Efficient Management of Ephemeral Data in Serverless Computing

Patrick Stuedi
IBM Research
Serverless Analytics

- Serverless frameworks are increasingly being used for interactive analytics
Serverless Analytics

- Serverless frameworks are increasingly being used for **interactive analytics**
  - Exploit massive parallelism with large number of serverless tasks
Challenge: Data Sharing

- Serverless analytics involve multiple stages of execution.
- Serverless tasks need an efficient way to communicate *intermediate data* between different stages.

**Ephemeral data**

User query & input data → Result
In traditional analytics,. 

- Ephemeral data is exchanged directly between the tasks.
In serverless analytics..

- Direct communication between serverless tasks is difficult
  - Tasks are short lived and stateless

mapper_0  mapper_1  mapper_2  mapper_3

?          reducer_0

reducer_1
In serverless analytics..

- Direct communication between serverless tasks is difficult
  - Tasks are short lived and stateless
In serverless analytics..

- Direct communication between serverless tasks is difficult
  - Tasks are short lived and stateless

mapper_0
mapper_1
mapper_2
reducer_0
reducer_1
Requirements for Ephemeral Storage

1. High performance for a wide range of object sizes

Thus, an ephemeral storage system should support high throughput and low latency.
Requirements for Ephemeral Storage

1. High performance for a wide range of object sizes
2. Fine grain, pay what you use resource billing

Example of performance-cost tradeoff for a serverless video analytics application

Finding the Pareto optimal resource allocation is non-trivial...and gets harder with multiple jobs.
Requirements for Ephemeral Storage

1. High performance for a wide range of object sizes
2. Fine grain, pay what you use resource billing
3. Fault-tolerance
Requirements for Ephemeral Storage

1. High performance for a wide range of object sizes
2. Fine grain, pay what you use resource billing
3. Fault-tolerance

Existing cloud storage systems do not meet the elasticity, performance and cost demands of serverless analytics jobs
Serverless Analytics: 2 Projects

1. Serverless Spark
   - Add serverless properties to Spark (elasticity, on-demand scaling, etc)

2. Pocket: elastic ephemeral storage for the cloud
   - Improve applications running on serverless frameworks in the cloud (AWS λ, IBM Cloud Functions, etc)
Project 1: Spark Serverless Motivation

Example: Sorting 100GB

Spark/On-Premise++: Running Apache Spark on a High-Performance Cluster using RDMA and NVMe Flash, Spark Summit'17
Why is it so hard?

• **Scheduler**: when to best add/remove resources?

• **Container startup**: may have to dynamically spin up containers

• **Storage overheads**:
  - Input data needs to be fetched from remote storage (e.g., S3)
  - Intermediate needs to be temporarily stored on remote storage (S3, Redis)
I/O Overhead: Sorting 100GB

Shuffle overheads are significantly higher when intermediate data is stored remotely
Instead of improving performance of serverless frameworks, can we...

* ...add serverless properties to Spark?
  - Elasticity, on-demand scaling
  - Sharing of a Spark resources (compute, memory) among users

* Use Case:
  - Enable sharing of Spark deployment among many users in a company, research lab, etc.

* Challenge:
  - Maintain original Spark performance
Spark Serverless: What’s missing?

SparkContext

Task

Executor

Worker

In-memory cache (MemStore)

Filesystem
Spark Serverless: What’s missing?

SparkContext

Task
Executor
Worker

In-memory cache (MemStore)
Filesystem

Scale-down event

loosing Intermediate data/state!
Disaggregation of Ephemeral Data

- SparkContext
- Task
- Executor
- Worker
- Network Fabric
- DRAM tier
- Flash tier
- Disaggregated Storage
Disaggregation of Ephemeral Data

- SparkContext
- Worker
- Executor
- Task
- DRAM tier
- Flash tier
- Network Fabric
- Disaggregated Storage
- Scale-down event
- Data/state available when executor disappears
Spark-Serverless Architecture

Driver -> HCS/Scheduler
- send schedule
- register
- send job DAG

HCS/Scheduler -> Executor
- assign application
- register
- launch task
- register

Executor -> Intermediate data
- send job DAG
- register

Intermediate data -> Flash tier
- register

Intermediate data -> DRAM tier
- register

Apache Crail
- crail.apache.org

Diagram notes:
- Crail manages the data flow between the Driver, HCS/Scheduler, Executor, Flash tier, and DRAM tier.
Architecture Overview

Driver

HCS/Scheduler

send schedule

register

send job DAG

launch task

assign application

register

Intermediate data

Apache Crail

Flash tier

DRAM tier

Intermediate data

Executor

crail.apache.org
Architecture Overview

Driver

HCS/Scheduler

send schedule

register

send job DAG

register

assign application

register

Intermediate data

Apache Crail

crail.apache.org

Flash tier

DRAM tier

Executor

Intermediate data

Intermediate data

launch task
Architecture Overview

Driver

HCS/Scheduler

send schedule

register

send job DAG

Intermediate data

Apache Crail

Flash tier

DRAM tier

Intermediate data

Executor

Executors get dynamically re-assigned to apps/drivers

crail.apache.org
Video: Putting things together

Application 1: GridSearch

Application 2: SQL TPC-DS

Application 3: SQL TPC-DS

HCL Scheduler

Resource view: fraction of resources each app consumes

Executor view: Which app an executor currently runs
Let’s look at performance...

• Compute cluster size: 8 nodes: IBM Power8 Minsky
• Storage cluster size: 8 nodes, IBM Power8 Minsky
• Cluster hardware:
  – DRAM: 512 GB
  – Storage: 4x 1.2 TB NVMe SSD
  – Network: 10Gb/s Ethernet, 100Gb/s RoCE
  – GPU: NVIDIA P100, NVLink
• Workload
  – SQL: TCP-DS
Efficiently disaggregating ephemeral data enables Spark cluster to grow and shrink without a performance cost.
Spark-SQL: TPC-DS (Query #3) (short running query)

Short-running queries benefit from the shared (already-up) Spark deployment
Project 2: 
Serverless Analytics in the Cloud

• Context: Serverless analytics in the cloud
  - AWS λ, IBM Cloud Functions, Azure Functions

• Current practice for storing ephemeral data:
  - S3:
    • High latencies for small data sets
  - Redis, AWS ElasticCache:
    • Inconvenient for storing large objects
    • No dynamic scaling
    • Costly (DRAM)

• Can we use Apache Crail?
  - Not as is, no dynamic scaling
Pocket

- An elastic distributed data store for ephemeral data sharing in serverless analytics

Pocket dynamically rightsizes storage resources (nodes, media) in an attempt to find a spot with a good performance price ratio.
How Pocket works

1. Throughput allocation
2. Capacity allocation
3. Choice of storage tier(s)

Optional hints about job:
- Latency sensitivity
- Maximum # of concurrent tasks
- Total ephemeral data capacity
- Peak aggregate bandwidth required

Controller
app-driven resource allocation & scaling

Metadata server(s)
request routing

Storage server
CPU
Net
HDD

Storage server
CPU
Net
Flash

Storage server
CPU
Net
DRAM

Storage server
CPU
Net
DRAM
Pocket: Resource Utilization

• Comparing Pocket to S3 and Redis

Pocket achieves similar performance to Redis but uses NVMe Flash

<table>
<thead>
<tr>
<th>MapReduce sort job hints</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Ephemeral capacity</td>
<td>100 GB</td>
</tr>
<tr>
<td>Latency sensitive</td>
<td>False</td>
</tr>
<tr>
<td>Aggregate peak throughput</td>
<td>100 Gb/s</td>
</tr>
</tbody>
</table>
Autoscaling a Pocket Cluster

The controller elastically scales resources to meet the requirements of multiple jobs

<table>
<thead>
<tr>
<th>Job hints</th>
<th>Job1: Sort</th>
<th>Job2: Video analytics</th>
<th>Job3: Sort</th>
</tr>
</thead>
<tbody>
<tr>
<td>Latency sensitive</td>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>Ephemeral data capacity</td>
<td>10 GB</td>
<td>6 GB</td>
<td>10 GB</td>
</tr>
<tr>
<td>Aggregate throughput</td>
<td>3 GB/s</td>
<td>2.5 GB/s</td>
<td>3 GB/s</td>
</tr>
</tbody>
</table>
Conclusion

- Serverless frameworks are increasingly being used for **interactive analytics**
- Efficiently managing ephemeral data is important for serverless analytics

2 Projects:

- **Spark-serverless**
  - Add support to Spark for fine-grained on-demand scaling
  - Permit growing/shrinking of Spark executors by disaggregating shuffle data using Apache Crail

- **Pocket**
  - Elastic distributed data store for ephemeral data sharing in serverless analytics
  - Can be used together with frameworks like AWS λ, IBM Cloud Functions, etc.
References

• Pocket: Ephemeral Storage for Serverless Analytics, OSDI’18
• Navigating Storage for Serverless Computing, Usenix ATC’18
• Crail: A High-Performance I/O Architecture for Distributed Data Processing, IEEE Data Bulletin 2017
• Running Apache Spark on a High-Performance Cluster Using RDMA and NVMe Flash, Spark Summit’17
• Serverless Machine Learning using Crail, Spark Summit’18
• Apache Crail, http://crail.apache.org
Thanks to

Ana Klimovic, Yawen Wang, Michael Kaufmann, Adrian Schuepbach, Jonas Pfefferle, Animesh Trivedi, Bernard Metzler

Slides (Intro & Pocket) from Pocket presentation (OSDI’18, Ana Klimovic)