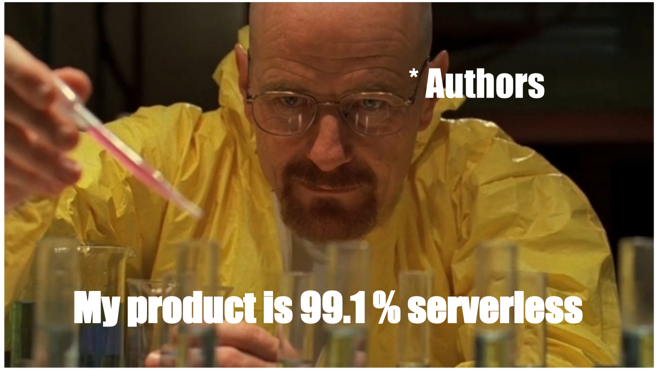
	#Workers	Storage Layer	
		VMs	S3
Pocket [27]	250	58 s	98 s
	500	28 s	
	1000	18 s	
Locus [38]	dynamic		80 s to 140 s
Qubole [41]	400		580 s
Lambada	250		22 s
	500		15 s
	1000		13 s

A4. a) Lambada’s exchange operator is more scalable as it uses multiple buckets.

b) Subject to slow down by stragglers

Discussion - what did they get right?

- Identifying the right workload
- Purely serverless - “no additional infrastructure”
- “Multi-level” efficient cloud query functions
  - The exchange op is pretty cool!
- Extensive Evaluation



Discussion - critique

- Handling faults
  - What is the cost of failure during multi-level batch invocation and shuffle?
- Handling stragglers
  - Can use the retry mechanism as in Starling
- Pipelining worker tasks?
  - They do mention that their work is similar to Starling which uses pipelining
- Omitted details about parallelised query plans.

Thank you!!!