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

中国科学技术大学
University of Science and Technology of China

香港中文大學
The Chinese University of Hong Kong

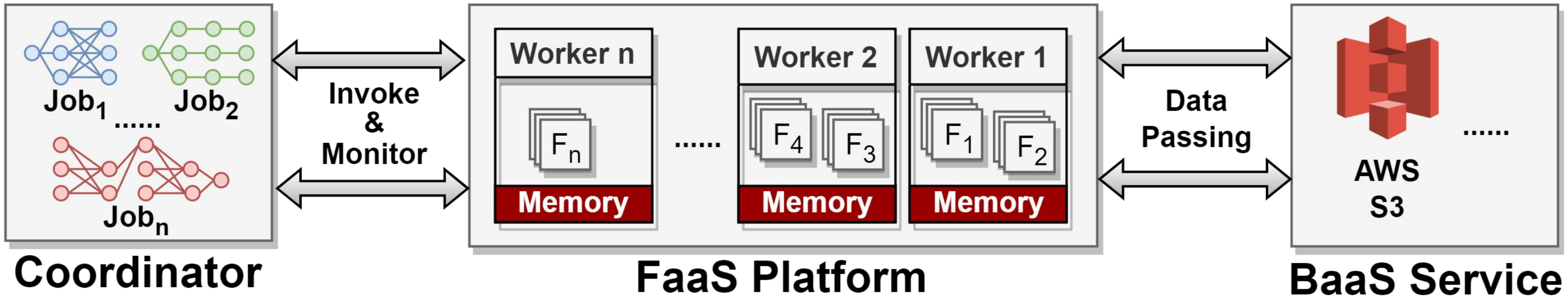
USENIX FAST 2024

Background

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

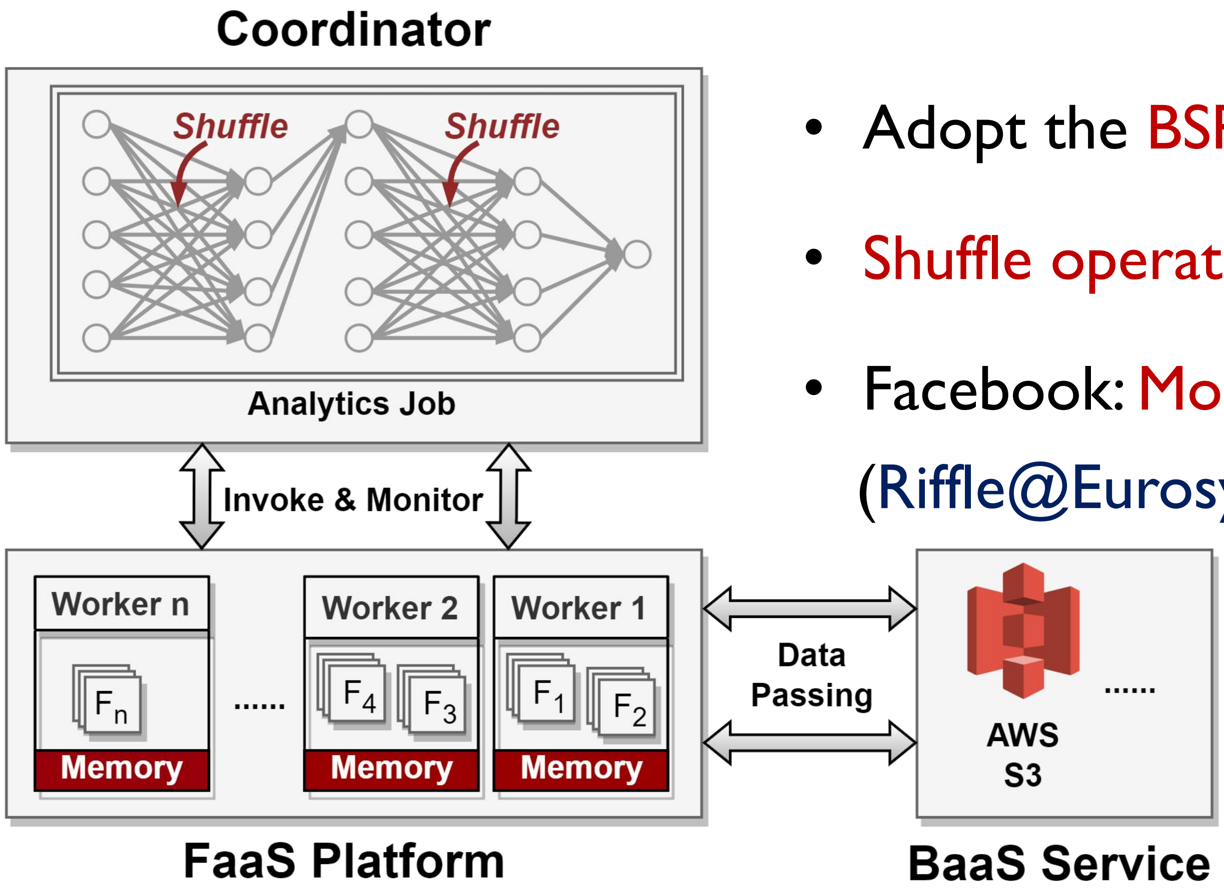
AWS Lambda
Google Cloud Functions
Azure Functions
IBM Cloud Functions

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



Background

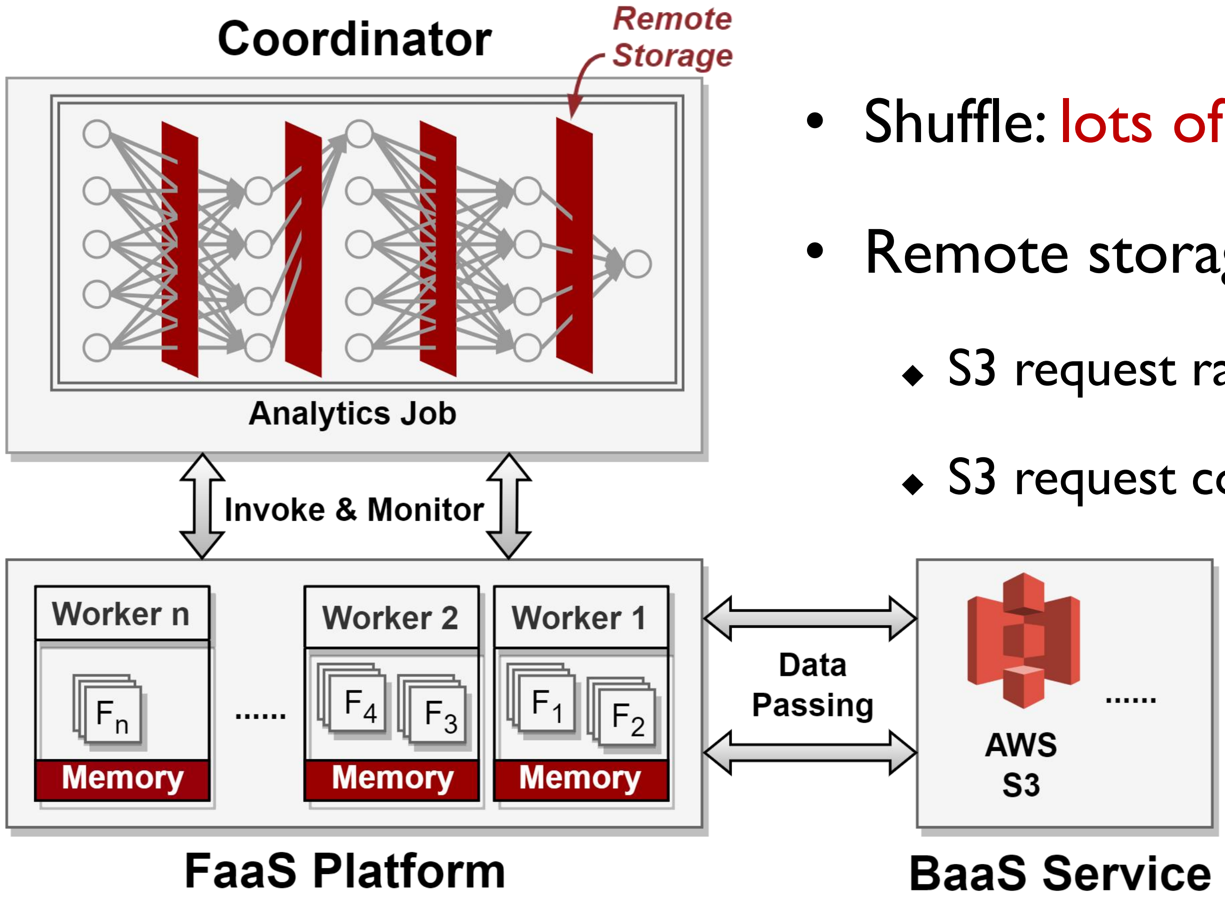
➢ Data analytics is a critical class of applications



- Adopt the BSP model
- Shuffle operation: all-to-all connection
- Facebook: More than 50% involve at least one shuffle (Riffle@Eurosys'18)

Background

➢ Serverless computing passes data via remote storage



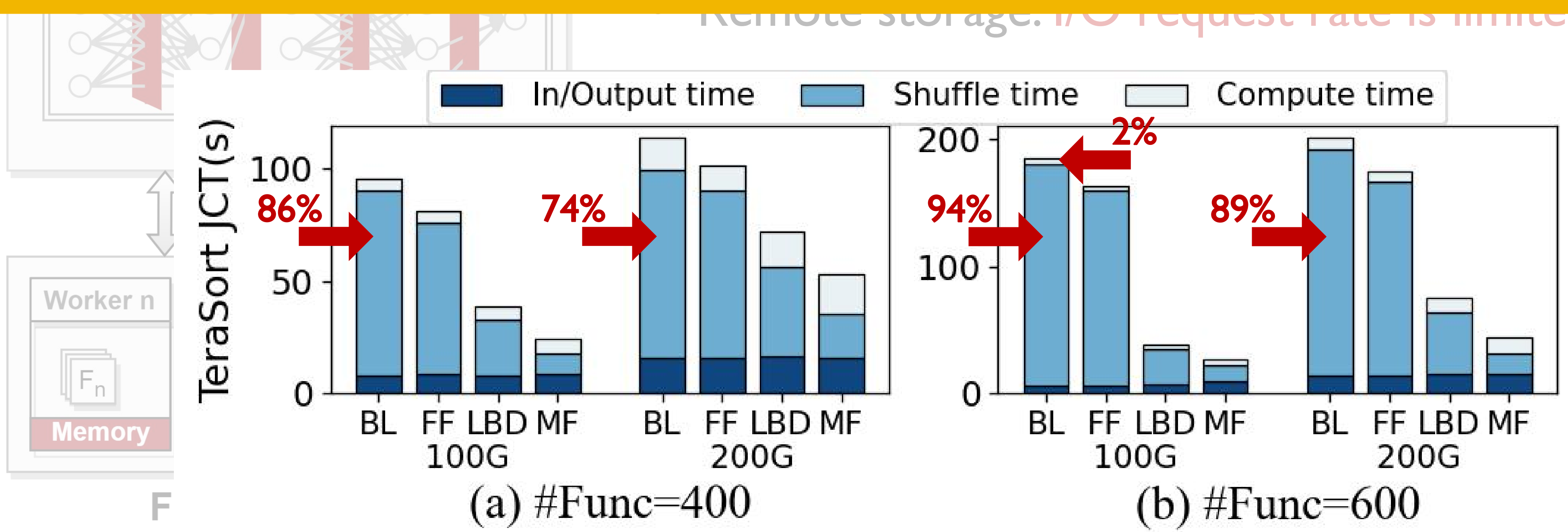
- Shuffle: lots of read/write requests
- Remote storage: I/O request rate is limited
  - S3 request rate: 3.5k and 5.5k req/s for writes and reads
  - S3 request cost: 0.005/0.0004 USD$ per 1k PUTs/GETs

Background

- 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



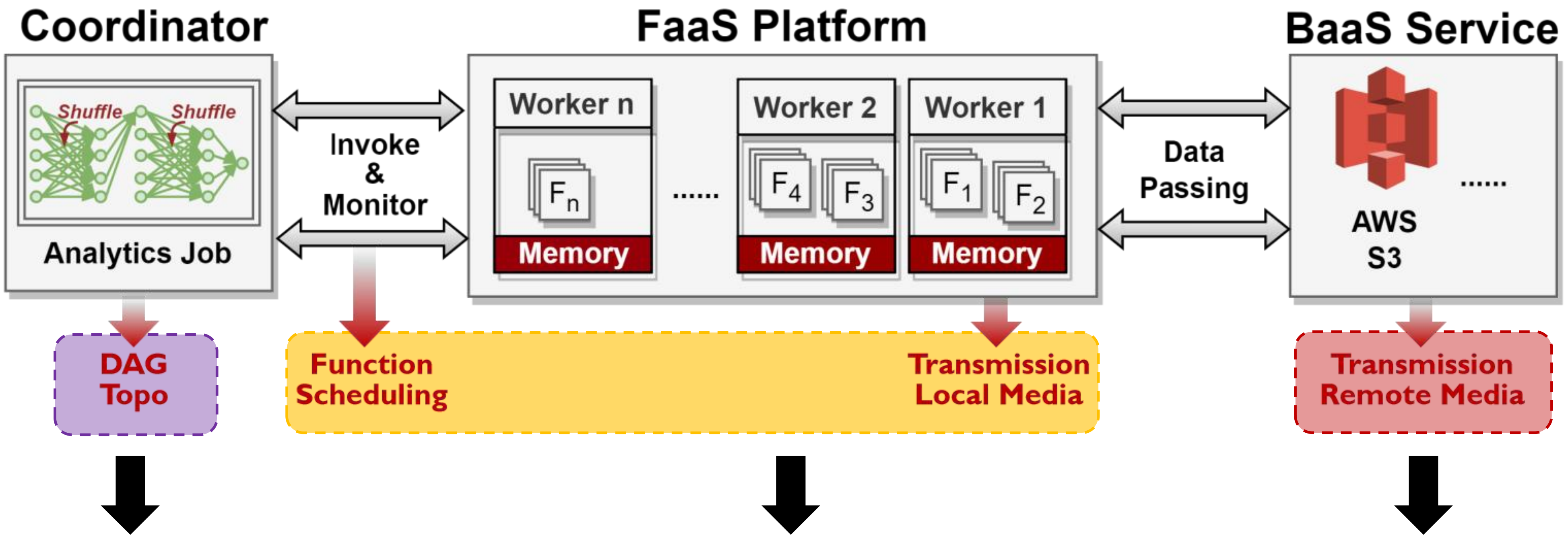
ınd reads

Ts/GETs

Key Issues

➢ How to improve the efficiency of data passing?

- **DAG topology**, **function scheduling**, and **transmission media**



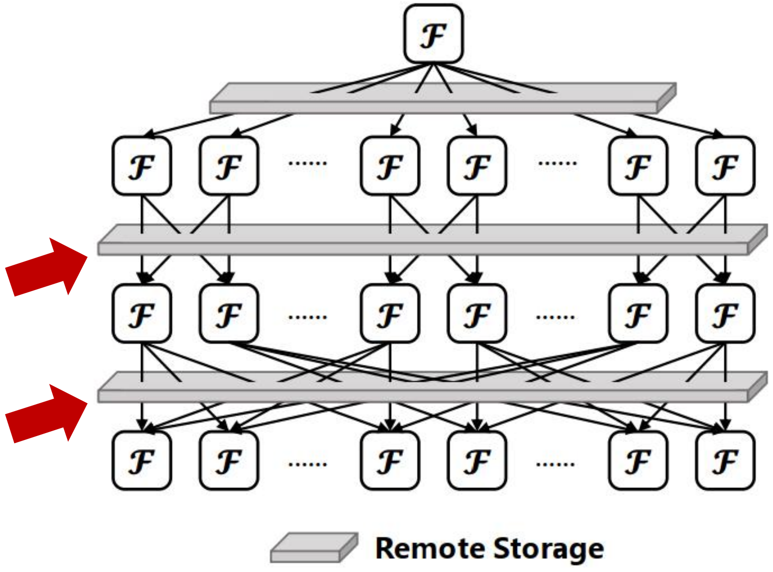
How to optimize the topology to reduce data passing requests?

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

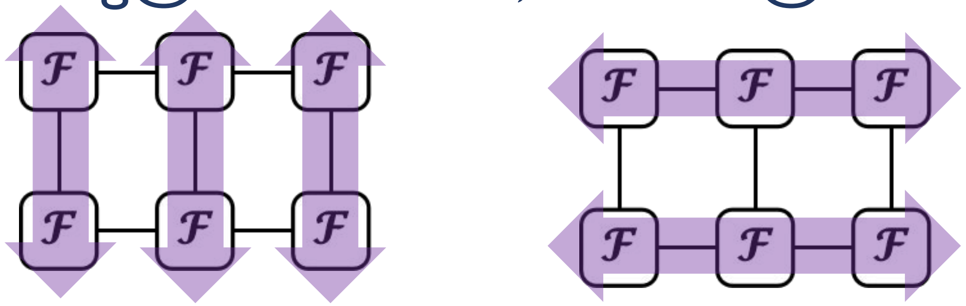
How to build the high-performance and cost-effcient remote storage?

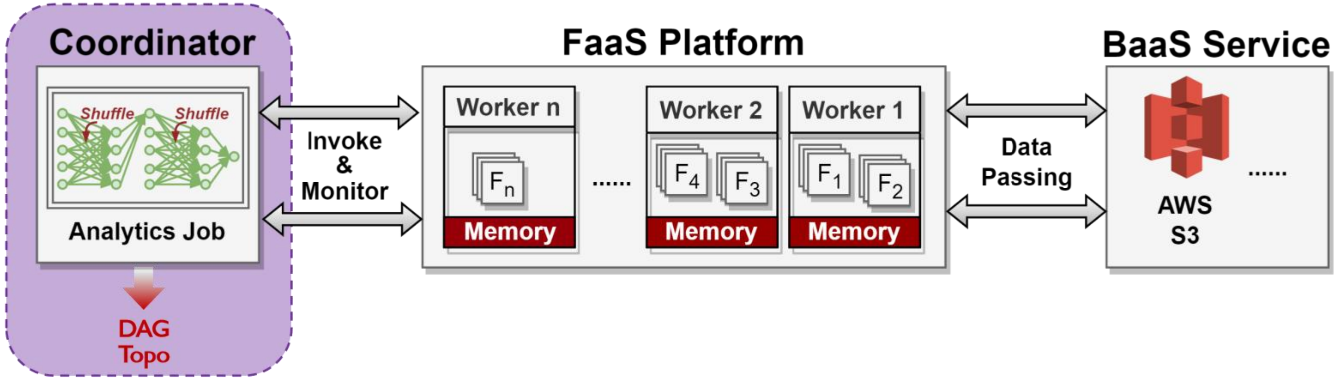
Existing Designs

➢ Two-level Shuffle



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





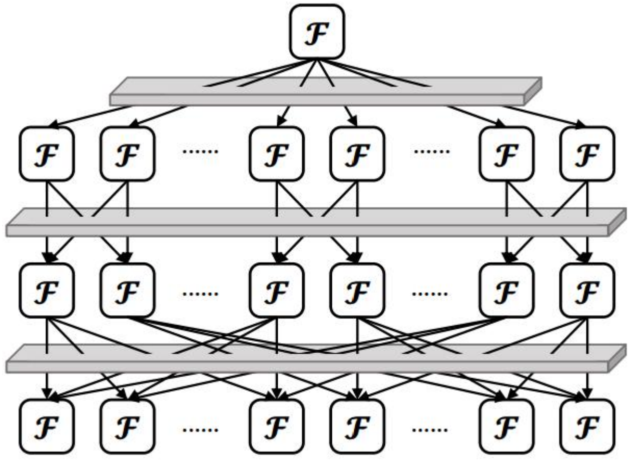
How to optimize the topology to reduce data passing requests?

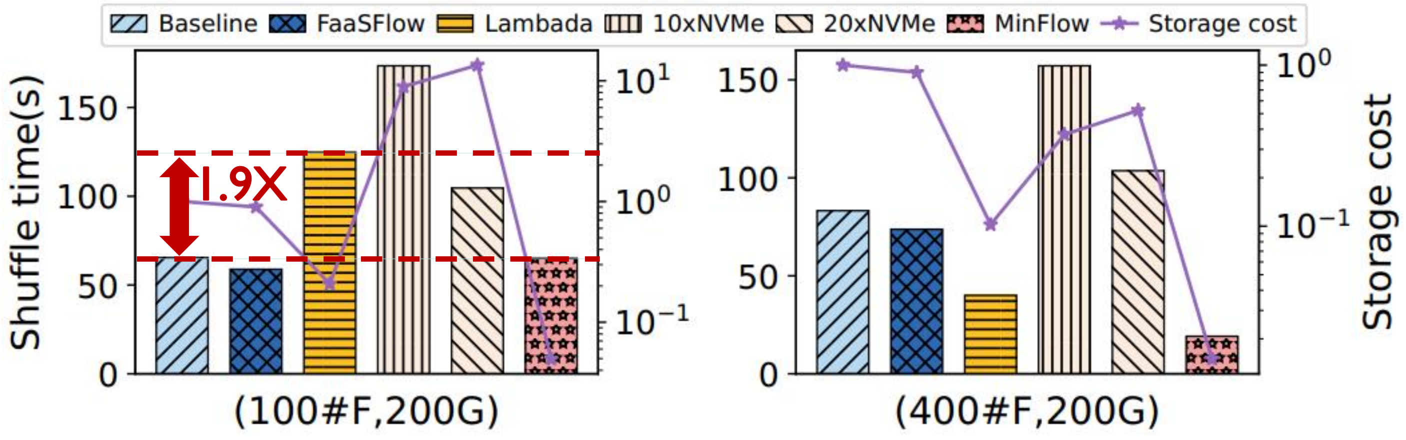
Existing Designs

- 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

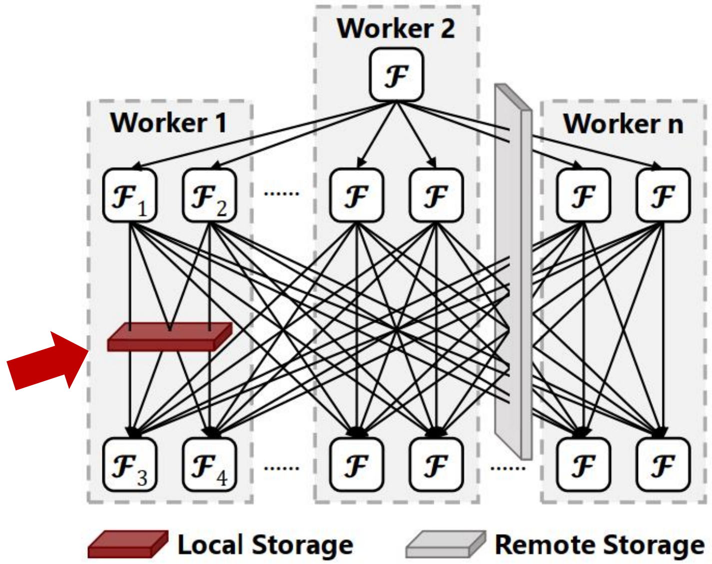




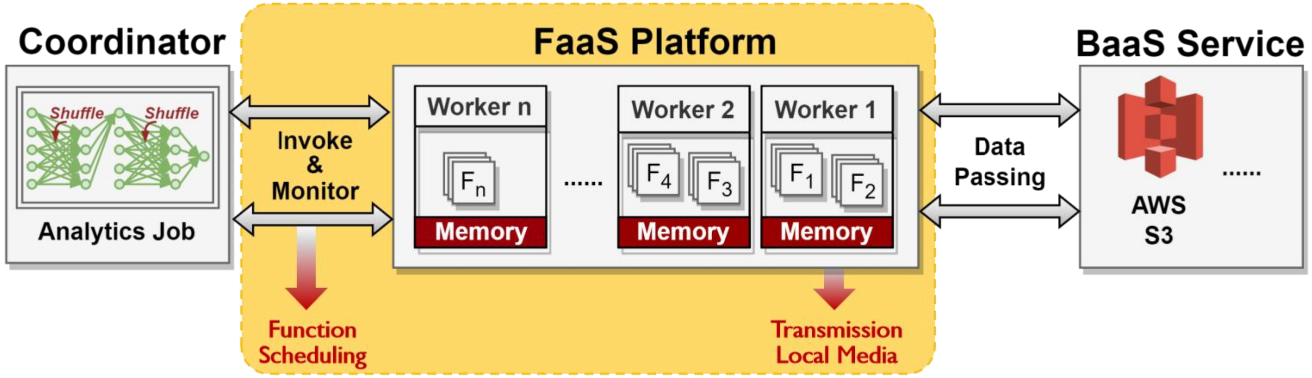
TeraSort Shuffle Time under Different Configurations

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?

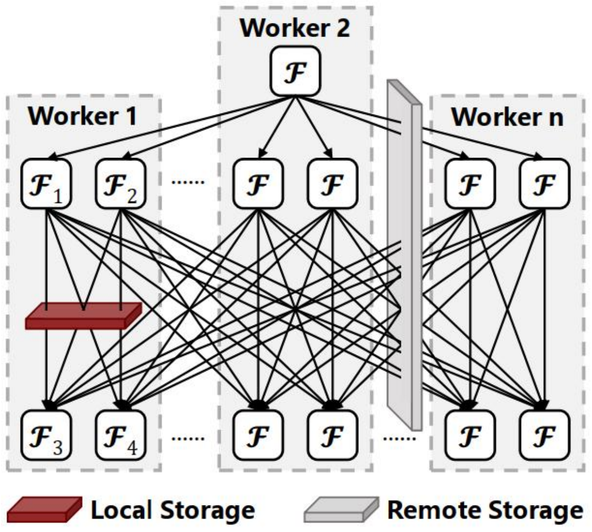
Existing Designs

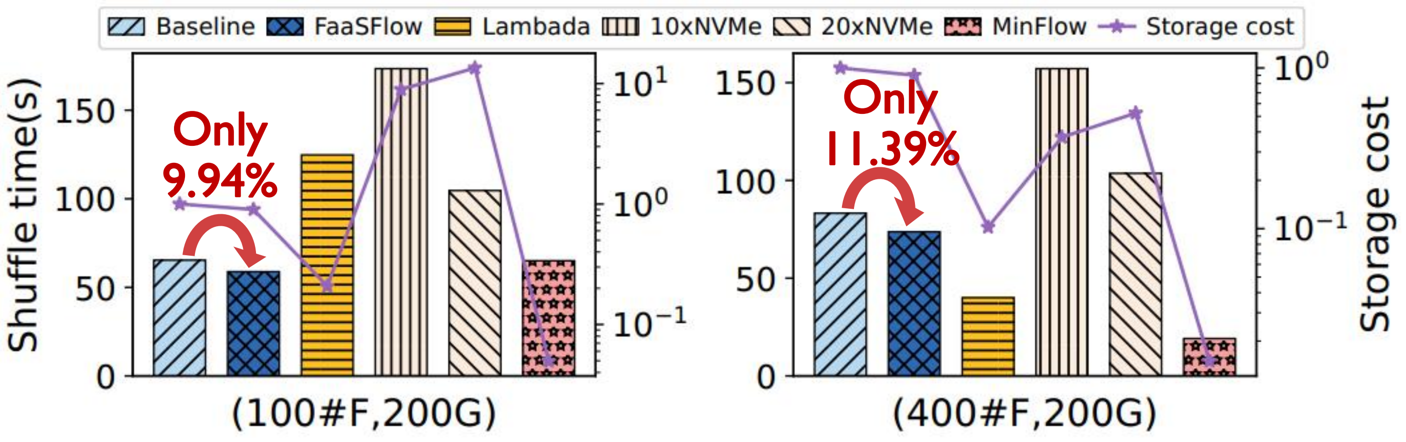
- Shuffle via intra-worker memory
  - Reclaim over-provisioned memory in workers to localize intra-worker traffic (Wukong@SoCC'20,FaaSFlow@ASPLOS'22)

Limitations:

I. Cross-worker traffic dominates and cannot be accelerated

II. Stragglers caused by slower remote storage

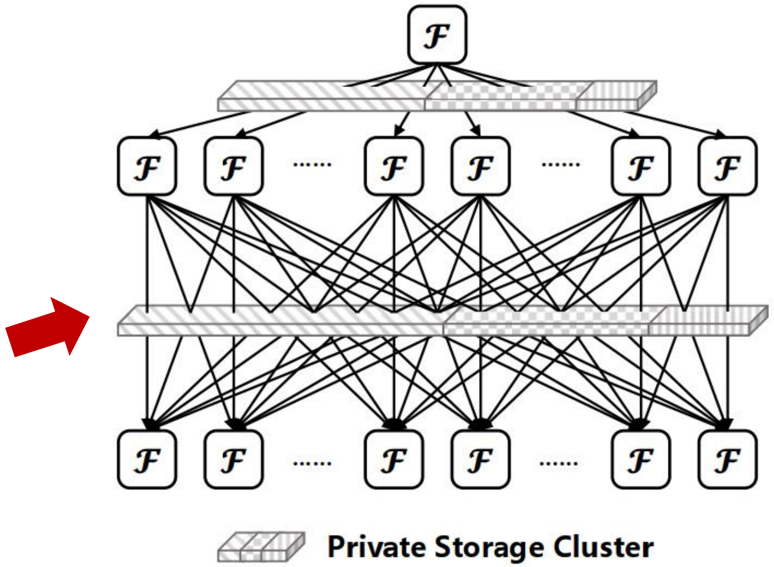




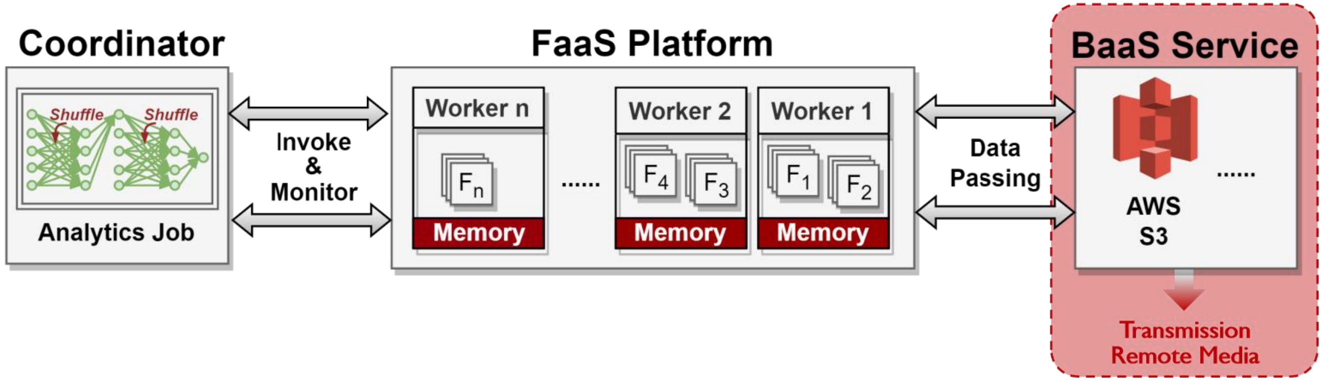
TeraSort Shuffle Time under Different Configurations

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



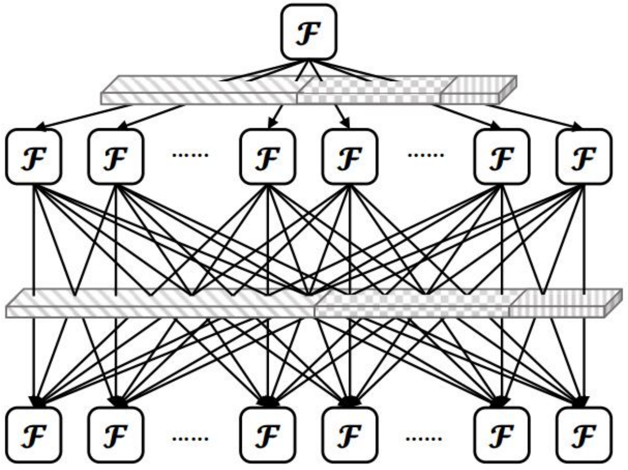
How to build the high-performance and cost-effcient remote storage?

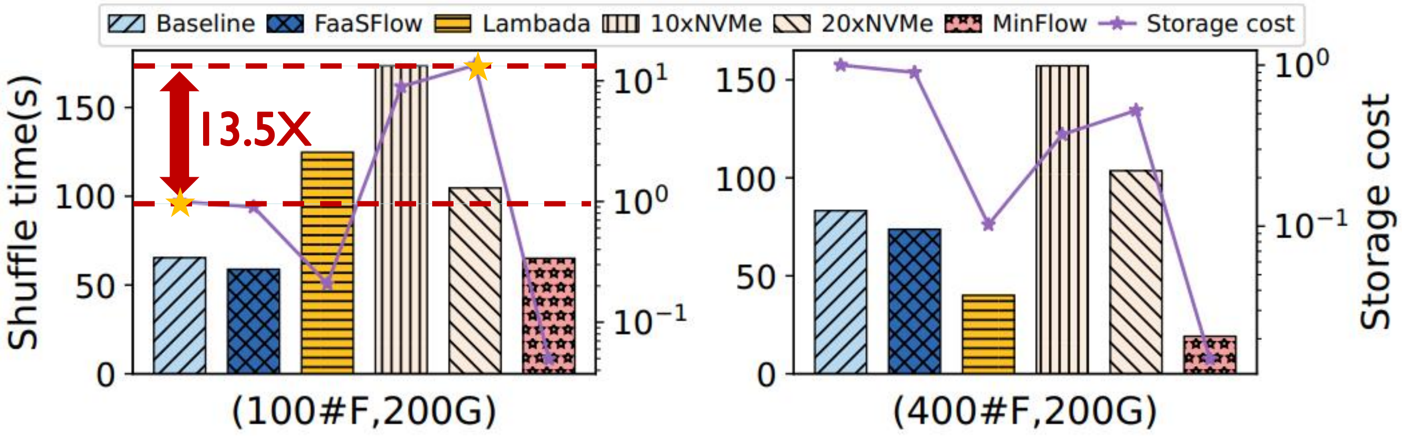
Existing Designs

- 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

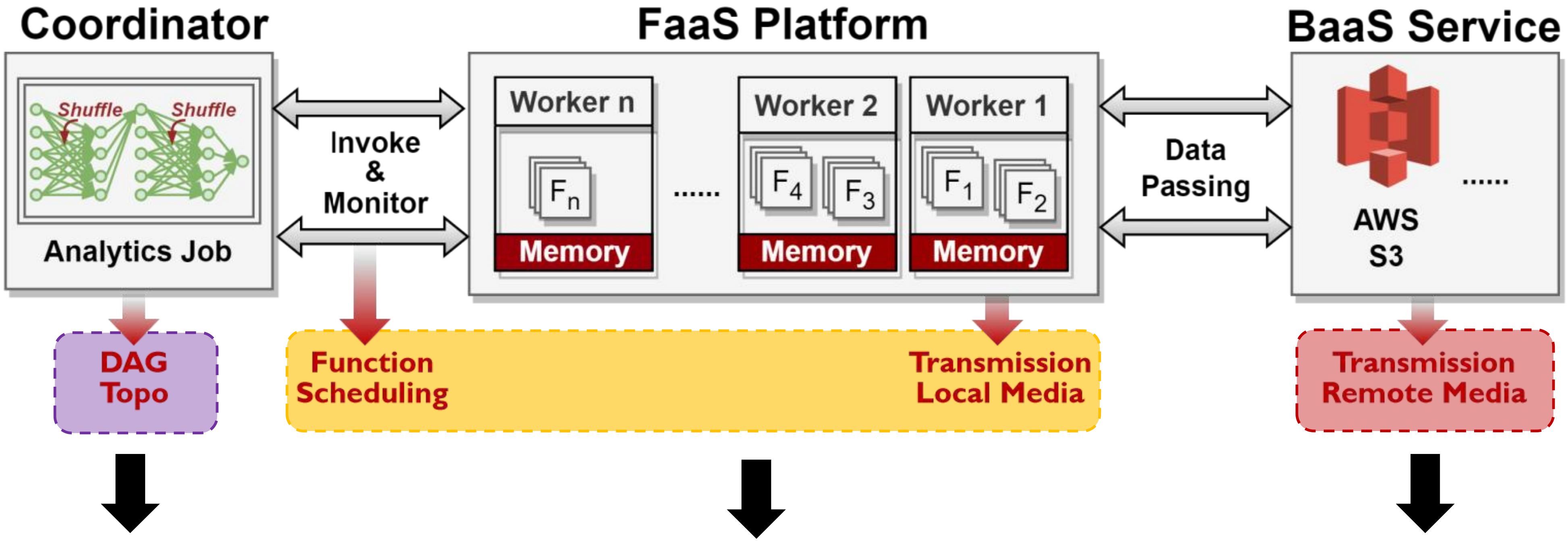




TeraSort Shuffle Time under Different Configurations

Motivation and Main Idea

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



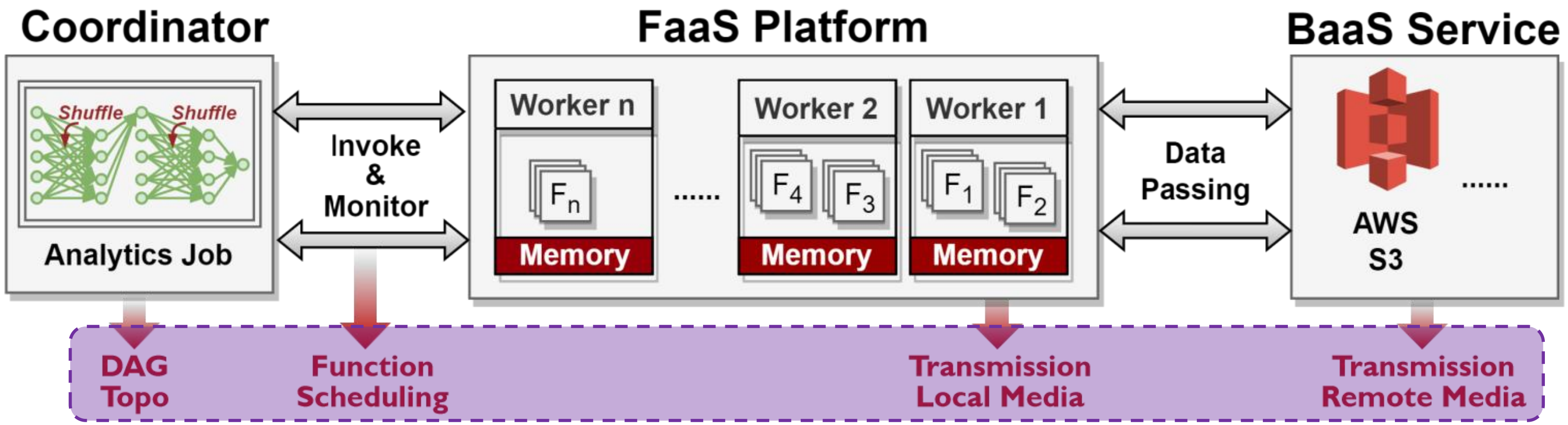
Two-level Shuffle

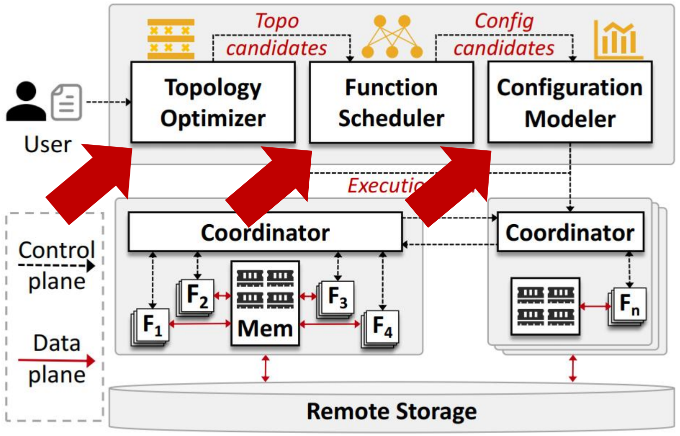
Intra-worker memory Shuffle

Private storage Shuffle

Motivation and Main Idea

Optimize DAG topo, function scheduling, transmission media in a unified way





Unified optimization

❶ Construct multi-level shuffle topology candidates

Decrease requests **Facilitate scheduling**

❷ Generate scheduling plan and optimize transmission media for each candidate topology and output config candidates

Maximize traffic localization **Balance load**

Avoid stragglers

❸ Model configs to select the optimal one form config candidates

Optimal configuration

Topology Optimizer

➢ How to construct the complete multi-level network topo space?

Progressively converging multi-level shuffle

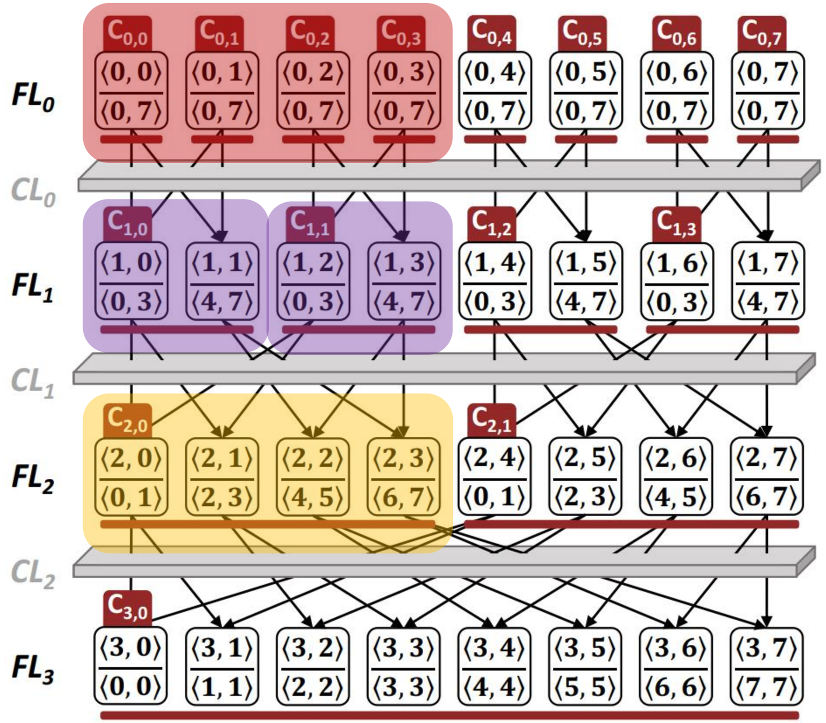
Step1. Divide functions in the $flevel\ i$ into g_i groups

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

Step2. Progressively converge groups

❶ Function linking:

keep all-to-all connection



Topology Optimizer

➢ How to construct the complete multi-level network topo space?

Progressively converging multi-level shuffle

Step1. Divide functions in the $flevel\ i$ into g_i groups

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

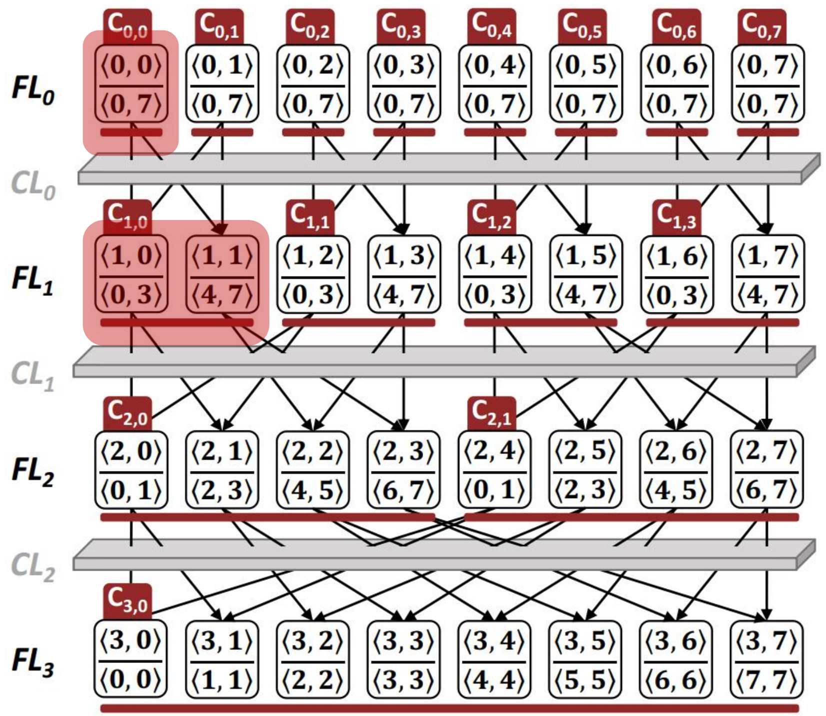
Step2. Progressively converge groups

❶ Function linking:

keep all-to-all connection

❷ Data passing:

shard data into continuous and equal-sized parts



Topology Optimizer

- ➢ How to construct the complete multi-level network topo space?

Progressively converging multi-level shuffle

Step1. Divide functions in the $flevel\ i$ into g_i groups

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

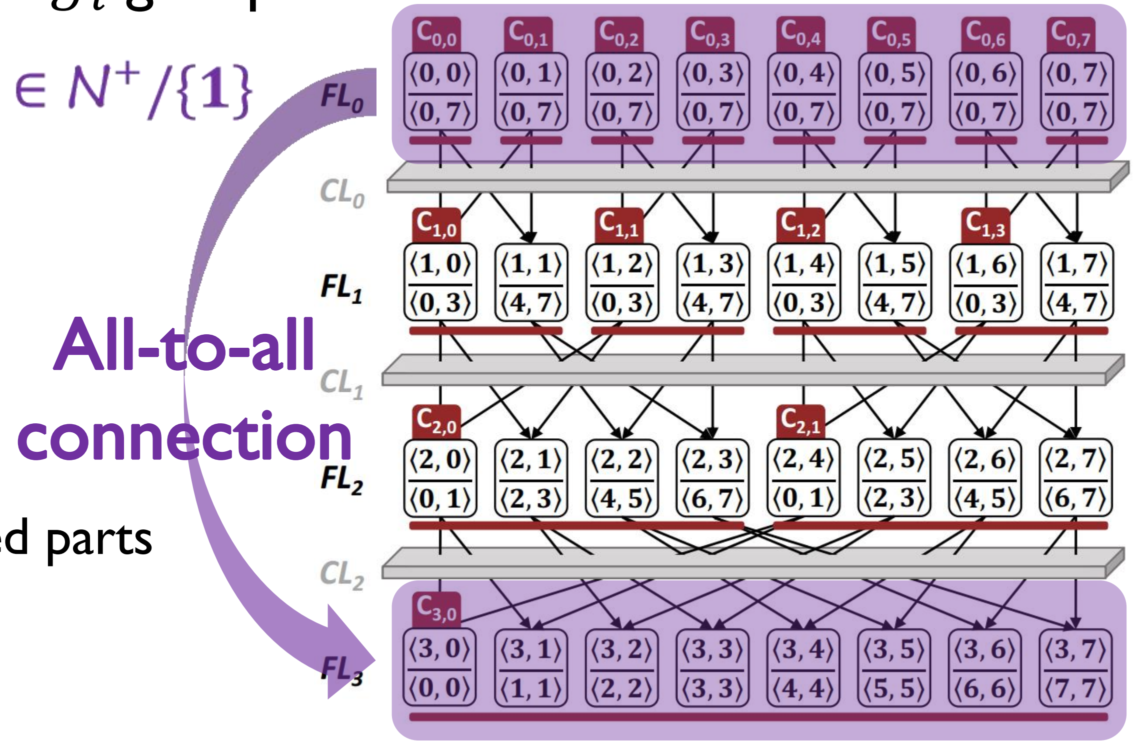
Step2. Progressively converge groups

❶ Function linking:

keep all-to-all connection

❷ Data passing:

shard data into continuous and equal-sized parts



Topology Optimizer

- How to select candidates among massive topologies?

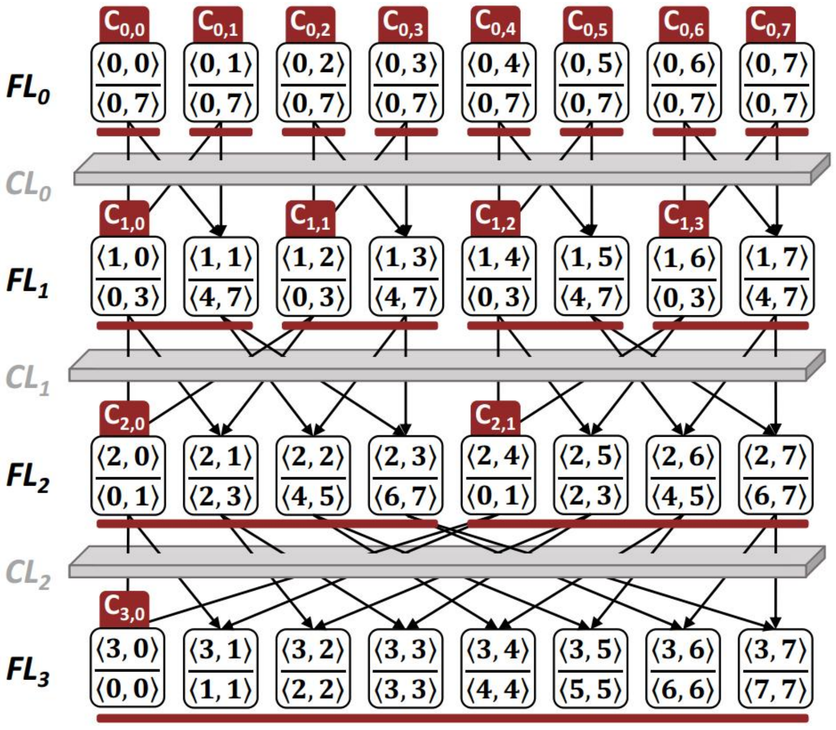
Lightweight candidates selection by dynamic programming

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

Step1. A series of optimization problems

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

Edges



Step2. Bottom-up dynamic programming

- Solve all problems at once with low overhead

$$MinSum(i,j) = \begin{cases} min_{n|i}\,(n + MinSum(i/n, j-1)) & j > 1 \\ i & j = 1 \end{cases}$$

Topology Optimizer

Conclusions:

I. Topology Optimizer outputs topology condidates, each has the fewest edges under their corresponding number of levels L

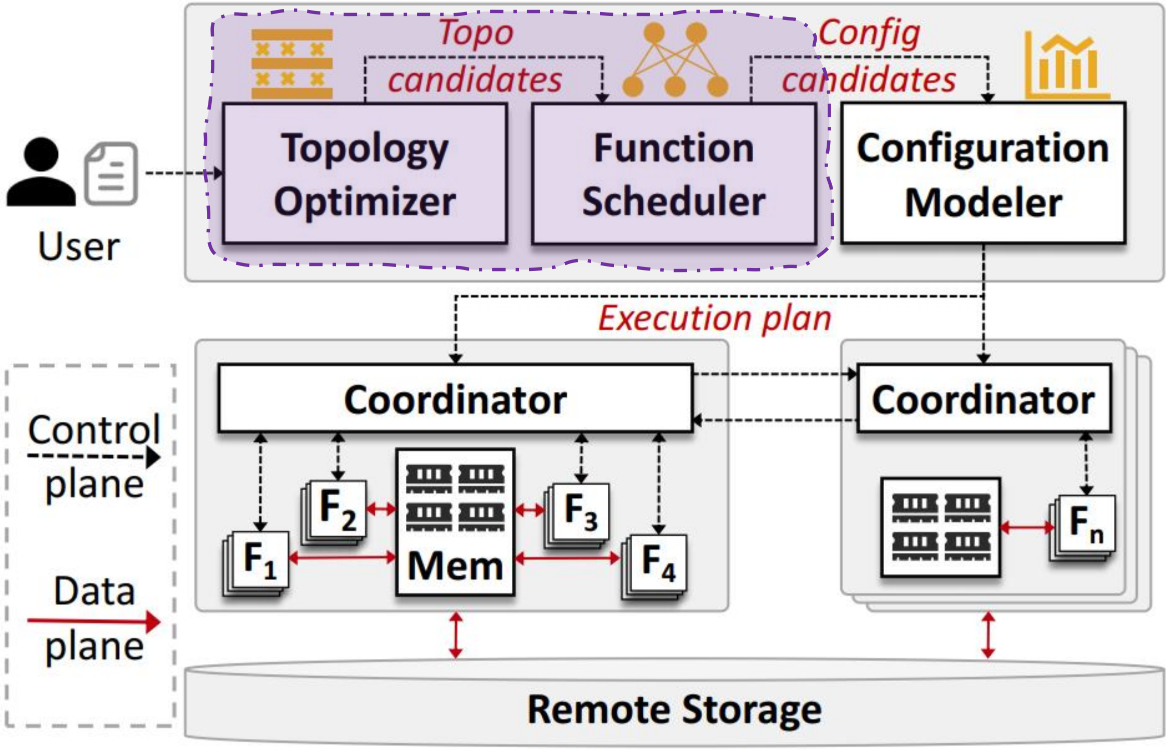
Step1. A series

For L ∈ [1,

Step2. Bottom-

- Solve all prob

$MinSum(i, j) =$

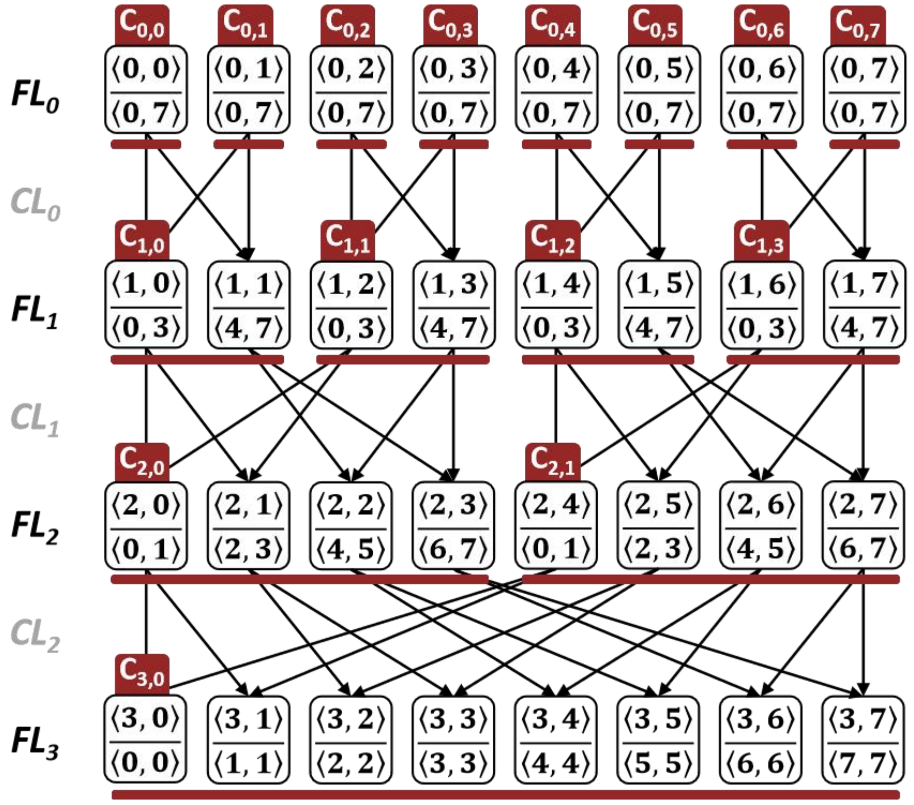


Function Scheduler

- 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

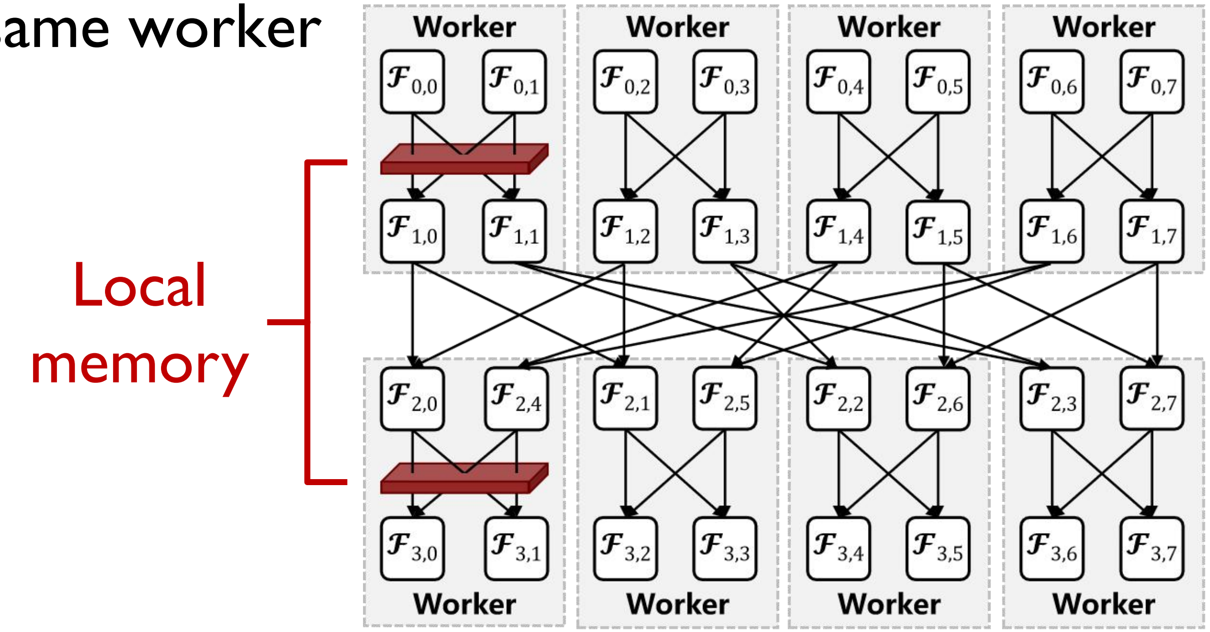
➢ How to meet all the scheduling requirements?

Interleaved complete bipartite graphs partitioning

➢ Adjacent function levels: complete bipartite graphs (CBG)

- Search the CBGs: schedule to the same worker

Maximize traffic localization



Function Scheduler

- How to meet all the scheduling requirements?

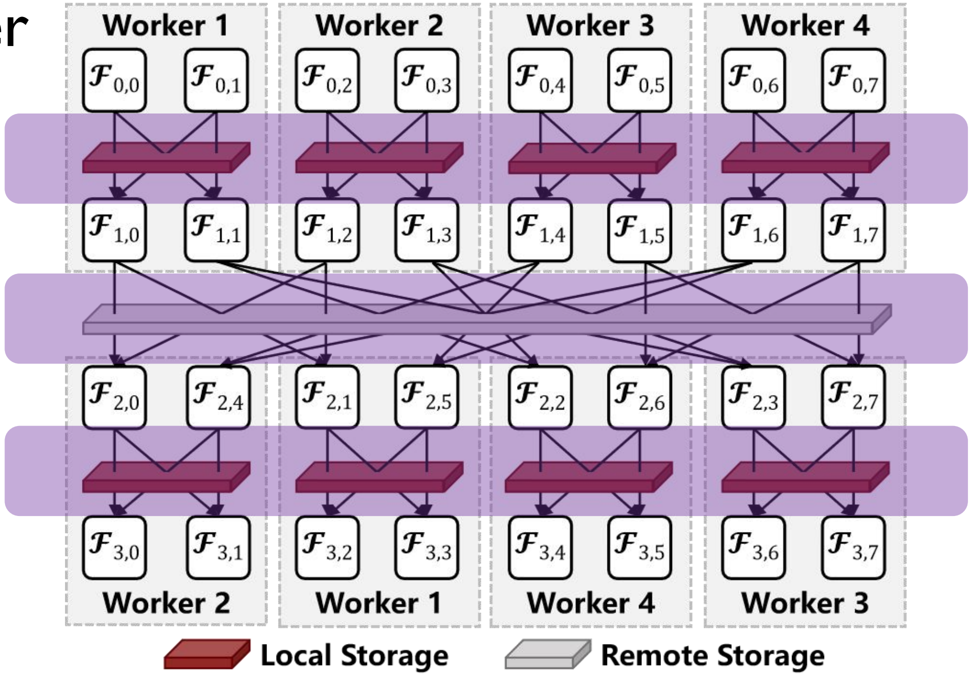
Interleaved complete bipartite graphs partitioning

- 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

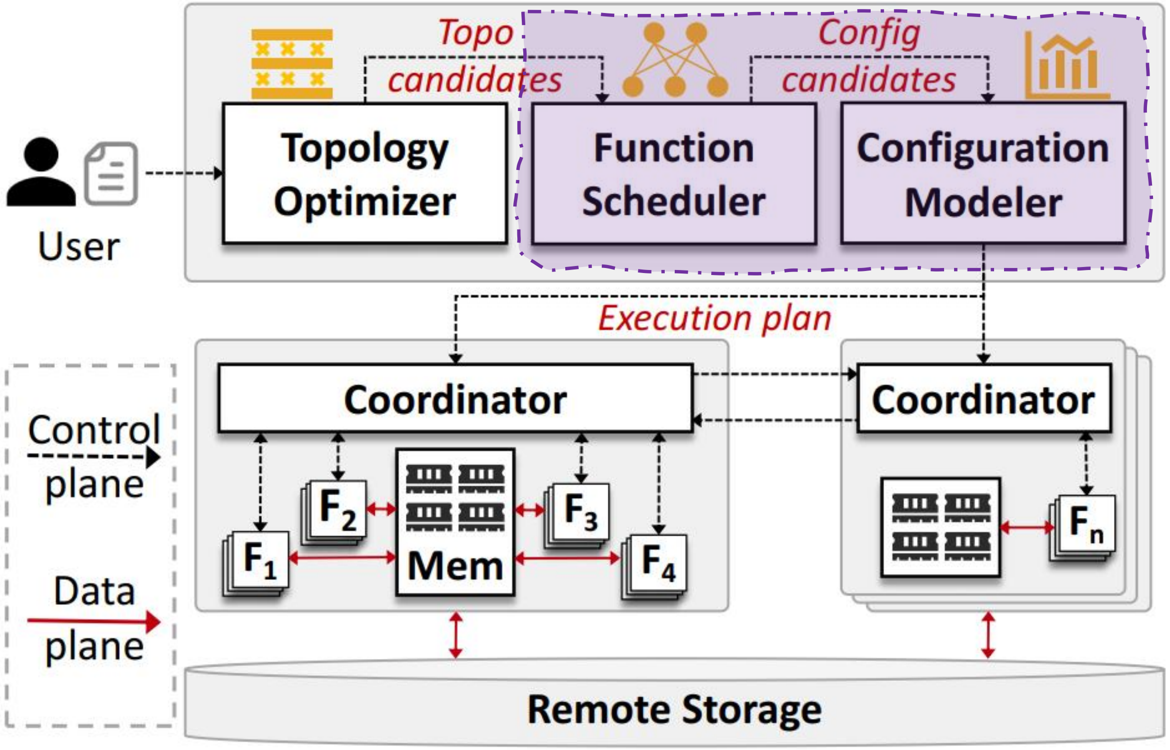


Function Scheduler

Conclusions:

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

- Adjacent function levels: complete bipartite graphs
  - Search the CB(
    Maximize traffic
  - Adopt the sam
    communication le
    Avoid stragglers
  - Employ interlea
    storage: across c
    Balance load

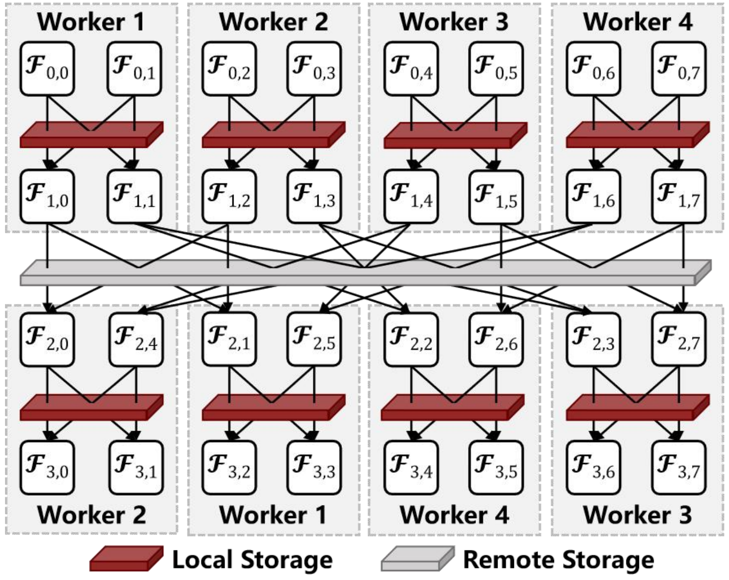


Configuration Modeler

➢ How to select the optimal configuration from config condidates?

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



function-side storage-side

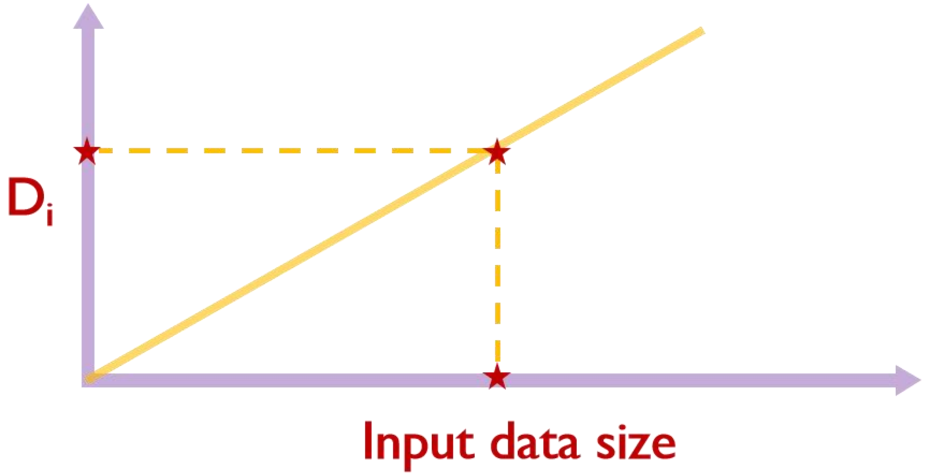
$$T = 2 * \sum_{i=0}^{L-1} \begin{cases} \max\left(\frac{D_i}{N * b_f}, \frac{R_i}{q_s}\right), \textit{S3 levels.} \\ \max\left(\frac{D_i}{M * b_t}, \frac{R_i}{M * q_t}\right), \textit{memory levels.} \end{cases}$$

Configuration Modeler

➢ How to select the optimal configuration from L config condidates?

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

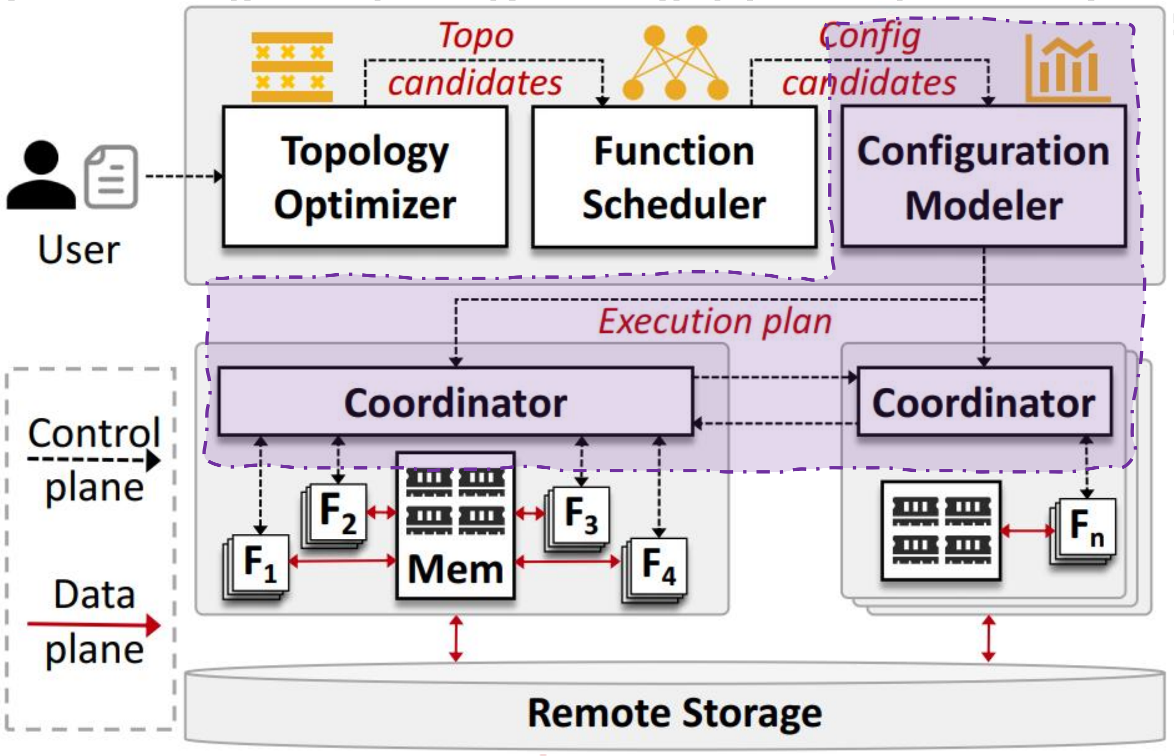


Configuration Modeler

Conclusions:

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

- The volume of
  - Input data s
  - Sample and



Experiments

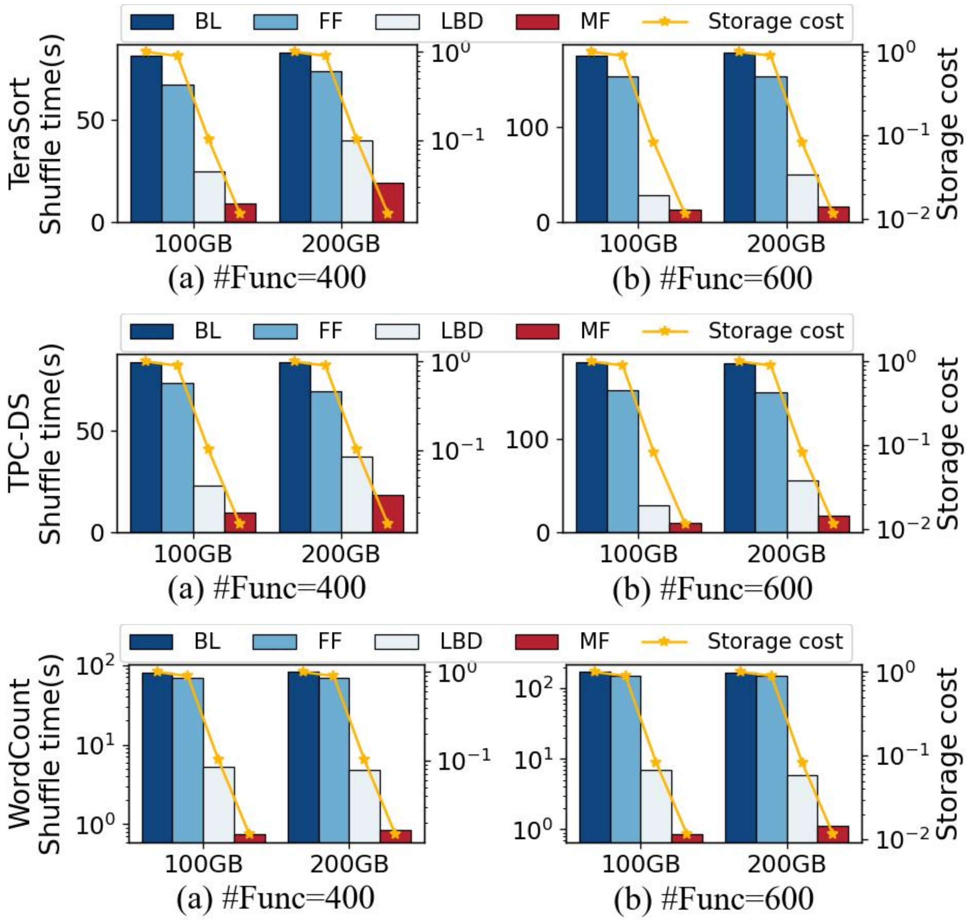
- Testbed:
  - 10 Amazon EC2 m6i.x24large instances

vCPU	Memory/GiB	Network bandwidth/Gib
96	384	37.5

- 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

Shuffle Time and Storage Cost

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



Conclusions:

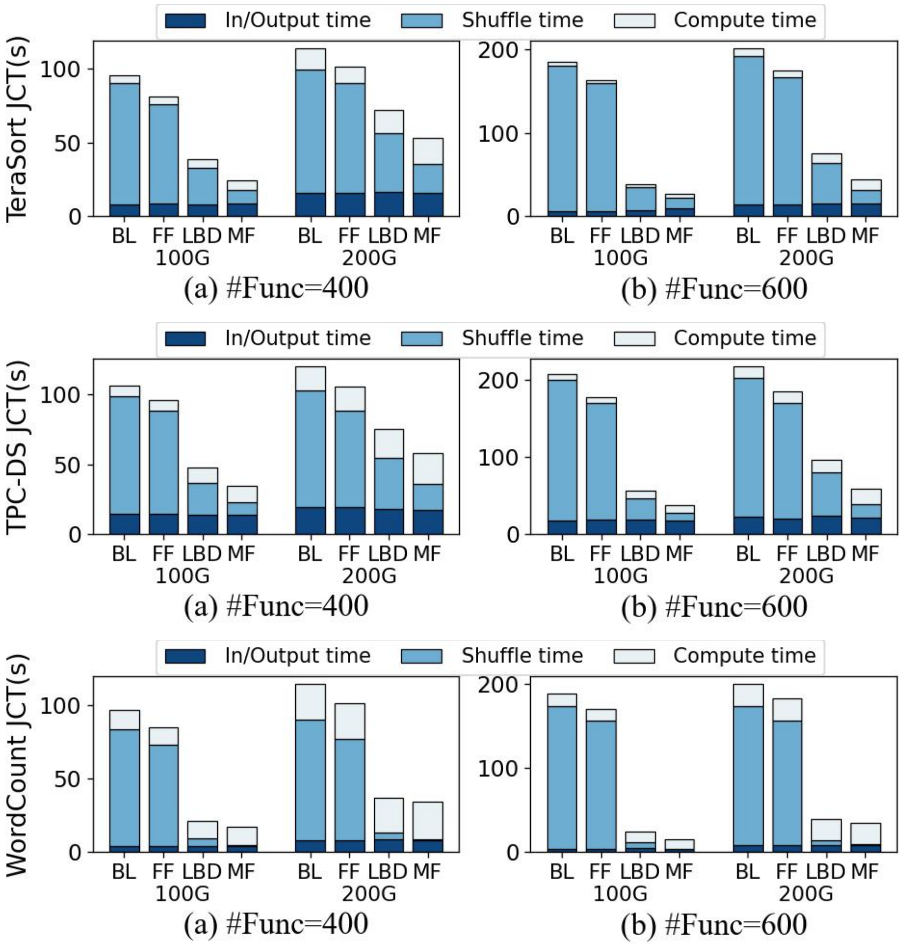
Under Terasort workload, compared to Baseline, FaaSFlow, and Lambada

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



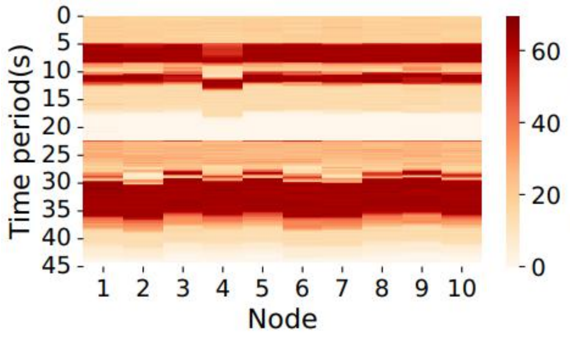
Conclusions:

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

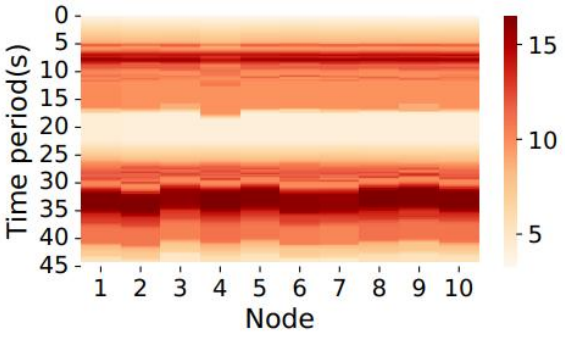
1. **MinFlow** reduces the job completion time up to 85.16%, 83.25%, and 41.35%, respec tively;

Load Balance

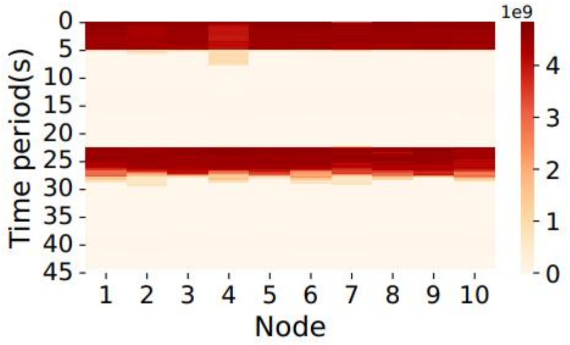
- ➢ Terasort workload: 200GB input data size, 600 functions



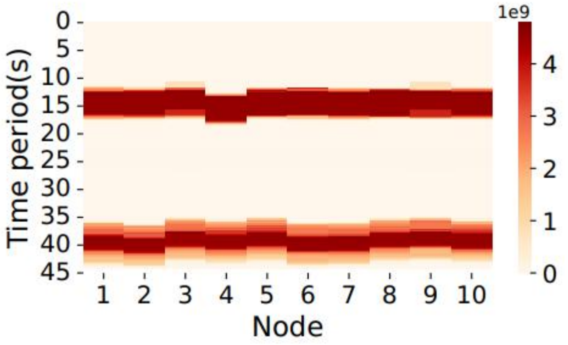
(a) CPU utilization



(b) Memory utilization



(c) Receive throughput (byte/s)



(d) Sent throughput (byte/s)

Conclusions:

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

Conclusions

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