FaaSGraph: Enabling Scalable, Efficient, and Cost-Effective Graph Processing with Serverless Computing

Yushi Liu[1], **Shixuan Sun[1],** Zijun Li[1], Quan Chen[1], Sen Gao[2], Bingsheng He[2], Chao Li[1], Minyi Guo[1]

Shanghai Jiao Tong University[1]; National University of Singapore[2]

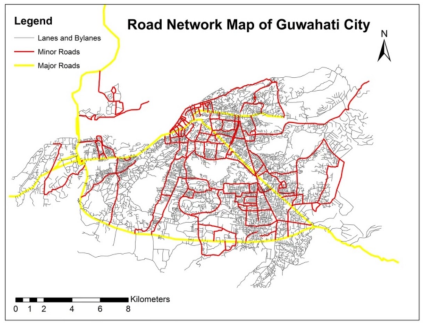
上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

Graph Processing Stack

Apps

Social Network Mining

Recommendation System



Road Analysis

Framework

Shared Memory: Ligra (PPoPP'13), Galois (SOSP'13), GraphIt (OOPSLA'18)

Distributed: Gemini (OSDI'16), Graphite (SIGMOD'20), GraphScope (VLDB'21)

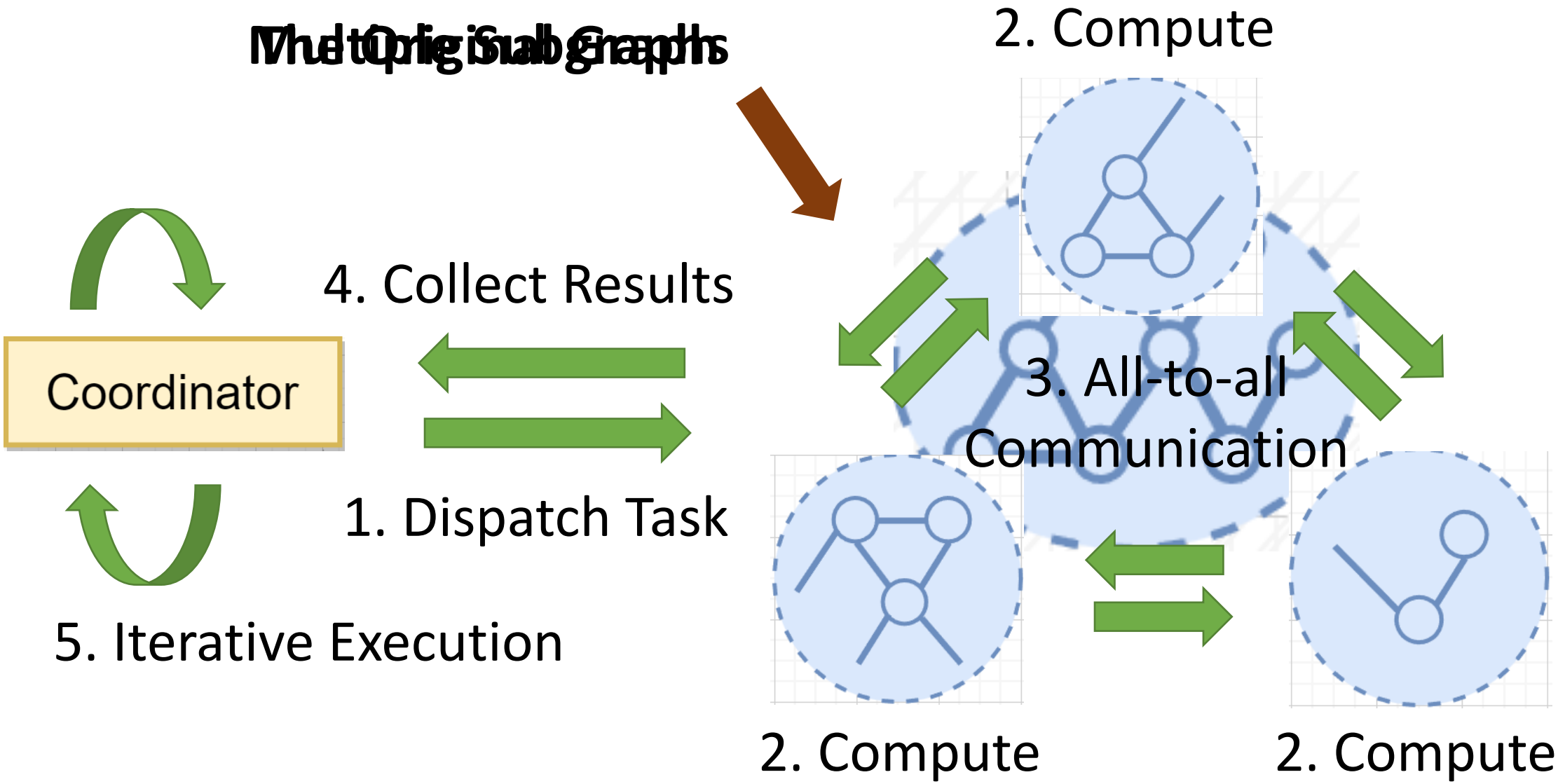
Resource

Physical Servers

aws Azure Google Cloud Alibaba Cloud

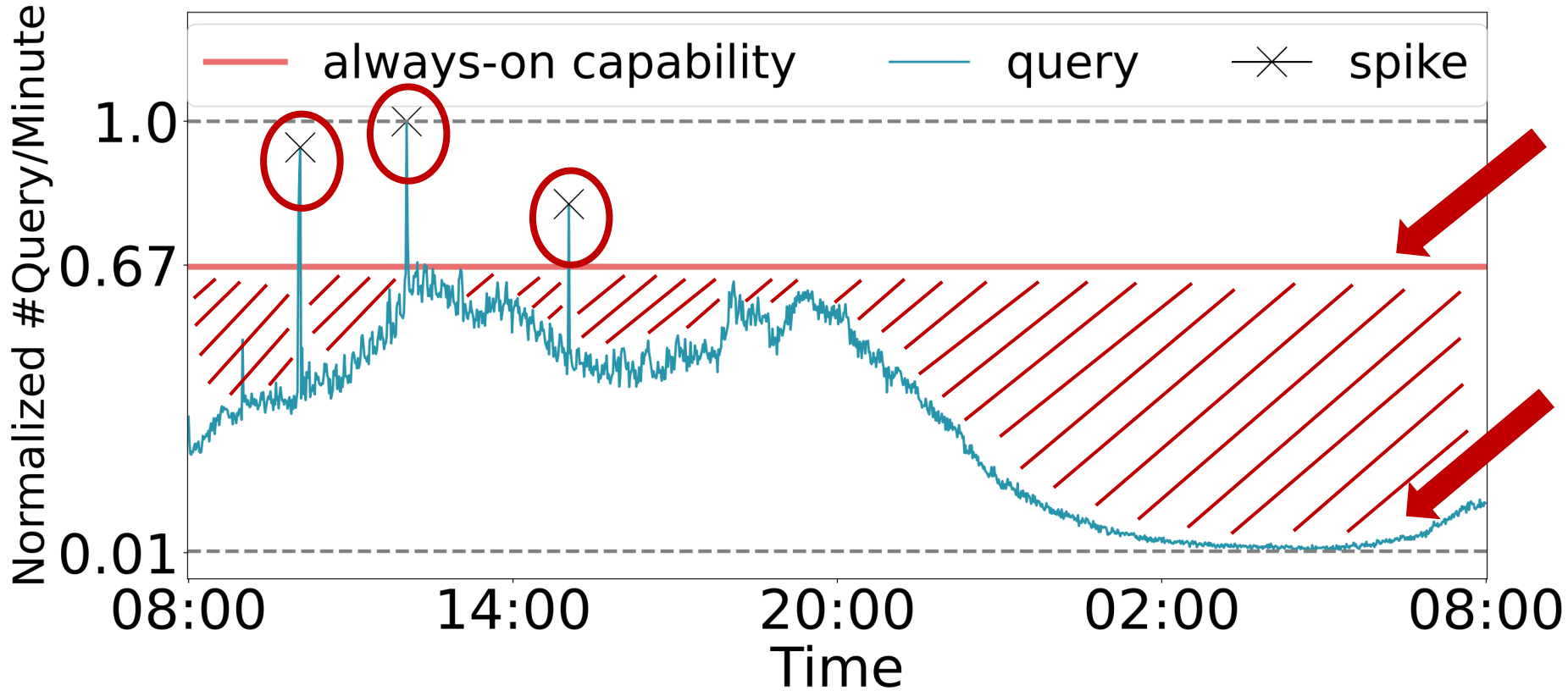
Cloud VMs (Infrastructure-as-a-Service)

Iterative Execution Flow



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 from our industry partner

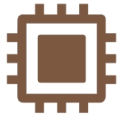
Introducing Serverless Computing

Auto-scaling

- The “Unbounded Scaling” policy[1].

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.

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

End-to-end Performance Degradation

Lessons Learned

- Direct migration can degrade performance.
- Graph IO & preprocessing dominates performance.

	Overall =	Resource Initialize +	IO&Preprocessing +	Compute
Gemini	148.5s	**20.7s (25.9x)**	113.2s	14.6s
S - R	**661.3s (⬇4.5x)**	0.8s	**401.3s (⬇3.5x)**	**259.1s (⬇17.7x)**
S - D	261.0s (⬇1.8x)	0.8s	142.7s (⬇1.3x)	117.5s (⬇8.0x)

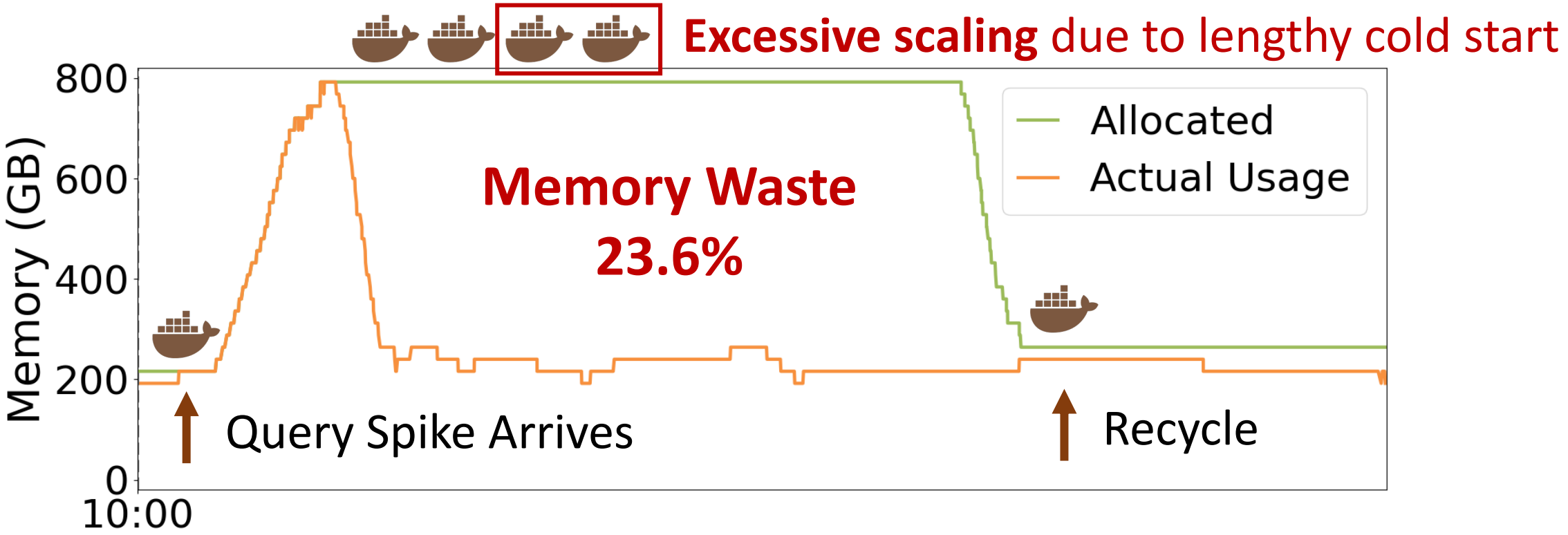
* S-R: Gemini[2] in Serverless; S-D: Gemini in Serverless with Direct Communication

[2] Zhu, Xiaowei, et al. "Gemini: A Computation-Centric distributed graph processing system." In OSDI, 2016.

Memory Waste

Lessons Learned

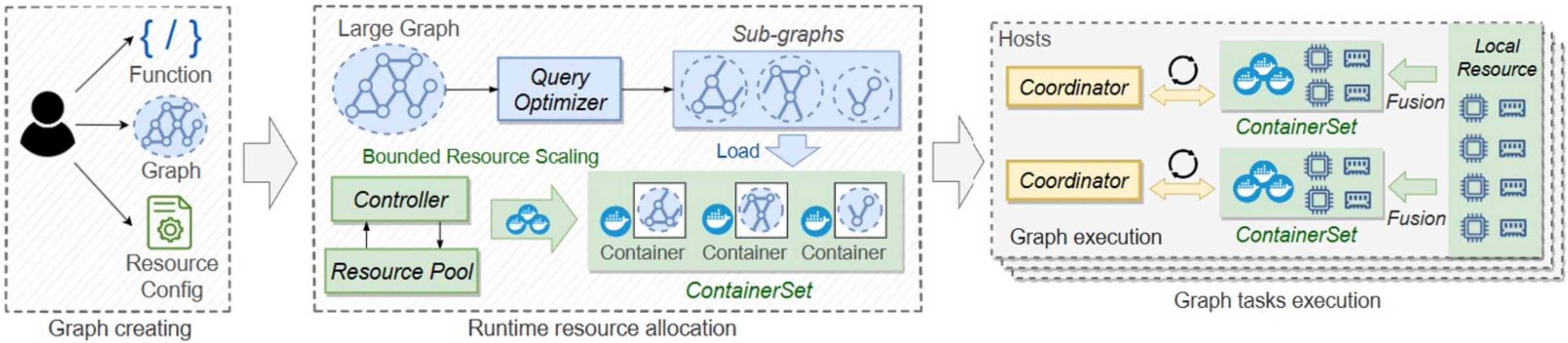
- Unbounded scaling allocates excessive resources, which are commonly wasted.



FaaSGraph: Serverless-native Graph Processing

- **Co-designed** scheme that combines graph processing and serverless architecture.

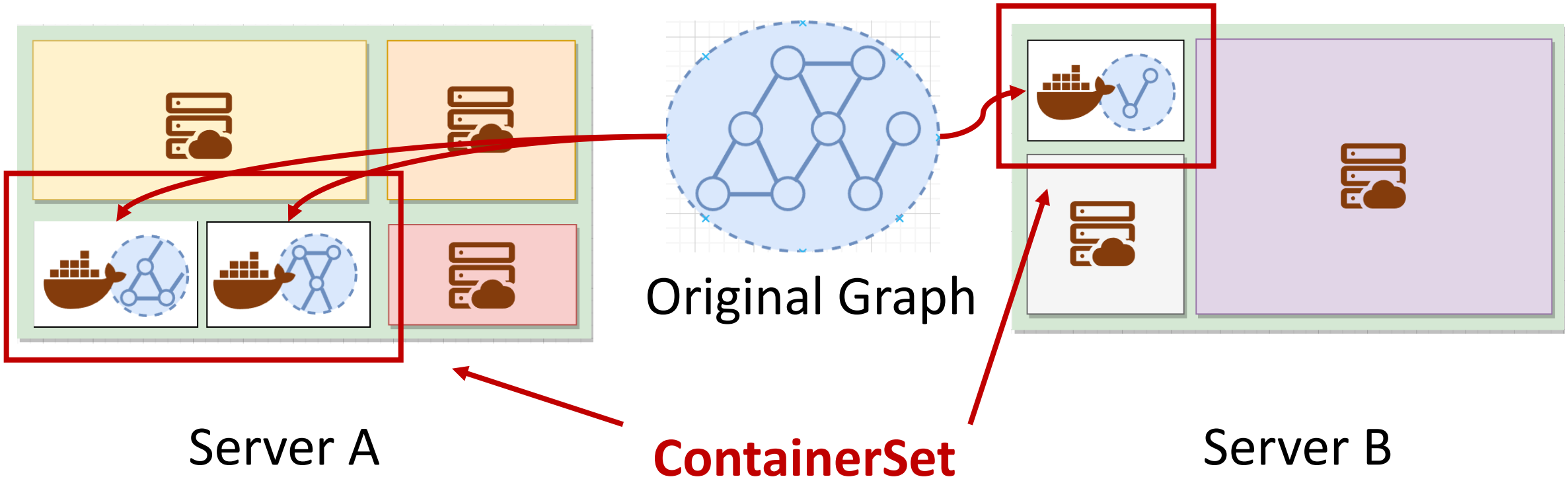
Resource Fusion | **Bounded Scaling** | **Graph Optimizations**



ContainerSet as Resource Abstraction

Insights Use unified, fine-grained containers to execute graph processing tasks while maintaining scheduling flexibility.

Benefits Improve cluster resource utilization.



Locality-aware Resource Fusion

Insights Reintroduce the “illusion of locality” through resource fusion.

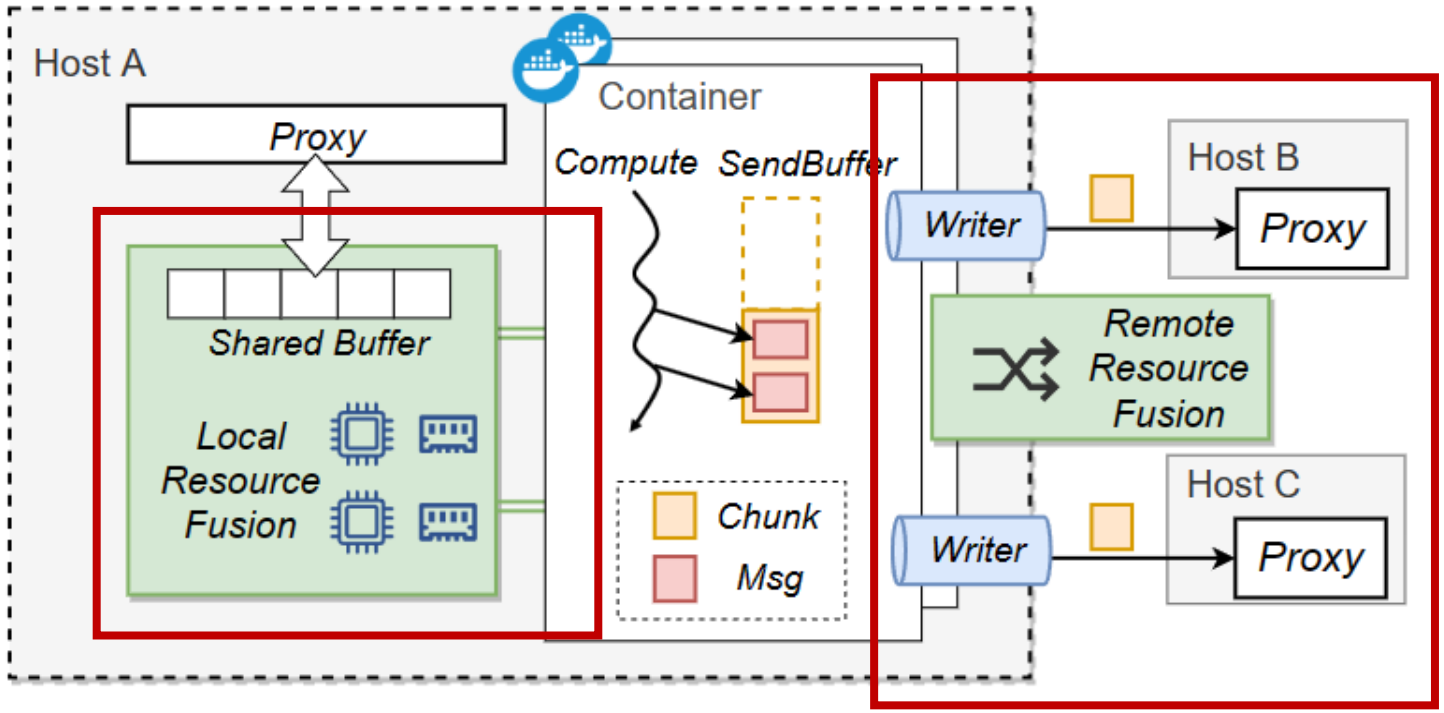
Benefits Improve comp. & comm. performance.

Local:

- CPU sharing: better load balancing
- Memory sharing: fast data sync channel

Remote:

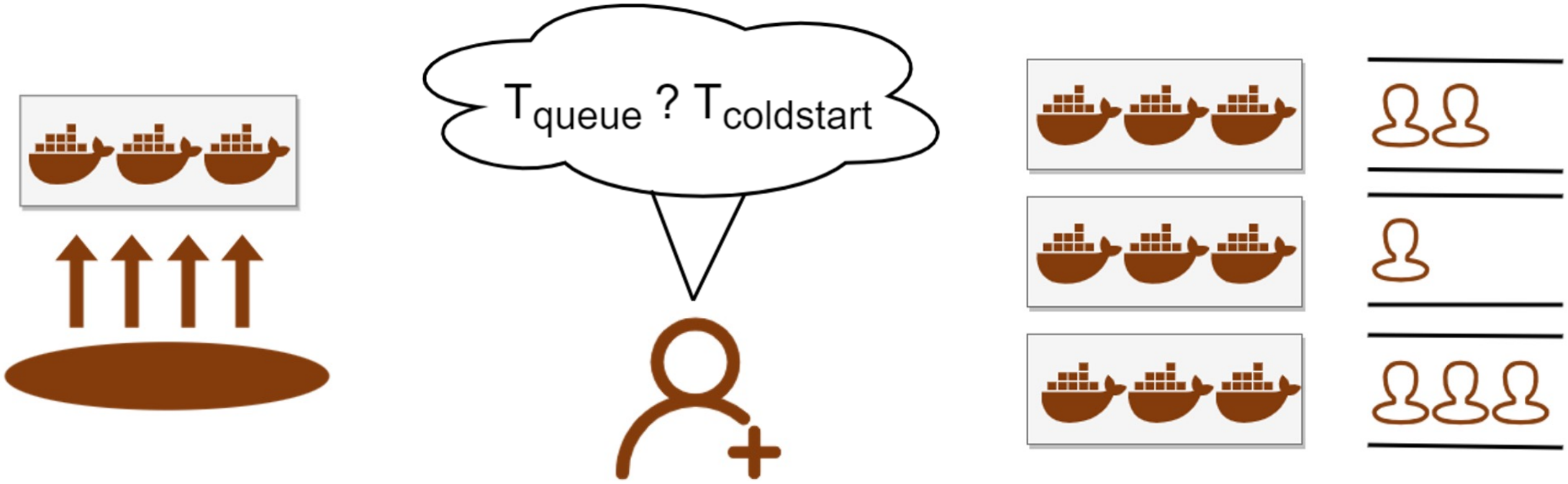
- Message consolidation
- Async transmission



Bounded Scaling

Insights Rather than immediately allocating new resources, wait for the current occupied resources to become available.

Benefits Reduce memory footprint, reduce latency.



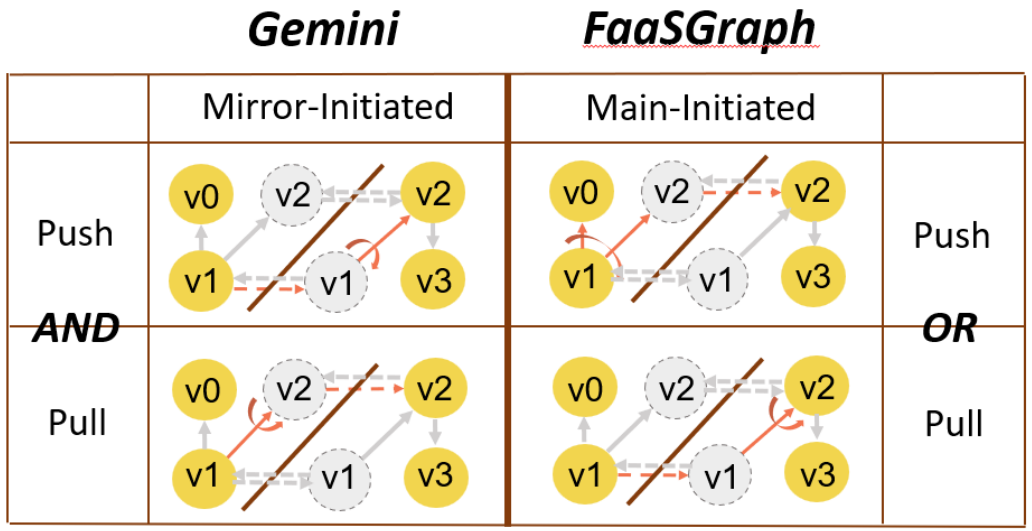
Cold Start

Queueing

Graph Optimizations

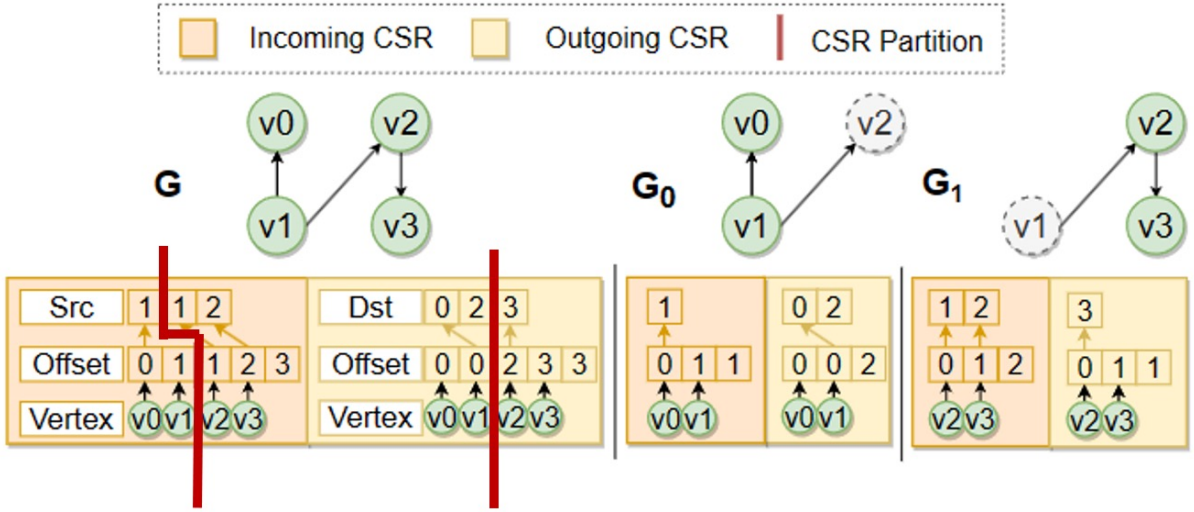
Single Mode, Main Initiated Message Passing

Benefits Reduce memory usage.



Fast Graph Loading

Benefits Reduce preprocessing overhead.



Experiment Settings

Workloads

Graphs[3]: 4 SNAP datasets + road graph

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

Environments

Single Query Evaluation: 4-server cluster

Large Scale Case Study: 34-server cluster

Baselines

Gemini (OSDI'16), Graphite (SIGMOD'20), Graphit (OOPSLA'18), GraphScope(VLDB'21)

	Configuration			
Hardware	CPU: Intel Xeon(Ice Lake) Platinum 8369B @3.5GHz Cores: 24, DRAM: 96GB, NAS: bandwidth 300MB/s			
Software	OS: Linux with kernel 5.15.0 Golang: 1.18.4, Docker: 20.10.17			
Container	Container Runtime: Golang 1.18-alpine Resource Limit and Lifetime: 2core, 3G, 15min			
Benchmark	Graph	Vertices	Edges	CSR Size
	amazon(am)	334,863	925,872	18Mb
	livejournal(lj)	4,847,571	68,993,773	1.1Gb
	twitter(tw)	41,652,230	1,468,364,884	23Gb
	friendster(fr)	65,608,366	1,806,067,135	28Gb
	road(rd)	3,996,221	4,246,845	176Mb

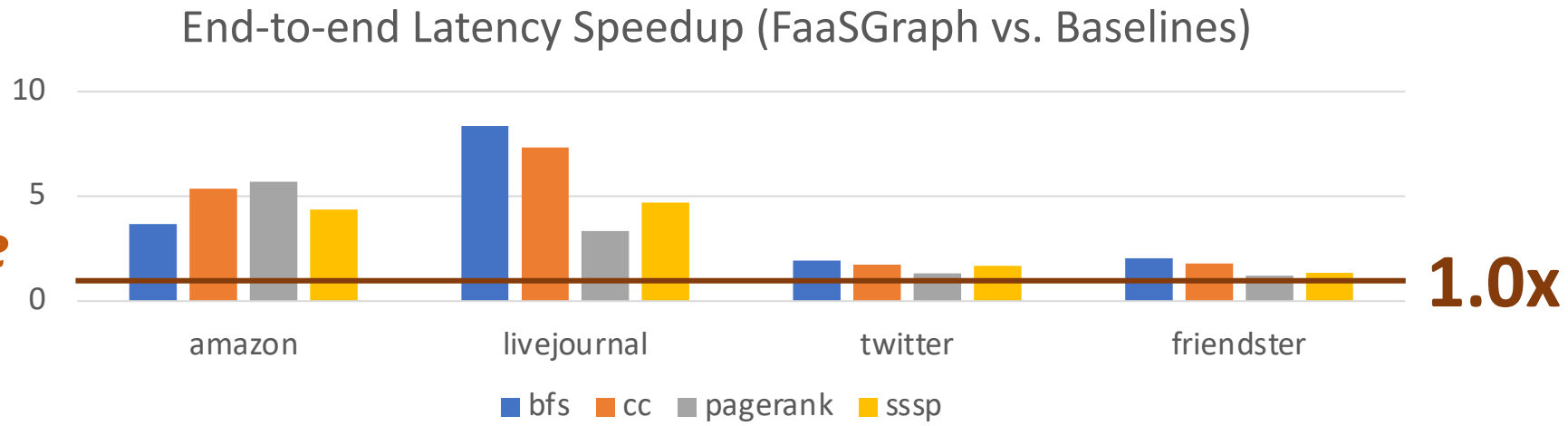
[3] https://snap.stanford.edu/data

Experiment Results

Max

End-to-end performance improvement

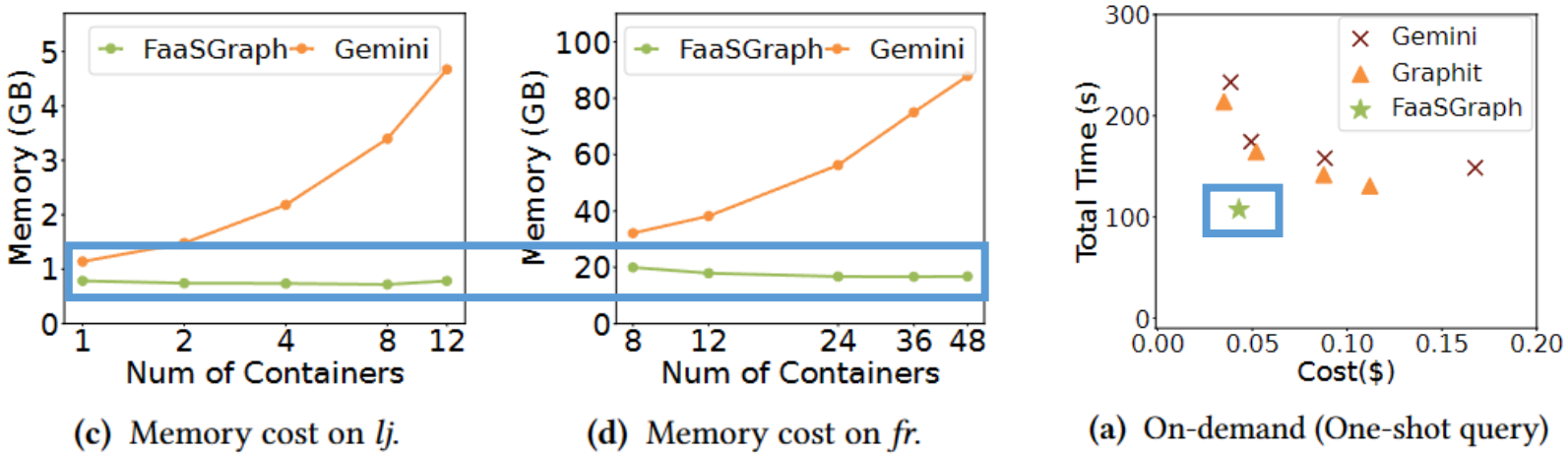
8.3×



Max

Reduced memory consumption

52.4%



(c) Memory cost on *lj*. (d) Memory cost on *fr*. (a) On-demand (One-shot query)

Low memory scaling overhead ***Cost-Performance Optimal***

Real-World Case Study

Constant Resource Config → **Scale Every 10 Minutes**

Gemini	Gemini-Naïve-Scaling	FaaSGraph
$283.96	$179.24	$40.74

Max

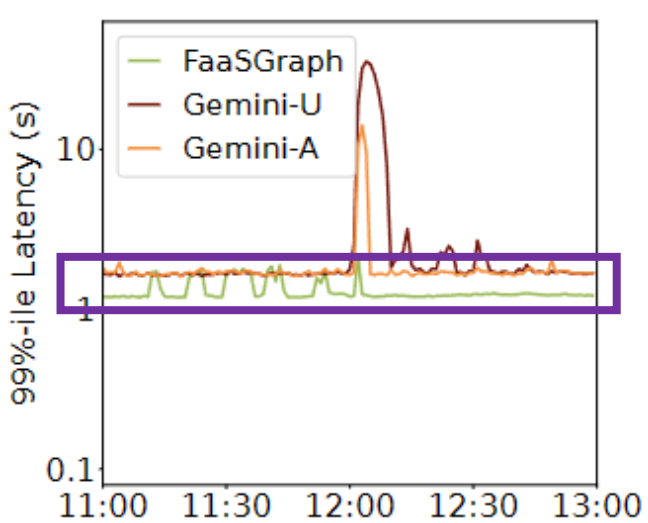
85.7%

Real-world one-day monetary cost savings

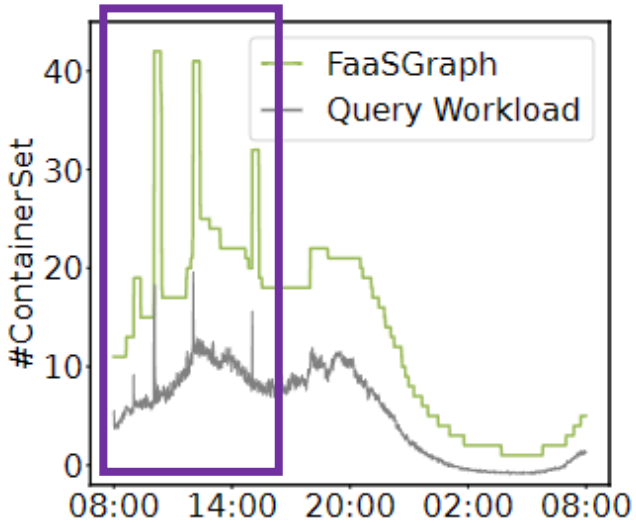
Max

6.6×

Tail latency improvement during query spike



(a) 99%-ile latency



(b) #ContainerSet

Steady 99%-ile Latency

Auto-scaling with Bounded Scaling Policy

Conclusion

- IaaS-based systems perform poorly under fluctuating workloads.
- We introduce FaaSGraph, a scalable, efficient and cost-effective billion-edge graph processing engine.

Resource Fusion **Bounded Scaling** **Graph Optimizations**

- FaaSGraph achieves significant improvements in latency, memory, and cost compared to state-of-the-art systems.

Thanks!

Source Code: https://github.com/ziliuziliu/FaaSGraph