Locus
Shuffling *Fast* and *Slow*
on Serverless Architecture

Qifan Pu
Shivaram Venkataraman
Ion Stoica
Serverless Computing
Serverless Analytics

Launch short-lived cloud workers with transparent elasticity and fine-grain usage billing

User function $F()$

Data

Results

i.e., Map()
Serverless Analytics

So far, a great fit for embarrassingly parallel analytics

- ExCamera (NSDI’17): video encoding
- PyWren (SoCC’17): scaling python functions
- NumPyWren: large-block matrix computation
- Stanford gg compiler: distributed compiling
- AWS Redshift Spectrum: ETL
General Serverless Analytics

To enable general analytics, one needs to implement shuffle.
- Key operation for `join()` and `groupby()`
- 70+% of TPC-DS queries use shuffle
- Most expensive operation in production clusters

How to perform cost-efficient shuffle on a serverless architecture?
An Shuffle Example: Cloud Sort

Lowest cost to sort 100TB of data

• Record: Apache Spark, 50min / $144
Traditional Analytics

Data communicated directly between servers that execute tasks.
Traditional Analytics

Data communicated directly between servers that execute tasks.
How to shuffle in serverless?

Tasks are short-lived
How to shuffle in serverless?

Tasks are short-lived

Shuffling directly between tasks is difficult in serverless.
How to shuffle in serverless?

How about using S3?
- Many frameworks already use it for input/results.
- Cheap capacity and elastic bandwidth
How to shuffle in serverless?

Shuffle with S3

Server
mapper task

Server
mapper task

Server
mapper task

mapper task

reducer task

reducer task

Compute

S3
How to shuffle in serverless?

Shuffle with S3

How much does it cost for CloudSort?
How to shuffle in serverless?

Shuffle with S3

How many requests does it take?
An Shuffle Example: Cloud Sort

- Total number of transfers:
  - Assuming 1GB serverless containers
  - Num partition/merge tasks = 100TB/1GB = $10^5$
  - Number of files: $10^{10} = 10$ billion files
An Shuffle Example: Cloud Sort

- **Total number of transfers:**
  - Assuming 1GB serverless containers
  - Num partition/merge tasks = 100TB/1GB = $10^5$
  - Number of files: $10^{10} = 10$ billion files
  - $0.000005/\text{op} \rightarrow \$50k$
An Shuffle Example: Cloud Sort

Partition Task

merge task

merge task

merge task

Bottlenecked at 4400 ops/sec!

Takes $\frac{10^{10}}{4400}$ seconds = 26 days
An Shuffle Example: Cloud Sort

AWS S3

ElastiCache (Redis)

Azure Blob

Azure Cache

Google Cloud Storage

Google MemoryStore

Bottlenecked at 4400 ops/sec!

S3 lacks cheap and elastic IOPS.
An Shuffle Example: Cloud Sort

Require 158 large (r5.24xlarge) instances!
Costs $1.6K/hour.

Capacity alone is 10x more expansive than record.
An Shuffle Example: Cloud Sort

Partition Task

Partition Task

Partition Task

merge task

merge task

merge task

... 

... 

... 

Require 158 large (r5.24xlarge) instances!

Redis capacity is too expensive for large intermediate data.
## Cloud Storage

<table>
<thead>
<tr>
<th>Service</th>
<th>Capacity</th>
<th>IOPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slow Storage</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>Fast Storage</td>
<td>Low</td>
<td>High</td>
</tr>
</tbody>
</table>

*Can we leverage the strengths of two storages types?*
Locus

- Hybrid *slow* and *fast* cloud storage to achieve cost-efficient shuffle performance
  - Intuition: use fast storage to absorb IOPS and create bigger chunks for slow storage.
Hybrid sort (100TB)

Round1: 5TB $\rightarrow$ partition $\rightarrow$ 5TB $\rightarrow$ merge $\rightarrow$ 5TB

num reqs = num mergers = $10^5$

Clean cache after each round

Round2: 5TB $\rightarrow$ partition $\rightarrow$ 5TB $\rightarrow$ merge $\rightarrow$ 5TB

Round20: 5TB $\rightarrow$ partition $\rightarrow$ 5TB $\rightarrow$ merge $\rightarrow$ 5TB

- Total number of S3 reqs: $20 \times 10^5$ (vs. $10^{10}$)
- Total memory cache needed: 5TB (vs. 100TB)
Hybrid sort (100TB)

Round1: 5TB partition 5TB merge 5TB num reqs = num mergers = 10^5

Clean cache after each round

Round2: 5TB partition 5TB merge 5TB

... 5TB partition 5TB merge 5TB

Round20: 5TB partition 5TB merge 5TB

• Spark record: 50min + $144
• Locus: 50min + $163
Hybrid sort (100TB)

Round1: 5TB partition 5TB merge 5TB
num reqs = num mergers = $10^5$

Clean cache after each round

Round2: 5TB partition 5TB merge 5TB

... 5TB partition 5TB merge 5TB

Round20: 5TB partition 5TB merge 5TB

Locus achieves same performance with 5X less resource.

Locus: $50mm + $105$
Locus

• Hybrid *slow* and *fast* cloud storage to achieve cost-efficient shuffle performance
  – Intuition: use fast storage to absorb IOPS and create bigger chunks for slow storage.

• When to use hybrid shuffle?
  – Alongside, many other configurations
    – Amount of fast cache
    – Level of parallelism
    – Worker size
Navigating cost-performance with different compute and storage configurations is difficult.
Navigating cost-performance with different compute and storage configurations is difficult.

– How many serverless workers? Which size?
Navigating cost-performance with different compute and storage configurations is difficult.

- How many serverless workers? Which size?

![Graph showing the relationship between worker memory size and shuffle time for 20GB and 1TB storage configurations.](image)
Navigating cost-performance with different compute and storage configurations is difficult.

- How many serverless workers? Which size?
Performance model example: slow-storage only shuffle

\[ T = \max(T1, T2) \]

\[ T1 = \frac{S}{B \times p} \]

\[ T2 = \frac{s^2}{w^2 \times q} \]
Evaluation

- Implement Locus on top of PyWren
  - AWS Lambda
  - S3 (slow)
  - Redis (fast)

- Workloads:
  - CloudSort
  - TPC-DS
  - Big data benchmark

- Baseline:
  - PyWren
  - Apache Spark on VMs
  - Redshift
Evaluation

TPC-DS

Reduce resource usage by up to 59% while achieving comparable performance!
Related Works

Pocket (OSDI’18)

• New elastic store that scales to application demand
Locus

• By judiciously combining *slow* and *fast* cloud storage, one can achieve cost-efficient shuffle for serverless analytics.

• Locus achieves this by:
  – Design algorithms that leverage storage characteristics
  – A performance model that captures the performance and cost metrics of shuffle operations.

• 4×-500× performance improvements over baseline and reduce resource usage by up to 59% while achieving comparable performance with traditional analytics