MinFlow: High-performance and Cost-efficient Data Passing for I/O-intensive Stateful Serverless Analytics

Tao Li†, Yongkun Li†, Wenzhe Zhu†, Yinlong Xu†, John C. S. Lui‡

†University of Science and Technology of China
‡The Chinese University of Hong Kong

USENIX FAST 2024
Serverless computing benefits

- Low operational overhead
- Fine-grained "pay-as-you-go" billing (1ms)
- Fast scaling (<1s)

Serverless computing framework

- Separate computation and storage
- FaaS: containerized functions; BaaS: cloud storage (typically S3)
Background

Data analytics is a critical class of applications

- Adopt the BSP model
- Shuffle operation: all-to-all connection
- Facebook: More than 50% involve at least one shuffle (Riffle@Eurosys’18)
Serverless computing passes data via remote storage

- **Shuffle**: lots of read/write requests
- **Remote storage**: I/O request rate is limited
  - S3 request rate: 3.5k and 5.5k req/s for writes and reads
  - S3 request cost: 0.005/0.0004 USD$ per 1k PUTs/GETs
Serverless computing passes data via remote storage.

Data passing severely impedes the elasticity and economy of serverless analytics.

Remote storage: I/O request rate is limited

TeraSort JCT(s)

In/Output time
Shuffle time
Compute time

(a) #Func=400
(b) #Func=600
Key Issues

- How to improve the efficiency of data passing?
  - DAG topology, function scheduling, and transmission media

- How to optimize the topology to reduce data passing requests?
- How to decide the function scheduling plan to leverage over-provisioned local memory?
- How to build the high-performance and cost-efficient remote storage?
Existing Designs

- **Two-level Shuffle**
  - Use mesh-based two-level Shuffle to decrease the number of data passing requests
  - Starling@SIGMOD’20, Lambada@SIGMOD’20

How to optimize the topology to reduce data passing requests?
Existing Designs

- **Two-level Shuffle**
  - Use mesh-based two-level Shuffle to decrease the number of data passing requests (Starling@SIGMOD’20, Lambada@SIGMOD’20)

**Limitations:**

I. Bring about multiplied extra data volume due to the additional level
II. Cannot extend to a general multi-level network algorithm
Existing Designs

- Shuffle via intra-worker memory

- Reclaim over-provisioned memory in workers to localize intra-worker traffic
- Wukong@SoCC’20, FaaSFlow@ASPLOS’22

How to decide the function scheduling plan to leverage over-provisioned local memory?
Existing Designs

- Shuffle via intra-worker memory
  - Reclaim over-provisioned memory in workers to localize intra-worker traffic (Wukong@SoCC’20, FaaSFlow@ASPLOS’22)

**Limitations:**

I. Cross-worker traffic dominates and cannot be accelerated
II. Stragglers caused by slower remote storage
Existing Designs

- Shuffle via private storage

- Combine high-end and cheap remote storage media to achieve better trade-offs between performance and cost

- Pocket@OSDI’18, Locus@NSDI’19

How to build the high-performance and cost-efficient remote storage?
Existing Designs

- Shuffle via private storage
  - Combine high-end and cheap remote storage media to achieve better trade-offs between performance and cost (Pocket@OSDI’18, Locus@NSDI’19)

Limitations:
I. Entail high costs due to extra high-end storage
II. The network bandwidth of VMs is limited
Motivation and Main Idea

- Existing approaches: independent optimizations in different components
  - performance/cost/ease-of-use degradation

Diagram:

- Coordinator
- FaasS Platform
- BaaS Service
- Two-level Shuffle
- Intra-worker memory Shuffle
- Private storage Shuffle
Optimize **DAG topo, function scheduling, transmission media** in a unified way

**Motivation and Main Idea**

1. **Construct multi-level shuffle topology candidates**
   - Decrease requests
   - Facilitate scheduling

2. **Generate scheduling plan and optimize transmission media** for each candidate topology and output config candidates
   - Maximize traffic localization
   - Balance load
   - Avoid stragglers

3. **Model configs to select the optimal one**
   - Optimal configuration
How to construct the complete multi-level network topo space?

**Step 1.** Divide functions in the \( f_{level \ i} \) into \( g_i \) groups

\[
g_0 = N, g_L = 1, g_i = d_i \times g_{i+1} \text{ where } d_i \in N^+/\{1\}
\]

**Step 2.** Progressively converge groups

1. Function linking:
   - keep all-to-all connection

![Diagram of multi-level shuffle](image)
How to construct the complete multi-level network topo space?

Step 1. Divide functions in the $f_{level} i$ into $g_i$ groups

$$g_0 = N, g_L = 1, g_i = d_i \times g_{i+1} \text{ where } d_i \in N^+/\{1\}$$

Step 2. Progressively converge groups

1. Function linking:
   - keep all-to-all connection

2. Data passing:
   - shard data into continuous and equal-sized parts
How to construct the complete multi-level network topo space?

**Step 1.** Divide functions in the $f_{level}$ into $g_i$ groups

$$g_0 = N, g_L = 1, g_i = d_i \times g_{i+1} \text{ where } d_i \in N^+/\{1\}$$

**Step 2.** Progressively converge groups

1. **Function linking:**
   - keep all-to-all connection

2. **Data passing:**
   - shard data into continuous and equal-sized parts

**Progressively converging multi-level shuffle**

![Diagram of multi-level shuffle](image)

17
Topology Optimizer

How to select candidates among massive topologies?

Lightweight candidates selection by dynamic programming

- Find networks with the fewest edges under each possible number of levels $L$

Step 1. A series of optimization problems

For $L \in [1, p]$, \[
\begin{align*}
\text{minimize} & \quad N \times \sum_{i=0}^{L-1} d_i \\
\text{subject to} & \quad \prod_{i=0}^{L-1} d_i = N
\end{align*}
\]

Step 2. Bottom-up dynamic programming

- Solve all problems at once with low overhead

\[
\text{MinSum}(i, j) = \begin{cases} 
\min_{n|\text{i}} (n + \text{MinSum}(i/n, j - 1)) & j > 1 \\
\text{i} & j = 1 
\end{cases}
\]
Conclusions:
I. Topology Optimizer outputs topology candidates, each has the fewest edges under their corresponding number of levels L.

Step 1. A series
For \( L \in [1, 7] \)

Step 2. Bottom-up dynamic programming
• Solve all problems at once with low overhead

\[ \text{MinSum}(i, j) = \]
Function Scheduler

How to meet all the scheduling requirements?

Interleaved complete bipartite graphs partitioning

Scheduling requirements

- Maximize traffic localization
- Avoid transmission stragglers
- Ensure load balancing
Function Scheduler

➤ How to meet all the scheduling requirements?

Interleaved complete bipartite graphs partitioning

➤ Adjacent function levels: complete bipartite graphs (CBG)
  • Search the CBGs: schedule to the same worker

Maximize traffic localization

Local memory
How to meet all the scheduling requirements?

Interleaved complete bipartite graphs partitioning

Adjacent function levels: complete bipartite graphs

- Search the CBGs: schedule to the same worker
  Maximize traffic localization
- Adopt the same transmission media: within a communication level
  Avoid stragglers
- Employ interleaved local memory and remote storage: across communication levels
  Balance load
How to meet all the scheduling requirements?

Adjacent function levels: complete bipartite graphs

- Search the CBGs:
  - Schedule to the same worker
  - Adopt the same transmission media: within a communication level
  - Employ interleaved local memory and remote storage: across communication levels

Maximize traffic

Avoid stragglers

Balance load

Conclusions:

1. Function Scheduler outputs configuration candidates, each has the fewest edges under their corresponding number of levels and meets all scheduling requirements.

Function Scheduler
How to select the optimal configuration from config candidates?

Estimate data passing time of candidate configurations

- Model application characteristics and platform features to data passing time
  - Within a level: maximum of function and storage
  - Across levels: S3-based and memory-based

\[ T = 2 \times \sum_{i=0}^{L-1} \left\{ \max\left( \frac{D_i}{N_b f}, \frac{R_i}{q_s} \right), S3 \text{ levels.} \right\} \]

\[ \max\left( \frac{D_i}{M_b t}, \frac{R_i}{M_q t} \right), \text{memory levels.} \]
How to select the optimal configuration from L config candidates?

**Estimate candidate configuration’s data passing time**

- Model *data passing time* for S3-based and memory-based level
- The volume of intermediate data $D_i$: available at the runtime
  - Input data size and $D_i$: linear/non-linear but deterministic
  - Sample and profile
How to select the optimal configuration from L config candidates?

- Model data passing time for S3-based and memory-based levels
- The volume of intermediate data available at runtime
- Input data size and linear/non-linear but deterministic
- Sample and...

Estimate candidate configuration's data passing time

Conclusions:

1. Configuration Modeler outputs the optimal configuration and dispatch it to distributed coordinators
Experiments

Testbed:
- 10 Amazon EC2 m6i.x24large instances

Workloads
- TeraSort, TPC-DS, WordCount

Comparisons:
- **Baseline**: use single-level shuffle and transfer all data via S3
- **FaaSFlow**: adopt the intra-worker memory shuffle
- **Lambada**: employ the mesh-based two-level shuffle

<table>
<thead>
<tr>
<th>vCPU</th>
<th>Memory/GiB</th>
<th>Network bandwidth/Gib</th>
</tr>
</thead>
<tbody>
<tr>
<td>96</td>
<td>384</td>
<td>37.5</td>
</tr>
</tbody>
</table>
Shuffle Time and Storage Cost

Three workloads: 100GB/200GB input data size, 400/600 functions

Conclusions:
Under Terasort workload, compared to Baseline, FaaSFlow, and Lambada

I. **MinFlow** improves the shuffle speed up to 14.1X, 12.4X, and 3X respectively;

II. **MinFlow** reduces the storage cost up to 98.84%, 98.71%, and 86%, respectively
Job Completion Time

- Three workloads: 100GB/200GB input data size, 400/600 functions

Conclusions:
Under Terasort workload, compared to Baseline, FaaSFlow, and Lambada

1. **MinFlow** reduces the job completion time up to 85.16%, 83.25%, and 41.35%, respectively;
Load Balance

- Terasort workload: 200GB input data size, 600 functions

Conclusions:
1. All types of resource (i.e., CPU utilization, Memory utilization, Receive throughput, and Sent throughput) are load-balanced among workers.
Conclusions


• Progressively converging multi-level shuffle: minimize data passing requests
• Interleaved complete bipartite graph scheduling: maximize traffic localization
• Estimate data passing time: select the optimal configuration

More evaluation results and analysis are in the paper

The source code is at https://github.com/lt2000/MinFlow

• Reproduce all results with Amazon cloud: tens of hours and thousands of dollars
Thanks for your attention!

Q&A

Contact email:
little314@mail.ustc.edu.cn