FaaSBoard: Efficient Graph Processing with a Disaggregated Architecture on Serverless Services

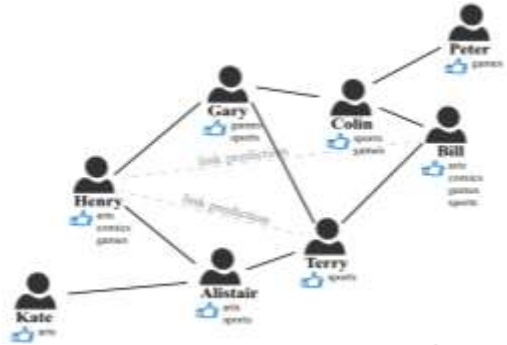
Yushi Liu*[1], **Yikang Ruan***[1], Letian Ruan[1], Zijun Li[2], Sen Gao[3], Weihao Cui[1], Shixuan Sun[1], Quan Chen[1], Shuo Quan[4], Jie Wu[4], Bingsheng He[3], Minyi Guo[1]

Shanghai Jiao Tong University[1]; Nanyang Technological University[2]
National University of Singapore[3]
China Telecom Cloud Computing Research Institute[4]


Graph Processing Stack

Apps

Social Network Mining



Recommendation System

Road Analysis

Framework

Shared Memory	Distributed
Ligra (PPoPP'13), GridGraph (ATC'13), GraphIt (OOPSLA'18)	Gemini (OSDI'16), Graphite (SIGMOD'20), GraphScope (VLDB'21)

Resource

Physical Servers

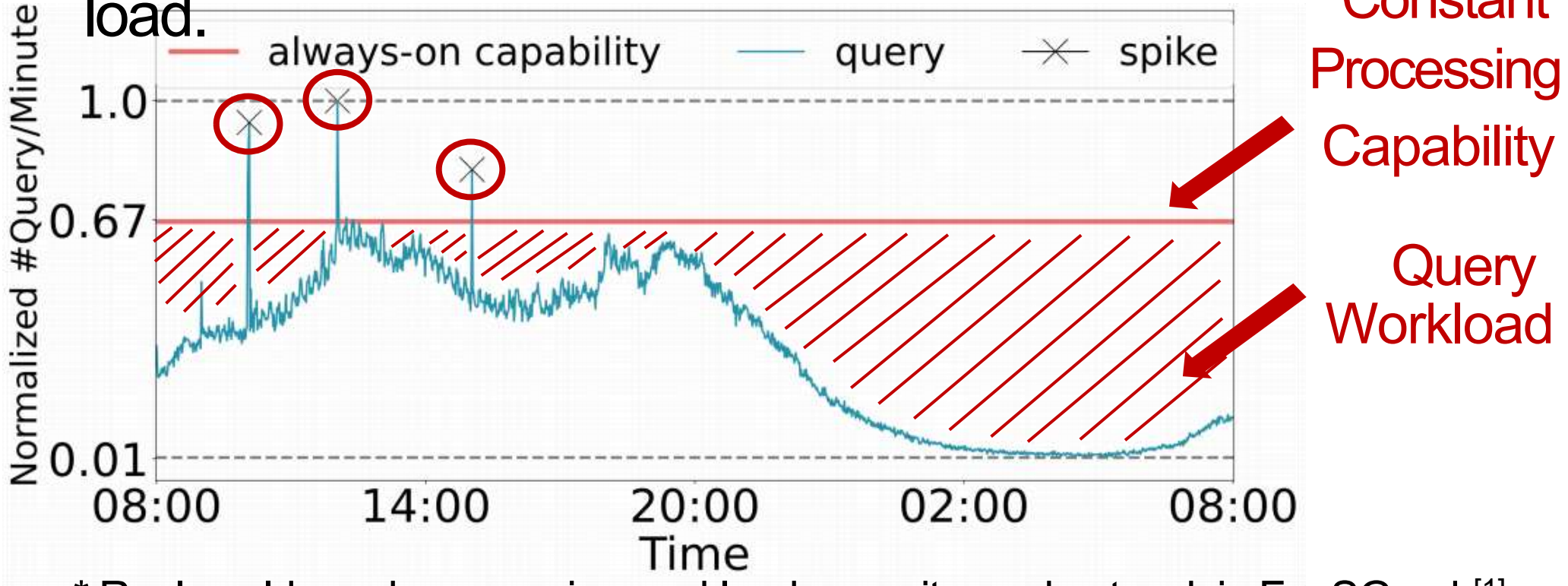
aws Azure Google Cloud Alibaba Cloud

Cloud VMs (Infrastructure-as-a-Service)

IaaS-based Graph Systems are not Elastic

Tail latency surge during query spikes.

Resource waste under low load.



* Real-world graph processing workload on a city road network in FaaSGraph[1]

[1] Liu, Yushi, et al. "FaaSGraph: Enabling Scalable, Efficient, and Cost-Effective Graph Processing with Serverless Computing." In ASPLOS, 2024.

Introducing Serverless Computing

Auto-scaling

- The “Unbounded Scaling” policy[2].

Pay-per-use billing

- Fine-grained billing (<1ms).
- Charge based on resource usage, not resource allocation.

Ease of management

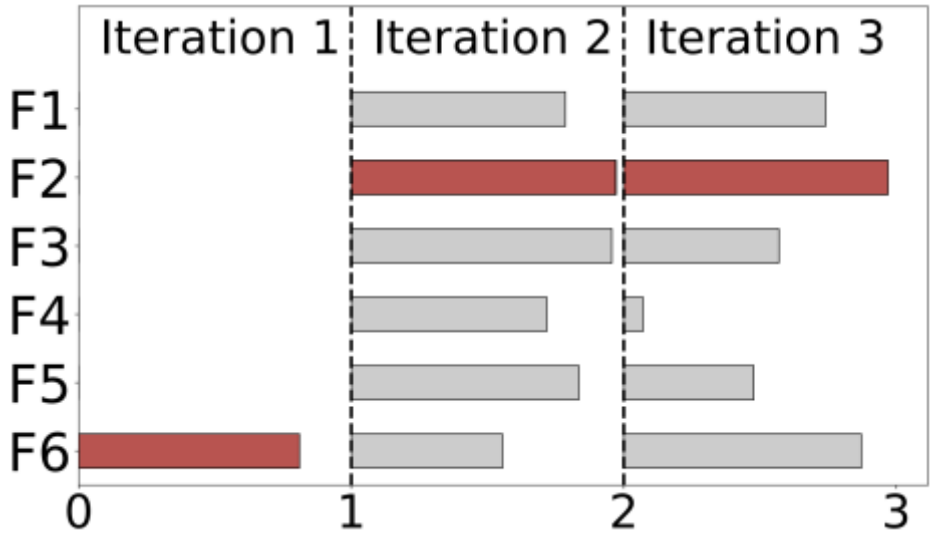
- Cloud manage underlying infrastructures.
- Users upload only the source code and data.

[2] https://docs.aws.amazon.com/lambda/latest/dg/lambda-concurrency.html

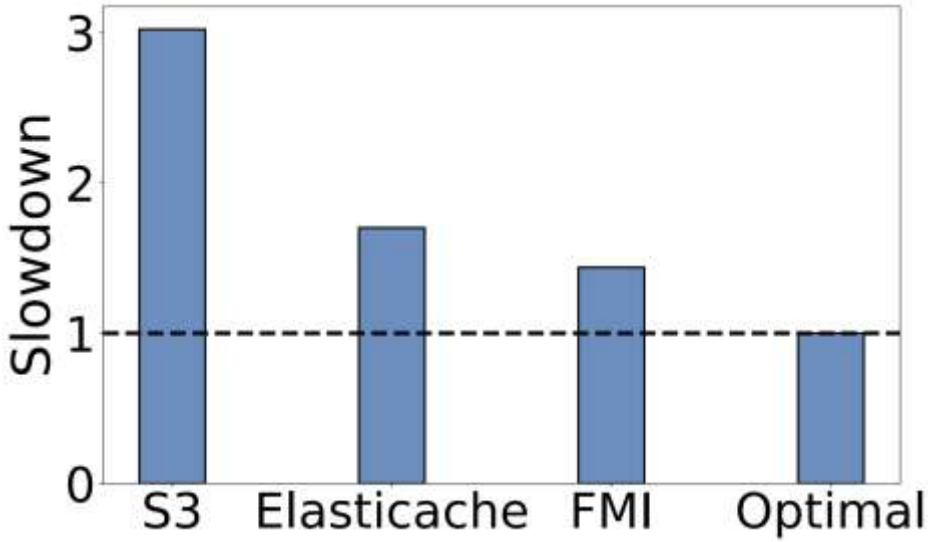
Inefficiencies

Lessons Learned from Profiling FaaSGraph

- Ignoring intra-job elasticity incurs additional monetary costs.
- Communication through storage or direct function links is slow.



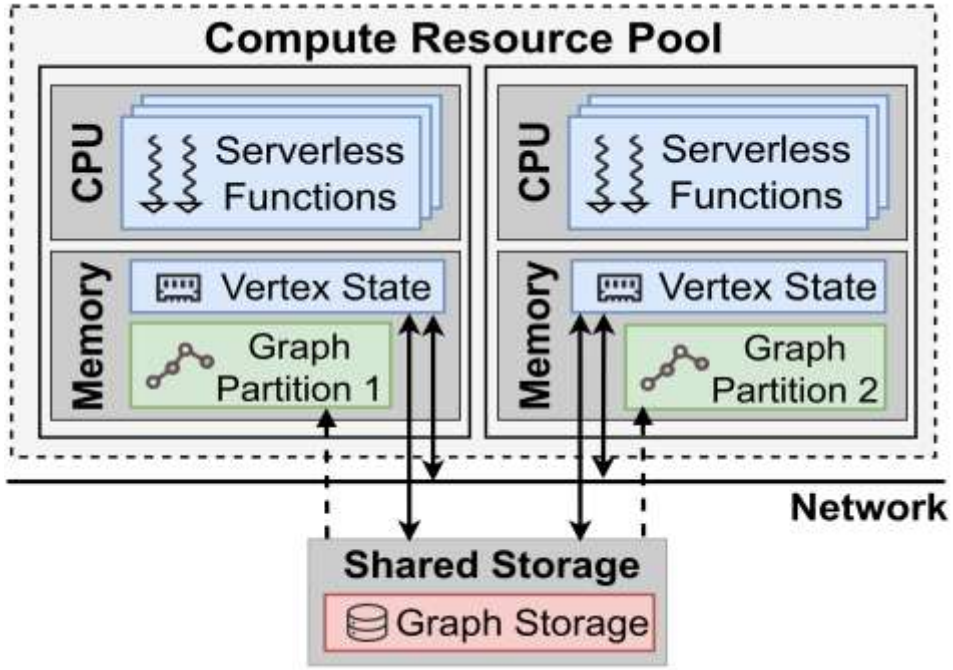
Normalized execution time in three consecutive iterations of tw-BFS



Performance when applying various communication methods

Claim

- Monolithic Function Architecture adopted by existing serverless graph processing system[1, 3] is the root cause, which encapsulates data loading, computation, and communication into a single function.

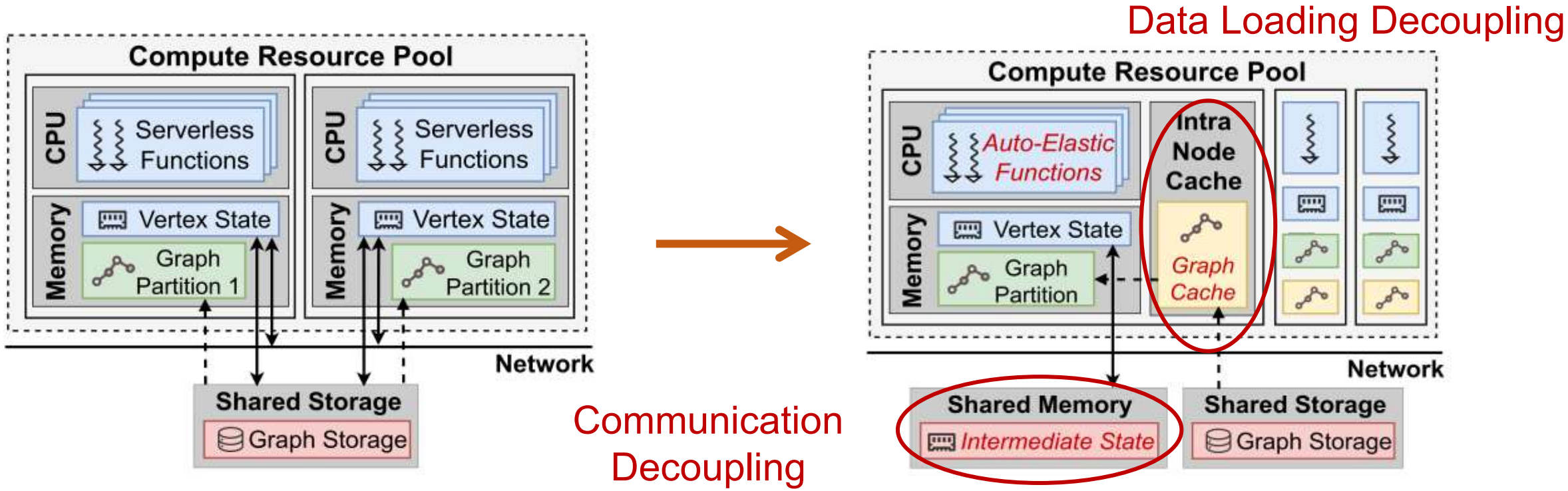


Monolithic Function Architecture

[1] Liu, Yushi, et al. "FaaSGraph: Enabling Scalable, Efficient, and Cost-Effective Graph Processing with Serverless Computing." In ASPLOS, 2024.
[3] Lucain, Toader, et al. “Graphless: Toward Serverless Graph Processing” In ISPDC, 2019.

FaaSBoard: Disaggregated Serverless Architecture

- We propose a Disaggregated Serverless Architecture that decouples data loading, computation, and communication.

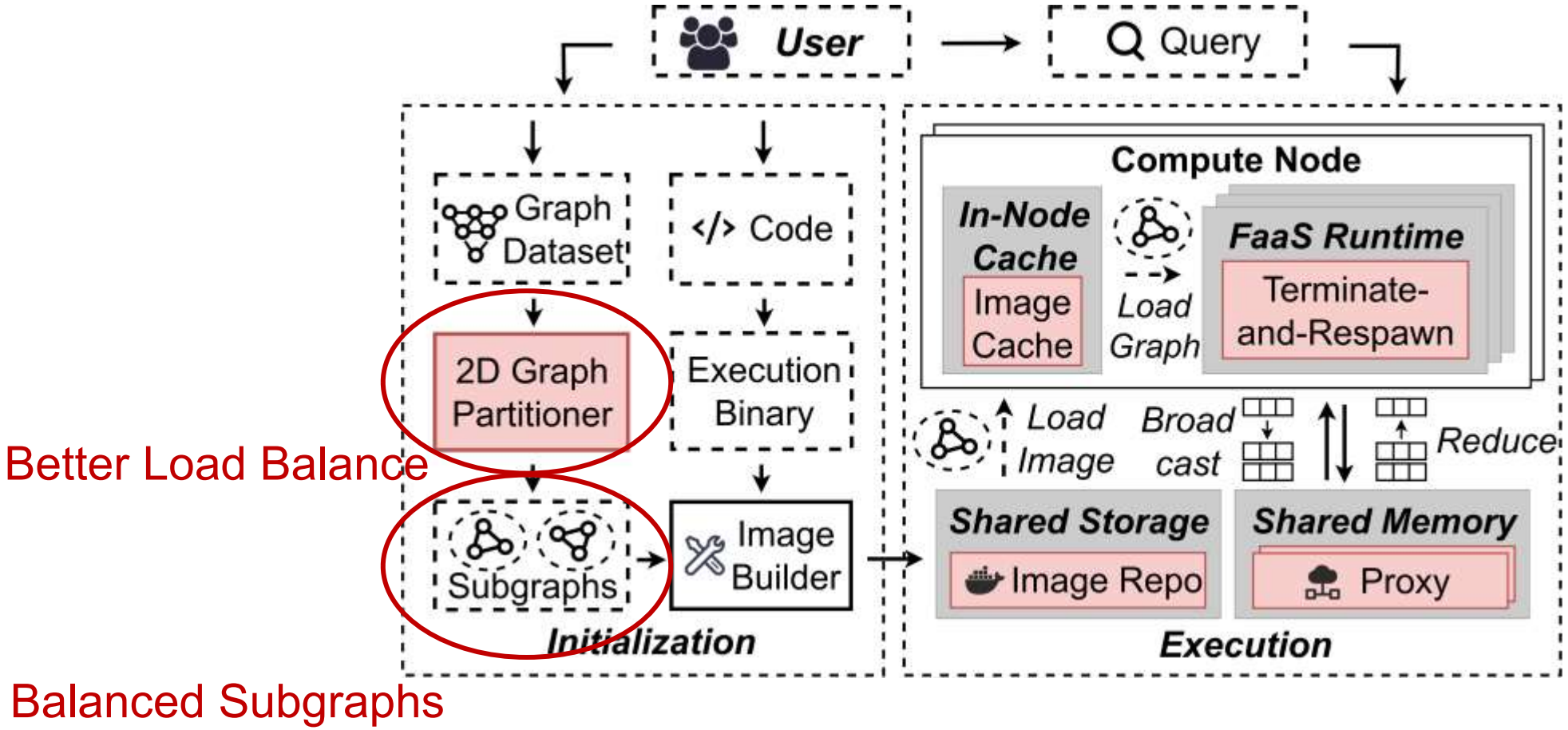


Monolithic Function Architecture

Disaggregated Serverless Architecture

FaaSBoard Workflow

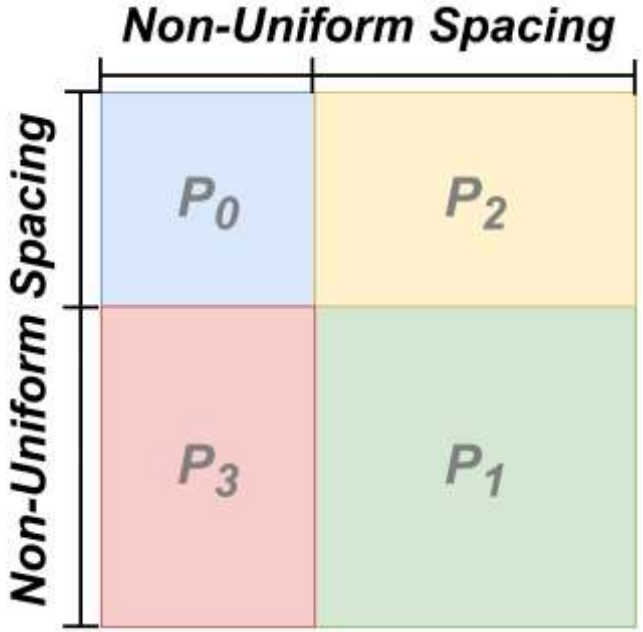
- FaaSBoard is built entirely on industrial serverless services, provided by AWS.



2D Balanced Graph Partitioning

Insights Combine the low communication complexity $O(\sqrt{p})$ of 2D-chunk based partitioning and the load balance of non-uniform chunking space.

Benefits Suitable for resource-constrained serverless environments, improving computational performance while avoiding OOM.



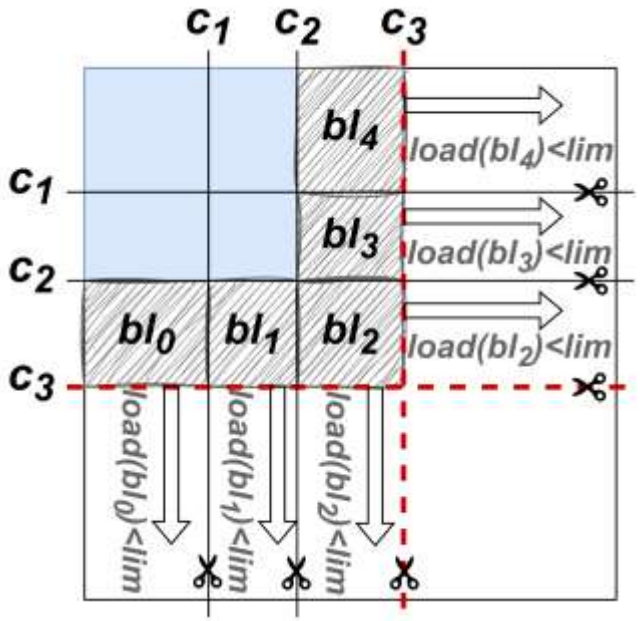
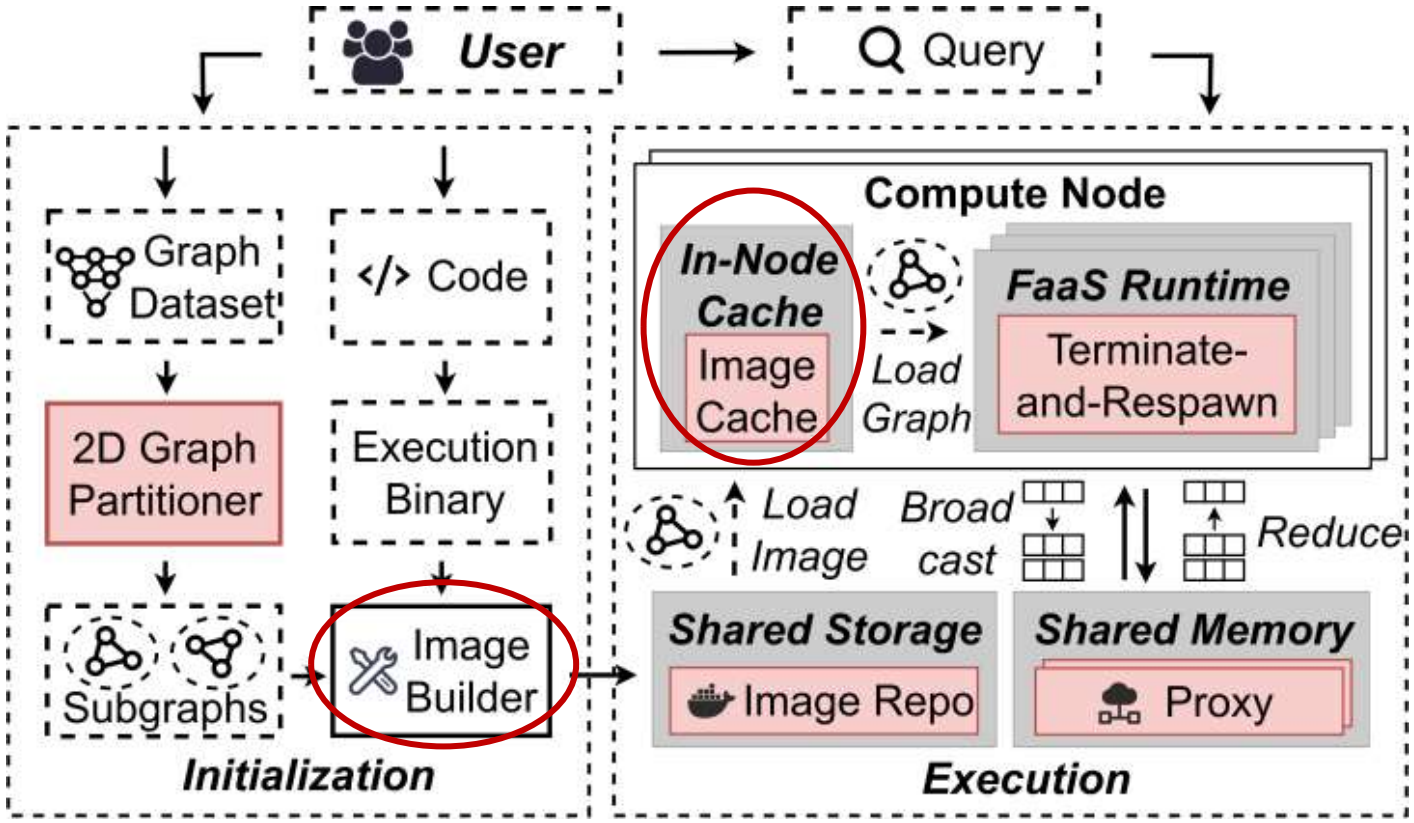


Image-based Graph Loading

Insights Package the computation code and graph into one image.

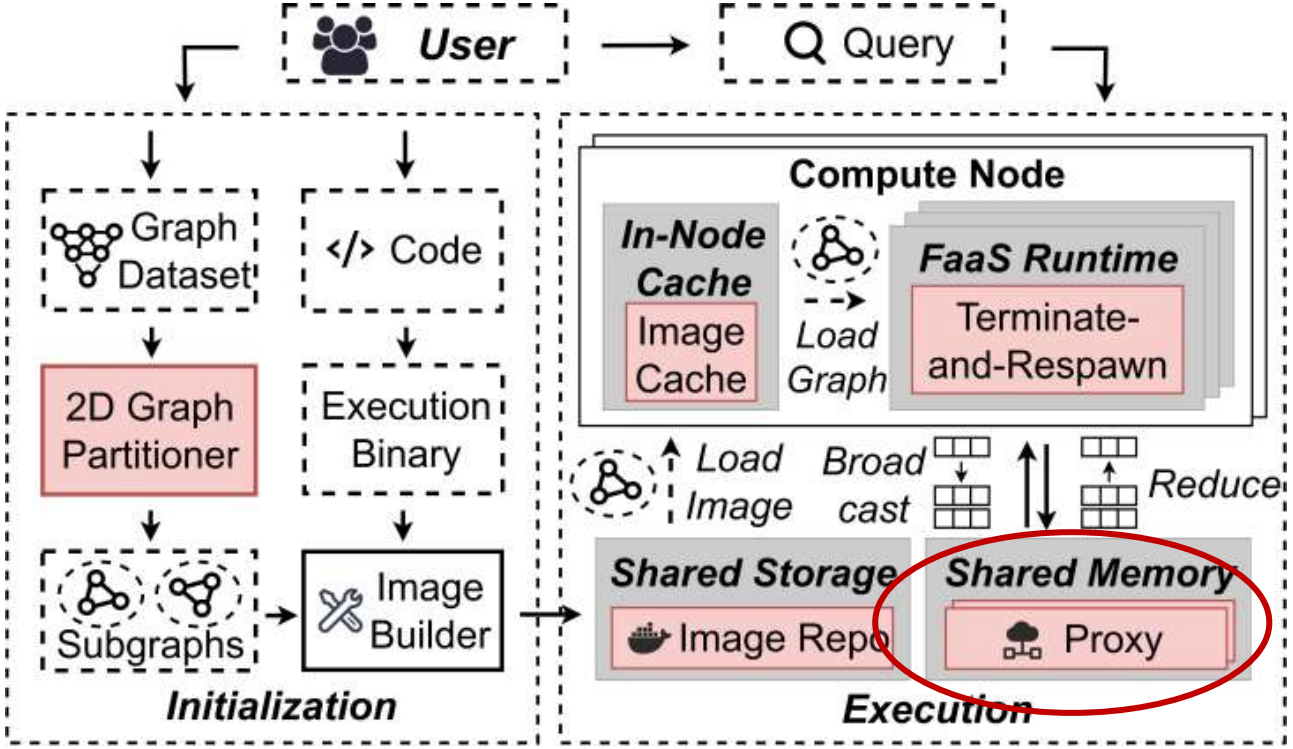
Benefits Shorter loading path and hot data reuse.



Proxy-based Collective Communication

Insights Leverage the higher bandwidth of proxy compared to functions (7.0–8.5× for AWS Fargate to AWS Lambda), and utilize proxy's computing resources for auxiliary computation.

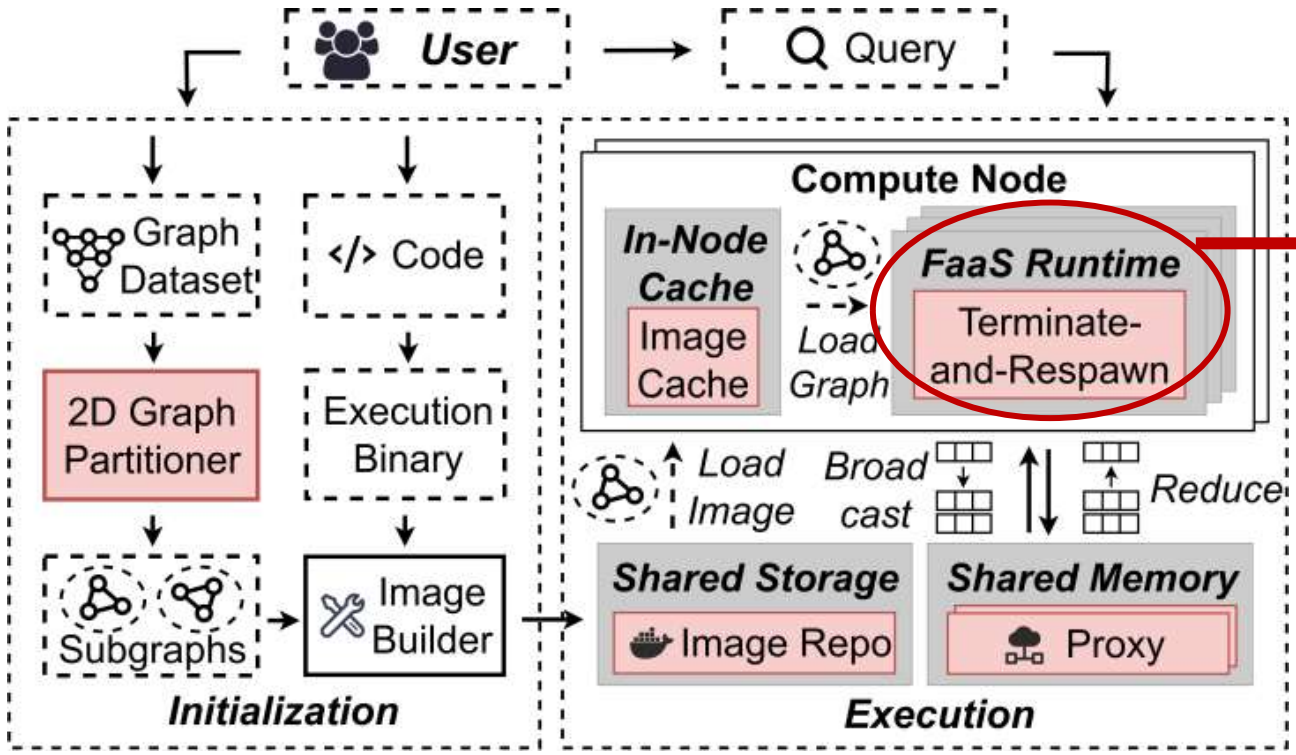
Benefits Better communication performance.

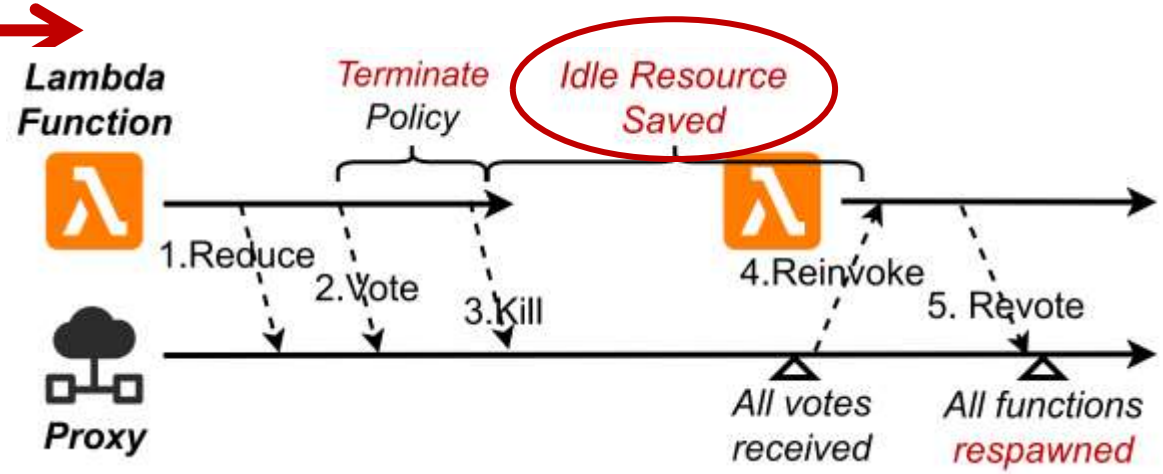


Proactive Terminate-and-Respawn

Insights Leverage intra-job elasticity of graph processing.

Benefits Less resource idling and lower monetary costs.





Experiment Settings

Workload

Graphs[4]: 3 SNAP datasets + RMAT-27

Apps: breadth-first-search (bfs), connected-components (cc), pagerank (pr), single-source-shortest-path(sssp)

Environments

Serverless function -> AWS Lambda

Proxy -> AWS Fargate

Baselines

FaaSGraph(ASPLOS'24)

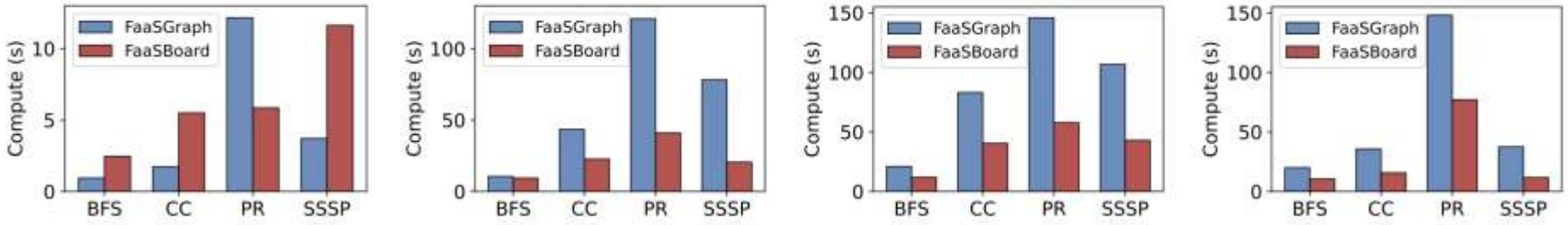
	Configuration			
Hardware	*FaaSGraph:* CPU: Intel Xeon E5-2676 V3 @2.40Ghz, Instance: m4.10xlarge, Cores: 40, DRAM: 160GB			
Software	*FaaSGraph:* Ubuntu 22.04, Golang 1.18.4, Docker 20.10.17 *FaaSBoard:* Image public.ecr.aws/lambda/provided:al2023			
Service	*FaaSBoard:* AWS Lambda, AWS Fargate			
Function	*FaaSGraph&FaaSBoard:* Cores: 2, DRAM: 3538MB			
Benchmark	Graph	Vertices	Edges	#Functions
	Livejournal	4.84M	68.9M	1(BFS,CC,PR,SSSP)
	Twitter	41.6M	1.46B	6(BFS,PR),12(CC,SSSP)
	Friendster	65.6M	1.80B	8(BFS,PR),14(CC,SSSP)
	Rmat27	134M	2.14B	9(BFS,PR),17(CC,SSSP)

[4] https://snap.stanford.edu/data

Overall Results

End-to-end Latency Speedup (FaaSBoard vs. FaaSGraph)

Max *3.8x Speedup*



(a) Livejournal (b) Twitter (c) Friendster (d) Rmat27

Max *61.5% Cost Savings*

Dataset	System	BFS	CC	PR	SSSP
lj	FaaSGraph	**0.01**	**0.01**	**0.07**	**0.02**
	FaaSBoard	0.09	0.21	0.22	0.44
tw	FaaSGraph	**0.37**	3.01	4.18	5.40
	FaaSBoard	0.63	**2.30**	**2.74**	**2.08**
fr	FaaSGraph	0.97	6.70	6.72	8.60
	FaaSBoard	**0.93**	**4.10**	**4.53**	**4.36**
rm	FaaSGraph	1.04	3.49	7.67	3.67
	FaaSBoard	**0.88**	**2.04**	**5.49**	**1.52**

Monetary cost (¢) comparison (FaaSBoard vs. FaaSGraph)

Conclusion

- We present FaaSBoard, a graph processing system on serverless cloud services which adopts a disaggregated serverless architectural design.
- FaaSBoard achieves significant improvements in computing performance and monetary costs compared to FaaSGraph, which is the SOTA serverless graph processing system.

Thanks!

Paper

Code

Source Code: https://github.com/SJTU-Liquid/FaaSBoard