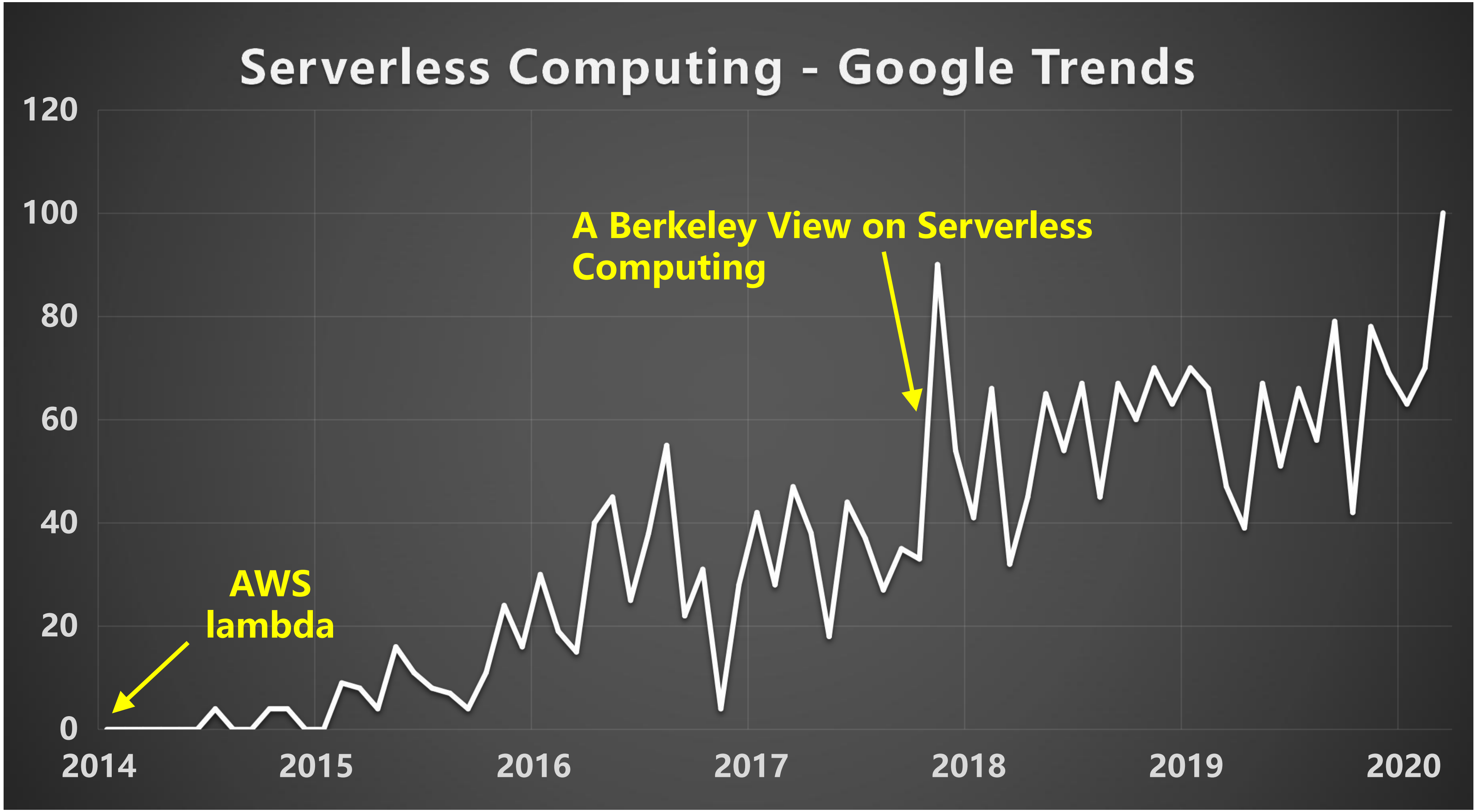
Understanding, Predicting and Scheduling Serverless Workloads under *Partial Interference*

Laiping Zhao, Yanan Yang, Yiming Li,
Xian Zhou and Keqiu Li

Tianjin University

Serverless Computing



Serverless Workloads

Scheduled Background (BG)

- IoT data collection, monitoring etc.

Short-term Computing (SC)

- Mapreduce/Spark, linear algebra etc.

Latency Sensitive (LS)

- Search, e-commerce, social network.

Small-sized

0.125-3GB Memory

Short-lived

≤900 s

Serverless Workloads

Small-sized → **High density**

- A server can accommodate hundreds of functions.

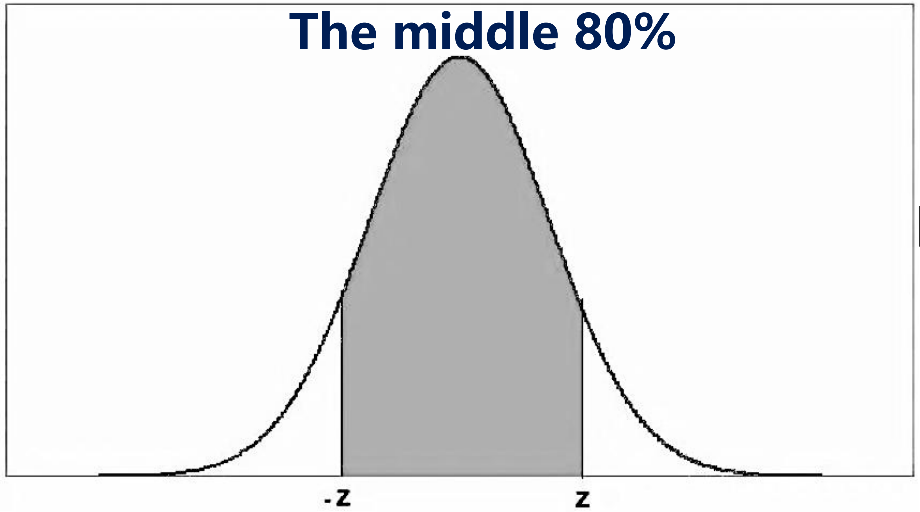
Short-lived → **High dynamic**

- Functions are started or released at all times.

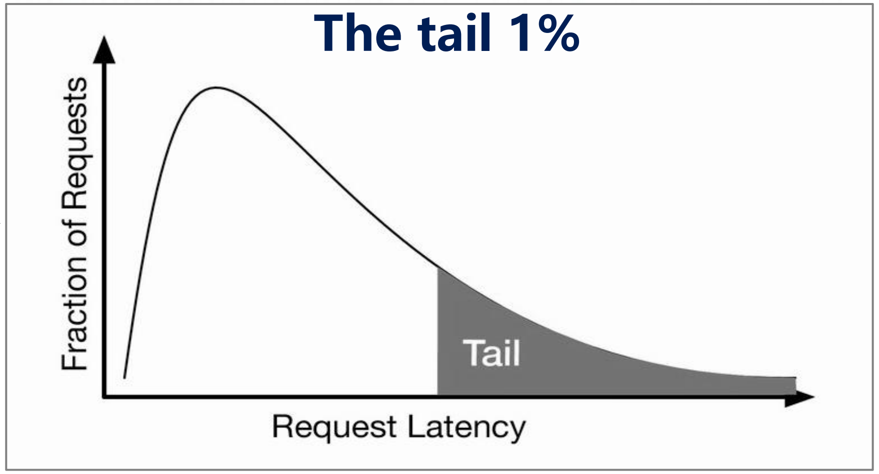
→ **interference**

Interference

The average latency



The tail latency



DOI:10.1145/2408776.2408794

Software techniques that tolerate latency variability are vital to building responsive large-scale Web services.

BY JEFFREY DEAN AND LUIZ ANDRÉ BARROSO

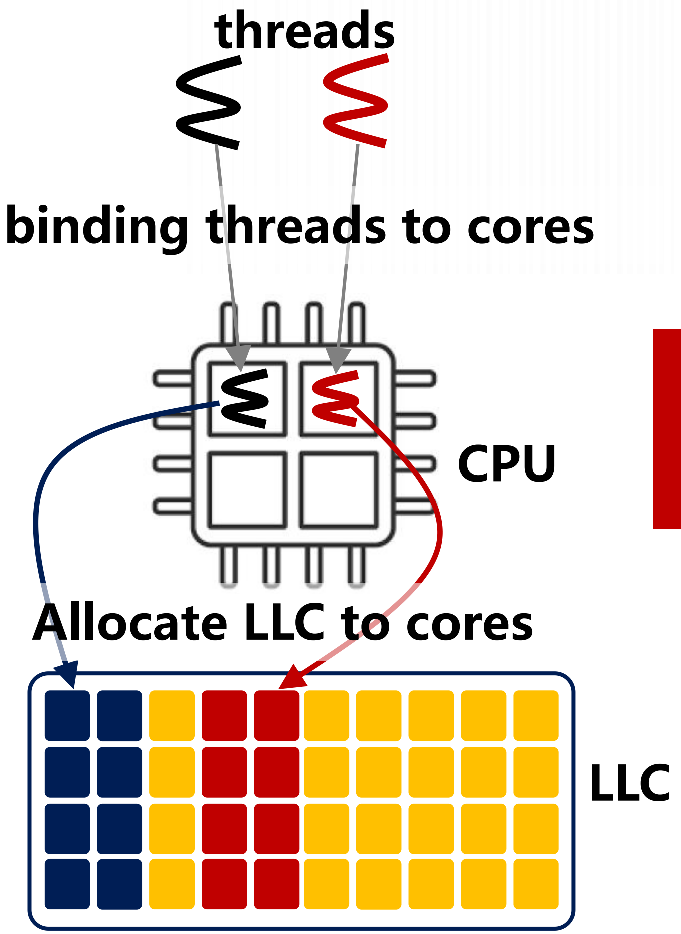
The Tail at Scale

[Jeffrey Dean2013]

- **Interference causes high tail latency.**

Cutting Tail Latency

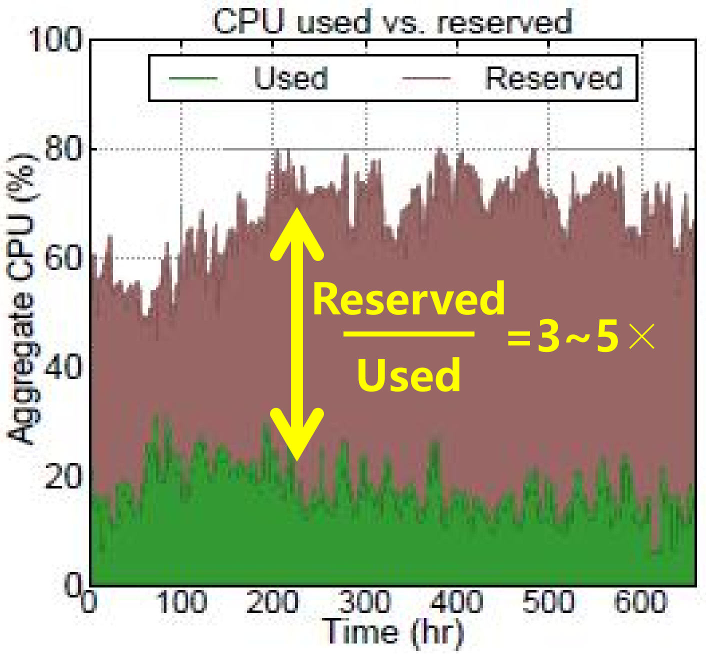
- **Physical isolation**



Core-level isolation

- **Intel RDT**

- **Software optimization**
  - VM + Container



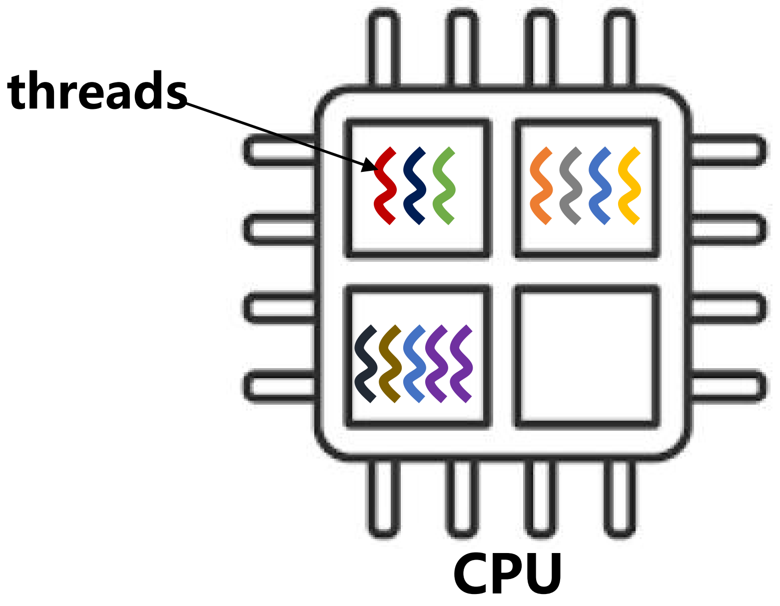
[Delimitrou and Kozyrakis, 2014]

Low resource efficiency

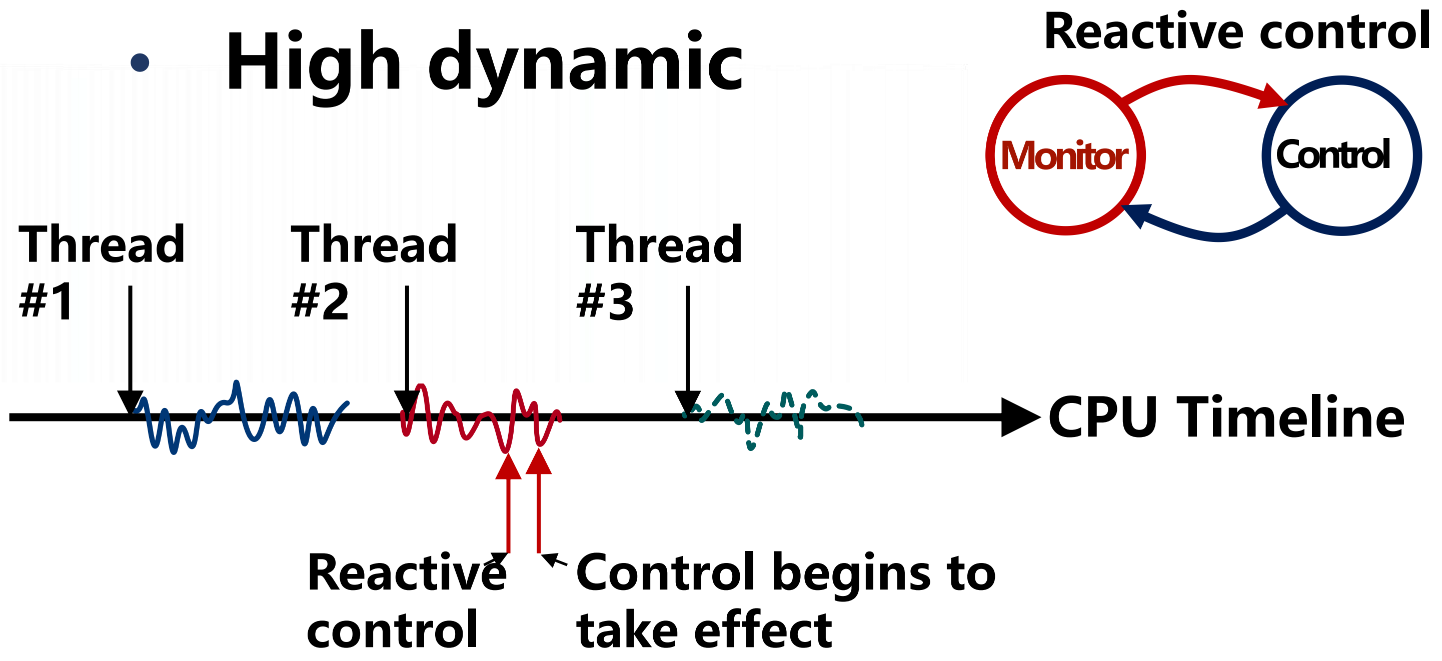
- **Overprovisioning**

Cutting Tail Latency

- High density

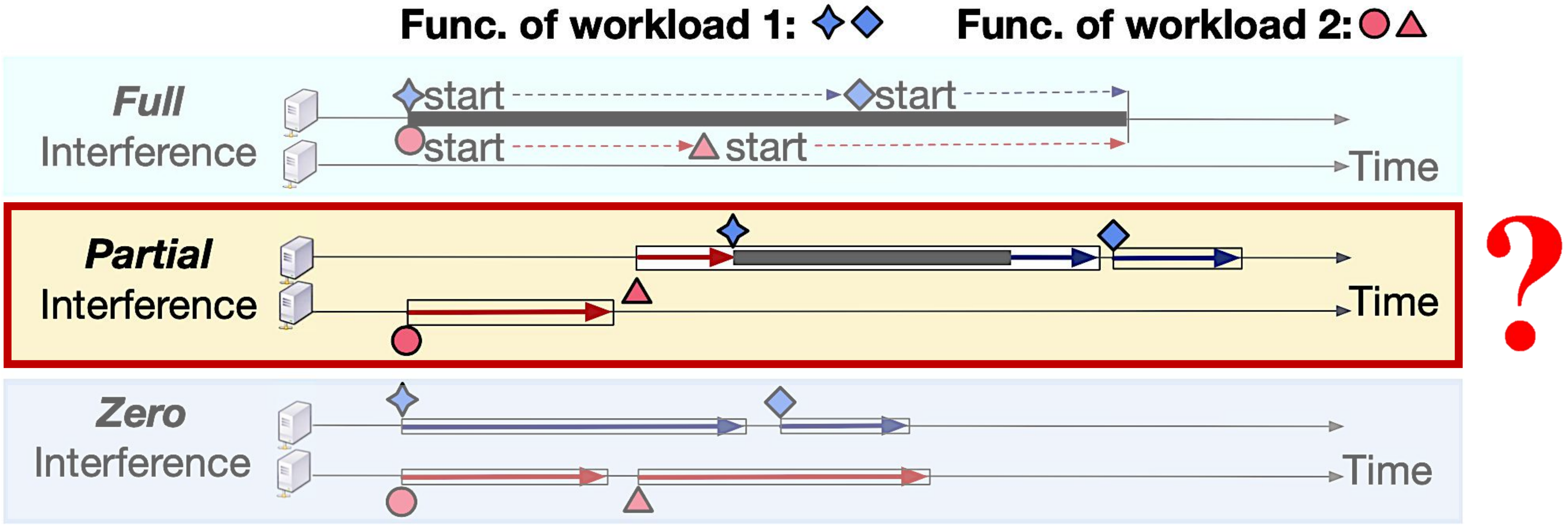


- High dynamic



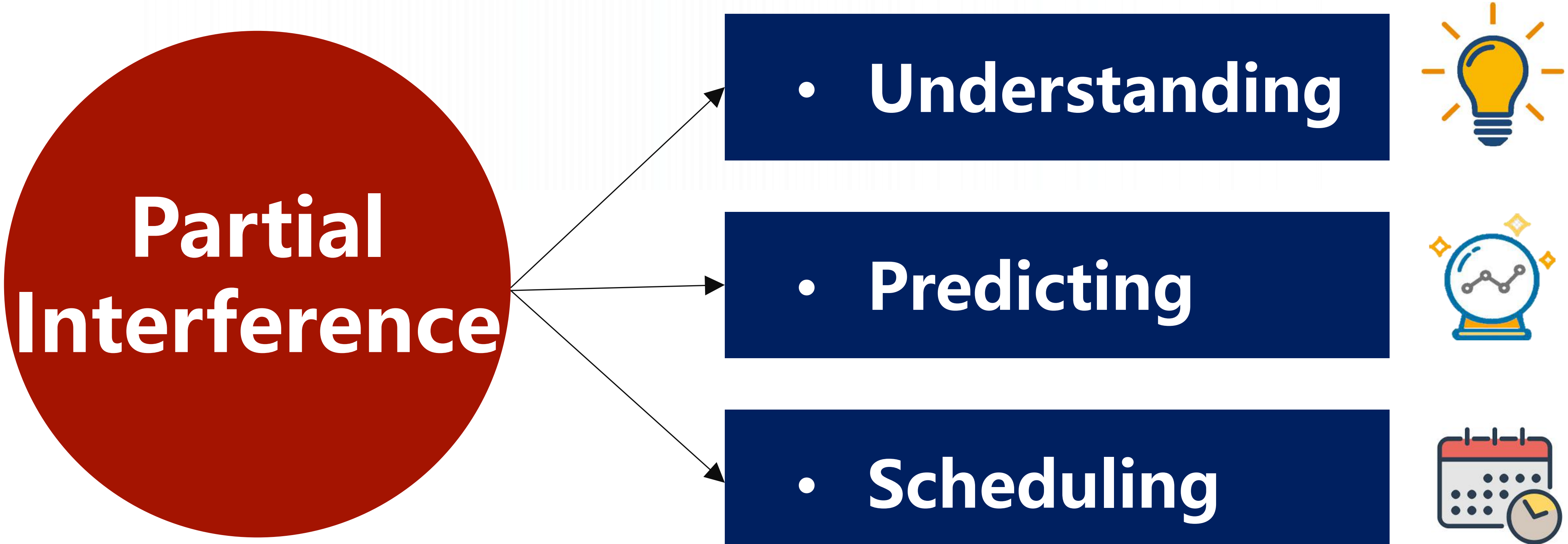
- Only **fine-grained** and **proactive control** can provide good performance and high throughput for serverless.

Partial Interference



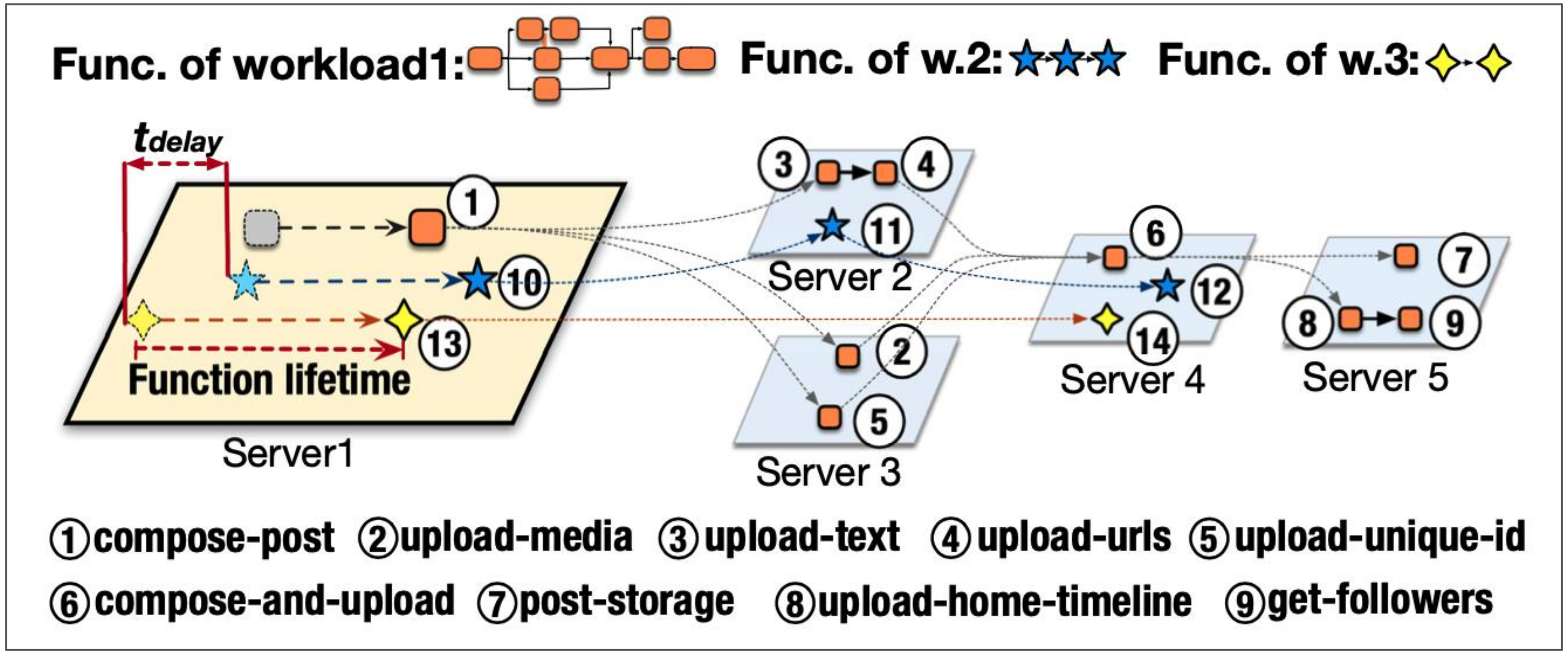
- **Interference occurs only at some parts, but not all, of the workloads.**

Partial Interference in Serverless



Partial Interference in Serverless

- LS: Social network (9 functions)



- BG/SC:
  - matrix multiplication,
  - video processing,
  - iperf,
  - dd,
  - LR,
  - Kmeans etc.

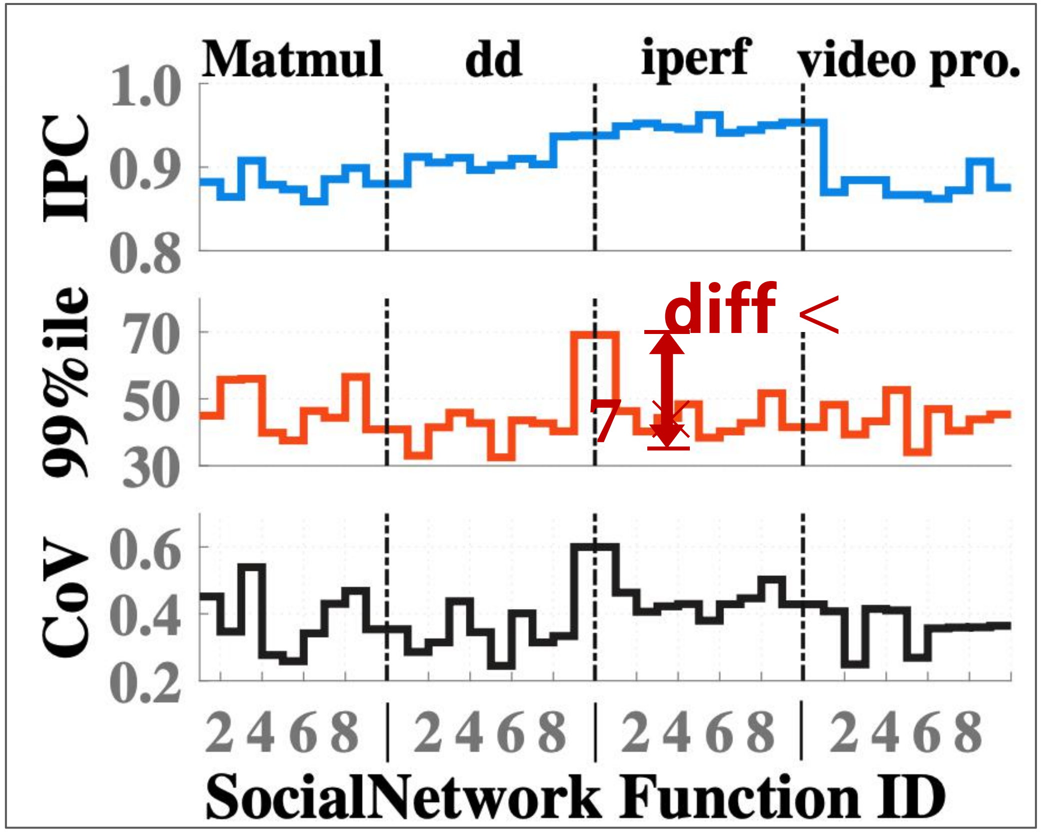
- **Serverless makes partial interference particularly severe.**

Understanding

1 High volatility

- Serverless functions are diverse in terms of execution behavior and resource consumption, making partial interference more volatile.

Partial interference scenarios



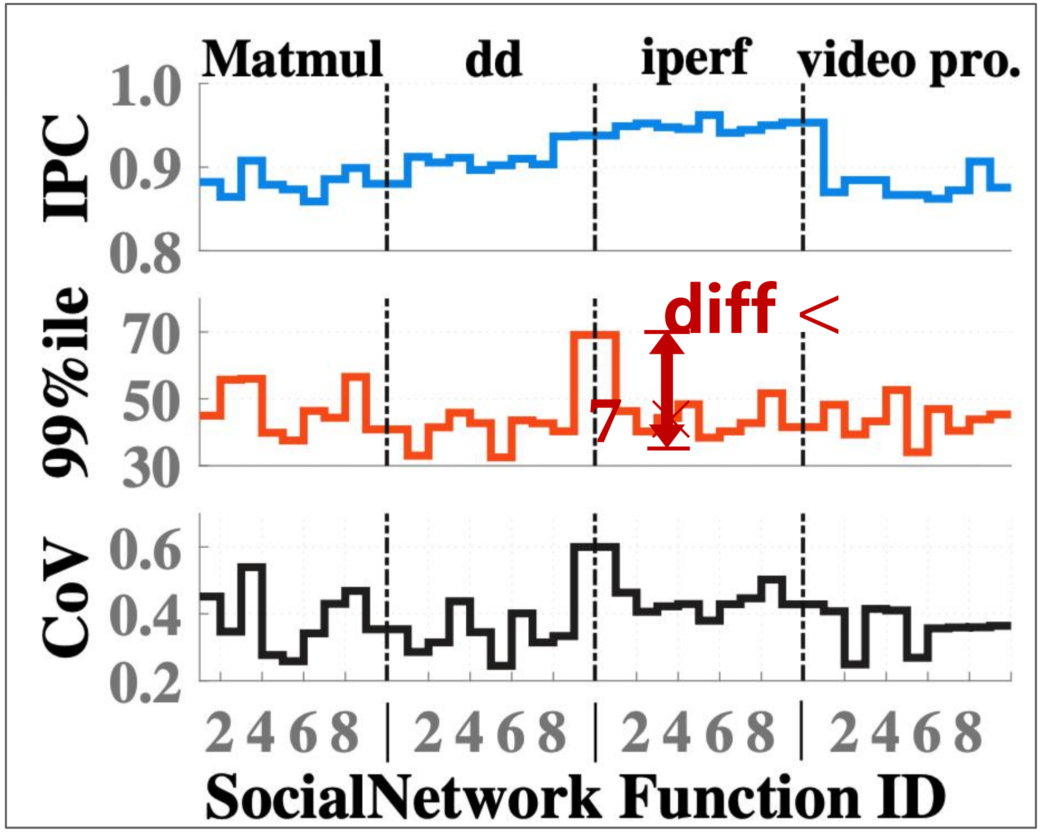
- Either as strong as *Full interference*, or as weak as Zero interference.
- The difference in the 99th percentile latency among these scenarios reaches 7×.

Understanding

2 Spatial variation

- Serverless functions are small in size and stateless, making partial interference spatially varied.

Partial interference scenarios

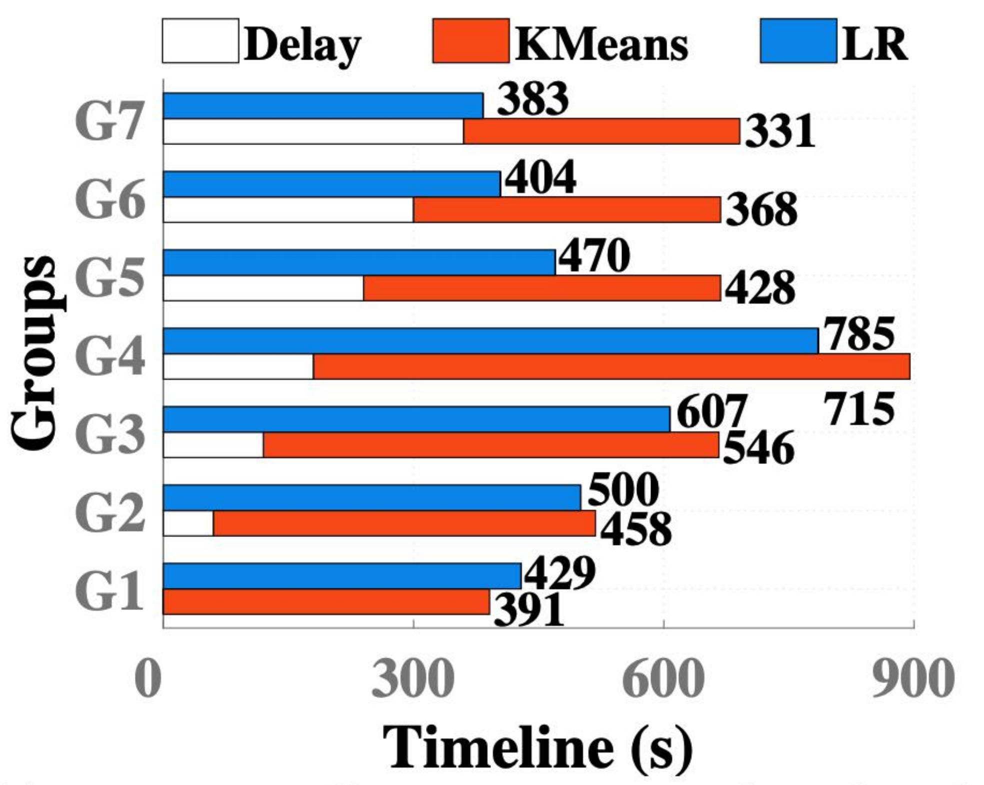


- Interference on the critical path (e.g., 1-2-6-8-9) generates a much more severe impact than interference on the non-critical path.

Understanding

3 Temporal variation

- Serverless functions are short-lived, making partial interference temporally varied.



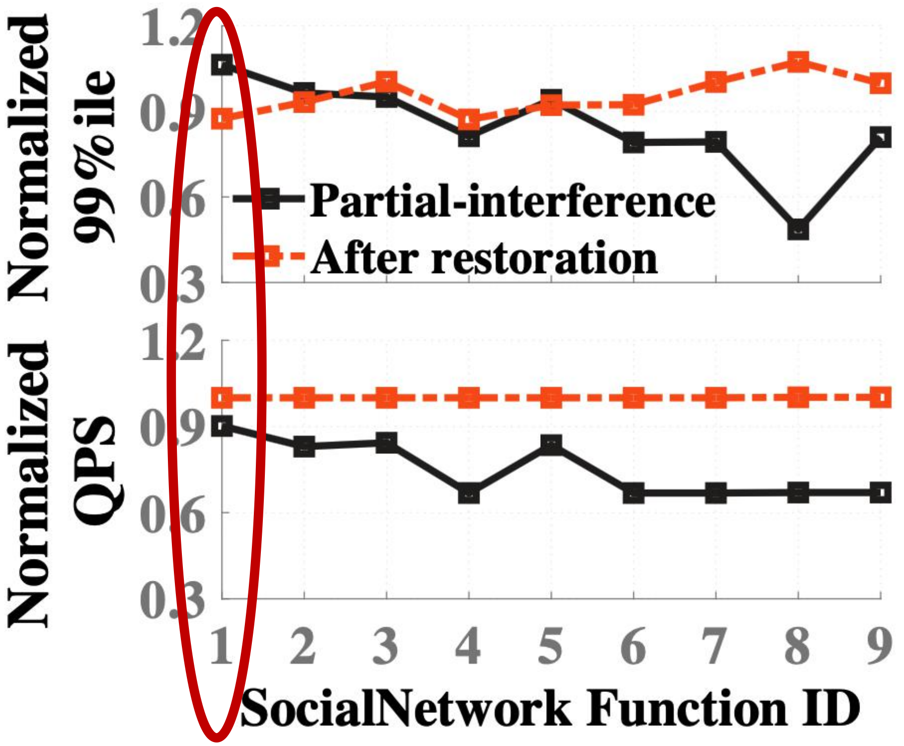
(b) Inter. at ⑥compose-and-upload

- The maximum difference in JCT of colocated Logistic Regression (LR) and Kmeans is more than 2×.

Understanding

4 Hotspot propagation

- Partial interference triggers a chain reaction across the function call path and leads to diametrically opposite effects.



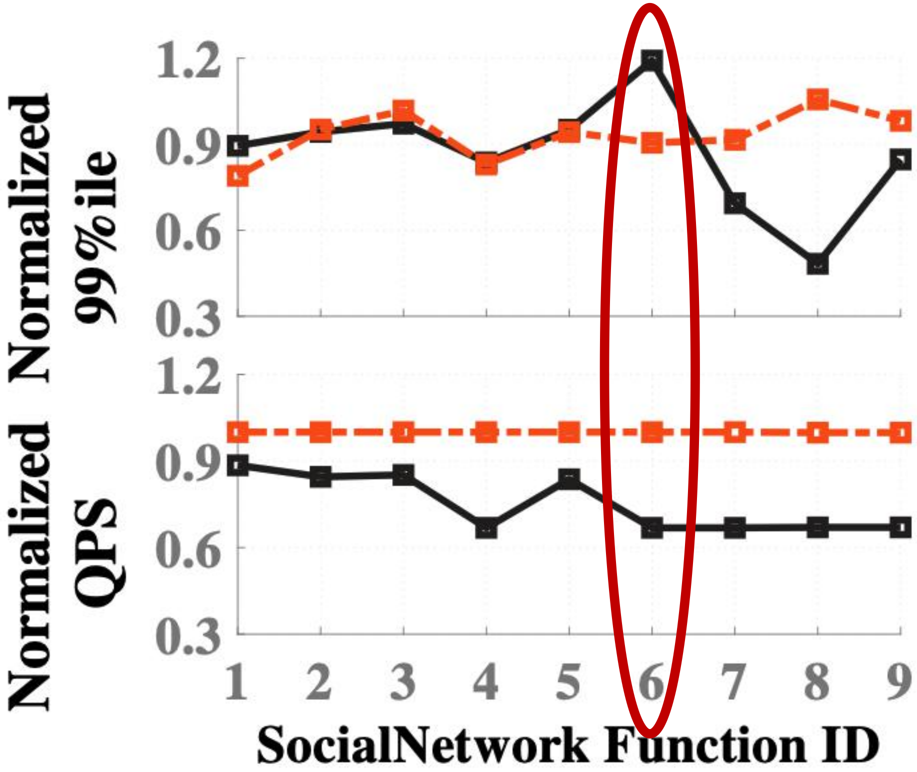
(a) Inter. at ①compose-post

- The QPS of the subsequent invoked functions decreases.
- The bottlenecked gateway degrades the invocation speeds of all other functions.

Understanding

5 Restoring propagation

- The local control of partial interference suffers from impact propagation.



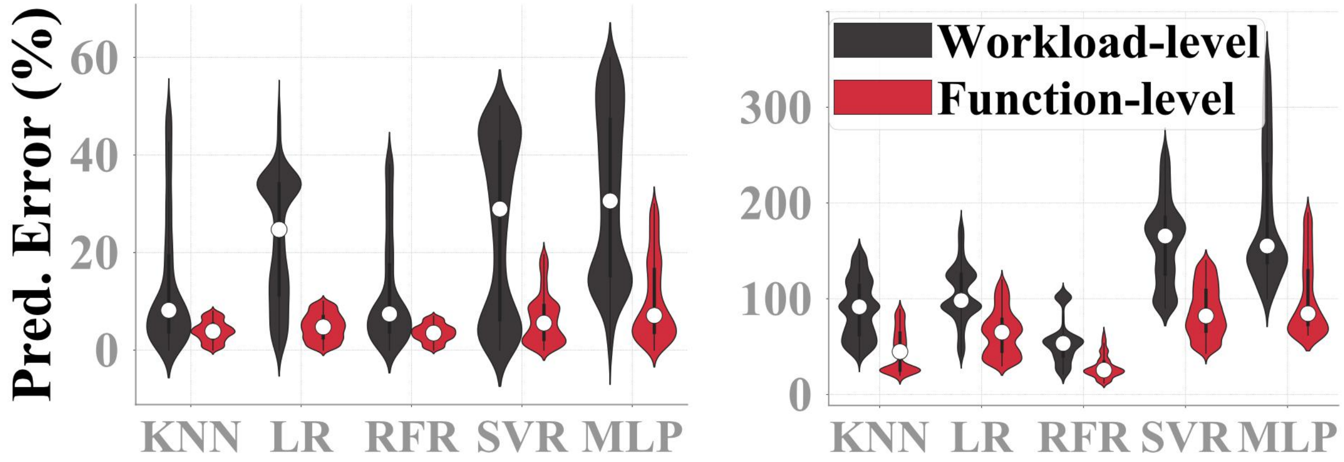
(b) Inter. at ⑥compose-and-upload

- Local interference control increase the other functions' latencies due to the restored invocations.

Predicting

6 Predictability in serverless

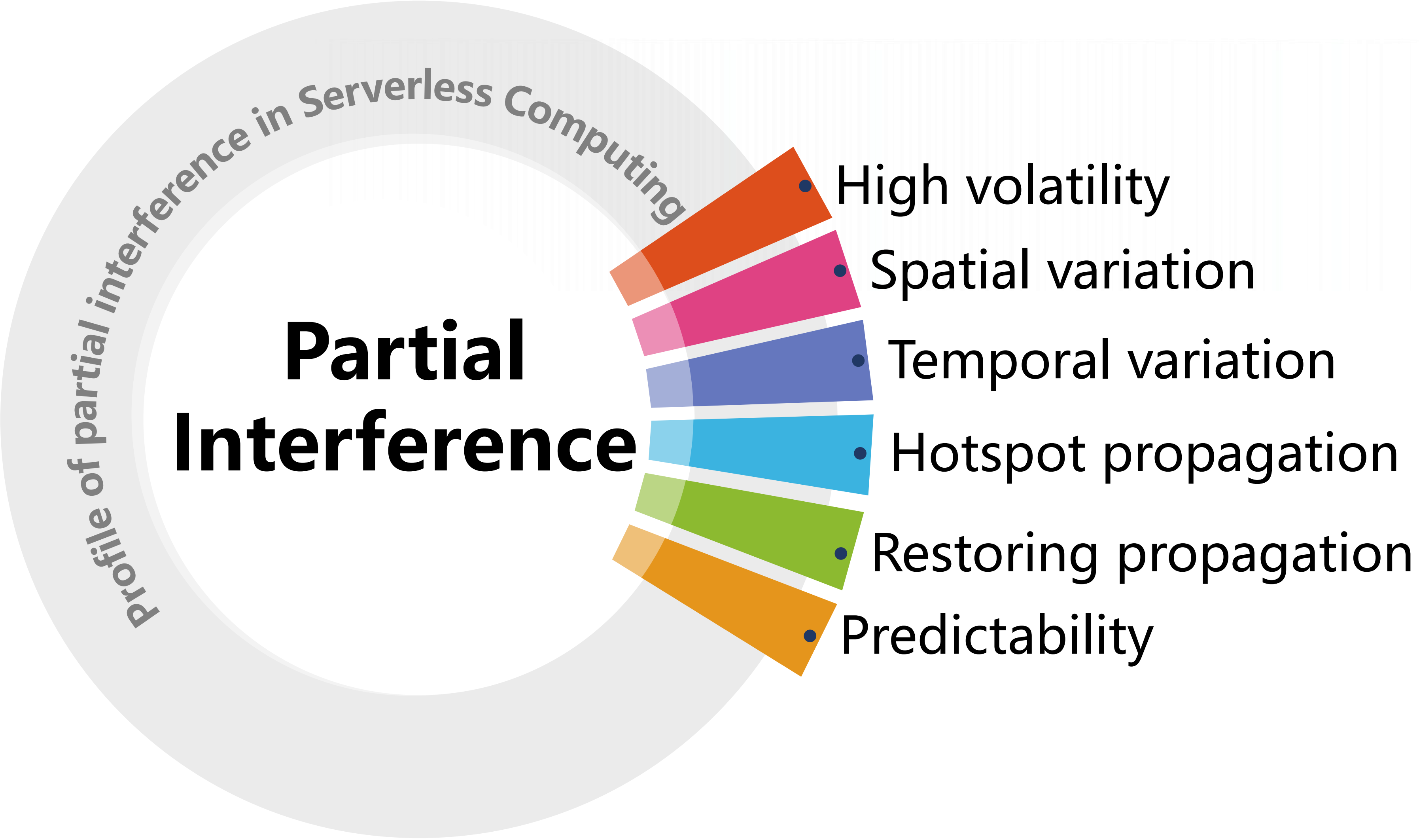
- Accurate partial interference prediction is enabled by function-level profiles, thereby improving the QoS of workloads.



(a) End-to-end ave. IPC (b) 99%ile latency

- **Function-level profiles** produce an average median that is 2× lower than that by **workload-level profiles**.

Predicting



Predicting

spatial-temporal interference aware

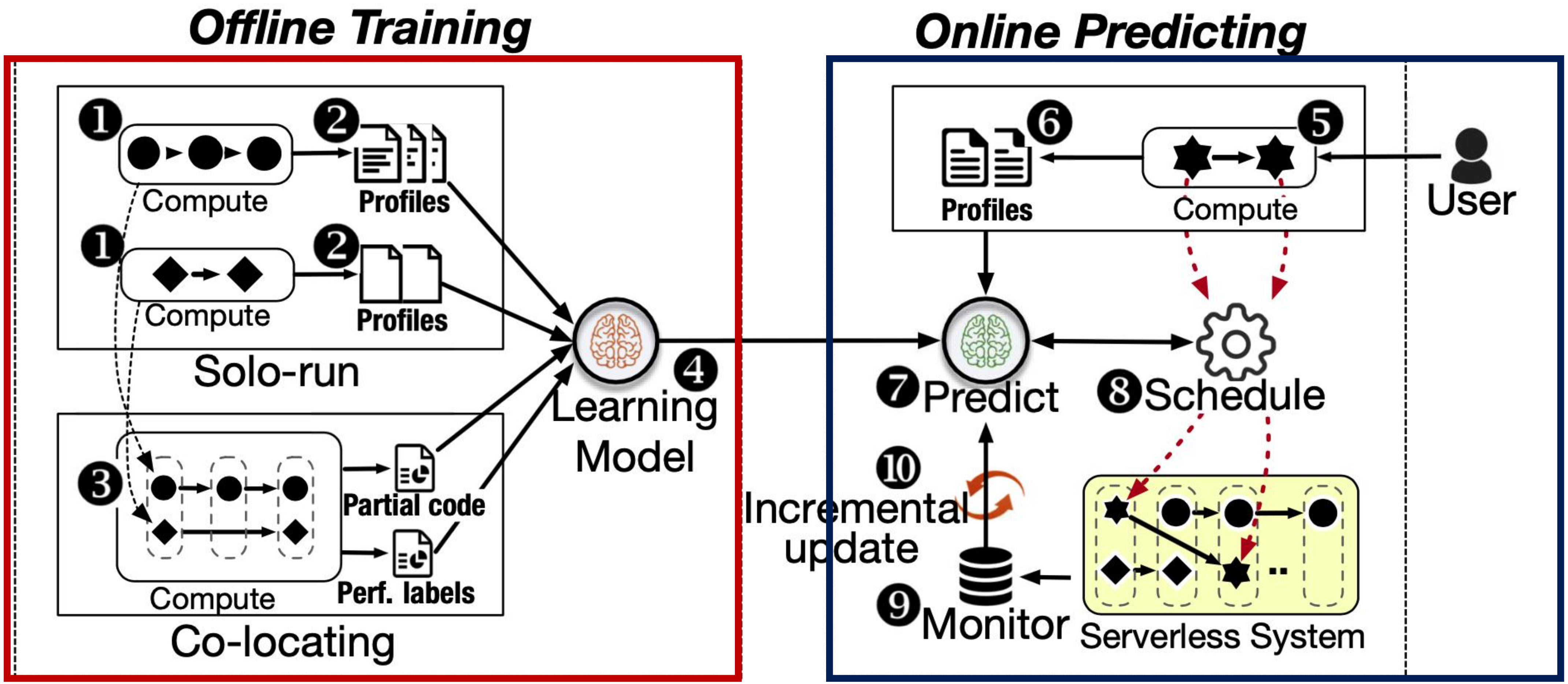
holistic method

Function profiles

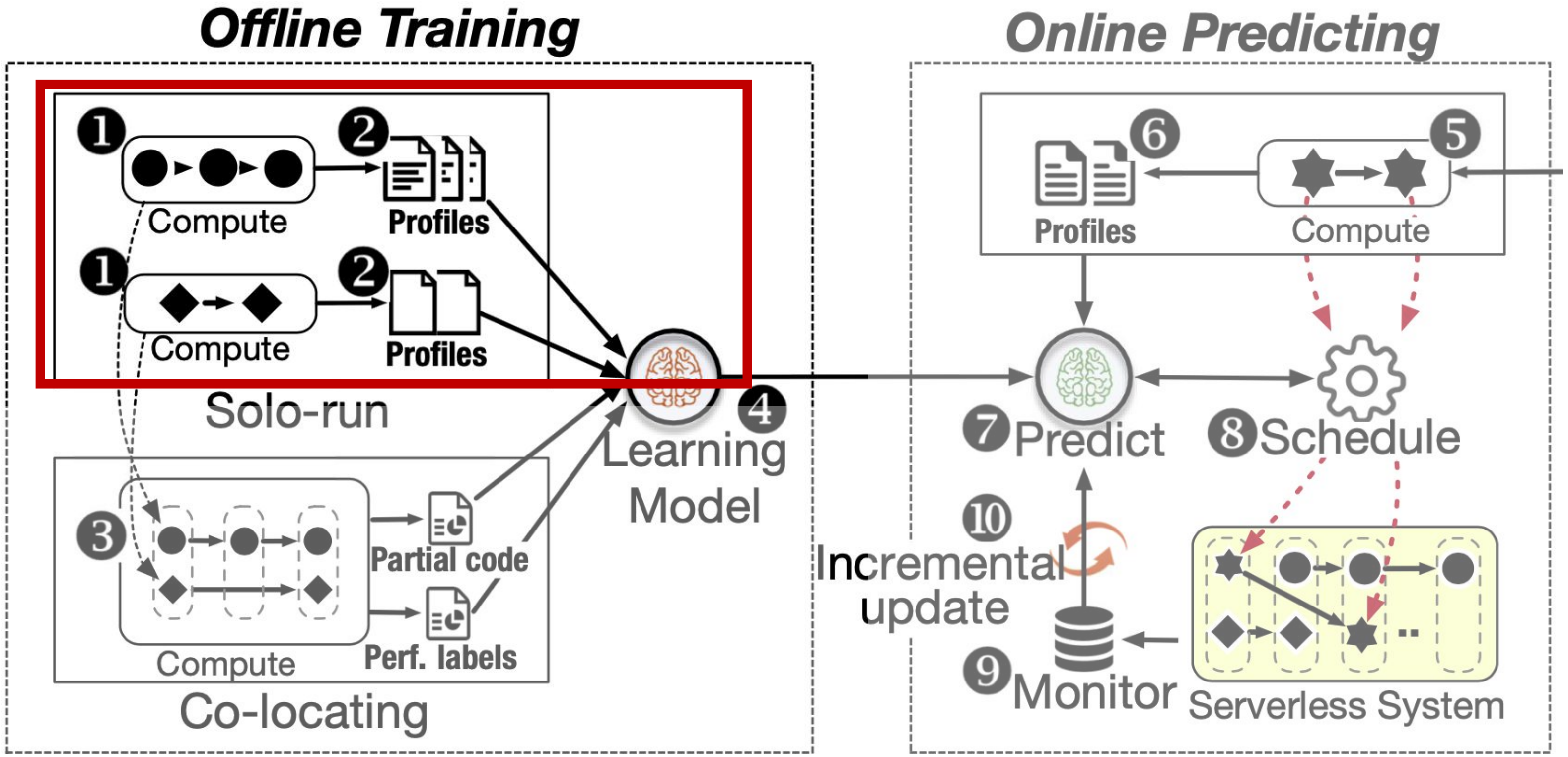
Gsight Predictor

- A **"spatial-temporal interference" - aware incremental learning predictor**, which can converge quickly by training on the **profiles of functions** along an **end-to-end call path**.

Gsight Predictor



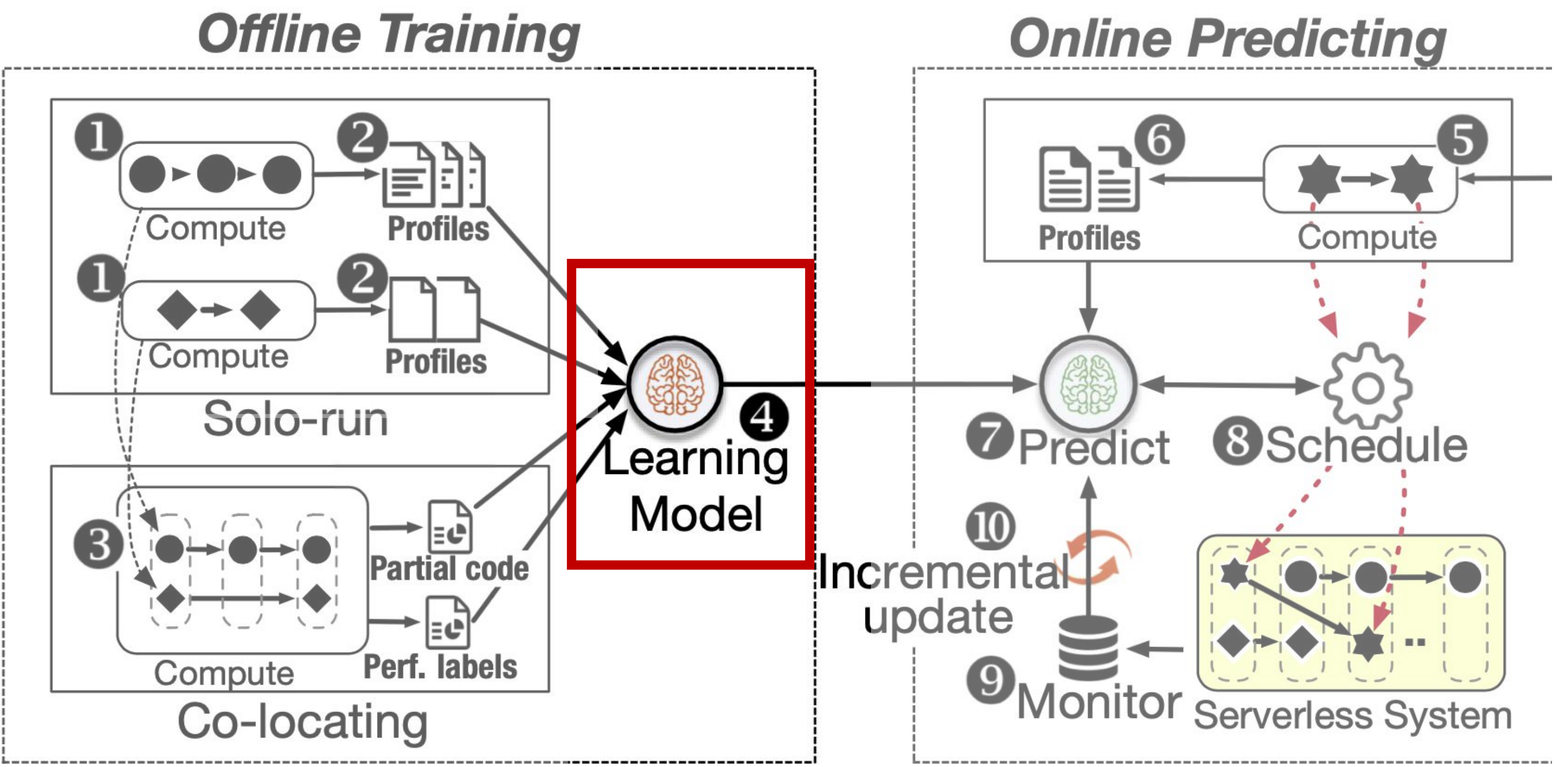
Gsight Predictor



- Function-level profiling
  - **solo-run way**
  - **non-intrusive**
    - system-layer
    - microarchitecture-layer

- 16 metrics: CPU, memory, network, I/O, branch MPKI, context-switches, LLC, CPU frequency, L2/L3 MPKI, etc.

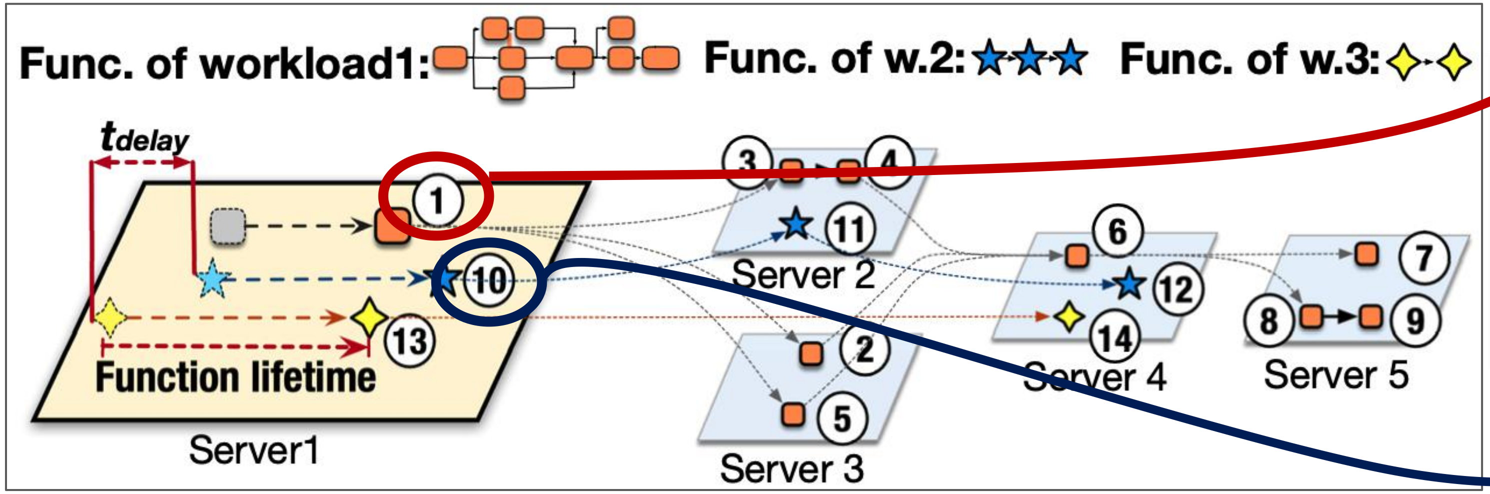
Gsight Predictor



- Incremental learning
  - **regression model**
  - **temporal overlap coding**
  - **spatial overlap coding**

$$P_{A\cup\{B,C,\ldots\}} = RM(\underbrace{R_A, R_B, R_C, \ldots}_{AllocatedRes}, \underbrace{U_A, U_B, U_C, \ldots}_{Utilization}, \underbrace{D_A, D_B, D_C, \ldots}_{Delay}, \underbrace{T_A, T_B, T_C, \ldots}_{Lifetime})$$

Gsight Predictor



# of rows = # of servers

$$\mathbf{U_1}\begin{vmatrix} u^{1}_{1,1} & .. & u^{16}_{1,1} \\ u^{1}_{1,\{34\}} & .. & u^{16}_{1,\{34\}} \\ u^{1}_{1,\{25\}} & .. & u^{16}_{1,\{25\}} \\ u^{1}_{1,6} & .. & u^{16}_{1,6} \\ u^{1}_{1,\{789\}} & .. & u^{16}_{1,\{789\}} \end{vmatrix}$$

$$\mathbf{U_2}\begin{vmatrix} u^{1}_{2,10} & .. & u^{16}_{2,10} \\ u^{1}_{2,11} & .. & u^{16}_{2,11} \\ 0 & .. & 0 \\ u^{1}_{2,12} & .. & u^{16}_{2,12} \\ 0 & ... & 0 \end{vmatrix}$$

- **Temporal and spatial overlap code**
  - **(D_2, D_3)** $= (0, t_{delay})$
  - **U_1, U_2.**

u^{k}_{ij}: the *k*th metric measured when workload *i* is deployed on server *j*.

Gsight Predictor

- Learning models
  - IKNN, IRFR, IMLP, ILR, ISVR, ESP [Mishra2017], Pythia [Xu2018]
  - The prediction error of IPC generated by **IRFR** is as low as **1.71%.**

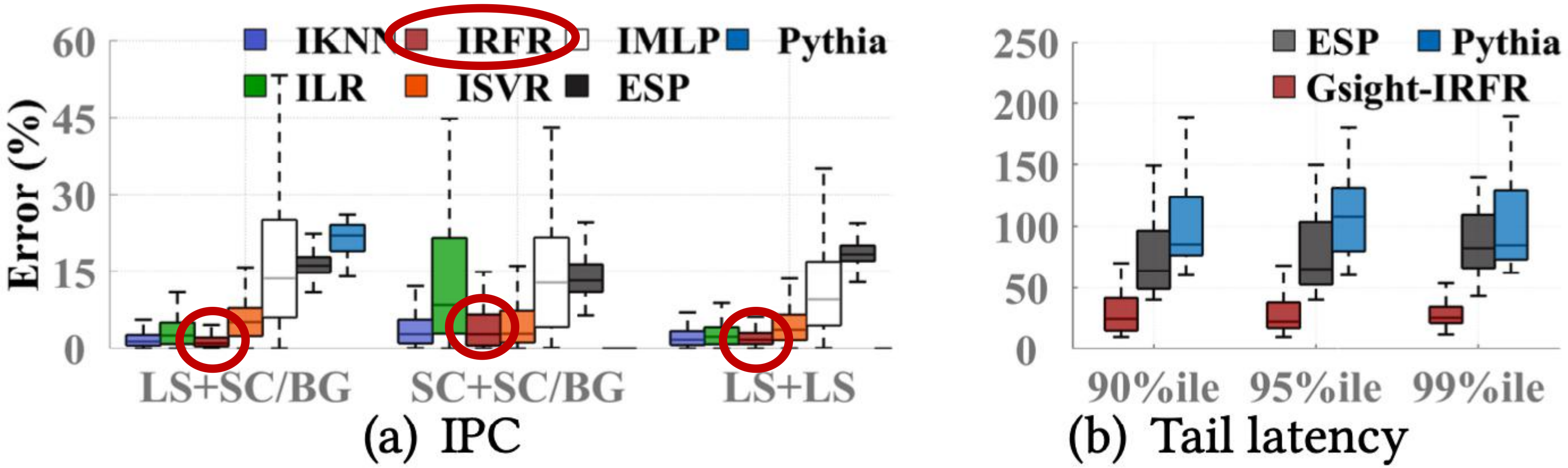
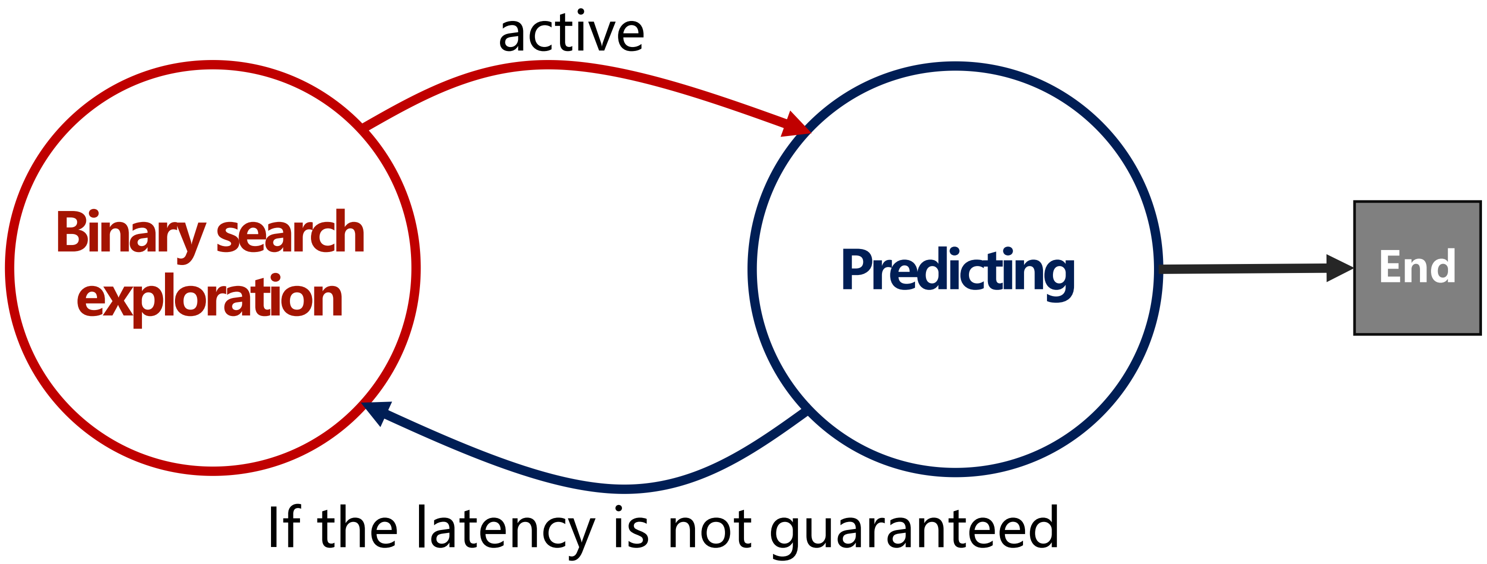


Figure 9. The prediction errors of (a) IPC and (b) tail latency.

Scheduling

- A simple binary search scheduling algorithm
  - maximize resource efficiency, by deploying function instances on a min number of active servers, while guaranteeing the QoS of colocated workloads.
  - Start from the full overlap.



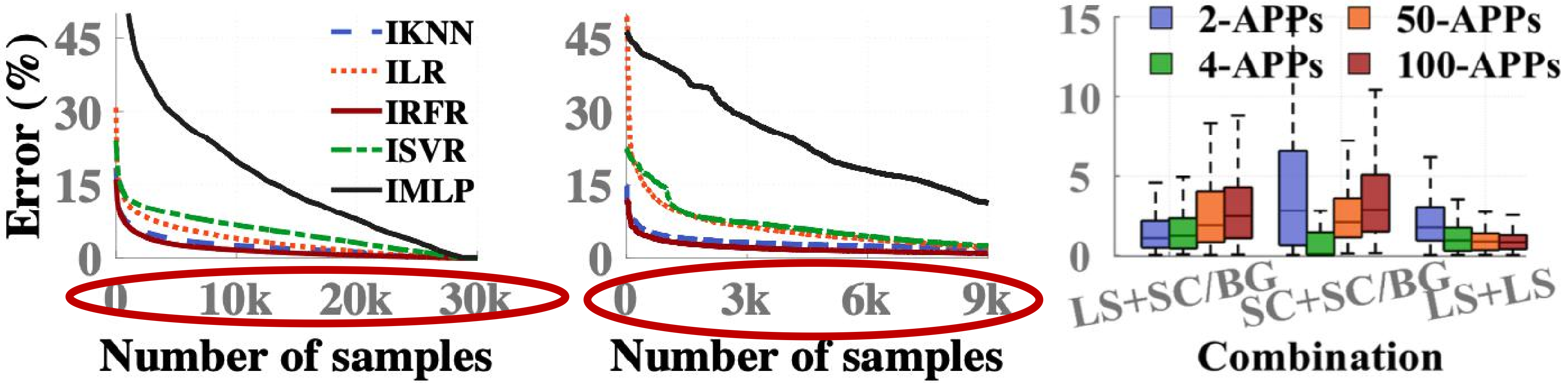
Experiments

- Workloads: [*Gsight and data available: https://github.com/tjulym/gray*]
  - BG/SC: ServerlessBench [Yu2020], FunctionBench [Kim2019];
  - LS: social network, e-commerce;
  - Azure trace [Shahrad2020]
- Testbed:
  - Openfaas

Component	Specification	Component	Specification
CPU model	Intel Xeon E7-4820v4	Shared LLC Size	25MB
Number of sockets	4	Memory Capacity	256GB
Processor BaseFreq.	2.00 GHz	Operating System	Ubuntu 14.04.5LTS
Threads	80 (40 physical cores)	SSD Capacity	960GB
Private L1&L2 Cache	64 KB and 256 KB	Number of Nodes	8

Experiments

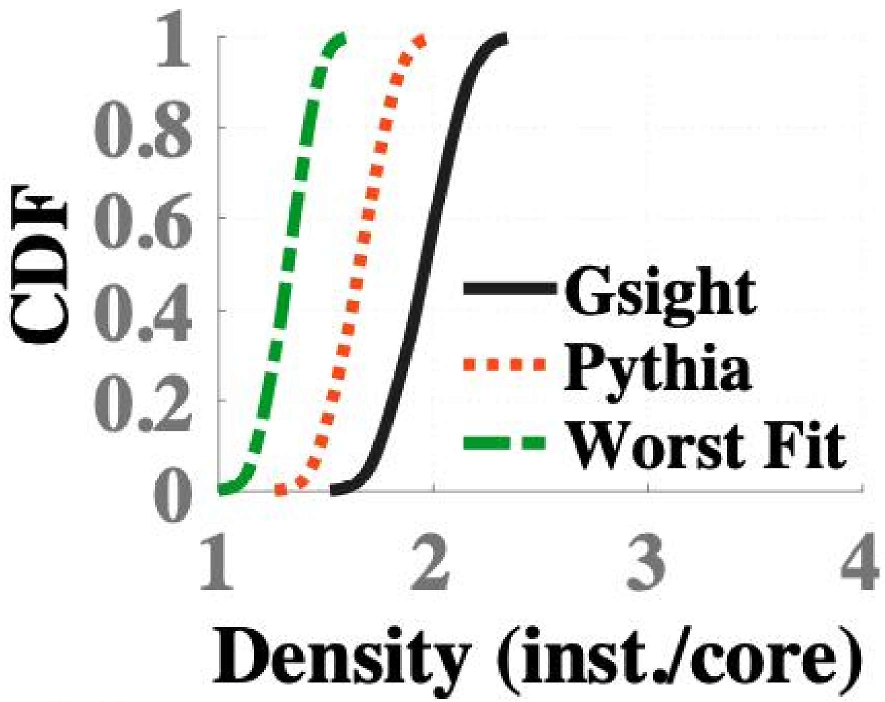
- **Fast and stable convergence**:
  - Function-level profiles enable Gsight to converge quickly. Its precision is also stable after convergence.
  - To achieve the same prediction error, Gsight-IRFR only requires 1/3 samples compared to serverful system.



(a) Serverful (b) Serverless (c) Multi-apps

Experiments

- Function density

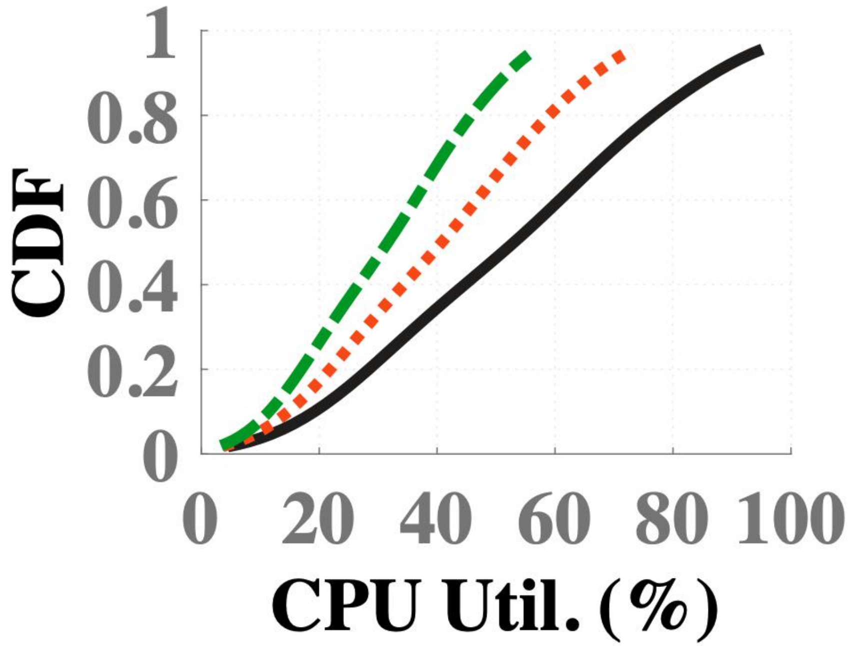


(a) Function density

- Gsight scheduling can improve the function density by an average of **18.79%** and by **48.48%** over those of *Pythia* and *Worst Fit*.

Experiments

- CPU utilization

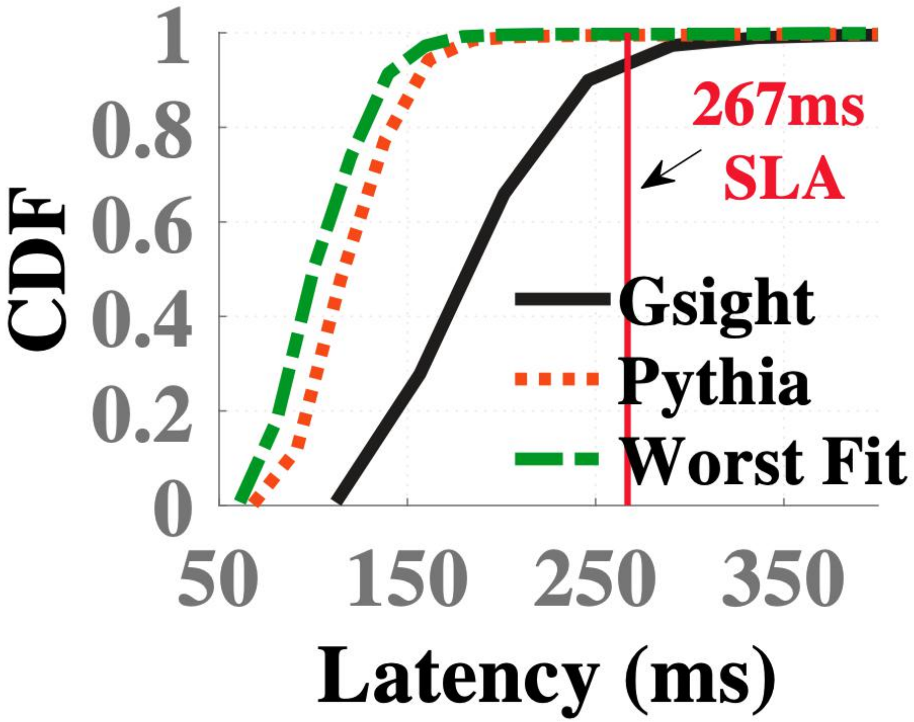


(b) CPU utilization

□Gsight scheduling can improve CPU utilization by **30.02%** and **67.51%** on average compared with that of *Pythia* and *Worst Fit*, respectively.

Experiments

- SLA guarantee

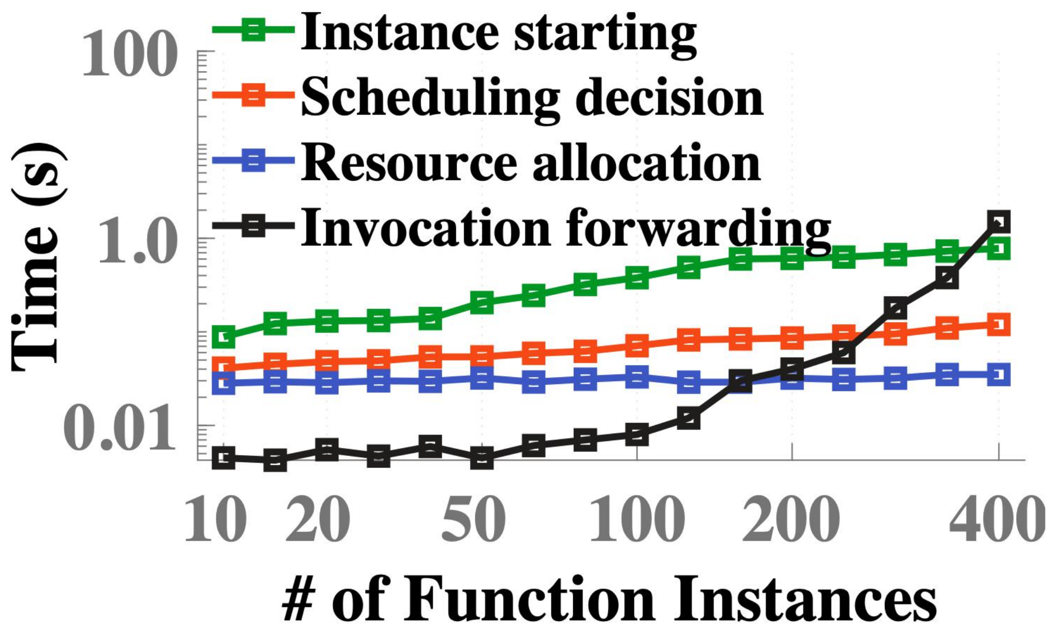


(a) Social network

- Gsight scheduling can guarantee the SLA of social network 95.39% of the time.

Experiments

- Overhead



- Training:
  - 3,000 samples takes us less than 2 person-hours, the training process takes only < 10 minutes.
- Gsight predicting takes 3.48 ms.
  - Cold start is slow.
  - A scalable gateway is also required.

Conclusion

- **From interference to partial interference, we are moving forward to understand more about tail latency.**
- **Serverless makes the resource management more challengeable.**
- **Only fine-grained and proactive control can provide good performance and high throughput for serverless.**
- **Gsight is just a start!**

Thank you