Serverless Data Analytics with Flint

YOUNGBIN KIM AND JIMMY LIN
Large-scale analytical data processing

- Spark adoption is booming
- Many use cases
Large-scale analytical data processing

- Spark adoption is booming
- Many use cases
  - Requirement: Pre-installed on a cluster before they can be used for analytics
    - On-premise data center or cluster of virtual instances in the cloud
Large-scale analytical data processing

- Problem: Cluster management can be difficult
  - monitoring the health of worker nodes
  - troubleshoot a variety of issues
  - fixing/replacing underperforming nodes

May not be feasible for many small startups/researchers with the limited resources!
Large-scale analytical data processing

- How about scaling?
Managed big data frameworks

- Current solution: Managed big data frameworks
- Example: Amazon Elastic Map Reduce (EMR)

Advantages:
- Reduces the burden of cluster management
- Save costs (automatically terminated)

Limitations:
- Time is wasted in cluster initialization/rescaling/teardown
- Need to choose the details of the managed cluster
- There are still management overheads & idle costs.
Serverless analytics

- Serverless analytics to the rescue!
Flint

- Flint: prototype execution engine for serverless PySpark
  - PySpark with serverless backend by simply specifying a config file
  - No costs for idle capacity
  - Simplicity
  - Use cases: ad hoc analytics and exploratory data analysis
Flint architecture

- Spark tasks are executed in AWS Lambda
- Intermediate data are held in Amazon’s Simple Queue Service (SQS)
- Reuses as many existing Spark components as possible
  - Query planning and optimization
  - Many different types of RDD transformations
Workflow

```
val sc = new SparkContext(conf)
val rdd = sc.cassandraTable(...)
  .map(...)  
  .filter(...)  
  .keyBy(...)  
  .reduceByKey(...)  
  .cache()
```
Flint architecture

- Spark Context
- Flint Scheduler Backend
- Client
- Amazon Web Services
- Intermediate Stage
- Final Stage
- Data Movement
- Control Flow
- Input Partition
- Input Partition
- Input Partition
- Output Partition
- Output Partition
- Flint Executor
- Flint Executor
- Flint Executor
- Flint Executor
- Flint Executor
- Queue
- Queue
- S3
- Lambda
- SQS
Flint architecture

- The Flint scheduler coordinates Flint executors to execute a particular physical plan
  - Function registration
  - Queue initialization
  - Serialization
  - Invocation using thread pool
  - Process the response from an executor
Flint executor

- Flint executor is a python process running inside an Amazon Lambda function
- Each serverless compute function invocation processes a single task
  - Simplifies the communication requirement between an executor and a driver
  - Less affected by the limitation of execution time
Remote storage for shuffling

- No permanent storage
- Small ephemeral disk space (~512 MB)
- Execution time limitation
  => Cannot guarantee the Flint executors from the previous stage are still alive to pass data
- Communication between Lambda functions

- Amazon’s Simple Queue Service (SQS)
  - highly-scalable
  - reliable
Experimental Evaluation

- A Spark cluster running the Databricks Unified Analytics Platform (Standard)
- 11 m4.2xlarge instances (one driver and ten workers) - 80 vCores
- 80 max concurrent invocations (~ 80 vCores)
Experimental Evaluation

- NYC taxi dataset (215 GB)
- Pick-up and drop-off dates/time, trip distance, payment type, tip amount
- Queries inspired by an exploratory data analysis task described in a popular blog post by Todd Schneider
Experimental Evaluation

- Q0: Line count
- Q1: Taxi drop-offs at the Goldman Sachs headquarters (hourly aggregation)

```scala
arr = src.map(lambda x: x.split(',')) 
.filter(lambda x: inside(x, goldman)) 
.map(lambda x: (get_hour(x), 1)) 
.reduceByKey(add, 30) 
.collect()
```
Experimental Evaluation

- Q2: Similar to Q1, but for Citigroup headquarters
- Q3: Goldman Sachs taxi drop-offs with tips greater than $10
- Q4: Cash vs. credit card payments
- Q5: Yellow taxi vs. green taxi, monthly aggregation
- Q6: Effect of precipitation on taxi trips
Experimental Evaluation

<table>
<thead>
<tr>
<th>Query Latency (s)</th>
<th>Estimated Cost (USD)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Flint</td>
</tr>
<tr>
<td>0</td>
<td>101 [93 - 109]</td>
</tr>
<tr>
<td>1</td>
<td>190 [186 - 197]</td>
</tr>
<tr>
<td>2</td>
<td>203 [201 - 205]</td>
</tr>
<tr>
<td>3</td>
<td>165 [161 - 169]</td>
</tr>
<tr>
<td>4</td>
<td>132 [122 - 142]</td>
</tr>
<tr>
<td>5</td>
<td>159 [142 - 177]</td>
</tr>
<tr>
<td>6</td>
<td>277 [272 - 281]</td>
</tr>
</tbody>
</table>

TABLE I
QUERY LATENCY AND COST COMPARISONS.
Experimental Evaluation

- Q1: Taxi drop-offs at the Goldman Sachs headquarters (hourly aggregation)
- Q3: Goldman Sachs taxi drop-offs with tips greater than $10
  tradeoff of concurrency between the latency and the cost
Lambda Limitations

- Most serverless platforms currently have several limitations
  - Memory size (e.g. 3008 MB for AWS)
  - Execution time limitation (5 ~ 9 minutes)
  - Cold start problem
Lambda Limitations

Execution time limitation
Lambda Limitations

Figure 3.4: PySpark internals on standard Spark cluster [14]. Workers rely on Java/Scala Spark running on JVM
Lambda Limitations

Other constraints

- Memory (3008 MB)
- Request size: 6 MB
  - Metadata
- Response size: 6 MB
  - Collect
Related Work

- **Iris:**
  - The origin of Flint (A course project, UWaterloo, Fall 2016)
  - Distributed computation framework supporting a subset of Spark API
  - In-browser data analytics backed by serverless backend

- **Amazon Athena**
  - Per-query pricing with zero idle costs
  - Only supports SQL
  - Presto distributed SQL engine

- **Databricks Serverless**
  - Automatically managed pools of cloud resources
    - auto-configured & auto-scaled
Related Work

- PyWren
  - Framework built from scratch on top of serverless compute functions and persistent storage

- Qubole Spark on Serverless
  - Ported the existing Spark executor infrastructure onto AWS Lambda, whereas Flint is a from-scratch implementation
    - Communication model
    - AWS Lambda limitations
Future Work

- Intensive shuffling tasks
- Robustness
- Higher level libraries (e.g. MLlib)