Understanding Ephemeral Storage for Serverless Analytics

Ana Klimovic*, Yawen Wang*, Christos Kozyrakis*, Patrick Stuedi†, Jonas Pfefferle†, Animesh Trivedi†

*Stanford University, †IBM Research
Introduction

- Serverless computing enables launching short-lived tasks with *high elasticity* and *fine-grain resource billing*.
- This makes serverless computing appealing for *interactive analytics*.
Introduction

- Serverless computing enables launching short-lived tasks with *high elasticity* and *fine-grain resource billing*
- This makes serverless computing appealing for *interactive analytics*
- **The challenge:** tasks (‘lambdas’) need an efficient way to communicate intermediate results
In traditional analytics...

- Ephemeral data is exchanged directly between tasks
In traditional analytics...

- Ephemeral data is exchanged directly between tasks
Direct communication between lambdas is difficult:
  - Lambdas are short-lived and stateless
  - Users have no control over lambda scheduling
In serverless analytics...

- Direct communication between lambdas is difficult:
  - Lambdas are short-lived and stateless
  - Users have no control over lambda scheduling

mapper\textsubscript{0} \hspace{1cm} reducer\textsubscript{0}
mapper\textsubscript{1} \hspace{1cm} ?
mapper\textsubscript{2} \hspace{1cm} reducer\textsubscript{1}
mapper\textsubscript{3}
In serverless analytics...

- The natural approach is to share data through a common data store
In serverless analytics...

- The natural approach is to share data through a common data store

mapper\_0
mapper\_1
mapper\_2
mapper\_3

reducer\_0
reducer\_1
In serverless analytics...

- The natural approach is to share data through a **common data store**

However, it is not clear whether existing storage systems are a good fit for ephemeral data sharing.
Questions:

1. What are the ephemeral I/O characteristics of serverless analytics applications?

2. How do applications perform using existing systems (e.g., S3, Redis) for ephemeral I/O?

3. What storage media (DRAM, Flash, HDD) satisfies I/O requirements at the lowest cost?
1. Application Ephemeral I/O Patterns

Application Type

- Distributed
- Compilation

**Ephemeral I/O Throughput:** Write (dotted), Read (solid)

- High throughput and IOPS due to high parallelism: lambdas each compile independent files

- Archiving and linking lambdas are serialized as they depend on previous lambdas → low parallelism, low I/O rate

**Ephemeral Data Capacity**

- 0.85 GB

<table>
<thead>
<tr>
<th>Application Type</th>
<th>Ephemeral Data Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distributed</td>
<td>0.85 GB</td>
</tr>
<tr>
<td>Compilation</td>
<td></td>
</tr>
</tbody>
</table>
1. Application Ephemeral I/O Patterns

Application Type
- Distributed
- Compilation
- MapReduce

Ephemeral I/O Throughput: Write (dotted), Read (solid)

Ephemeral Data Capacity
- 0.85 GB
- 100 GB

High throughput due to high I/O intensity and parallelism (up to 7.5 GB/s with 500 concurrent lambdas)
1. Application Ephemeral I/O Patterns

<table>
<thead>
<tr>
<th>Application Type</th>
<th>Ephemeral Data Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distributed Compilation</td>
<td>0.85 GB</td>
</tr>
<tr>
<td>MapReduce</td>
<td>100 GB</td>
</tr>
<tr>
<td>Video Analytics</td>
<td>6 GB</td>
</tr>
</tbody>
</table>
1. Application Ephemeral I/O Patterns

Thus, an ephemeral storage system should support high throughput and low latency.

Wide range of I/O sizes (bytes to 100s of MBs)
2. Existing Storage Systems

We focus on three different categories:
2. Existing Storage Systems

We focus on three different categories:

1. **Cloud object storage system (e.g. Amazon S3)**
   - Pay only for the capacity and throughput you use
   - Resources managed by cloud provider

2. **In-memory key-value store (e.g. Redis)**
   - High performance at the higher cost of DRAM
   - Manually select and scale storage instance

3. **Distributed Flash-based data store (e.g. Crail-ReFlex)**
   - Use Flash vs. DRAM for high bandwidth at lower cost
   - Manually select and scale storage instances
2. Existing Storage Systems

We focus on three different categories:

1. **Cloud object storage system (e.g. Amazon S3)**
   - Pay only for the capacity and throughput you use
   - Resources managed by cloud provider

2. **In-memory key-value store (e.g. Redis)**
   - High performance at the higher cost of DRAM
   - Manually select and scale storage instance
2. Existing Storage Systems

We focus on three different categories:

1. **Cloud object storage system (e.g. Amazon S3)**
   - Pay only for the capacity and throughput you use
   - Resources managed by cloud provider

2. **In-memory key-value store (e.g. Redis)**
   - High performance at the higher cost of DRAM
   - Manually select and scale storage instance

3. **Distributed Flash-based data store (e.g. Crail-ReFlex)**
   - Use Flash for high bandwidth at lower cost
   - Manually select and scale storage instances
Latency sensitivity

- Distributed compilation job shows some sensitivity to latency due to small I/Os.

As concurrency increases, job runtime becomes dominated by the sequential portion of the application.
The impact of application parallelism

Distributed compilation (gg-cmake) with up to 650 concurrent lambdas using S3

Figure based on Fig. 6 in “A thunk to remember: make -j1000 (and other jobs) on functions-as-a-service infrastructure (preprint).” Fouladi, S., et al.
The impact of application parallelism

Distributed compilation (gg-cmake) with up to 650 concurrent lambdas using Redis

- Ephemeral read I/O
- Compute
- Ephemeral write I/O

But job runtime is the same as with S3

Each lambda spends less time on I/O

Figure based on Fig. 6 in “A thunk to remember: make -j1000 (and other jobs) on functions-as-a-service infrastructure (preprint).” Fouladi, S., et al.
The impact of application parallelism

Distributed compilation (gg-cmake) with up to 650 concurrent lambdas using Redis

Runtime is limited by dependencies on compute-bound lambdas

Figure based on Fig. 6 in “A thunk to remember: make -j1000 (and other jobs) on functions-as-a-service infrastructure (preprint).” Fouladi, S., et al.
The impact of application parallelism

Distributed compilation (gg-cmake) with up to 650 concurrent lambdas using Redis

- Ephemeral read I/O
- Compute
- Ephemeral write I/O

Applications with inherently limited parallelism have lower ephemeral I/O throughput demands

Figure based on Fig. 6 in “A thunk to remember: make -j1000 (and other jobs) on functions-as-a-service infrastructure (preprint).” Fouladi, S., et al.
High I/O intensity

MapReduce sort (100 GB) demands high throughput
High I/O intensity

MapReduce sort (100 GB) demands high throughput

S3 does not provide sufficient throughput

S3 also does not provide sufficient IOPS scalability
High I/O intensity

MapReduce sort (100 GB) demands high throughput

Average Time per Lambda (s)

S3
250 lambdas

Redis
500 lambdas

Crail-ReFlex
1000 lambdas

Original input/output data I/O

Compute

Ephemeral data I/O

Similar performance with Flash and DRAM
High I/O and compute intensity

Video analytics has both high I/O and compute intensity
3. Choice of storage media

- Compare throughput:capacity ratios of DRAM, Flash, HDD

<table>
<thead>
<tr>
<th>Storage Media</th>
<th>Throughput (GB/s)</th>
<th>Capacity (GB)</th>
<th>Throughput:Capacity Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRAM</td>
<td>20</td>
<td>64</td>
<td>0.3</td>
</tr>
<tr>
<td>Flash</td>
<td>3.2</td>
<td>500</td>
<td>0.006</td>
</tr>
<tr>
<td>Disk</td>
<td>0.7</td>
<td>6000</td>
<td>0.0001</td>
</tr>
</tbody>
</table>

**Diagram:**

- gg-cmake
- Sort100GB
- video-analytics
3. Choice of storage media

- Compare throughput:capacity ratios of DRAM, Flash, HDD

<table>
<thead>
<tr>
<th>Storage Media</th>
<th>Throughput (GB/s)</th>
<th>Capacity (GB)</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRAM</td>
<td>20</td>
<td>64</td>
<td>0.3</td>
</tr>
<tr>
<td>Flash</td>
<td>3.2</td>
<td>500</td>
<td>0.006</td>
</tr>
<tr>
<td>Disk</td>
<td>0.7</td>
<td>6000</td>
<td>0.0001</td>
</tr>
</tbody>
</table>

Application throughput:capacity ratios are in DRAM - Flash regimes
3. Choice of storage media

- Compare throughput:capacity ratios of DRAM, Flash, HDD

<table>
<thead>
<tr>
<th>Storage Media</th>
<th>Throughput (GB/s)</th>
<th>Capacity (GB)</th>
<th>Throughput:Capacity Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRAM</td>
<td>20</td>
<td>64</td>
<td>0.3</td>
</tr>
<tr>
<td>Flash</td>
<td>3.2</td>
<td>500</td>
<td>0.006</td>
</tr>
<tr>
<td>Disk</td>
<td>0.7</td>
<td>6000</td>
<td>0.0001</td>
</tr>
</tbody>
</table>

Application throughput:capacity ratios are in DRAM - Flash regimes

Using Flash vs. DRAM, jobs achieve similar performance at lower cost per bit
Putting it all together...

- Ephemeral storage wishlist for serverless analytics:
  - ★ High throughput and IOPS
  - ★ Low latency, particularly important for small requests
  - ★ Fine-grain, elastic scaling to adapt to elastic application load
  - ★ Automatic rightsizing of resource allocations
  - ★ Low cost, pay-what-you-use

- Existing systems provide some but not all of these properties
Conclusion

- Our analysis motivates the design of an ephemeral storage service that supports automatic, fine-grain storage capacity and throughput allocation
- Ephemeral I/O requirements depend on a job’s latency sensitivity, inherent parallelism and its I/O vs. compute intensity
- Flash is an appealing storage media for ephemeral I/O performance-cost requirements