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

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

**Platform-as-a-service**
- Build and Deploy applications on cloud
- Amazon Elastic bean, Google App services

**Function-as-a-service**
- Ultimate granularity, resource utilization
- AWS Lambda, Google Cloud Functions.
Serverless - SWOT

<table>
<thead>
<tr>
<th>Good for</th>
<th>Bad for</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low-latency queries (interactive)</td>
<td>Long running analytics</td>
</tr>
<tr>
<td>Sporadic query load (ie, likely cold data)</td>
<td>High sustained query load</td>
</tr>
</tbody>
</table>
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?
Q. How to 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 the same row if possible
  - Chunk-size vs latency trade-off

- Can use computational push-down as well
Q. How to exchange data amongst workers efficiently?

First cut:

```
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 ← DrumPartitioning(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
```

**Issues:**
- Quadratic number of requests!!
- Billing on number of requests
Prob #3: Efficient Exchange Op

Q. How to exchange data amongst workers efficiently?

❖ Multi-level exchange

Algorithm 2 Two-level S3-based exchange operator.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>#reads</th>
<th>#writes</th>
<th>#lists</th>
<th>#scans</th>
</tr>
</thead>
<tbody>
<tr>
<td>1l</td>
<td>$p^2$</td>
<td>$p^2$</td>
<td>$O(P)$</td>
<td>1</td>
</tr>
<tr>
<td>1l-wc</td>
<td>$p^2$</td>
<td>$P$</td>
<td>$O(P)$</td>
<td>1</td>
</tr>
<tr>
<td>2l</td>
<td>$2P\sqrt{P}$</td>
<td>$2P\sqrt{P}$</td>
<td>$O(P)$</td>
<td>2</td>
</tr>
<tr>
<td>2l-wc</td>
<td>$2P\sqrt{P}$</td>
<td>$2P$</td>
<td>$O(P)$</td>
<td>2</td>
</tr>
<tr>
<td>3l</td>
<td>$3P\sqrt{P}$</td>
<td>$3P\sqrt{P}$</td>
<td>$O(P)$</td>
<td>3</td>
</tr>
<tr>
<td>3l-wc</td>
<td>$3P\sqrt{P}$</td>
<td>$3P$</td>
<td>$O(P)$</td>
<td>3</td>
</tr>
</tbody>
</table>

❖ Write combining

▶ Each worker writes all data to be shared in a single file.

Q. How does this compare with starling?
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

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

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!!!